



CREATING SECURED MULTI-CLOUDS BY CONTINUOUS AUTOMATED AUDITING USING TERNARY HASH TREE IN AWS

A PROJECT REPORT

Submitted by

JAYALAKSHMI S [REGISTER NO:211417104089]

SAI VEENA K [REGISTER NO:211417104122]

HARINI A [REGISTER NO:211417104503]

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2021

BONAFIDE CERTIFICATE

Certified that this project report “**CREATING SECURED MULTI-CLOUDS BY CONTINUOUS AUTOMATED AUDITING USING TERNARY HASH TREE IN AWS**” is the bonafide work of “**JAYALAKSHMI S [REGISTER NO: 211417104089], SAI VEEN K [REGISTER NO:211417104122], HARINI A [REGISTER NO:211417104503]**” who carried out the project work under my supervision.

SIGNATURE

Dr.S. MURUGAVALLI,

M.E., Ph.D.,

HEAD OF THE DEPARTMENT

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

SIGNATURE

Dr. KAVITHA SUBRAMANI,

M.E., Ph.D.,

SUPERVISOR

DEPARTMENT OF CSE,
PANIMALAR ENGINEERING COLLEGE,
NASARATHPETTAI,
POONAMALLEE,
CHENNAI-600 123.

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on.....

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P. CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors **Tmt.C. VIJAYARAJESWARI, Dr.C. SAKTHIKUMAR, M.E., Ph.D** and **Tmt. SARANYASREE SAKTHIKUMAR B.E., M.B.A.,** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K. Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr. S. MURUGAVALLI, M.E., Ph.D.,** for the support extended throughout the project.

We would like to thank our guide **Dr. KAVITHA SUBRAMANI M.E., Ph.D** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

NAME OF THE STUDENTS

JAYALAKSHMI S

SAI VEENA K

HARINI A

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	6
	LIST OF TABLES	7
	LIST OF FIGURES	7
	LIST OF SYMBOLS, ABBREVIATIONS	8
1.	INTRODUCTION	9
	1.1 Overview	10
	1.2 Problem Definition	11
2.	LITERATURE SURVEY	12
3.	SYSTEM ANALYSIS	20
	3.1 Existing System	21
	3.2 Proposed system	22
	3.3 Requirement Analysis and Specification	23
	3.3.1 Input Requirements	23
	3.3.2 Output Requirements	23
	3.3.3 Functional Requirements	23
	3.4 Hardware Environment	24
	3.5 Software Environment	24
4.	SYSTEM DESIGN	36
	4.1. ER diagram	37
	4.2 Data dictionary	38

CHAPTER NO.	TITLE	PAGE NO.
	4.3 Data Flow Diagram	39
	4.4 UML Diagrams	41
5.	SYSTEM ARCHITECTURE	46
	5.1 Architecture Overview	47
	5.2 Module Design Specification	49
	5.3 Program Design Language	51
6.	SYSTEM IMPLEMENTATION	61
	6.1 Client-side coding	62
	6.2 Server-side coding	69
7.	SYSTEM TESTING	81
	7.1 Unit Testing	82
	7.2 Integration Testing	82
	7.3 Test Cases & Reports	83
8.	CONCLUSION	87
	8.1 Conclusion and Future Enhancements	88
	APPENDICES	89
	A.1 Sample Screens	90
	A.2 Publications	97
	REFERENCES	98

ABSTRACT

There are countless choices for the way to store and access the data. There's the disc drive on the portable computer, the external disc drive you utilize for backing up and transferring information, network file shares, USB drives, and a lot of. With numerous storage choices accessible, what makes cloud storage more captivating is that files are accessed and altered with ease. Accessing files from any device, anywhere is the key for cloud. Most of the organizations are moving to cloud storage to acquire advantage of the larger information availability, vital value savings, and information redundancy compared to conventional infrastructure. An important element of cloud data security is data integrity — preventing unauthorized modification or deletion and guaranteeing that knowledge remains unchanged since originally uploaded. This can be sometimes monitored by Third-party auditors appointed by the organizations. However, it can lead to high risks associated related to knowledge integrity which incorporates human errors(procrastination), corporate executive threats (the corruption of files), malicious intruders, compromised hardware, and configuration error, etc. This solution aims at resolving the preceding issues. In this work, we tend to plan a unique integrity verification framework in a multi-cloud atmosphere for securing cloud storage with the help of Ternary Hash Tree (THT) and Replica-based Ternary Hash Tree (R-THT). This helps to perform continuous data auditing at fixed time intervals which will be set by the admin. Differing from existing work, the auditing occurs at 3 levels-Block- level, File-level, and Replica-level with tree block ordering, storage block ordering for preserving data-integrity and guaranteeing data availableness within the cloud. The File recovery process will be carried out by the checker automatically if the information gets corrupted while checking. Users will cavil the cloud for file recovery. The outcomes depict that the planned secure cloud auditing framework is very secure and economical in storage, communication, and computation prices.

LIST OF TABLES

TABLE 1.1 USER	44
TABLE 1.2 ADMIN	44
TABLE 1.3 ADMIN LOGIN TEST REPORT	89
TABLE 1.4 USER LOGIN TEST REPORT	89
TABLE 1.5 FILE UPLOADING TEST REPORT	90
TABLE 1.6 AUDITING TEST REPORT	91

LIST OF FIGURES

Figure 1.1 WORKING FOR JAVA	32
Figure 1.2 JAVA VIRTUAL MACHINE	33
Figure 1.3 ER DIAGRAM	43
Figure 1.4 DATA FLOW DIAGRAM (LEVEL 0)	45
Figure 1.5 DATA FLOW DIAGRAM (LEVEL 1)	46
Figure 1.6 DATA FLOW DIAGRAM (LEVEL 2)	46
Figure 1.7 USE CASE DIAGRAM	48
Figure 1.8 SEQUENCE DIAGRAM	49
Figure 1.9 ACTIVITY DIAGRAM	50
Figure 2.0 COLLABORATION DIAGRAM	51
Figure 2.1 ARCHITECTURE DIAGRAM	54
Figure 2.2 MD5 ALGORITHM	58

Figure 2.3	BLOCK GENERATION	59
Figure 2.4	DYNAMIC BLOCK GENERATION ALGORITHM	60

LIST OF ABBREVIATIONS

IP	Internet Protocol
CSP	Cloud Service Provider
THT	Ternary Hash Tree
TPA	Third Party Auditor
JDK	Java Development ToolKit
DEX	Dalvik Executables
TCP	Transmission Control Protocol
HTTP	Hyper Text Transfer Protocol
ADT	Android Development Tool
FATFS	File Allocation Table File System

INTRODUCTION

CHAPTER 1

1.INTRODUCTION

1.1OVERVIEW

Cloud Computing enables the remote users to access data, services, and applications in on-demand from the shared pool of configurable computing resources, without the consideration of storage, hardware and software management. On the other hand, it is not easy for cloud users to identify whether Cloud Service Provider's (CSP) tag along with the data security legal expectations. So, cloud users could not rely on CSP's in terms of trust. So, it is significant to build a secure and efficient data auditing framework for increasing and maintaining cloud users trust with CSP. Researchers suggested introducing Third Party Auditor (TPA) on behalf of cloud user for verifying the outsourced data integrity, which may reduce the computation overhead of cloud users. In this work, we proposed a novel integrity verification framework for securing cloud storage based on Ternary Hash Tree (THT) and Replica based Ternary Hash Tree (R-THT), which will be used by TPA to perform data auditing. Differing from existing work, the proposed framework performs Block-level, File-level and Replica-level auditing with tree block ordering, storage block ordering for verifying the data integrity and ensuring data availability in the cloud. We further extend our framework to support error localization with data correctness, dynamic updates with block update, insert and delete operations in the cloud. The structure of THT and R-THT will reduce the computation cost and provide efficiency in data updates compared to the existing schemes. The security analysis of the proposed public auditing framework indicates the achievement of desired properties and performance has been evaluated with the detailed experiment set. The results show that the proposed secure cloud auditing

framework is highly secure and efficient in storage, communication and computation costs.

1.2 Problem Definition

In this new normal world, cloud storage is a boon for most of the organizations. Work from home option is possible only because of cloud storage. On the other hand, it is not easy for cloud users to identify whether Cloud Service Provider's (CSP) tag along with the data security legal expectations. So, cloud users could not rely on CSP's in terms of trust. A crucial component of cloud data security is data integrity — preventing unauthorized modification or deletion, and ensuring that data remains as it was when originally uploaded. This will be usually monitored by Third party auditor appointed by the organizations. But it may result in top risks related to data integrity which includes human errors(procrastination), insider threats (corruption of files), malicious intruders, compromised hardware and configuration error etc. And reliability of certificate given by a Third-party auditor may put in doubt due to its long validity period.

LITERATURE SURVEY

CHAPTER 2

2. LITERATURE SURVEY

[1] Remote data integrity checking is of crucial importance in cloud storage. It can make the clients verify whether their outsourced data is kept intact without downloading the whole data. In some application scenarios, the clients have to store their data on multi cloud servers. At the same time, the integrity checking protocol must be efficient in order to save the verifier's cost. From the two points, we propose a novel remote data integrity checking model: ID-DPDP (identity-based distributed provable data possession) in multi cloud storage. The formal system model and security model are given. Based on the bilinear pairings, a concrete ID-DPDP protocol is designed. The proposed ID-DPDP protocol is provably secure under the hardness assumption of the standard CDH (computational Diffie-Hellman) problem. In addition to the structural advantage of elimination of certificate management, our ID-DPDP protocol is also efficient and flexible. Based on the client's authorization, the proposed ID-DPDP protocol can realize private verification, delegated verification, and public verification.

[2] In cloud computing, data owners host their data on cloud servers and users (data consumers) can access the data from cloud servers. Due to the data outsourcing, however, this new paradigm of data hosting service also introduces new security challenges, which requires an independent auditing service to check the data integrity in the cloud. Some existing remote integrity checking methods can only serve for static archive data and, thus, cannot be applied to the auditing service since the data in the cloud can be dynamically updated. Thus, an efficient and secure dynamic auditing protocol is desired to convince data owners that the data are correctly stored in the cloud. In this paper, we first design an auditing framework for cloud storage systems and

propose an efficient and privacy-preserving auditing protocol. Then, we extend our auditing protocol to support the data dynamic operations, which is efficient and provably secure in the random oracle model. We further extend our auditing protocol to support batch auditing for both multiple owners and multiple clouds, without using any trusted organizer. The analysis and simulation results show that our proposed auditing protocols are secure and efficient, especially it reduces the computation cost of the auditor.

[3] A growing number of online service providers offer to store customers' photos, email, the system backups, and other digital assets. Currently, customers cannot make informed decisions about the risk of losing data stored with any particular service provider, reducing their incentive to rely on these services. We argue that third-party auditing is important in creating an online service-oriented economy, because it allows customers to evaluate risks, and it increases the efficiency of insurance-based risk mitigation. We describe approaches and system hooks that support both internal and external auditing of online storage services, describe motivations for service providers and auditors to adopt these approaches, and list challenges that need to be resolved for such auditing to become a reality.

[4] We introduce a model for provable data possession (PDP) that allows a client that has stored data at an untrusted server to verify that the server possesses the original data without retrieving it. The model generates probabilistic proofs of possession by sampling random sets of blocks from the server, which drastically reduces I/O costs. The client maintains a constant amount of metadata to verify the proof. The challenge/response protocol transmits a small, constant amount of data, which minimizes network communication. Thus, the PDP model for remote data checking supports large data sets in widely-distributed storage systems. We present

two provably-secure PDP schemes that are more efficient than previous solutions, even when compared with schemes that achieve weaker guarantees. In particular, the overhead at the server is low (or even constant), as opposed to linear in the size of the data. Experiments using our implementation verify the practicality of PDP and reveal that the performance of PDP is bounded by disk I/O and not by cryptographic computation.

[5] A growing number of online services, such as Google, Yahoo!, and Amazon, are starting to charge users for their storage. Customers often use these services to store valuable data such as email, family photos and videos, and disk backups. Today, a customer must entirely trust such external services to maintain the integrity of hosted data and return it intact. Unfortunately, no service is infallible. To make storage services accountable for data loss, we present protocols that allow a third-party auditor to periodically verify the data stored by a service and assist in returning the data intact to the customer. Most importantly, our protocols are privacy-preserving, in that they never reveal the data contents to the auditor. Our solution removes the burden of verification from the customer, alleviates both the customers and storage services fear of data leakage, and provides a method for independent arbitration of data retention contracts.

[6] Using cloud storage, users can remotely store their data and enjoy the on-demand high-quality applications and services from a shared pool of configurable computing resources, without the burden of local data storage and maintenance. However, the fact that users no longer have physical possession of the outsourced data makes the data integrity protection in cloud computing a formidable task, especially for users with constrained computing resources. Moreover, users should be able to just use the cloud storage as if

it is local, without worrying about the need to verify its integrity. Thus, enabling public auditability for cloud storage is of critical importance so that users can resort to a third-party auditor (TPA) to check the integrity of outsourced data and be worry free. To securely introduce an effective TPA, the auditing process should bring in no new vulnerabilities toward user data privacy, and introduce no additional online burden to user. In this paper, we propose a secure cloud storage system supporting privacy-preserving public auditing. We further extend our result to enable the TPA to perform audits for multiple users simultaneously and efficiently. Extensive security and performance analysis show the proposed schemes are provably secure and highly efficient. Our preliminary experiment conducted on Amazon EC2 instance further demonstrates the fast performance of the design.

[7] In this paper, we propose a dynamic audit service for verifying the integrity of an untrusted and outsourced storage. Our audit service is constructed based on the techniques, fragment structure, random sampling, and index-hash table, supporting provable updates to outsourced data and timely anomaly detection. In addition, we propose a method based on probabilistic query and periodic verification for improving the performance of audit services. Our experimental results not only validate the effectiveness of our approaches, but also show our audit system verifies the integrity with lower computation overhead and requiring less extra storage for audit metadata.[7]

[8] With cloud data services, it is commonplace for data to be not only stored in the cloud, but also shared across multiple users. Unfortunately, the integrity of cloud data is subject to skepticism due to the existence of hardware/software failures and human errors. Several mechanisms have been designed to allow both data owners and public verifiers to efficiently audit

cloud data integrity without retrieving the entire data from the cloud server. However, public auditing on the integrity of shared data with these existing mechanisms will inevitably reveal confidential information-identity privacy-to public verifiers. In this paper, we propose a novel privacy-preserving mechanism that supports public auditing on shared data stored in the cloud. In particular, we exploit ring signatures to compute verification metadata needed to audit the correctness of shared data. With our mechanism, the identity of the signer on each block in shared data is kept private from public verifiers, who are able to efficiently verify shared data integrity without retrieving the entire file. In addition, our mechanism is able to perform multiple auditing tasks simultaneously instead of verifying them one by one. Our experimental results demonstrate the effectiveness and efficiency of our mechanism when auditing shared data integrity.

[9] Cloud Computing has been envisioned as the next-generation architecture of IT Enterprise. It moves the application software and databases to the centralized large data centers, where the management of the data and services may not be fully trustworthy. This unique paradigm brings about many new security challenges, which have not been well understood. This work studies the problem of ensuring the integrity of data storage in Cloud Computing. In particular, we consider the task of allowing a third party auditor (TPA), on behalf of the cloud client, to verify the integrity of the dynamic data stored in the cloud. The introduction of TPA eliminates the involvement of the client through the auditing of whether his data stored in the cloud are indeed intact, which can be important in achieving economies of scale for Cloud Computing. The support for data dynamics via the most general forms of data operation, such as block modification, insertion, and deletion, is also a significant step toward practicality, since services in Cloud Computing are

not limited to archive or backup data only. While prior works on ensuring remote data integrity often lacks the support of either public auditability or dynamic data operations, this paper achieves both. We first identify the difficulties and potential security problems of direct extensions with fully dynamic data updates from prior works and then show how to construct an elegant verification scheme for the seamless integration of these two salient features in our protocol design. In particular, to achieve efficient data dynamics, we improve the existing proof of storage models by manipulating the classic Merkle Hash Tree construction for block tag authentication. To support efficient handling of multiple auditing tasks, we further explore the technique of bilinear aggregate signature to extend our main result into a multiuser setting, where TPA can perform multiple auditing tasks simultaneously.

[10] Cloud storage enables users to remotely store their data and enjoy the on-demand high quality cloud applications without the burden of local hardware and software management. Though the benefits are clear, such a service is also relinquishing users' physical possession of their outsourced data, which inevitably poses new security risks toward the correctness of the data in cloud. In order to address this new problem and further achieve a secure and dependable cloud storage service, we propose in this paper a flexible distributed storage integrity auditing mechanism, utilizing the homomorphic token and distributed erasure-coded data. The proposed design allows users to audit the cloud storage with very lightweight communication and computation cost. The auditing result not only ensures strong cloud storage correctness guarantee, but also simultaneously achieves fast data error localization, i.e., the identification of misbehaving server. Considering the cloud data are dynamic in nature, the proposed design further supports secure and efficient dynamic operations on outsourced data, including block

modification, deletion, and append. Analysis shows the proposed scheme is highly efficient and resilient against Byzantine failure, malicious data modification attack, and even server colluding attacks.

SYSTEM ANALYSIS

CHAPTER 3

3. SYSTEM ANALYSIS

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could see it as the application of systems theory to product development.

3.1 EXISTING SYSTEM

In cloud computing, remote data integrity checking is an important security problem. The client's massive data is outside his control. The malicious cloud server may corrupt the client's data in order to gain more benefits. However, cloud services are part of an ever-changing environment, resulting from fast technology life cycles and inherent cloud computing (CC) characteristics, like on-demand provisioning and entangled supply chains. Hence, such long validity periods may put in doubt reliability of issued certifications. And also cloud service customers do not longer possess their data locally, assuring that their data is being correctly stored and integrity is maintained in cloud environments is of critical importance. Data integrity may be threatened by, for example, malicious insiders, data loss, technical failures, and by external attackers.

This work has its foundation from previous works where the data of the client has been stored in the cloud and accessed with ease without the need to download the entire data in order to check the integrity of the data. The system mentioned above relays on a third- party auditor, who continuously checks the integrity of the data. Auditing protocols put forward are reliable and coherent, especially it curbs the charge of auditor. All the protocols used are privacy-preserving which doesn't reveal the data to the auditor. It removes the burden of verification from the customer and eliminates the fear of data leakage. It involves dividing the data into blocks and storing it into multiple servers. Of all these advantages, the disadvantages including procrastination of auditing by the auditor which may

lead to data corruption and data loss. He may also get compromised by some other dealers which may lead to the trade of our private data. It also involves higher costs for the dedicated protocols (security related) used in the existing systems. Moreover, the system does not immediately offer any guarantee on data integrity and availability and also insufficient to detect data corruption.

3.2 PROPOSED SYSTEM

In Multi-Cloud environment, remote data integrity checking is required to secure user's data. User will upload file to Cloud. This file is split into blocks using Dynamic Block generation Algorithm. The Blocks are stored in Ternary Hash Tree (THT) format. The blocks have a parent's node and child node. File Allocation Table (FAT) File System has proper Indexing and Metadata for the different Chunks of the Cloud Storage. Here the auditor agrees to inspect logs, which are routinely created during monitoring operations by services providers to assess certification adherence. If Attacker corrupts data in Multi-Cloud, the continuous auditing process helps the verifier to perform Block and File level checking for remote data Integrity Checking using Verifiable Data Integrity Checking Algorithm. The auditing processes have a flow, first the parent block checking. If the parent block have any corrupted file then the child node auditing. If the child nodes have any corrupted file the File recovery is done by the Verifier automatically if the data gets corrupted during checking. Users can complaint cloud for file recovery.

Advantages:

- The proposed framework performs Block-level, File-level and Replica-level auditing with tree block ordering.
- Our framework to support error localization with data correctness, dynamic updates with block update, insert and delete operations in the cloud.

3.3 REQUIREMENT ANALYSIS AND SPECIFICATION

3.3.1 INPUT REQUIREMENTS:

- **User personal details.**
- **User Id and password to register.**
- **Files to be stored in cloud.**

3.3.2 OUTPUT REQUIREMENTS:

- **Auditing at given time intervals.**
- **File recovery, if corrupted.**

3.3.3 FUNCTIONAL REQUIREMENTS:

A **Functional Requirement (FR)** is a description of the service that the software must offer. It describes a software system or its component. A function is nothing but inputs to the software system, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional Requirements are also called **Functional Specification**.

In software engineering and systems engineering, a Functional Requirement can range from the high-level abstract statement of the sender's necessity to detailed mathematical functional requirement specifications. Functional software requirements help you to capture the intended behavior of the system.

Different types of functional requirements in an SRS document are:

- **Authentication functions**

The user will be authenticated based upon his username and password provided.

- **Authorization levels**

The user will be authorized based upon his username and password provided.

- **Audit tracking**

The files will be audited for the time intervals set by the admin.

- **Searching or reporting requirements:**

User can upload or download the document whenever he wants to access.

- **Algorithms**

MD 5Algorithm

Dynamic Block Generation Algorithm

Base 64 Algorithm

- **Backup and recovery**

In case the file uploaded by the user has been corrupted, then the automatic recovery of the corrupted files will be initiated.

3.4 HARDWARE ENVIRONMENT

- Hard Disk : 200GB and Above
- RAM : 2GB and Above
- Processor : i3 and Above

3.5 SOFTWARE ENVIRONMENT

- Windows XP and Above
- JDK 1.7
- Virtual Tomcat 6.0(AWS)
- Virtual Tomcat 7.0(AWS)

- MySQL 5.0
- J2EE (JSP, Servlet)
- Struts Framework
- JavaScript, Ajax, HTML, CSS, jQuery
- Web Services (JAX -WS), JSON.

3.5.1 JAVA

Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C.

3.5.1.1 INTRODUCTION TO JAVA

Java has been around since 1991, developed by a small team of Sun Microsystems developers in a project originally called the Green project. The intent of the project was to develop a platform-independent software technology that would be used in the consumer electronics industry. The language that the team created was originally called Oak.

The first implementation of Oak was in a PDA-type device called Star Seven (*7) that consisted of the Oak language, an operating system called GreenOS, a user interface, and hardware. The name *7 was derived from the telephone sequence that was used in the team's office and that was dialled in order to answer any ringing telephone from any other phone in the office.

Around the time the First-Person project was floundering in consumer electronics, a new craze was gaining momentum in America; the craze was called "Web surfing." The World Wide Web, a name applied to the Internet's millions

of linked HTML documents was suddenly becoming popular for use by the masses. The reason for this was the introduction of a graphical Web browser called Mosaic, developed by ncSA. The browser simplified Web browsing by combining text and graphics into a single interface to eliminate the need for users to learn many confusing UNIX and DOS commands. Navigating around the Web was much easier using Mosaic.

It has only been since 1994 that Oak technology has been applied to the Web. In 1994, two Sun developers created the first version of Hot Java, and then called Web Runner, which is a graphical browser for the Web that exists today. The browser was coded entirely in the Oak language, by this time called Java. Soon after, the Java compiler was rewritten in the Java language from its original C code, thus proving that Java could be used effectively as an application language. Sun introduced Java in May 1995 at the Sun World 95 convention.

Web surfing has become an enormously popular practice among millions of computer users. Until Java, however, the content of information on the Internet has been a bland series of HTML documents. Web users are hungry for applications that are interactive, that users can execute no matter what hardware or software platform they are using, and that travel across heterogeneous networks and do not spread viruses to their computers. Java can create such applications.

3.5.1.2 WORKING OF JAVA

For those who are new to object-oriented programming, the concept of a class will be new to you. Simplistically, a class is the definition for a segment of code that can contain both data (called attributes) and functions (called methods).

When the interpreter executes a class, it looks for a particular method by the name of **main**, which will sound familiar to C programmers. The main method

is passed as a parameter an array of strings (similar to the `argv []` of C), and is declared as a static method.

To output text from the program, we execute the **`println`** method of **`System.out`**, which is java's output stream. UNIX users will appreciate the theory behind such a stream, as it is actually standard output. For those who are instead used to the Wintel platform, it will write the string passed to it to the user's program.

Java consists of two things:

- Programming language
- Platform

3.5.1.3 THE JAVA PROGRAMMING LANGUAGE

Java is a high-level programming language that is all of the following:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable
- High-performance
- Multithreaded

➤ Dynamic

The code can bring about changes whenever felt necessary. Some of the standards needed to achieve the above-mentioned objectives are as follows:

Java is unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called **Java byte codes** – the platform independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how it works:

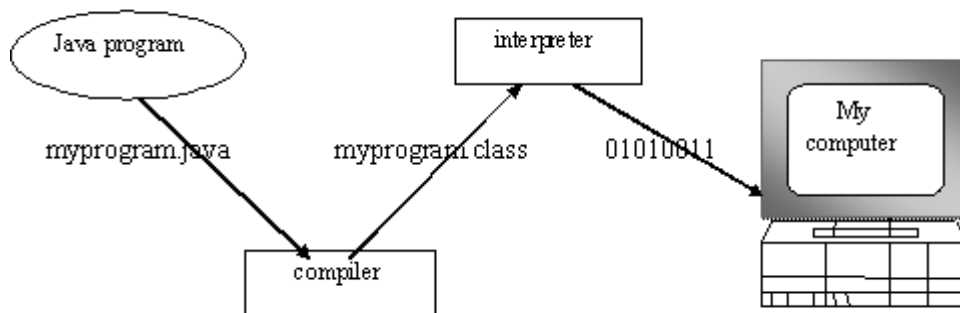


Figure 1.1 WORKING OF JAVA

You can think of Java byte codes as the machine code instructions for the **Java Virtual Machine (JVM)**. Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of JVM. That JVM can also be implemented in hardware. Java byte codes help make “write once, run anywhere” possible.

You can compile your Java program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the JVM. For example, that same Java program can be run on Windows NT, Solaris and Macintosh.

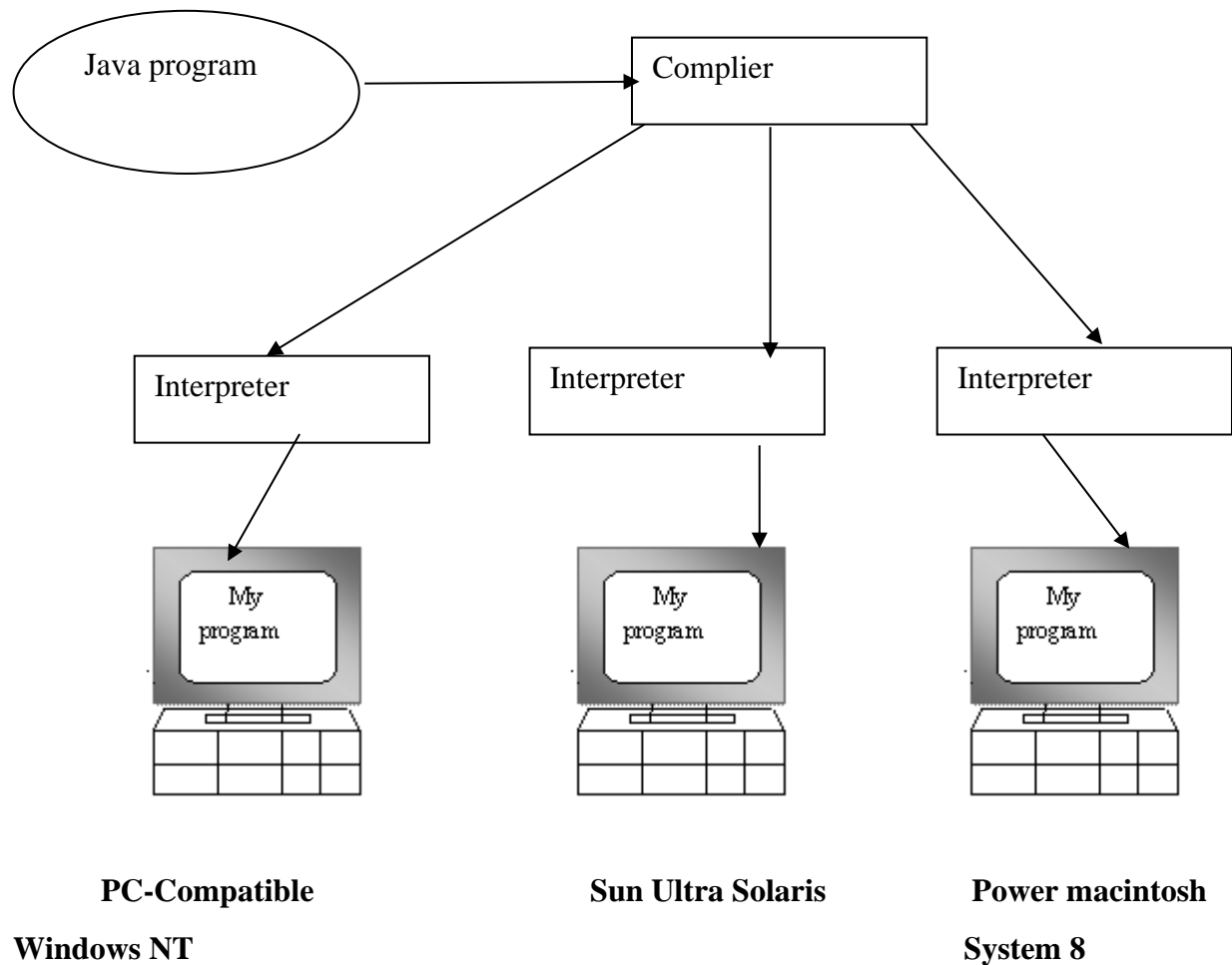


Figure 1.2 JAVA VIRTUAL MACHINE

3.5.1.4 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components :

- The Java Virtual Machine (JVM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the JVM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (**packages**) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, weel-tuned interpreters, and just-in-time byte compilers can bring Java's performance close to that of native code without threatening portability.

3.5.2 APACHE TOMCAT SERVER

Apache Tomcat (formerly under the Apache Jakarta Project; Tomcat is now a top-level project) is a web container developed at the Apache Software Foundation. Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Because Tomcat includes its own HTTP server internally, it is also considered a standalone web server.

Environment

Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets.

The Tomcat servlet engine is often used in combination with an Apache web server or other web servers. Tomcat can also function as an independent web

server. Earlier in its development, the perception existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception no longer exists; Tomcat is increasingly used as a standalone web server in high-traffic, high-availability environments.

Since its developers wrote Tomcat in Java, it runs on any operating system that has a JVM.

Product features

Tomcat 3.x (initial release)

- Implements the Servlet 2.2 and JSP 1.1 specifications
- servlet reloading
- basic HTTP functionality Tomcat 4.x
- implements the Servlet 2.3 and JSP 1.2 specifications
- servlet container redesigned as Catalina
- JSP engine redesigned as Jasper
- Coyote connector
- Java Management Extensions (JMX), JSP and Struts-based administration
- Tomcat 5.x
- implements the Servlet 2.4 and JSP 2.0 specifications
- reduced garbage collection, improved performance and scalability
- native Windows and Unix wrappers for platform integration
- faster JSP parsing

History

Tomcat started off as a servlet specification implementation by James Duncan Davidson, a software architect at Sun. He later helped make the project

open source and played a key role in its donation by Sun to the Apache Software Foundation.

Davidson had initially hoped that the project would become open-sourced and, since most open-source projects had O'Reilly books associated with them featuring an animal on the cover, he wanted to name the project after an animal. He came up with Tomcat since he reasoned the animal represented something that could take care of and fend for itself. His wish to see an animal cover eventually came true when O'Reilly published their Tomcat book with a tomcat on the cover.

3.5.3 Data Mining Introduction:

In general terms, “**Mining**” is the process of extraction of some valuable material from the earth e.g., coal mining, diamond mining etc. In the context of computer science, “**Data Mining**” refers to the extraction of useful information from a bulk of data or data warehouses. One can see that the term itself is a little bit confusing. In case of coal or diamond mining, the result of extraction process is coal or diamond. But in case of Data Mining, the result of extraction process is not data!! Instead, the result of data mining is the patterns and knowledge that we gain at the end of the extraction process. In that sense, Data Mining is also known as Knowledge Discovery or Knowledge Extraction.

Now a days, data mining is used in almost all the places where a large amount of data is stored and processed. For example, banks typically use ‘data mining’ to find out their prospective customers who could be interested in credit cards, personal loans or insurances as well. Since banks have the transaction details and detailed profiles of their customers, they analyze all this data and try to find out patterns which help them predict that certain customers could be interested in personal loans etc.

Main purpose of Data mining

Basically, the information gathered from Data Mining helps to predict hidden patterns, future trends and behaviors and allowing businesses to take decisions.

Technically, data mining is the computational process of analysing data from different perspective, dimensions, angles and categorizing/summarizing it into meaningful information.

Data Mining can be applied to any type of data e.g., Data Warehouses, Transactional Databases, Relational Databases, Multimedia Databases, Spatial Databases, Time-series Databases, World Wide Web.

Datamining as a whole process

The whole process of Data Mining comprises of three main phases:

1. Data Pre-processing – Data cleaning, integration, selection and transformation takes place.
2. Data Extraction – Occurrence of exact data mining
3. Data Evaluation and Presentation – Analysing and presenting results.

Why is data mining important?

So why is data mining important? You've seen the staggering numbers – the volume of data produced is doubling every two years. Unstructured data alone makes up 90 percent of the digital universe. But more information does not necessarily mean more knowledge.

Data mining allows you to:

- Sift through all the chaotic and repetitive noise in your data.
- Understand what is relevant and then make good use of that information to assess likely outcomes.

- Accelerate the pace of making informed decisions.

Learn more about data mining techniques in *Data Mining From A to Z*, a paper that shows how organizations can use predictive analytics and data mining to reveal new insights from data.

Data Mining Techniques

1. Classification:

This analysis is used to retrieve important and relevant information about data, and metadata. This data mining method helps to classify data in different classes.

2. Clustering:

Clustering analysis is a data mining technique to identify data that are like each other. This process helps to understand the differences and similarities between the data.

3. Regression:

Regression analysis is the data mining method of identifying and analysing the relationship between variables. It is used to identify the likelihood of a specific variable, given the presence of other variables.

4. Association Rules:

This data mining technique helps to find the association between two or more Items. It discovers a hidden pattern in the data set.

5. Outer detection:

This type of data mining technique refers to observation of data items in the dataset which do not match an expected pattern or expected behavior. This technique can be used in a variety of domains, such as intrusion, detection, fraud

or fault detection, etc. Outlier detection is also called Outlier Analysis or Outlier mining.

6. Sequential Patterns:

This data mining technique helps to discover or identify similar patterns or trends in transaction data for certain period.

7. Prediction:

Prediction has used a combination of the other data mining techniques like trends, sequential patterns, clustering, classification, etc. It analyzes past events or instances in a right sequence for predicting a future event.

Benefits of Data Mining:

- Data mining technique helps companies to get knowledge-based information.
- Data mining helps organizations to make the profitable adjustments in operation and production.
- The data mining is a cost-effective and efficient solution compared to other statistical data applications.
- Data mining helps with the decision-making process.
- Facilitates automated prediction of trends and behaviors as well as automated discovery of hidden patterns.
- It can be implemented in new systems as well as existing platforms
- It is the speedy process which makes it easy for the users to analyze huge amount of data in less time.

SYSTEM DESIGN

CHAPTER 4

4. SYSTEM DESIGN

4.1. ER diagram

An entity relationship model, also called an entity-relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.

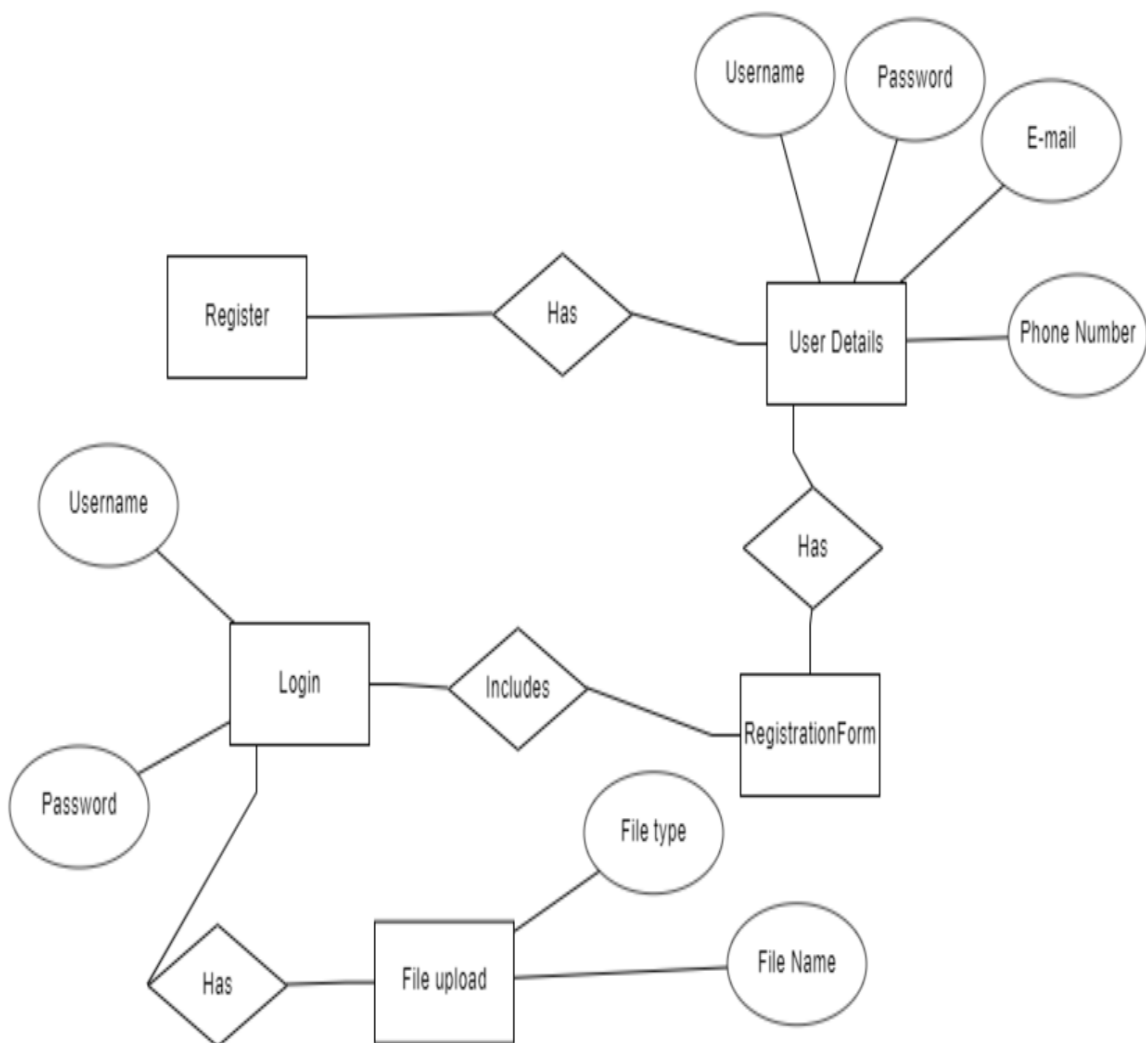


Figure 1.3 ER DIAGRAM

4.2 Data dictionary

A data dictionary, or metadata repository, as defined as a centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format. It is also defined as a collection of tables with metadata. The term can have one of several closely related meanings pertaining to databases. A document describing a database or collection of databases. An integral component of a DBMS that is required to determine its structure. A piece of middleware that extends or supplants the native data dictionary of a DBMS.

A. USER

COLUMN NAME	DATA TYPE	NULL VALUE ALLOWED
User Name	varchar (20)	No
Password	varchar (20)	No
Mail_ID	varchar (20)	No

TABLE 1.2

B. ADMIN

USER NAME	DATA TYPE	NULL VALUE ALLOWED
User Name	varchar (20)	No
Password	varchar (20)	No

TABLE 1.3

4.3 Data Flow Diagram

LEVEL 0:

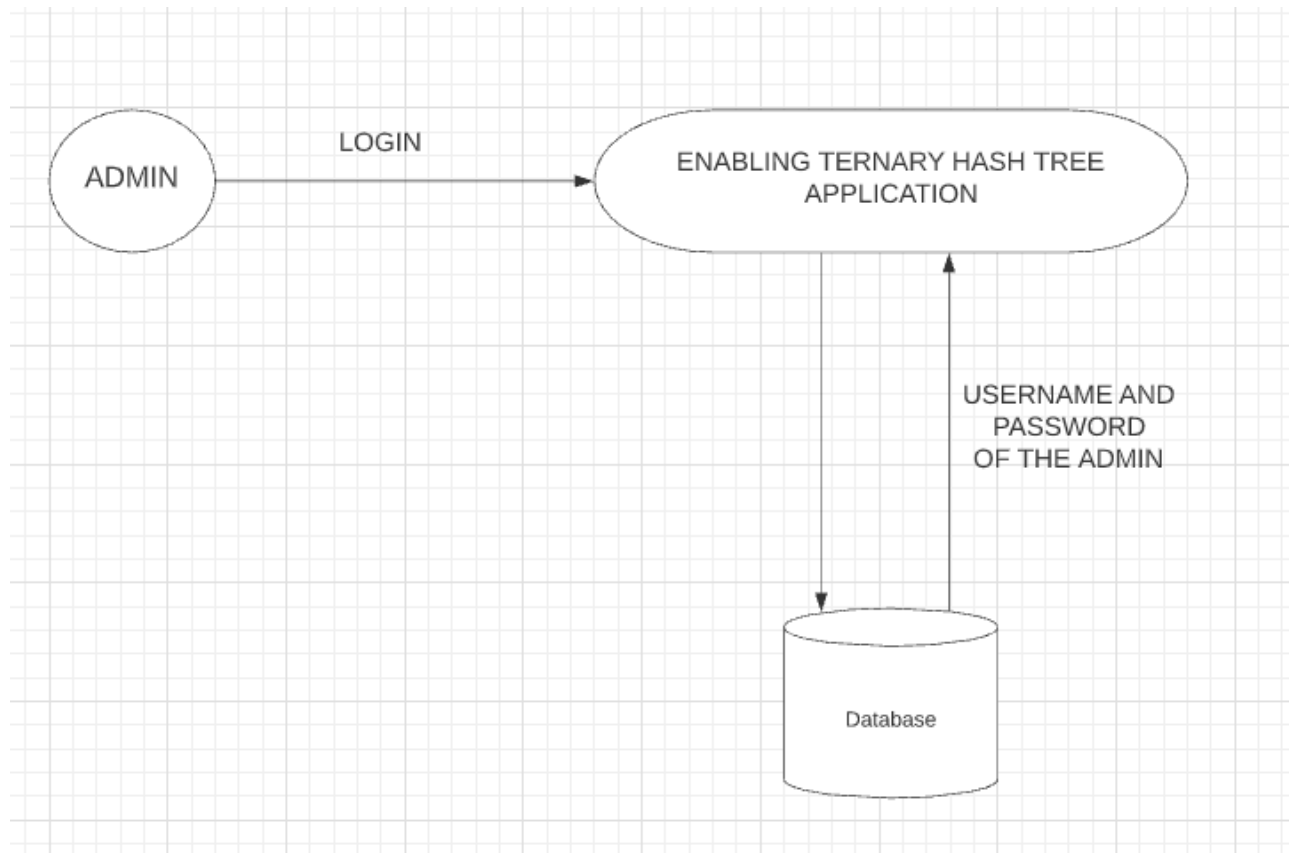


Figure 1.4 DFD (LEVEL 0)

LEVEL 1:

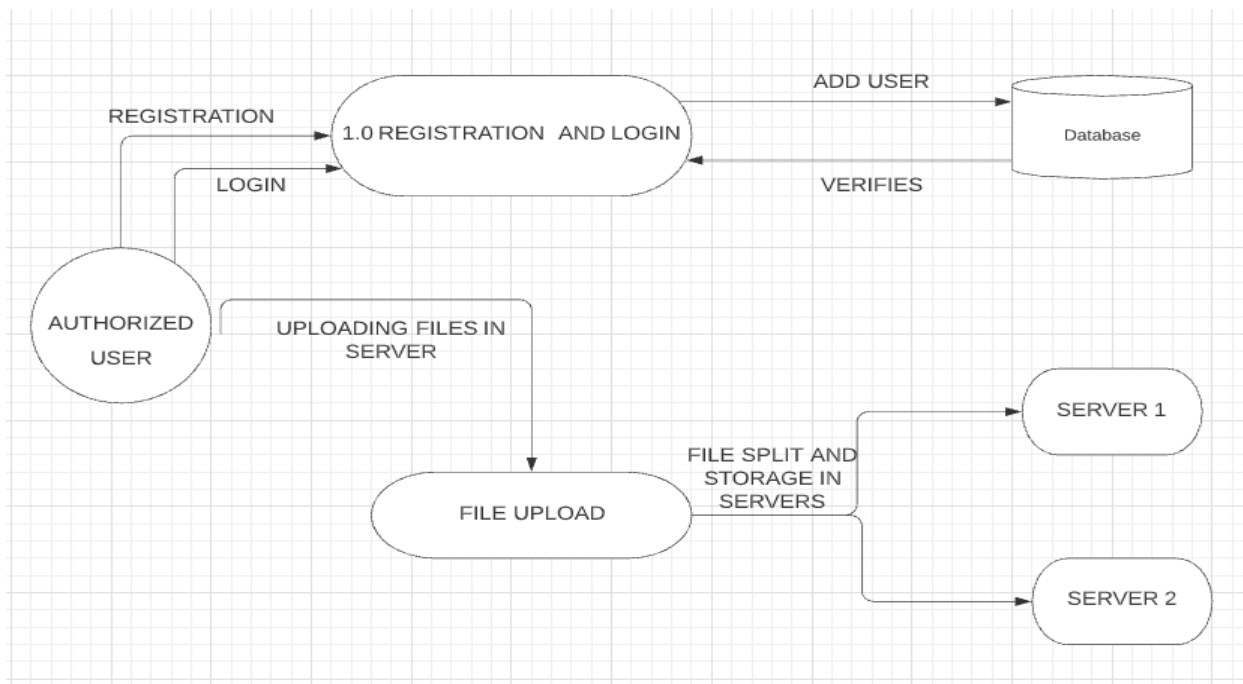


Figure 1.5 DFD (LEVEL 1)

LEVEL 2:

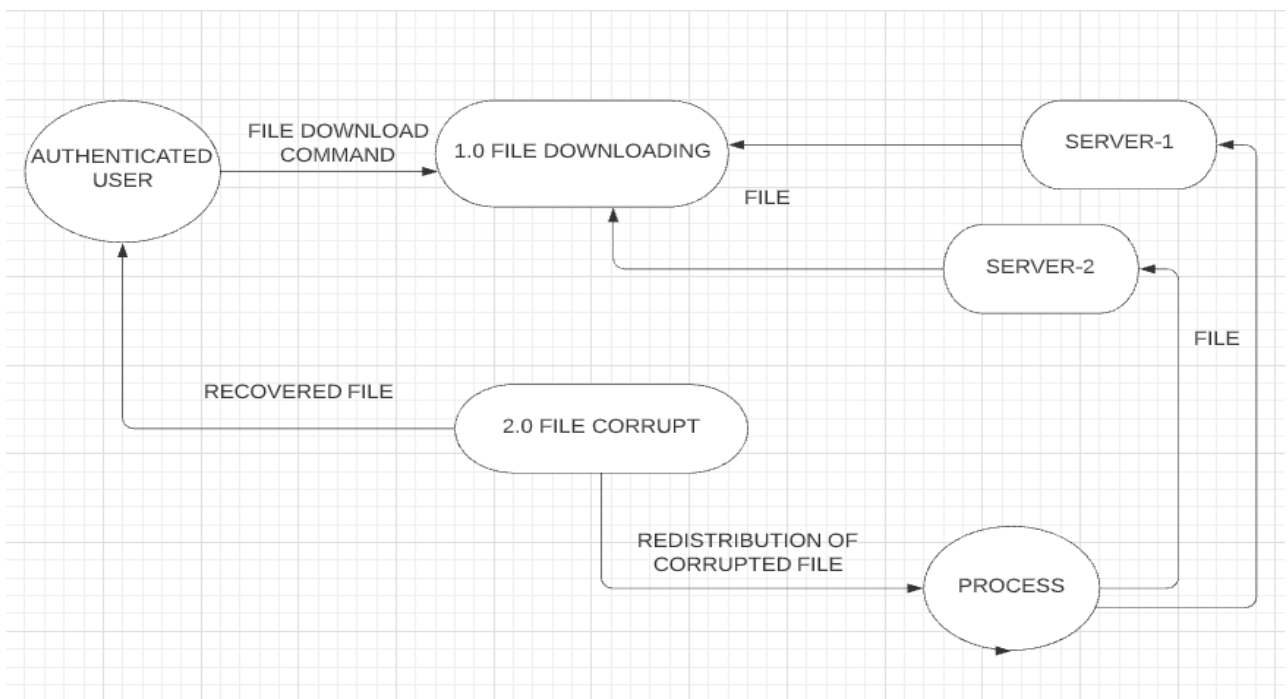


Figure 1.6 DFD (LEVEL 2)

4.4 UML Diagrams

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development.

4.4.1.2 USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases. Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.

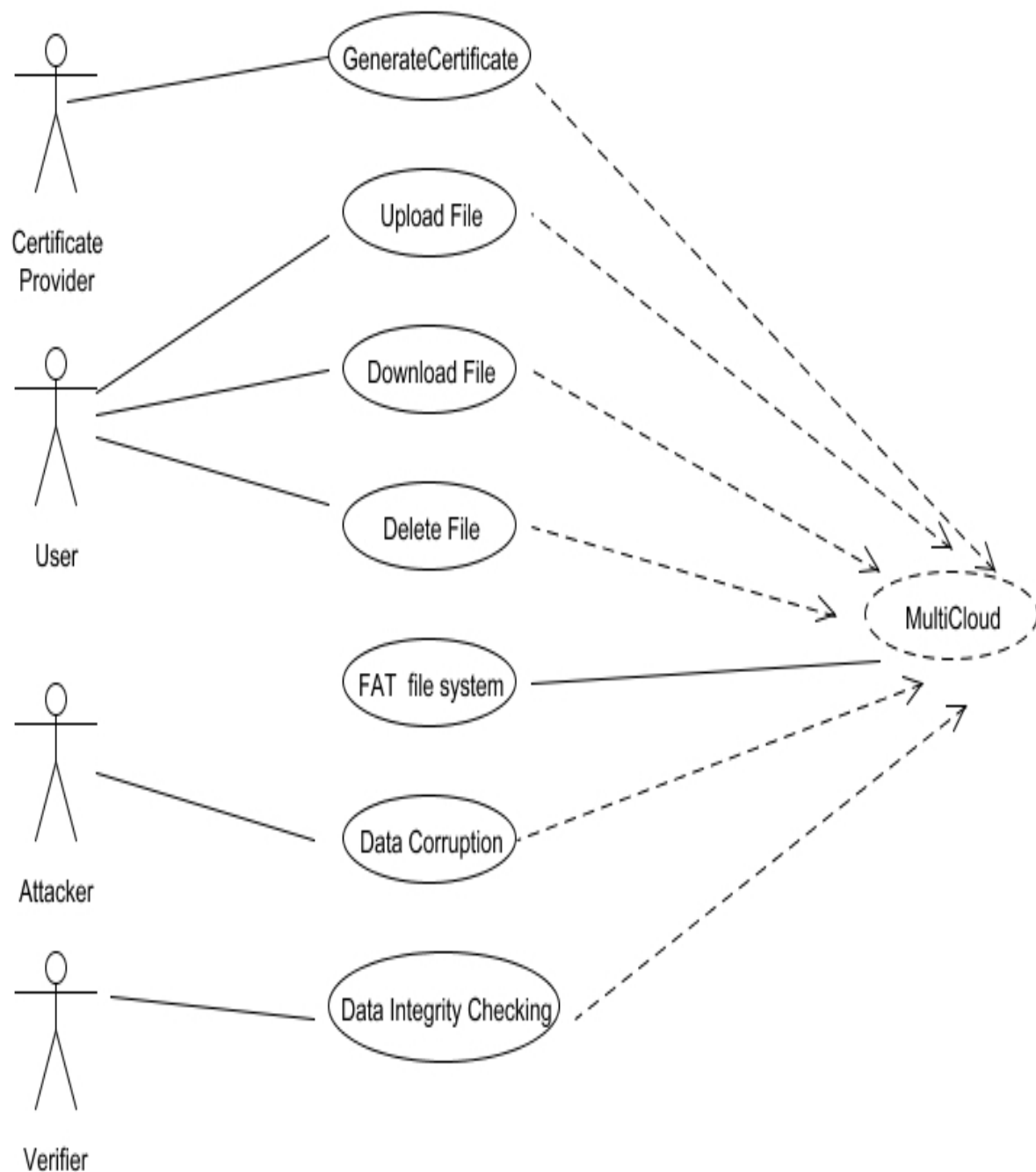


Figure 1.7 USECASE DIAGRAM

4.4.2 SEQUENCE DIAGRAM:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

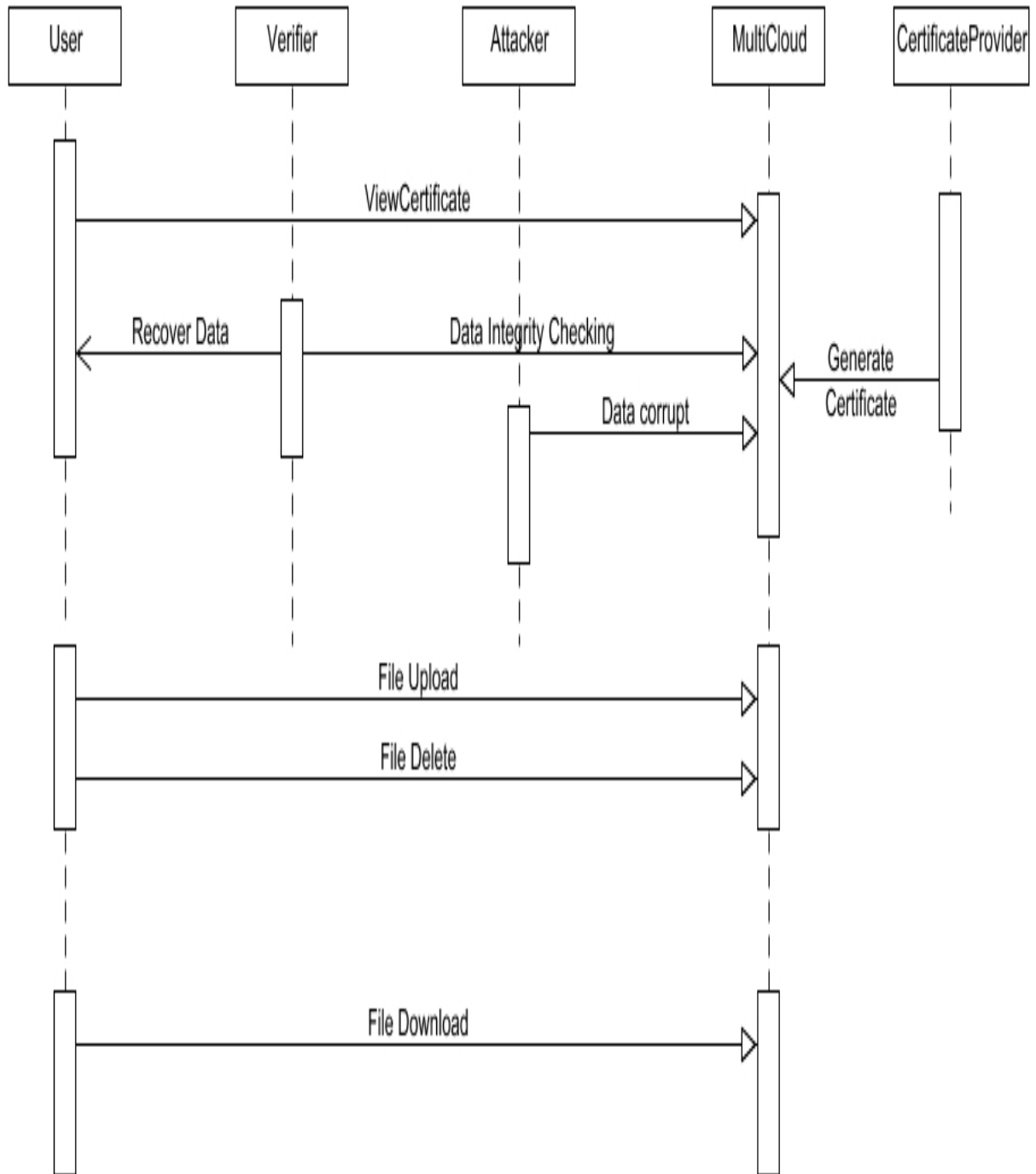


Figure 1.8 SEQUENCE DIAGRAM

4.4.3 ACTIVITY DIAGRAM:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.

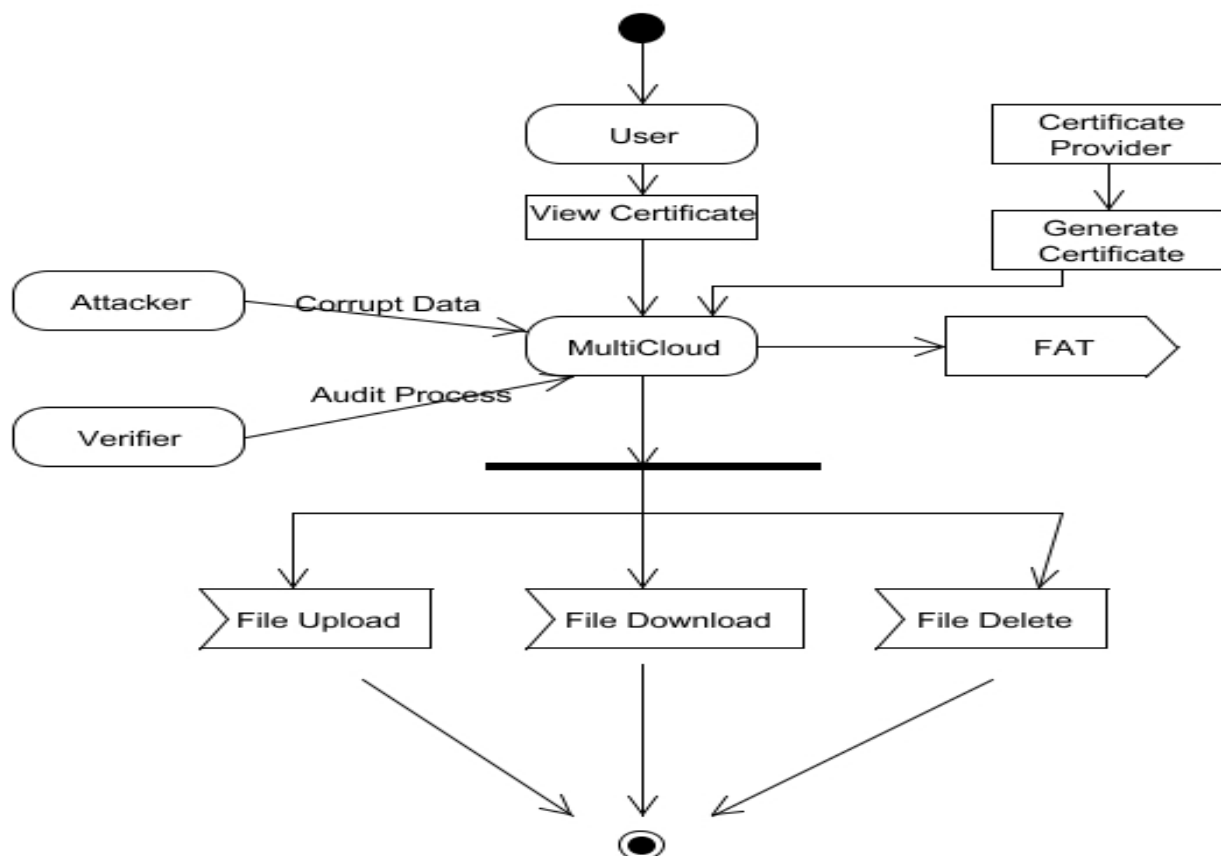


Figure 1.9 ACTIVITY DIAGRAM

4.4.4 COLLABORATION DIAGRAM:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.

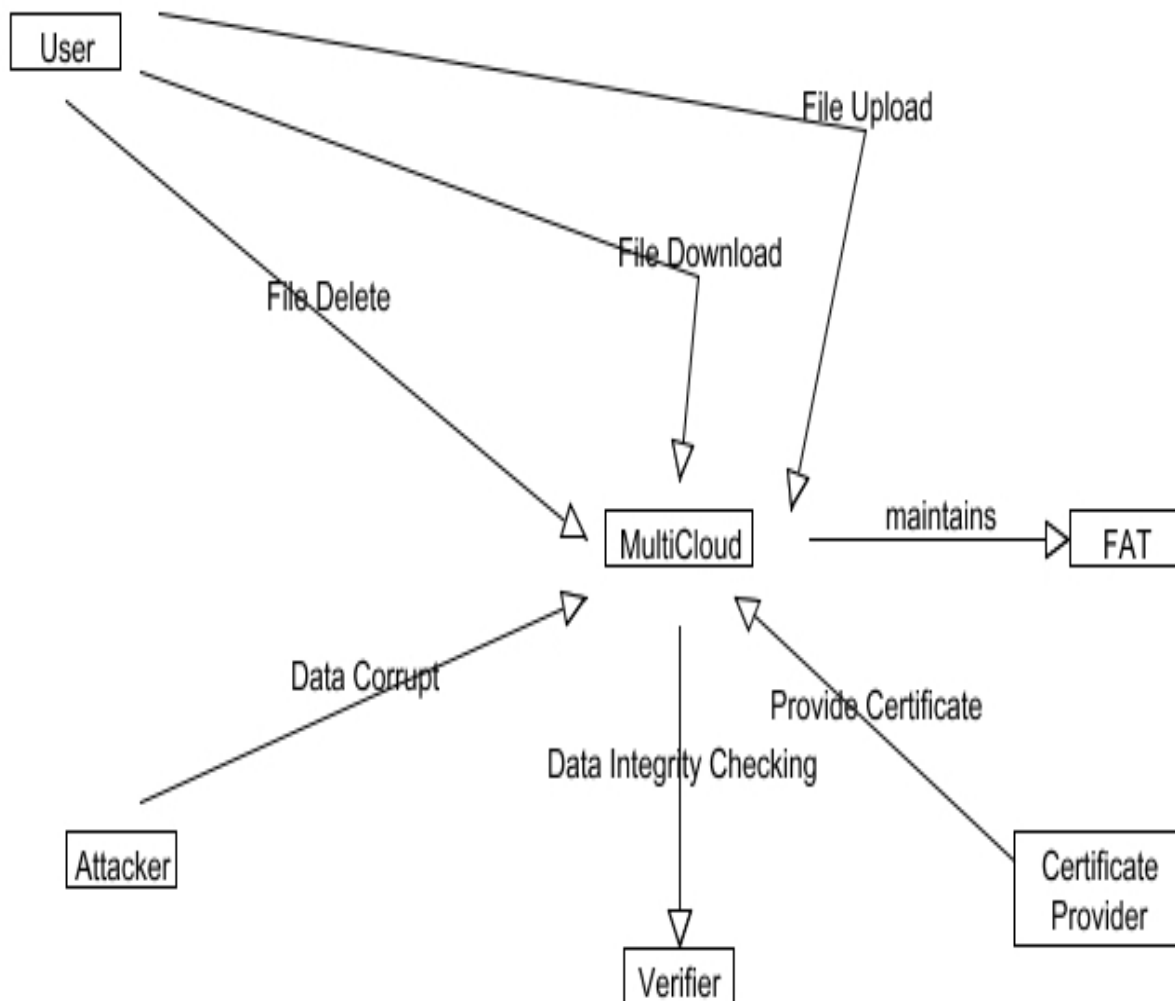


Figure 2.0 COLLABRATION DIAGRAM

SYSTEM ARCHITECTURE

CHAPTER 5

5.1 ARCHITECTURE OVERVIEW

In our solution, the file which is uploaded by the user will get split into blocks. This is done by Dynamic Block Generation algorithm. These blocks will be stored in multiple clouds in Ternary Hash Tree format. In this tree, the root hash value is used to check data integrity of the data blocks. FAT (FILE ALLOCATION TABLE FILE SYSTEM) has all the metadata with proper indexing for various chunks of stored file. Time interval will be fixed by the admin. The continuous auditing process will be carried out based on the fixed time interval. This helps the verifier to perform Block and File level checking. If the attacker corrupts the data, it will be notified in FAT file system as “File corrupted” and then the file will be recovered. The continuous auditing system has a hierarchy. Firstly, parent block will be checked. If it has any corrupted file, then child node auditing will be done. Verifier performs Remote Integrity Checking on Cloud Data. Cloud allocates random combination of all the blocks to the Verifier, instead of the whole file is retrieved during integrity checking. This is to protect user privacy from a third party (Verifier). Verifiable Data Integrity Checking Algorithm is done in two steps: Block Checking and File Checking. Attacker can corrupt data in any one of the cloud servers. On Data Integrity Checking done by the Verifier, Verifier informs Corrupted blocks to the Cloud. Recovery Process will be done by the verifier automatically when data gets corrupted. User can complaint to the Cloud if the user file gets corrupted (Verifier doesn't perform checking on this file). Whenever user access file, Blocks will be reallocated dynamically to provide access confidentiality in cloud and FAT File System will get updated. Auditor will monitor the cloud continuously and they provide the certificate based on the cloud performance. When new user joins in the cloud they will read the certificate and then they can create an account in the cloud.

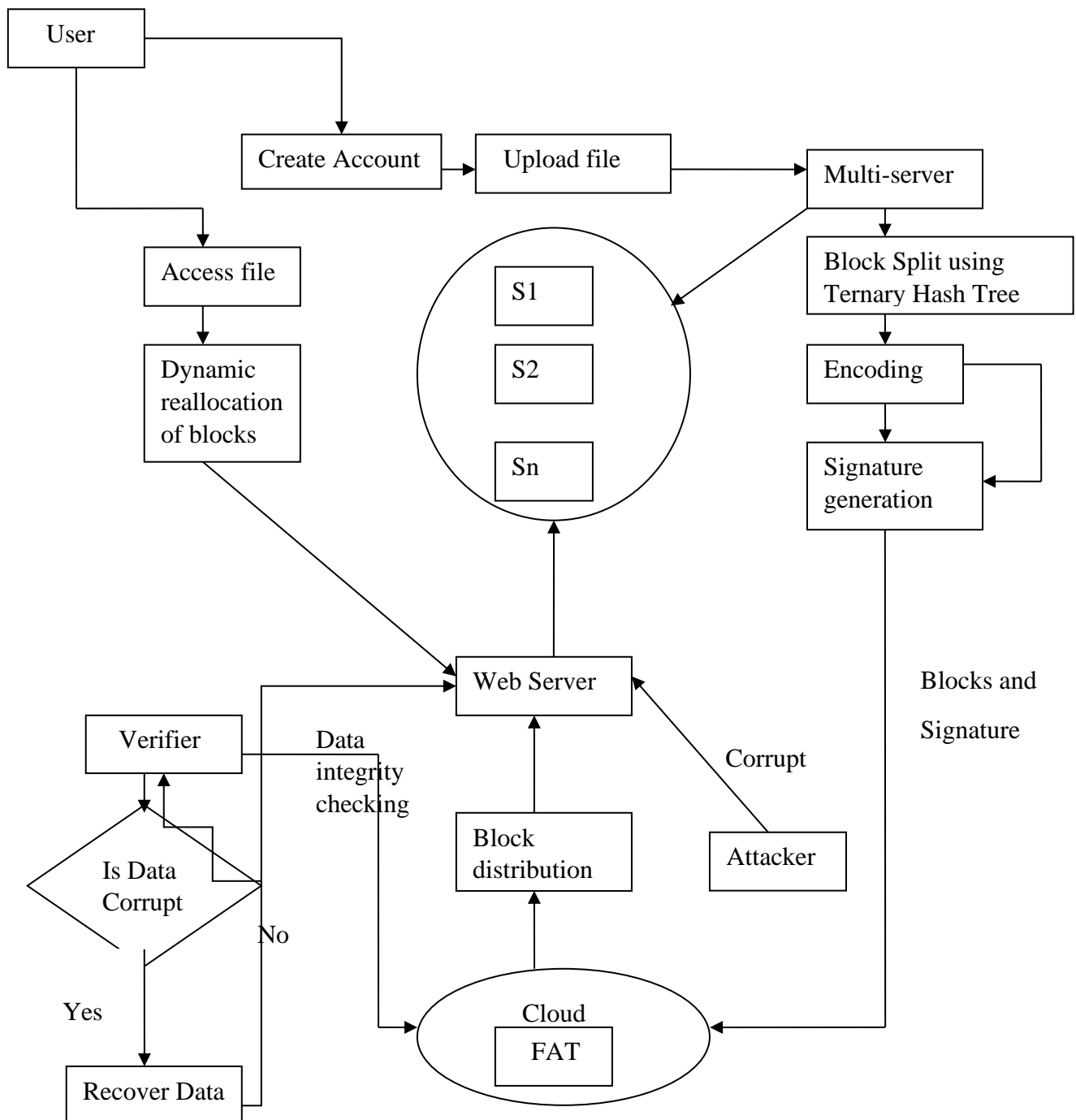


Figure 2.2 ARCHITECTURE DIAGRAM

5.2 MODULE DESIGN SPECIFICATION:

5.2.1 MODULES:

- Server Configuration
- Data Upload and Block Split
- Data Integrity Checking and Update
- File Recovery

5.2.2 MODULE EXPLANATION:

5.2.2.1 SERVER CONFIGURATION:

Admin configure Multi-Cloud server setup. Server IP Address and Port number is given by the admin for each Cloud. Now a Server Architecture is created for Multi-Cloud Storage. If the admin has to reconfigure the old Multi-Cloud server setup, it can be done. For old server setup, FAT file can be modified or remain same. Audit time will be set by the admin for Data Integrity checking process.

5.2.2.2 DATA UPLOAD AND BLOCK SPLIT:

User has an initial level Registration Process at the web end. The users provide their own personal information for this process. The server in turn stores the information in its database. After Registration, user can upload files to the server. Uploaded files will be stored in a Server. When the user uploads the data to different cloud by the time it is split into different blocks using dynamic block generation Algorithm and each block will be appended with Signatures before storing the data in FATFS. Signature generated using MD5 Algorithm. Also the data gets encoded using for Base64 Algorithm.

5.2.2.3 DATA INTEGRITY CHECKING AND UPDATE:

FATFS has proper Indexing and Metadata for the different Chunks of the Data that is being uploaded by User. Verifier performs Remote Integrity Checking on Cloud Data. Cloud allocates random combination of all the blocks to the Verifier, instead of the whole file is retrieved during integrity checking. This is to protect user privacy from a third party (Verifier). Verifiable Data Integrity Checking Algorithm is done in two steps: Block Checking and File Checking. In Block Checking step: Three signatures are generated for Block level Checking.

- A signature of a block retrieved from a FATFS
- A new signature is generated for block to be checked
- A Signature is retrieved from the block appended with the signature which is stored in the Cloud

The above three signatures are cross checked for Block level Integrity Checking. And the block contents are appended to verify with File level Integrity Checking.

5.2.2.4 FILE RECOVERY

Attacker can corrupt data in any one of the cloud servers. On Data Integrity Checking done by the Verifier, Verifier informs Corrupted blocks to the Cloud. Recovery Process will be done by the verifier automatically when data gets corrupted. User can complaint to the Cloud if the user file gets corrupted (Verifier doesn't perform checking on this file). Whenever user access file, Blocks will be reallocated dynamically to provide access confidentiality in cloud and FAT File System will get updated. Auditor will monitor the cloud continuously and they provide the certificate based on the cloud performance. When new user joins in

the cloud they will read the certificate and then they can create an account in the cloud.

5.3 PROGRAM DESIGN LANGUAGE:

5.3.1 ALGORITHM USED:

- Base64 Algorithm
- Message Digest (MD5)
- Dynamic block generation

5.3.1.1 BASE64 ALGORITHM:

In programming, **Base64** is a group of binary-to-text encoding schemes that represent binary data (more specifically, a sequence of 8-bit bytes) in an ASCII string format by translating the data into a radix-64 representation. The term *Base64* originates from a specific MIME_content_transfer_encoding. Each non-final Base64 digit represents exactly 6 bits of data. Three 8-bit bytes (i.e., a total of 24 bits) can therefore be represented by four 6-bit Base64 digits.

Common to all binary-to-text encoding schemes, Base64 is designed to carry data stored in binary formats across channels that only reliably support text content. Base64 is particularly prevalent on the World_Wide_Web where its uses include the ability to embed image_files or other binary assets inside textual assets such as HTML and CSS files.

Base64 is also widely used for sending e-mail attachments. This is required because SMTP - in its original form - was designed to transport 7-bit ASCII characters only. This encoding causes an overhead of 33–36% (33% by the encoding itself; up to 3% more by the inserted line breaks).

Base64 algorithm is designed to encode any binary data, an stream of bytes, into a stream of 64-printable characters.

Base64 encoding algorithm was first presented in "RFC 1421 - Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures" in 1993 by John Linn. It was later modified slightly in "RFC 1521 - MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies" in September 1993 by N. Borenstein.

The Base64 index table:

Ind ex	Bina ry	Ch ar	Ind ex	Bina ry	Ch ar	Ind ex	Bina ry	Ch ar	Ind ex	Bina ry	Ch ar
0	0000 00	A	16	0100 00	Q	32	1000 00	g	48	1100 00	W
1	0000 01	B	17	0100 01	R	33	1000 01	h	49	1100 01	X
2	0000 10	C	18	0100 10	S	34	1000 10	i	50	1100 10	Y
3	0000 11	D	19	0100 11	T	35	1000 11	j	51	1100 11	Z

4	0001 00	E		20	0101 00	U		36	1001 00	k		52	1101 00	0
5	0001 01	F		21	0101 01	V		37	1001 01	l		53	1101 01	1
6	0001 10	G		22	0101 10	W		38	1001 10	m		54	1101 10	2
7	0001 11	H		23	0101 11	X		39	1001 11	n		55	1101 11	3
8	0010 00	I		24	0110 00	Y		40	1010 00	o		56	1110 00	4
9	0010 01	J		25	0110 01	Z		41	1010 01	p		57	1110 01	5
10	0010 10	K		26	0110 10	A		42	1010 10	q		58	1110 10	6
11	0010 11	L		27	0110 11	B		43	1010 11	r		59	1110 11	7

12	0011 00	M	28	0111 00	C	44	1011 00	s	60	1111 00	8
13	0011 01	N	29	0111 01	D	45	1011 01	t	61	1111 01	9
14	0011 10	O	30	0111 10	E	46	1011 10	u	62	1111 10	+
15	0011 11	P	31	0111 11	F	47	1011 11	v	63	1111 11	/
Padding		=									

The Base64 encoding process is to:

- Divide the input bytes stream into blocks of 3 bytes.
- Divide 24 bits of each 3-byte block into 4 groups of 6 bits.
- Map each group of 6 bits to 1 printable character, based on the 6-bit value using the Base64 character set map.
- If the last 3-byte block has only 1 byte of input data, pad 2 bytes of zero (\x0000). After encoding it as a normal block, override the last 2 characters with 2 equal signs (==), so the decoding process knows 2 bytes of zero were padded.

- If the last 3-byte block has only 2 bytes of input data, pad 1 byte of zero (\x00). After encoding it as a normal block, override the last 1 character with 1 equal signs (=), so the decoding process knows 1 byte of zero was padded.
- Carriage return (\r) and new line (\n) are inserted into the output character stream. They will be ignored by the decoding process.

Example 1: Input data, 1 byte, "A". Encoded output, 4 characters, "QQ=="

Input Data	A			
Input Bits	01000001			
Padding	01000001 00000000 00000000			
	\	\	\	
Bit Groups	010000 010000 000000 000000			
Mapping	Q	Q	A	A
Overriding	Q	Q	=	=

Example 2: Input data, 2 bytes, "AB". Encoded output, 4 characters, "QUI="

Input Data	A	B		
Input Bits	01000001 01000010			
Padding	01000001 01000010 00000000			
	\	\	\	
Bit Groups	010000 010100 001000 000000			
Mapping	Q	U	I	A
Overriding	Q	U	I	=

Example 3: Input data, 3 bytes, "ABC". Encoded output, 4 characters, "QUJD"

Input Data	A	B	C
Input Bits	01000001	01000010	01000011
	\	\	\
Bit Groups	010000	010100	001001 000011
Mapping	Q	U	J D

5.3.1.2 MESSAGE DIGEST (MD5) ALGORITHM:

The **MD5 message-digest algorithm** is a widely used hash_function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities. It can still be used as a checksum to verify data_integrity, but only against unintentional corruption. It remains suitable for other non-cryptographic purposes, for example for determining the partition for a particular key in a partitioned database.

MD5 was designed by Ronald Rivest in 1991 to replace an earlier hash function MD4,^[4] and was specified in 1992 as RFC 1321.

One basic requirement of any cryptographic hash function is that it should be computationally infeasible to find two distinct messages that hash to the same value. MD5 fails this requirement catastrophically; such collisions can be found in seconds on an ordinary home computer.

The weaknesses of MD5 have been exploited in the field, most infamously by the Flame malware in 2012. The CMU Software Engineering Institute considers MD5 essentially "cryptographically broken and unsuitable for further use".

As of 2019, MD5 continues to be widely used, in spite of its well-documented weaknesses and deprecation by security experts.¹

Digest sizes: 128 bits

Block sizes: 512 bit

Rounds: 4

Structure: Merkle–Damgård construction

First published: April 1992

Designers: Ronald Rivest

Series: MD2, MD4, MD5, MD6

SECURITY:

The security of the MD5 hash function is severely compromised. A collision attack exists that can find collisions within seconds on a computer with a 2.6 GHz Pentium 4 processor (complexity of 2^{24}). Further, there is also a chosen-prefix collision attack that can produce a collision for two inputs with specified prefixes within seconds, using off-the-shelf computing hardware (complexity 2^{24}). The ability to find collisions has been greatly aided by the use of off-the-shelf GPUs. On an NVIDIA GeForce 8400GS graphics processor, 16–18 million hashes per second can be computed. An NVIDIA GeForce 8800 Ultra can calculate more than 200 million hashes per second.

These hash and collision attacks have been demonstrated in the public in various situations, including colliding document files and digital certificates. As of 2015, MD5 was demonstrated to be still quite widely used, most notably by security research and antivirus companies.

As of 2019, one quarter of widely used content management systems were reported to still use MD5 for password hashing.

MD5 digests have been widely used in the software world to provide some assurance that a transferred file has arrived intact. For example, file servers often provide a pre-computed MD5 (known as md5sum) checksum for the files, so that a user can compare the checksum of the downloaded file to it. Most unix-based operating systems include MD5 sum utilities in their distribution packages; Windows users may use the included PowerShell function "Get-FileHash", install a Microsoft utility, or use third-party applications. Android ROMs also use this type of checksum.

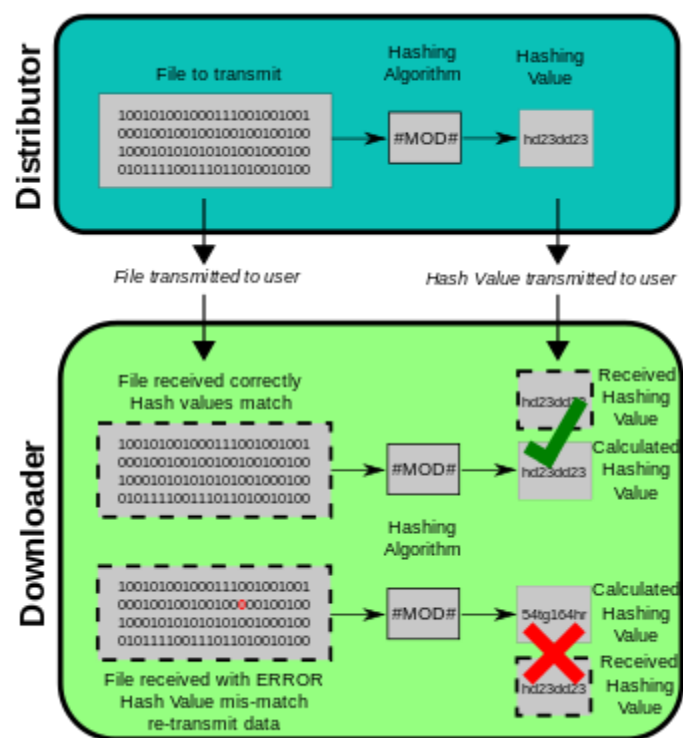


FIGURE 2.2 MD5 ALGORITHM

As it is easy to generate MD5 collisions, it is possible for the person who created the file to create a second file with the same checksum, so this technique cannot protect against some forms of malicious tampering. In some cases, the checksum

cannot be trusted (for example, if it was obtained over the same channel as the downloaded file), in which case MD5 can only provide error-checking functionality: it will recognize a corrupt or incomplete download, which becomes more likely when downloading larger files.

Historically, MD5 has been used to store a one-way hash of a password, often with key_stretching. NIST does not include MD5 in their list of recommended hashes for password storage.

MD5 is also used in the field of electronic discovery, in order to provide a unique identifier for each document that is exchanged during the legal discovery process. This method can be used to replace the Bates's stamp numbering system that has been used for decades during the exchange of paper documents. As above, this usage should be discouraged due to the ease of collision attacks.

5.3.1.3 DYNAMIC BLOCK GENERATION ALGORITHM:

This algorithm is used to split the data into different blocks using the Dynamic Block Generation algorithm. The blocks are divided and stored in different servers in order to preserve security. This algorithm comes into action again if the files get corrupted. In this case the blocks will be redistributed and stored in different servers automatically.

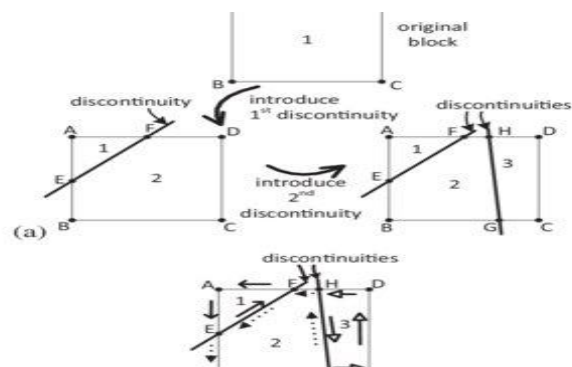


FIGURE 2.3 BLOCK GENERATION

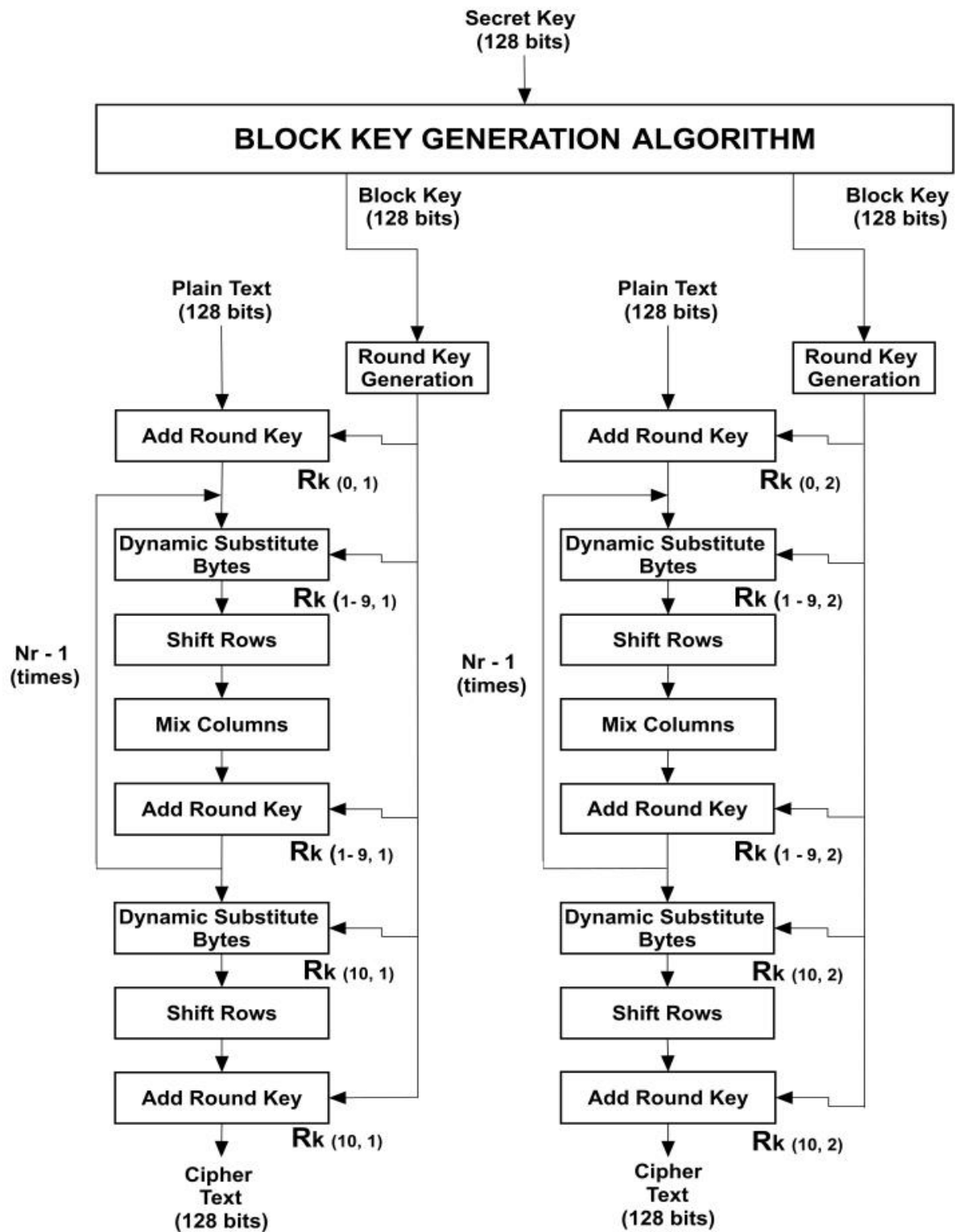


Figure 2.4 Dynamic block generation Algorithm

SYSTEM IMPLEMENTATION

CHAPTER 6

SYSTEM IMPLEMENTATION

6.1 Client-side coding:

1. UploadResponse.java

```
package service;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>
 * Java class for uploadResponse complex type.
 *
 * <p>
 * The following schema fragment specifies the expected content contained within
 * this class.
 *
 * <pre>
 * <complexType name="uploadResponse">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         </sequence>
 *     </restriction>
 *   </complexContent>
 * </complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "uploadResponse")
public class UploadResponse {

}
```

2. SaveFile.java

```
package service;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;

/**
```

```

* <p>
* Java class for saveFile complex type.
*
* <p>
* The following schema fragment specifies the expected content contained within
* this class.
*
* <pre>
* <complexType name="saveFile">
*   <complexContent>
*     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
*       <sequence>
*         <element name="arg0" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
*         <element name="arg1" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
*         <element name="arg2" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
*         <element name="arg3" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
*       </sequence>
*     </restriction>
*   </complexContent>
* </complexType>
* </pre>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "saveFile", propOrder = { "arg0", "arg1", "arg2", "arg3" })
public class SaveFile {

```

```

    protected String arg0;
    protected String arg1;
    protected String arg2;
    protected String arg3;

    /**
     * Gets the value of the arg0 property.
     *
     * @return possible object is {@link String }
     *
     */
    public String getArg0() {
        return arg0;
    }

    /**
     * Sets the value of the arg0 property.
     *
     */

```

```

* @param value
*     allowed object is {@link String }
*
*/
public void setArg0(String value) {
    this.arg0 = value;
}

/**
 * Gets the value of the arg1 property.
 *
 * @return possible object is {@link String }
 *
 */
public String getArg1() {
    return arg1;
}

/**
 * Sets the value of the arg1 property.
 *
 * @param value
 *     allowed object is {@link String }
 *
 */
public void setArg1(String value) {
    this.arg1 = value;
}

/**
 * Gets the value of the arg2 property.
 *
 * @return possible object is {@link String }
 *
 */
public String getArg2() {
    return arg2;
}

/**
 * Sets the value of the arg2 property.
 *
 * @param value
 *     allowed object is {@link String }
 *
 */
public void setArg2(String value) {
    this.arg2 = value;
}

```



```

/**
 * Gets the value of the arg3 property.
 *
 * @return possible object is {@link String }
 */
public String getArg3() {
    return arg3;
}

/**
 * Sets the value of the arg3 property.
 *
 * @param value
 *         allowed object is {@link String }
 */
public void setArg3(String value) {
    this.arg3 = value;
}
}

```

3. SaveFileResponse.java

package service;

```

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

```

```

/**
 * <p>
 * Java class for saveFileResponse complex type.
 *
 * <p>
 * The following schema fragment specifies the expected content contained within
 * this class.
 *
 * <pre>
 * <complexType name="saveFileResponse">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         <element name="return" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
 *       </sequence>
 *     </restriction>
 *   </complexContent>

```

```

* </complexType>
* </pre>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "saveFileResponse", propOrder = { "_return" })
public class SaveFileResponse {

    @XmlElement(name = "return")
    protected String _return;

    /**
     * Gets the value of the return property.
     *
     * @return possible object is {@link String }
     */
    public String getReturn() {
        return _return;
    }

    /**
     * Sets the value of the return property.
     *
     * @param value
     *         allowed object is {@link String }
     */
    public void setReturn(String value) {
        this._return = value;
    }

}

```

4. FetchFiles.java

```
package service;
```

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlType;
```

```

/**
 * <p>
 * Java class for fetchFiles complex type.
 *
 * <p>
 * The following schema fragment specifies the expected content contained within
 * this class.

```

```

*
* <pre>
* <complexType name="fetchFiles">
*   <complexContent>
*     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
*       <sequence>
*         <element name="arg0" type="{http://www.w3.org/2001/XMLSchema}string"
minOccurs="0"/>
*       </sequence>
*     </restriction>
*   </complexContent>
* </complexType>
* </pre>
*
*
*/
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "fetchFiles", propOrder = { "arg0" })
public class FetchFiles {

    protected String arg0;

    /**
     * Gets the value of the arg0 property.
     *
     * @return possible object is {@link String }
     */
    public String getArg0() {
        return arg0;
    }

    /**
     * Sets the value of the arg0 property.
     *
     * @param value
     *         allowed object is {@link String }
     */
    public void setArg0(String value) {
        this.arg0 = value;
    }

}

```

5. FetchFilesResponse.java

```

package service;

import java.util.ArrayList;

```

```

import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

/**
 * <p>
 * Java class for fetchFilesResponse complex type.
 *
 * <p>
 * The following schema fragment specifies the expected content contained within
 * this class.
 *
 * <pre>
 * <complexType name="fetchFilesResponse">
 *   <complexContent>
 *     <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       <sequence>
 *         <element name="return" type="{http://www.w3.org/2001/XMLSchema}anyType"
maxOccurs="unbounded" minOccurs="0"/>
 *       </sequence>
 *     </restriction>
 *   </complexContent>
 * </complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "fetchFilesResponse", propOrder = { "_return" })
public class FetchFilesResponse {

    @XmlElement(name = "return")
    protected List<Object> _return;

    /**
     * Gets the value of the return property.
     *
     * <p>
     * This accessor method returns a reference to the live list, not a
     * snapshot. Therefore any modification you make to the returned list will
     * be present inside the JAXB object. This is why there is not a
     * <CODE>set</CODE> method for the return property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     *
     * <pre>
     * getReturn().add(newItem);

```

```

* </pre>
*
*
* <p>
* Objects of the following type(s) are allowed in the list {@link Object }
*
*
*/
public List<Object> getReturn() {
    if (_return == null) {
        _return = new ArrayList<Object>();
    }
    return this._return;
}
}

```

6.2 Server-side coding:

1. CommonInter.java

```

package com.logic;

import java.sql.Timestamp;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Vector;

public interface CommonInter {

    public static Vector<Timestamp> timVector=new Vector<Timestamp>();
    public static LinkedHashMap<String,String> suspMap=new LinkedHashMap<String,
String>();
    void setSession(Map<String, Object> session);

}

```

2. DeleteFile.java

```

package com.logic;

import java.io.File;

public class DeleteFile {
    public void delete(String fName)
    {
        File dir=new File(fName);
        System.out.println("cleared "+fName);
        deleteDir(dir);
    }
}

```

```

        public static boolean deleteDir(File dir) {
            if (dir.isDirectory()) {
                String[] children = dir.list();
                for (int i = 0; i < children.length; i++) {
                    boolean success = deleteDir(new File(dir, children[i]));
                    if (!success) {
                        return false;
                    }
                }
            }
            return dir.delete();
        }
    }
}

```

3. DeleteFileDelegate.java

```

package com.logic;

import java.io.File;

@javax.jws.WebService(targetNamespace = "http://logic.com/", serviceName =
"DeleteFileService", portName = "DeleteFilePort1", wsdlLocation = "WEB-
INF/wsdl/DeleteFileService.wsdl")
public class DeleteFileDelegate {

    com.logic.DeleteFile deleteFile = new com.logic.DeleteFile();

    public void delete(String fName) {
        deleteFile.delete(fName);
    }

    public boolean deleteDir(File dir) {
        return deleteFile.deleteDir(dir);
    }

}

```

4. FileUpload.java

```

package com.logic;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;

```

```

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;

public class FileUpload implements CommonInter {
    HashMap<String,String> filemap=new HashMap<String,String>();
    boolean b=false;
    public void upload(String block,String cont,String fname)
    {
        try
        {
            String blo="webapps/IdentityBasedStorage/BLOCKS";

            File blocksfile=new File(blo);
            if(!blocksfile.exists())
            {
                blocksfile.mkdir();
                for(int i=0;i<=30;i++)
                {
                    new
File("webapps/IdentityBasedStorage/BLOCKS/BLOCK"+i).mkdir();
                }
                System.out.println("Cloud Space Created ");
            }
            if(blocksfile.listFiles().length==0)
            {
                for(int i=0;i<=30;i++)
                {
                    new
File("webapps/IdentityBasedStorage/BLOCKS/BLOCK"+i).mkdir();
                }
            }

            File f=new File("webapps/IdentityBasedStorage/BLOCKS");

            File fFodr=new File("webapps/IdentityBasedStorage/BLOCKS/"+block);
            if(fFodr.exists())
            {
                File blkF=new
File("webapps/IdentityBasedStorage/BLOCKS/"+block+"/"+block+"_"+fname);
                if(blkF.exists())
                {
                    blkF.delete();
                }
                FileWriter fw=new FileWriter(blkF);

```

```

        fw.write(cont);
        fw.close();
    }
    else
    {
        System.out.println("File Upload Encountered Pblm.....Check
Storage Blocks(30 cnt)");
    }
}
catch (Exception e)
{
    e.printStackTrace();
}
}
public ArrayList RwPackets(ArrayList blkName,String fName)
{
    ArrayList conlst=new ArrayList();
    try{
        b=true;
        for(int i=0;i<blkName.size();i++)
        {
            String bNo=blkName.get(i).toString();

            String blkNme="BLOCK"+bNo;
            String fName="BLOCK"+bNo+"_"+fName;
            String pckCon=fetchFile(blkNme, fName);
            String pckCont=pckCon.split("@")[0].toString();
            conlst.add(pckCont);
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    b=false;
    return conlst;
}
public ArrayList fetchFiles(String fdrNme)
{
    ArrayList al = new ArrayList();
    Try
    {
        File f=new File("webapps/IdentityBasedStorage/BLOCKS/"+fdrNme);
        File[] fArr=f.listFiles();

        for(int i=0;i<fArr.length;i++)
        {
            al.add(fArr[i].getName().toString().trim());
        }
    }
    if(al!=null)
    {

```



```

        System.out.println(al.toString());
    }
    // al=new ArrayList(Arrays.asList(fArr));
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return al;
}
public String fetchFile(String blkNme,String fNme)
{
    System.out.println("Fetching File Content in block "+blkNme+" & file "+fNme);
    Timestamp curtm=new Timestamp(new Date().getTime());
    long timDiff=0;
    timVector.add(curtm);
    if(timVector.size()==2)
    {
        timDiff=(timVector.get(1).getTime())-(timVector.get(0).getTime());
        timVector.remove(timVector.firstElement());
    }
    StringBuffer sb=new StringBuffer();
    try
    {
        File f=new
File("webapps/IdentityBasedStorage/BLOCKS/"+blkNme+"/"+fNme);
        FileReader fis=new FileReader(f);
        BufferedReader br=new BufferedReader(fis);
        String s="";
        while ((s=br.readLine())!=null) {
            sb.append(s);
        }
        // System.out.println("fetchFile "+sb);
        br.close();
        fis.close();
        //System.out.println("Time Difference id "+timDiff+" boolean "+b);
        if(timDiff>2000 && b==false)
        {
            suspMap.put(blkNme+"@"+fNme, sb.toString());
        }
        if(b)
        {
            System.out.println("File Deleted.....");
            f.delete();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return sb.toString();
}

```

```

    }
    public String saveFile(String blkNme,String fNme,String encData,String sign)
    {
        String status="";
        try
        {
            String contentToSave=encData+"@"+sign;
            File f=new
File("webapps/IdentityBasedStorage/BLOCKS/"+blkNme+"/"+fNme);
            FileOutputStream fos=new FileOutputStream(f);
            fos.write(contentToSave.getBytes());
            fos.close();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return status;
    }
    public String deleteUsrFile(String block,String fileName)
    {
        String status="";
        String blo="webapps/IdentityBasedStorage/BLOCKS";
        File blocksfile=new File(blo);
        if(blocksfile.exists())
        {
            File fFodr=new File("webapps/IdentityBasedStorage/BLOCKS/"+block);
            if(fFodr.exists())
            {
                File blkF=new
File("webapps/IdentityBasedStorage/BLOCKS/"+block+"/"+block+"_"+fileName);

                if(blkF.exists())
                {
                    blkF.delete();
                    status="success";
                }
            }
        }
        else
        {
            status="error";
        }
        //System.out.println("status in webservice file delete"+status);
        return status;
    }
    public void setSession(Map<String, Object> session) {
        // TODO Auto-generated method stub
    }
}

```

```
}
```

5. FileUploadDelegate.java

```
package com.logic;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;

@javax.jws.WebService(targetNamespace = "http://logic.com/", serviceName =
"FileUploadService1", portName = "FileUploadPort", wsdlLocation = "WEB-
INF/wsdl/FileUploadService1.wsdl")
public class FileUploadDelegate {

    com.logic.FileUpload fileUpload = new com.logic.FileUpload();

    public void upload(String block, String cont, String fname) {
        fileUpload.upload(block, cont, fname);
    }

    public ArrayList RwPackets(ArrayList blkName, String fName) {
        return fileUpload.RwPackets(blkName, fName);
    }

    public ArrayList fetchFiles(String fdrNme) {
        return fileUpload.fetchFiles(fdrNme);
    }

    public String fetchFile(String blkNme, String fNme) {
        return fileUpload.fetchFile(blkNme, fNme);
    }

    public String saveFile(String blkNme, String fNme, String encData,
        String sign) {
        return fileUpload.saveFile(blkNme, fNme, encData, sign);
    }

    public String deleteUsrFile(String block, String filName) {
        return fileUpload.deleteUsrFile(block, filName);
    }
}
```

```
}
```

6. PublicAudit.java

```
package com.logic;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.security.MessageDigest;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.Map;
```

```
public class PublicAudit implements CommonInter{
```

```
    String status="";
```

```
    public String startAudit(ArrayList aList)
```

```
    {
```

```
        boolean boolForList=true;
```

```
        String failedContent="";
```

```
        try{
```

```
            for(int i=0;i<aList.size();i++)
```

```
            {
```

```
                String mapping=aList.get(i).toString();
```

```
                String arr[]=mapping.split("@");
```

```
                String blkNme="BLOCK"+arr[0].toString();
```

```
                String fNme=blkNme+"_ "+arr[1].toString();
```

```
                String tagSign=arr[2].toString();
```

```
                String content=fetchFileCon(blkNme, fNme);
```

```
                String arrSplit[]=content.split("@");
```

```
                String packContent=arrSplit[0].toString();
```

```
                String packSign=arrSplit[1].toString();
```

```
                String curSig=genSignature(packContent);
```

```
                if(tagSign.equals(packSign) && tagSign.equals(curSig))
```

```
                {
```

```
                    System.out.println("Signature Verified For
```

```
Packet....."+blkNme);
```

```
                }
```

```
            } else
```

```
            {
```

```
                System.out.println("Signature Verification
```

```
Failed ....."+blkNme);
```

```
                failedContent=failedContent+arr[0].toString()+"@";
```

```

        boolForList=false;
    }
    Thread.sleep(1000);
}
if(boolForList==false)
{
    status=failedContent;
    String stat=finalizecheck();
    status=status+"-"+stat;
}
else
{
    status="success";
}
}
catch (Exception e) {
    e.printStackTrace();
}
return status;
}

public String fetchFileCon(String blkNme,String fNme)
{
    System.out.println("Fetching File Content for Audit Process in Block
"+blkNme);
    StringBuffer sb=new StringBuffer();
    try{
        File f=new
File("webapps/IdentityBasedStorage/BLOCKS/"+blkNme+"/"+fNme);
        FileReader fis=new FileReader(f);
        BufferedReader br=new BufferedReader(fis);
        String s="";
        while ((s=br.readLine())!=null) {
            sb.append(s);
        }
        // System.out.println("fetchFileCon "+sb);
        br.close();
        fis.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return sb.toString();
}

public String genSignature(String packet)
{
    StringBuffer sb=null;
    try
    {
        MessageDigest mdP = MessageDigest.getInstance("MD5");

```

```

        mdP.update(packet.getBytes());
        byte mdbytes[] = mdP.digest();
        sb = new StringBuffer();
        for (int i = 0; i < mdbytes.length; i++)
        {
            sb.append(Integer.toString((mdbytes[i] & 0xff) + 0x100,
16).substring(1));
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
    return sb.toString();
}
public String finalizecheck()
{
    boolean recStatus=false;
    String stat="";
    try
    {
        Iterator iFinal=suspMap.keySet().iterator();
        while (iFinal.hasNext()) {
            String key=iFinal.next().toString();
            String suspCon=suspMap.get(key).toString();
            String arr[]=key.split("@");
            String blkNme=arr[0].toString();
            String fNme=arr[1].toString();
            FileUpload fu=new FileUpload();
            fu.b=true;
            String cont=fu.fetchFile(blkNme, fNme);
            String spl[]=cont.split("@");
            String filePacket=spl[0].toString();
            String fileSign=spl[1].toString();
            String fileConSign=genSignature(filePacket);
            String suspSign=genSignature(suspCon.split("@")[0].toString());

            System.out.println("fileNewsign "+fileConSign+"
"+"fileSign"+fileSign+" "+"suspSign"+suspSign);

            if(!fileSign.equals(fileConSign)&&(suspSign.equals(fileSign)))
            {
                recStatus=true;
                String recContent=suspCon+"@"+suspSign;
                fu.saveFile(blkNme, fNme, recContent, suspSign);
                System.out.println("file recovered");
            }
            fu.b=false;
        }
    }
}

```

```

    }
    catch (Exception e) {
        e.printStackTrace();
    }
    if(recStatus)
    {
        stat="success";
    }
    else
    {
        stat="fail";
        System.out.println("suspicious map empty");
    }
    return stat;
}

public void setSession(Map<String, Object> session) {
    // TODO Auto-generated method stub

}

}

```

7. PublicAuditDelegate.java

```

package com.logic;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.security.MessageDigest;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;

@javax.jws.WebService(targetNamespace = "http://logic.com/", serviceName =
"PublicAuditService", portName = "PublicAuditPort", wsdlLocation = "WEB-
INF/wsdl/PublicAuditService.wsdl")
public class PublicAuditDelegate {

    com.logic.PublicAudit publicAudit = new com.logic.PublicAudit();

    public String startAudit(ArrayList aList) {
        return publicAudit.startAudit(aList);
    }
}

```

```

        public String fetchFileCon(String blkNme, String fNme) {
            return publicAudit.fetchFileCon(blkNme, fNme);
        }

        public String genSignature(String packet) {
            return publicAudit.genSignature(packet);
        }

        public void finalizecheck() {
            publicAudit.finalizecheck();
        }
    }
}

```

8. RecoverJob.java

```

package com.logic;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class RecoverJob implements ServletContextListener{

    public void contextDestroyed(ServletContextEvent arg0)
    {
        System.out.println("Stopping Application successfully");
        try{
            PublicAudit pa=new PublicAudit();
            pa.finalizecheck();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public void contextInitialized(ServletContextEvent arg0)
    {
        System.out.println("Initializing Application successfully");

        try{

        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}

```


SYSTEM TESTING

CHAPTER 7

7. SYSTEM TESTING

7.1 UNIT TESTING:

Unit Testing is a level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. In our application the modules are taken separately and tested. For example, in the capture module when the unauthorized user tries to uninstall applications the front camera of the smart phone is enabled and captured image is sent to the linked account. As a part of unit testing the time taken for sending this captured image is tested for various trials of the same module. An aggregate time after a few trials is taken to be the minimum time for this particular module. Likewise, the other modules are tested separately to test their functionality. Unit testing is usually performed by the developer.

7.2 INTEGRATION TESTING

Integration Testing is a level of the software testing process where individual units are combined and tested as a group. In this testing, a Big Bang approach is used where the separate modules are combined together as a single bundle and then tested. In our application the entire app is tested once the functionalities are combined together as a single application component. The modules like the scan mobile, capture module, scan installed apps are tested separately during unit testing for proper functionality. Then they are confined into a single application for applying integration testing. This testing is performed at the developer's end in our case.

VALIDATION TESTING

The final step involves Validation testing, which determines whether the software function as the user expected. The end-user rather than the system developer conducts this most software developers as a process called “Alpha and Beta Testing” to uncover that only the end users seem to find. The compilation of the entire project is based on the full satisfaction of the end users. In this project, validation testing is made in various forms. In the register form, mobile number field is required to have ten numbers only. If this condition is overruled then a pop-up occurs saying "Invalid mobile no". Similarly, the email-id field requires a valid mail id to be entered.

7.3 TEST CASES & REPORTS

A test case is a detailed procedure that fully tests a feature or an aspect of a feature. Whereas the test plan describes what to test, a test case describes how to perform a particular test. You need to develop a test case for each test listed in the test plan. The below table shows the possible Inputs and their corresponding Expected and Obtained output for the given modules with their status.

TEST CASE DESIGN

A test case is a detailed procedure that fully tests a feature or an aspect of a feature. Whereas the test plan describes what to test, a test case describes how to perform a particular test. You need to develop a test case for each test listed in the test plan. The below table shows the possible Inputs and their corresponding Expected and Obtained output for the given modules with their status.

TABLE 1.4 ADMIN LOGIN TEST REPORT**MODULE: SERVER CONFIGURATION****SCREEN: (1) ADMIN LOGIN**

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULTS
1	Enter valid username	Username: Admin	Admin	Admin	PASSED
2	Enter valid password	Password: *****	*****	*****	PASSED
3	Check user name and password with the database	Username: Admin Password: *****	“Login successful ”and Go to next page	“Login successful” and Configuration page displayed	PASSED

TABLE 1.5 USER LOGIN TEST REPORT**MODULE: DATA UPLOAD AND BLOCK SPLIT****SCREEN:(5) USER LOGIN**

S.N O	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL OUTPUT	TEST RESULTS
1	(Registration page) Enter valid username, password, Retype password, Gender, Age, Address,	Username: XXXXXX Password: ***** Retype Password: ***** Gender: Male/Female Age:25	“Registered successfully”	“Registered successfully”	PASSED

	Contact no, Email.	Address: Chennai Contact no: 987654321 0 Email: xxx@gmail.com			
2	(User login page) Enter valid username and password	Username: XXXXXX Password: abcdefgh	“Password should contain one uppercase letter, one lowercase, one special character(@\$*) , one digit and should contain atleast of length 6”	“Password should contain one uppercase letter,one lowercase,one special character(@\$*),one digit and should contain atleast of length 6”	PASSED
3	(User login page) Enter valid username and password	Username: XXXXXX Password: *****	Display “Login successful” and Go to File uploading page	Display “Login successful” and Go to File uploading page	PASSED

TABLE 1.6 FILE UPLOADING TEST REPORT

MODULE: DATA UPLOAD AND BLOCK SPLIT

SCREEN:(6) FILE UPLOAD

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL INPUT	TEST RESULTS
1.	Files need to be upload in the servers	A file of type pdf,jpg,png etc, need	File uploaded successfully.	File uploaded successfully.	Success

		to be uploaded to the server.			
2.	Display of hash tree	Clicking the link to display the ternary hash tree	Display of the ternary hash tree for the file uploaded.	Display of the ternary hash tree for the file uploaded.	Success
3.	Error, if the file gets corrupted	The user checking the file uploaded.	An error which indicates the corruption of the file.	An error which indicates the corruption of the file.	Success
4.	Redistribution of the blocks of the file uploaded	Clicking the button "Inform to admin" to redistribute the blocks of the file uploaded	FATFS table of the redistributed blocks of the file uploaded	FATFS table of the redistributed blocks of the file uploaded	Success

TABLE 1.7 AUDITING TEST REPORT

MODULE: DATA INTEGRITY CHECKING AND UPDATE

SCREEN:(9) FATFS

S.NO	ACTION	INPUT	EXPECTED OUTPUT	ACTUAL INPUT	TEST RESULTS
1	Display of the FATFS table	File uploading and block split	The display of the FATFS table regarding the blocks allocation	The display of the FATFS table regarding the blocks allocation	Success

CONCLUSION

CHAPTER 8

8.CONCLUSION

8.1 CONCLUSION:

The continuous File level, Block level and replica level auditing are performed for ensuring data integrity of the files which is uploaded. Since we are picking few random blocks to perform auditing, it helps in making sure the computational efficiency of our system. Data consistency is preserved with the help of replica level auditing in the cloud. When the corrupted blocks are localized during continuous auditing, it is corrected to fulfil the requirement of a real time application. Most importantly, privacy of the user data is preserved, since we choose blocks for checking in a random order. So that the TPA cannot access the complete data of the user. This helps our system to overcome the drawbacks of the existing work.

8.2 FUTURE ENHANCEMENT:

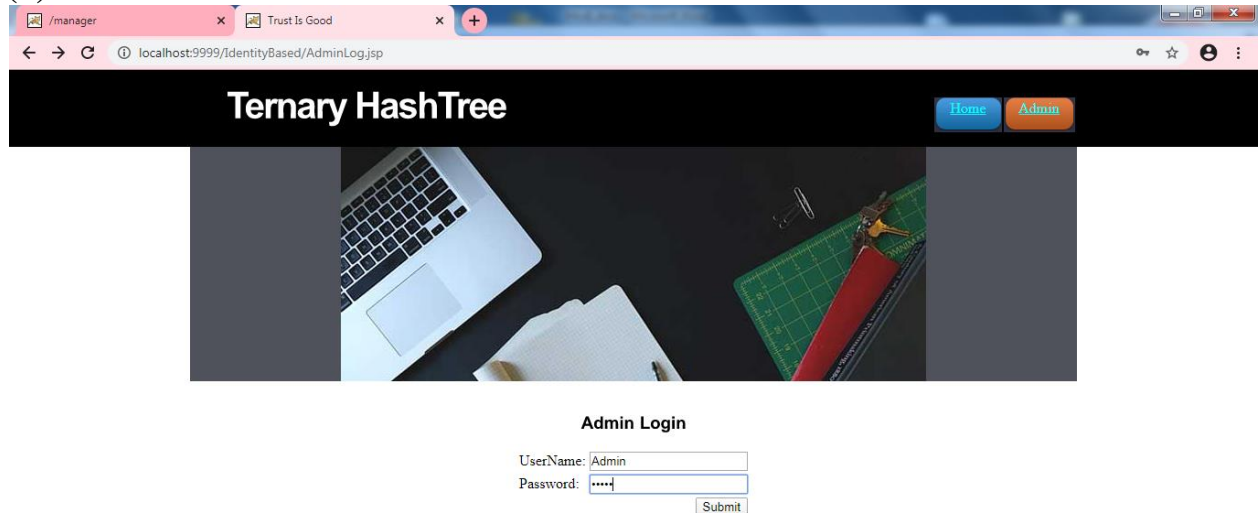
Our future goal is to implement this application for data sharing and therefore to ensure the integrity of the data which is shared and accessed by the group of users.

As of now we are using only two servers to store our data. In future we can extend to as many servers as we wish.

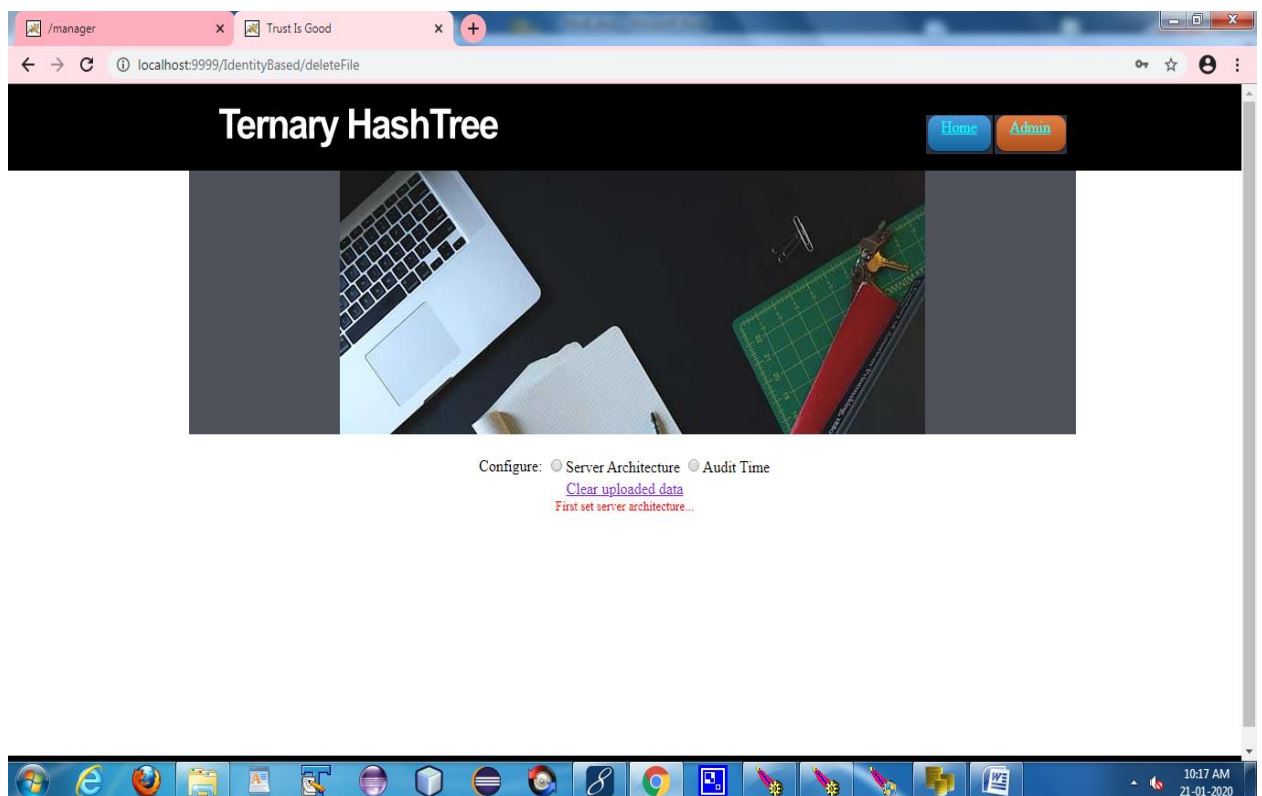
APPENDICES

A.1 SAMPLE SCREENS

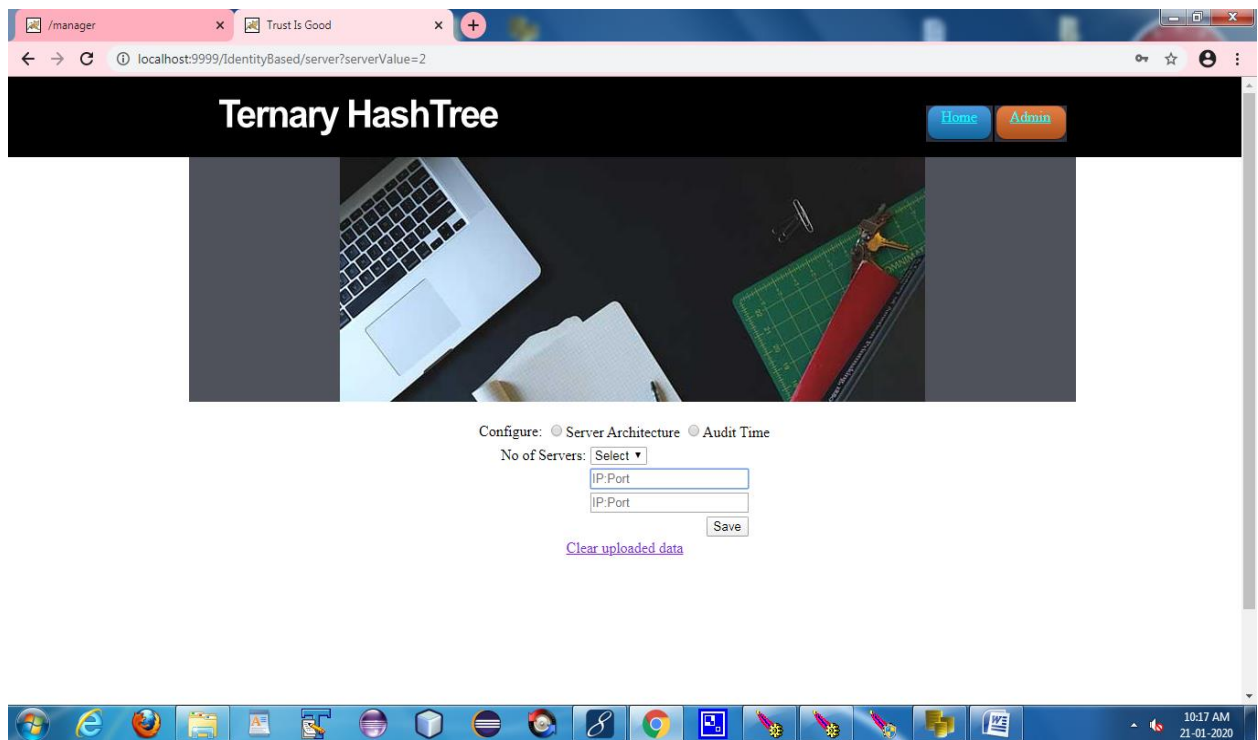
(1) ADMIN LOGIN:



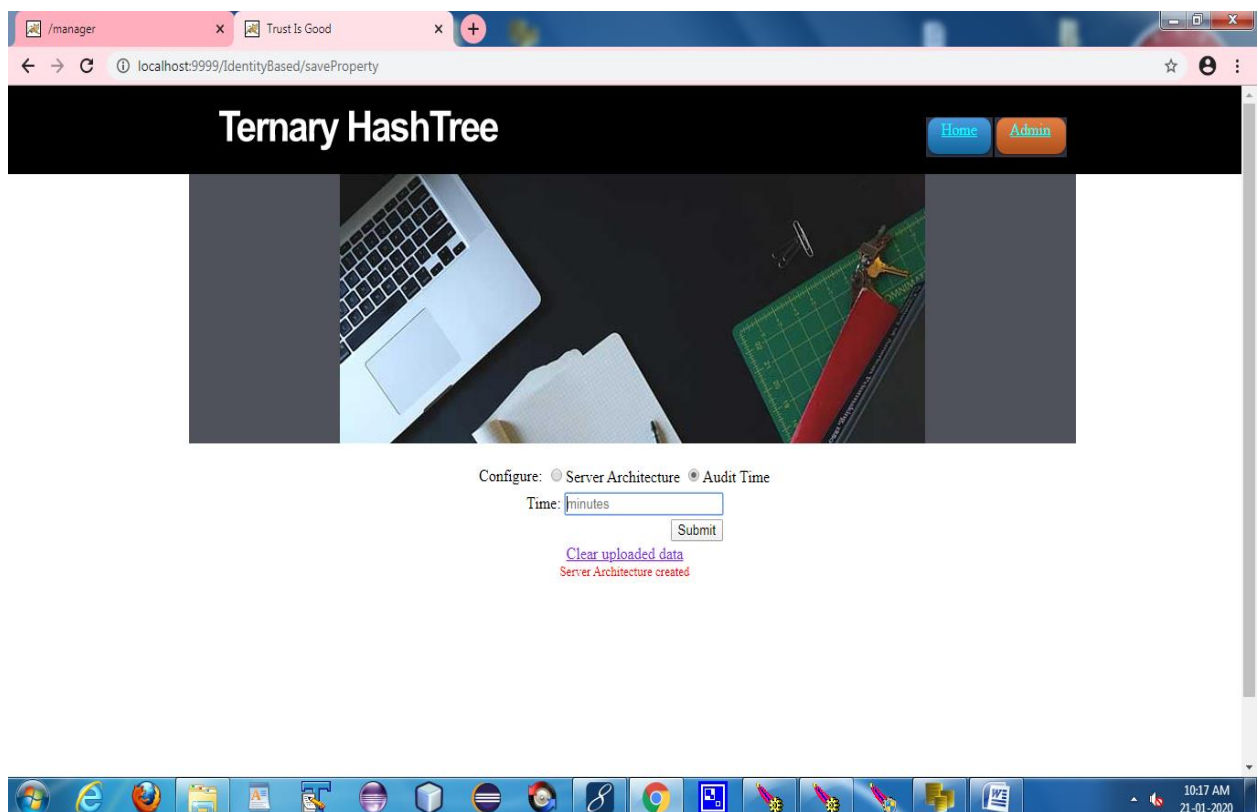
(2) SERVER CONFIGURATION:



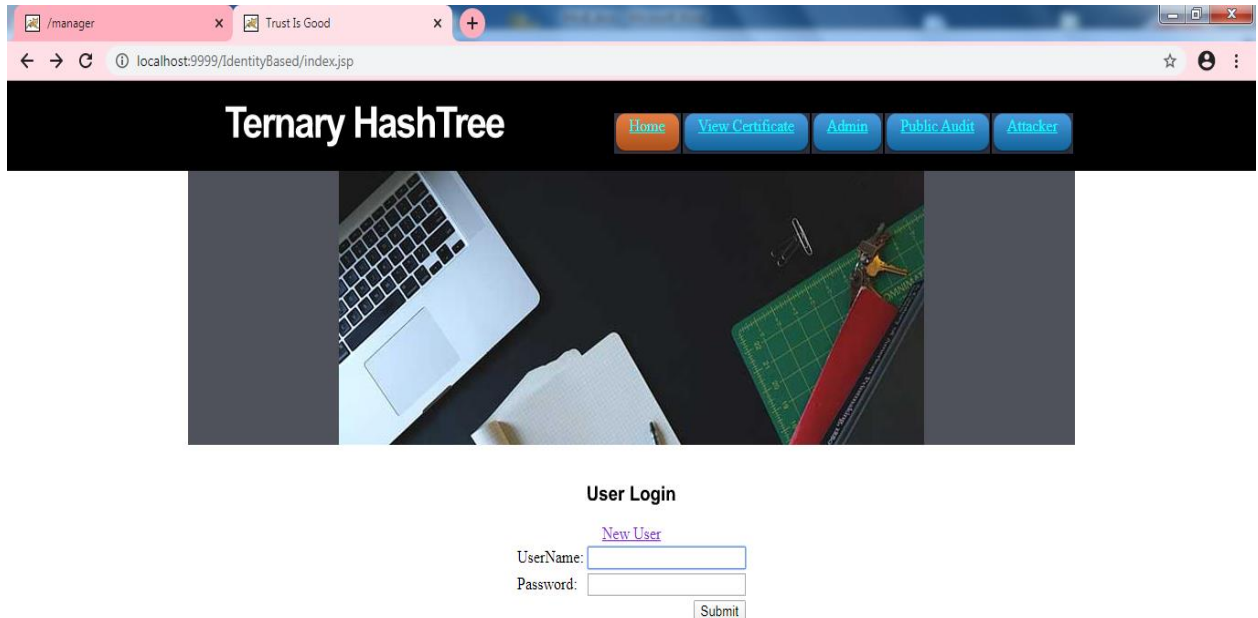
(3) SET NO OF SERVERS:



(4) SET AUDIT TIME:



(5) USER LOGIN:



The screenshot shows a web browser window with the URL `localhost:9999/identityBased/index.jsp`. The page has a black header with the title "Ternary HashTree" and navigation buttons: Home, View Certificate, Admin, Public Audit, and Attacker. Below the header is a large image of a laptop, papers, and a ruler. The main content area is titled "User Login" and contains a "New User" link, input fields for "UserName:" and "Password:", and a "Submit" button.

Ternary HashTree Home View Certificate Admin Public Audit Attacker

User Login

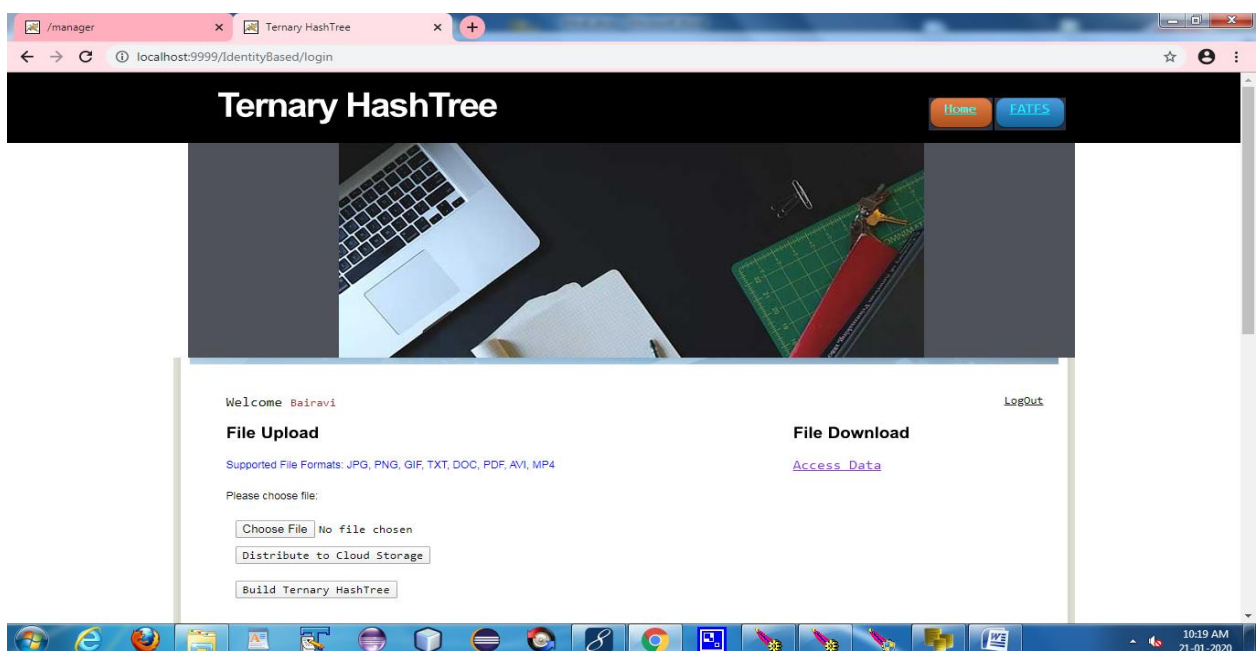
[New User](#)

UserName:

Password:



(6) UPLOAD FILE:



The screenshot shows a web browser window with the URL `localhost:9999/identityBased/login`. The page has a black header with the title "Ternary HashTree" and navigation buttons: Home and FATES. Below the header is a large image of a laptop, papers, and a ruler. The main content area is titled "File Upload" and contains a "Welcome Bairaavi" message, a "LogOut" link, and a "File Download" section with an "Access Data" link. The "File Upload" section includes a "Supported File Formats" list (JPG, PNG, GIF, TXT, DOC, PDF, AVI, MP4), a "Please choose file:" prompt, a "Choose File" button, a "No file chosen" status, a "Distribute to Cloud Storage" button, and a "Build Ternary HashTree" button.

Ternary HashTree Home FATES

Welcome Bairaavi [LogOut](#)

File Upload

Supported File Formats: JPG, PNG, GIF, TXT, DOC, PDF, AVI, MP4

Please choose file:

No file chosen

File Download

[Access Data](#)

(7) BLOCK SPLIT;

File Upload

Supported File Formats: JPG, PNG, GIF, TXT, DOC, PDF, AVI, MP4

Please choose file:

No file chosen

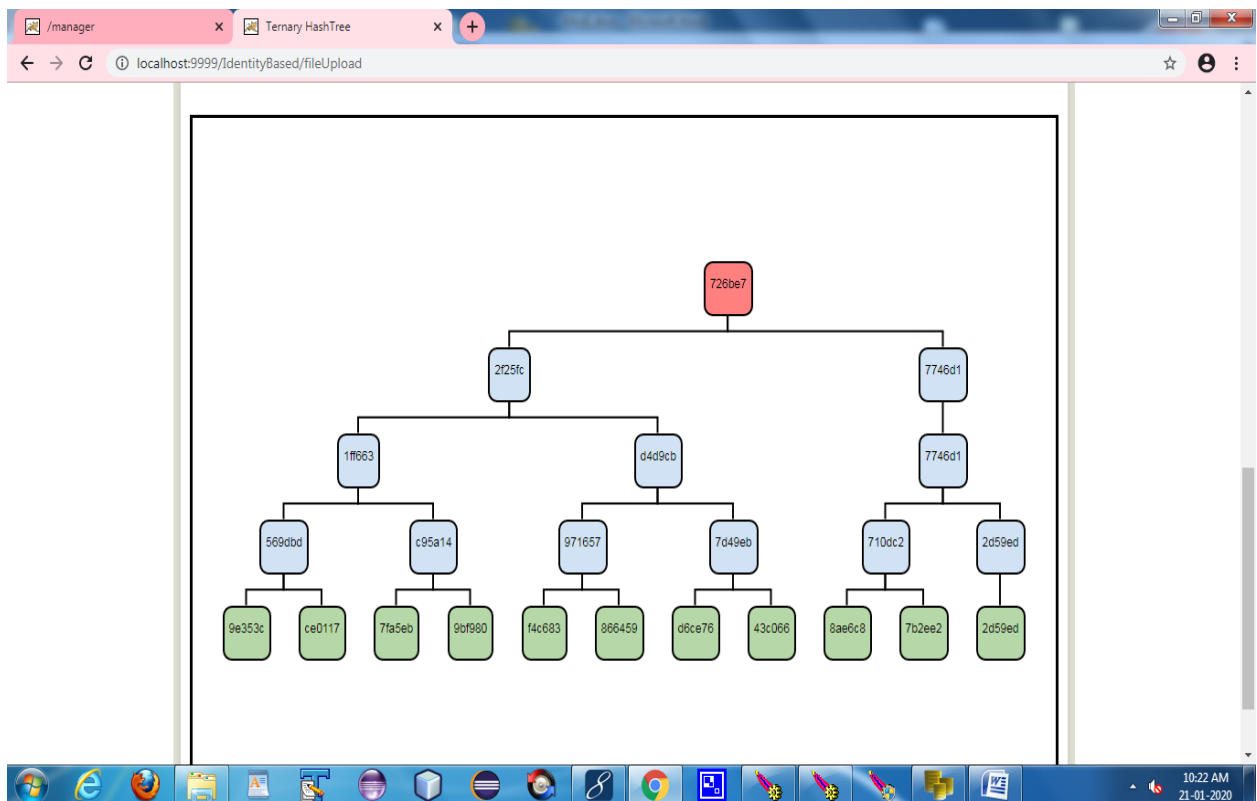
File Uploaded successfully

File Upload Info			
Server	File Name	Block No	Signature
10.0.0.36:8888	Penguins.jpg	BLOCK9	9e353ca3093b6fb52142a6fd80b2eb0e
10.0.0.36:8888	Penguins.jpg	BLOCK5	ce0117d250568b716d8a749079890b95
10.0.0.36:8888	Penguins.jpg	BLOCK1	7fa5eb85dfe814e6d408264740f30826
10.0.0.36:8888	Penguins.jpg	BLOCK2	9bf980bde535f61e4b6357668c60a7c3
10.0.0.36:9999	Penguins.jpg	BLOCK3	f4c683d4dada77b348284f12ee5f6230
10.0.0.36:9999	Penguins.jpg	BLOCK7	866459a35e39b29ff3d05af0906f986d
10.0.0.36:9999	Penguins.jpg	BLOCK6	d6ce764e62a4b9eb01dbfcd20fcc6858
10.0.0.36:9999	Penguins.jpg	BLOCK10	43c066c5ac49079662e81b29e5b378b2
10.0.0.36:9999	Penguins.jpg	BLOCK4	8ae6c88e2c45cb4f2c4868b767dbd922
10.0.0.36:9999	Penguins.jpg	BLOCK0	7b2ee2d85c8e1c394ab1fdc5b923beb6
10.0.0.36:9999	Penguins.jpg	BLOCK8	2d59ed7ef10be36dc12f5f5c363cac63

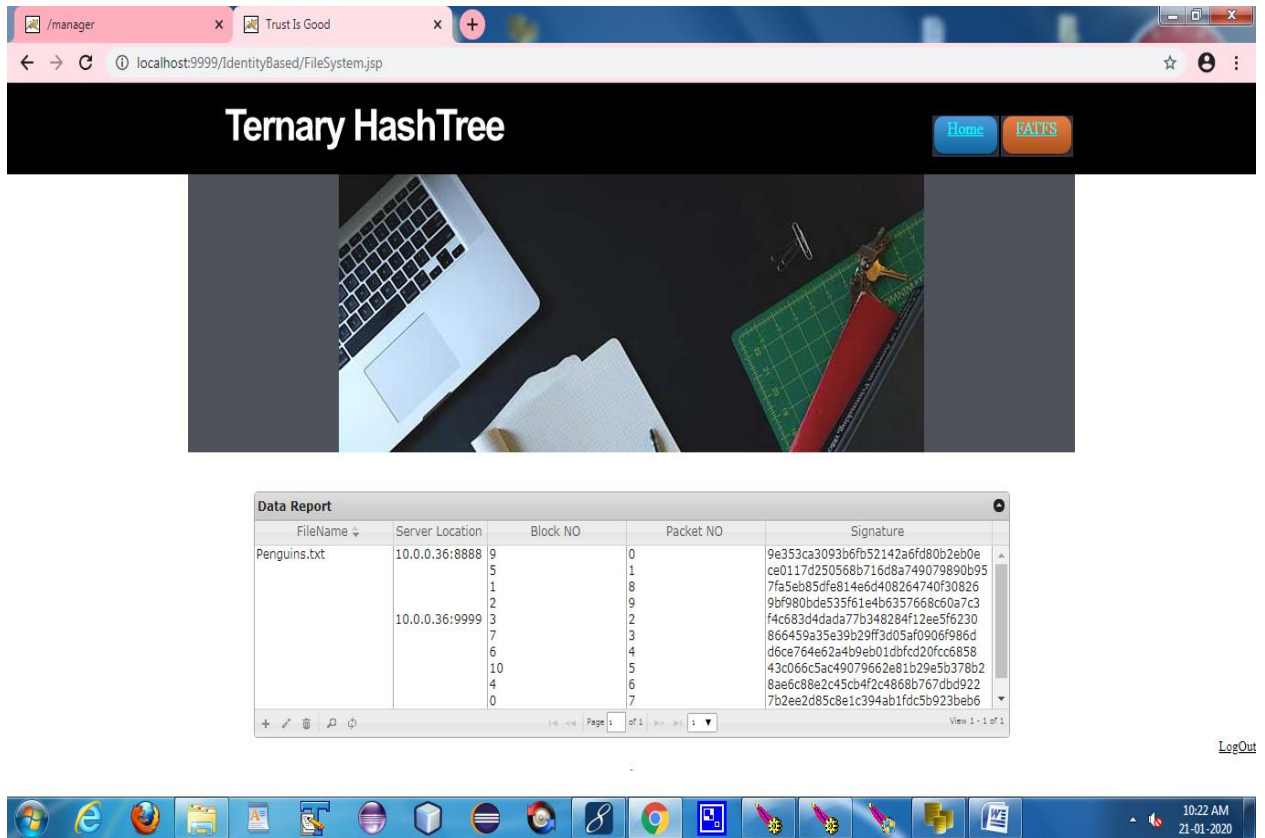
File Download

[Access Data](#)

(8) TERNARY HASH TREE:



(9) FATFS:

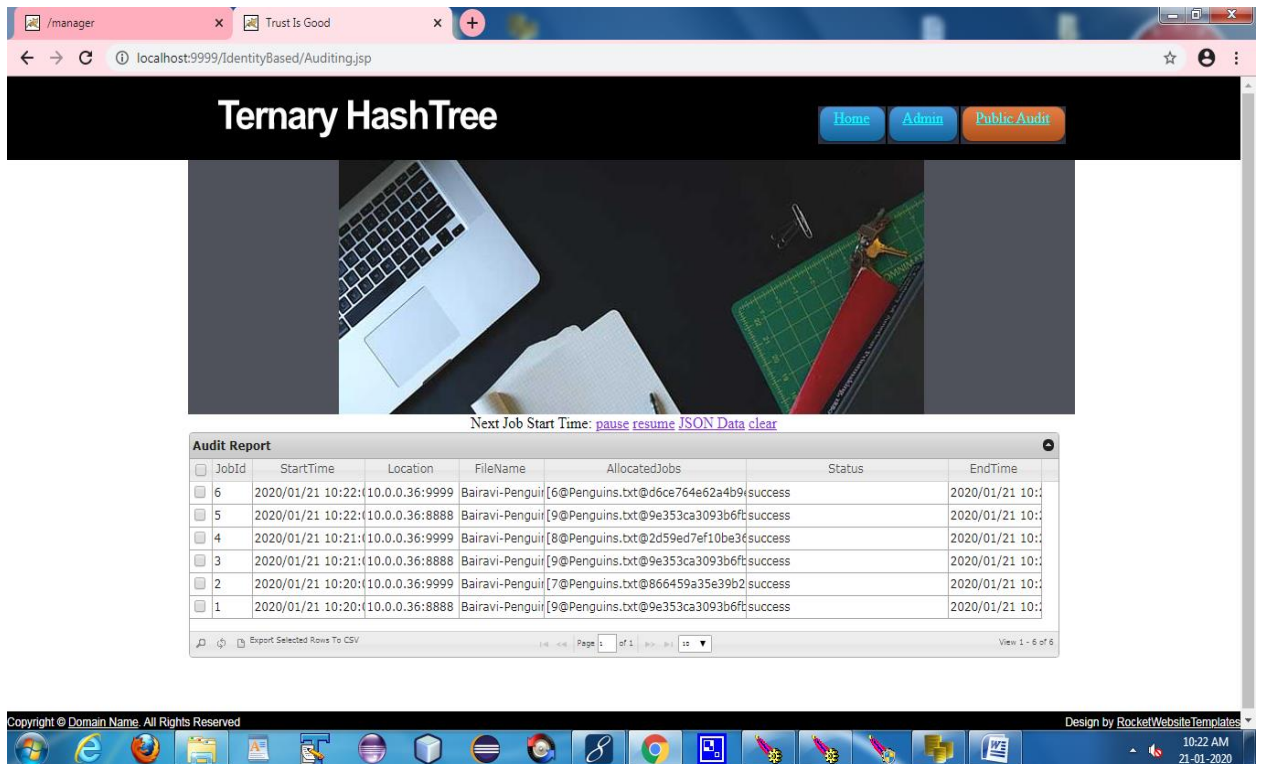


The screenshot shows the Ternary HashTree web application interface. The browser address bar indicates the URL is `localhost:9999/IdentityBased/FileSystem.jsp`. The page title is "Ternary HashTree". Below the title bar, there is a navigation menu with "Home" and "FATFS" buttons. The main content area displays a "Data Report" table with the following data:

FileName	Server Location	Block NO	Packet NO	Signature
Penguins.txt	10.0.0.36:8888	9	0	9e353ca3093b6fb52142a6fd80b2eb0e
		5	1	ce0117d250568b716d8a749079890b95
		1	8	7fa5eb85dfe814e6d408264740f30826
		2	9	9bf980bde535f61e4b6357668c0a7c3
	10.0.0.36:9999	3	2	f4c683d4dada77b348284f12ee5f6230
		7	3	866459a35e39b29ff3d05af0906f986d
		6	4	d6ce764e62a4b9eb01dbfcd20fcc6858
		10	5	43c066c5ac49079662e81b29e5b378b2
		4	6	8ae6c88e2c45cb4f2c4868b767db922
		0	7	7b2ee2d85c8e1c394ab1fdc5b923beb6

At the bottom right of the interface, there is a "Logout" button.

(10) FATFS STATUS

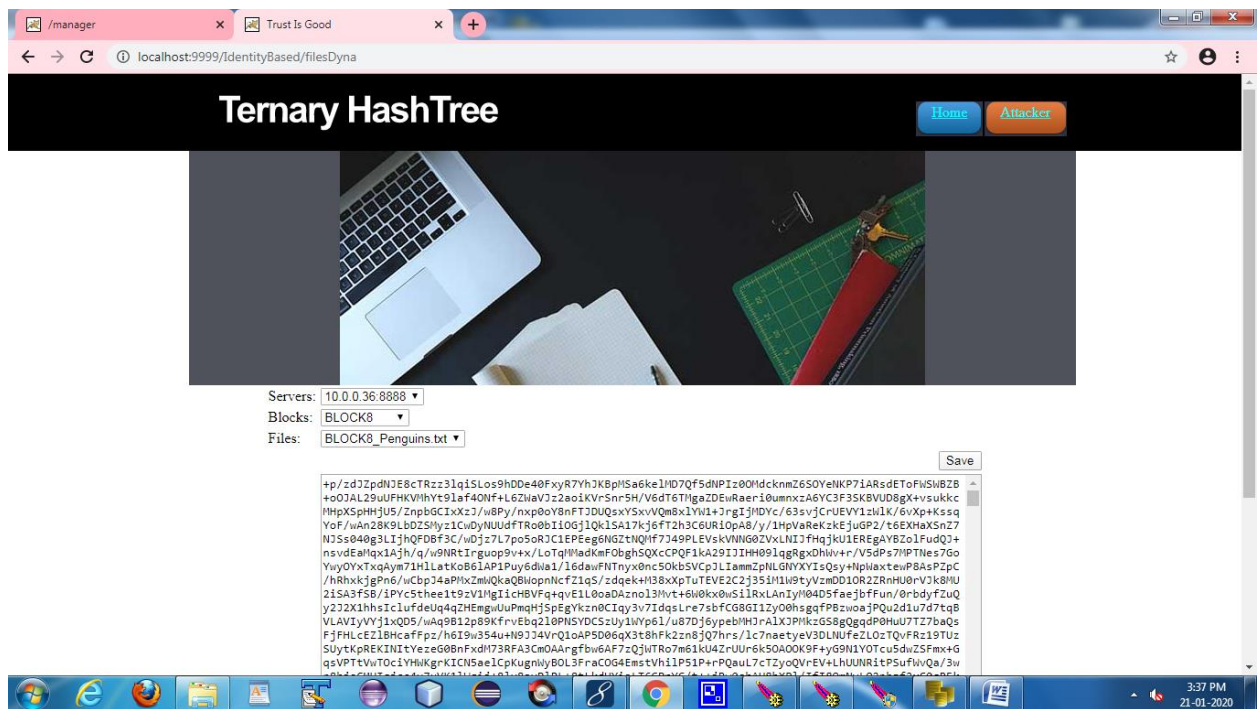


The screenshot shows the Ternary HashTree web application interface for the "FATFS STATUS" section. The browser address bar indicates the URL is `localhost:9999/IdentityBased/Auditing.jsp`. The page title is "Ternary HashTree". Below the title bar, there is a navigation menu with "Home", "Admin", and "Public Audit" buttons. The main content area displays an "Audit Report" table with the following data:

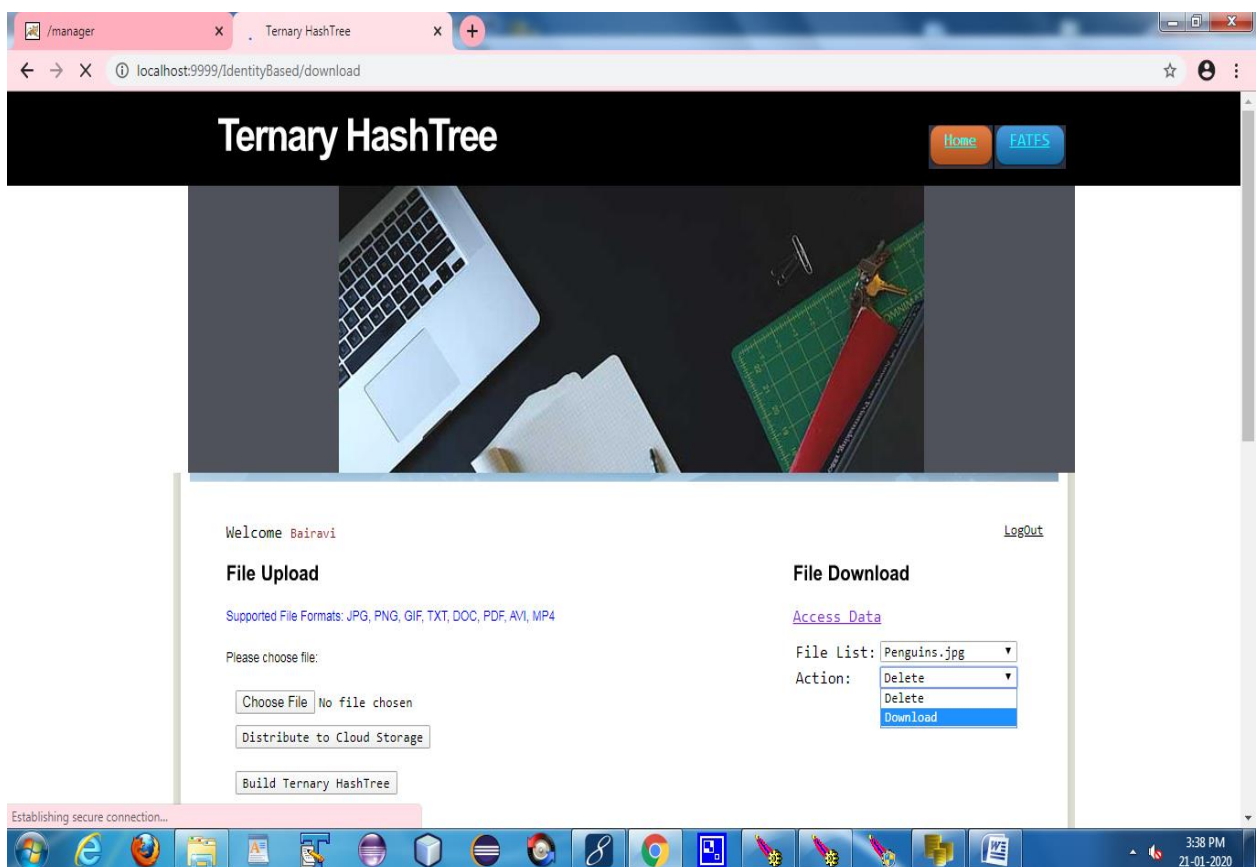
JobId	StartTime	Location	FileName	AllocatedJobs	Status	EndTime
6	2020/01/21 10:22:10	10.0.0.36:9999	Bairavi-Penguin[6@Penguins.txt@	d6ce764e62a4b9	success	2020/01/21 10:22:10
5	2020/01/21 10:22:10	10.0.0.36:8888	Bairavi-Penguin[9@Penguins.txt@	9e353ca3093b6f	success	2020/01/21 10:22:10
4	2020/01/21 10:21:10	10.0.0.36:9999	Bairavi-Penguin[8@Penguins.txt@	2d59ed7ef10be3	success	2020/01/21 10:21:10
3	2020/01/21 10:21:10	10.0.0.36:8888	Bairavi-Penguin[9@Penguins.txt@	9e353ca3093b6f	success	2020/01/21 10:21:10
2	2020/01/21 10:20:10	10.0.0.36:9999	Bairavi-Penguin[7@Penguins.txt@	866459a35e39b2	success	2020/01/21 10:20:10
1	2020/01/21 10:20:10	10.0.0.36:8888	Bairavi-Penguin[9@Penguins.txt@	9e353ca3093b6f	success	2020/01/21 10:20:10

Below the table, there is a "Next Job Start Time: pause resume JSON Data clear" link. At the bottom of the interface, there is a "Copyright © Domain Name. All Rights Reserved" footer and a "Design by RocketWebsiteTemplates" link.

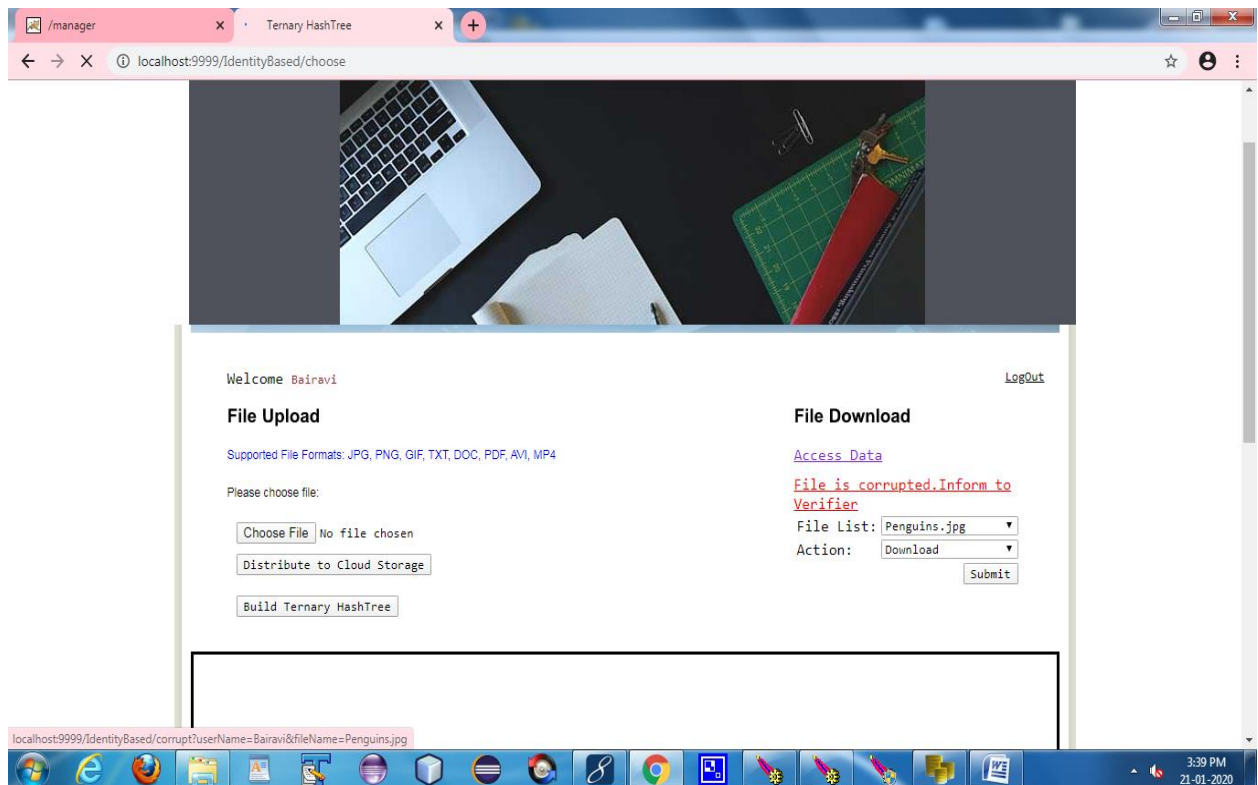
(11) ATTACKER:



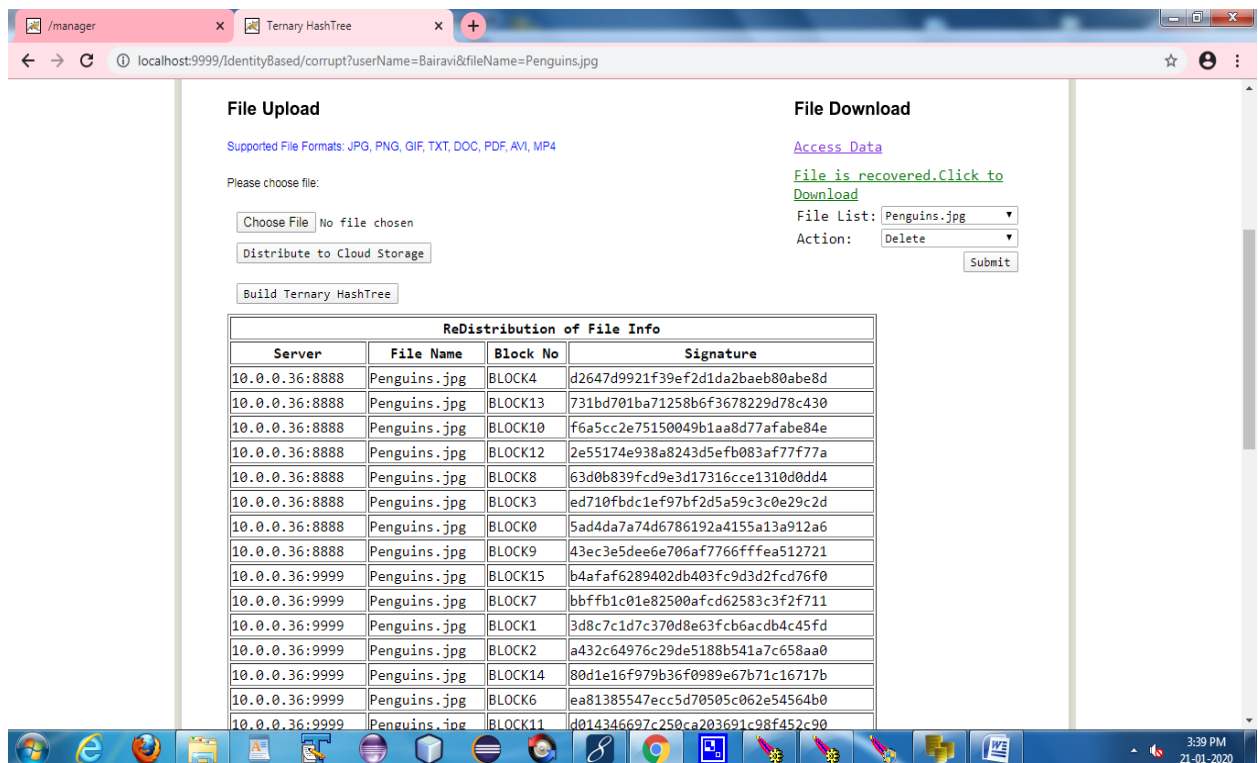
(12) FILE DOWNLOAD:



(13) FILE CORRUPTED NOTIFICATION:



(14) FILE RECOVERED NOTIFICATION:



A.2 PUBLICATIONS

Jayalakshmi S, Sai Veena K, Harini A, Kavitha Subramani, "Creating Secured Multi-Clouds By Continuous Automated Auditing Using Ternary Hash Tree In AWS", **International Journal of Emerging Technologies and Innovative Research (www.jetir.org)**, ISSN:2349-5162, Vol.8, Issue 4, page no.655-661, April2021, Available: <http://www.jetir.org/papers/JETIR2104287.pdf>

REFERENCES

REFERENCES

- [1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable Data Possession at Untrusted Stores,” Proc. 14th ACM Conf. Computer and Comm. Security (CCS’07), pp. 598-609, Oct. 2007.
- [2] M.A. Shah, M. Baker, J.C. Mogul, and R. Swaminathan, “Auditing to Keep Online Storage Services Honest,” Proc. 11th USENIX Workshop Hot Topics in Operating Systems (HotOS ’07), pp. 1-6, 2007.
- [3] M.A. Shah, R. Swaminathan, and M. Baker, “Privacy-Preserving Audit and Extraction of Digital Contents,” Cryptology ePrint Archive, Report 2008/186, <http://eprint.iacr.org>, 2008.
- [4] Vladimir A. Dobrushkin, “Methods in Algorithmic Analysis”, CRC Press, 2009.
- [5] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, “Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing”, IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 5, 847-859, 2011.
- [6] Hao, Z., Zhong, S., and Yu, N, “A privacy-preserving remote data integrity checking protocol with data dynamics and public verifiability”, IEEE Transactions on Knowledge and Data Engineering, Vol.23, 1432 – 1437, 2011.
- [7] Y. Zhu, G. J. Ahn, and Z. Hu, “Dynamic Audit Services for Integrity Verification of Outsourced Storages in Clouds”, IEEE Transactions on Services Computing, Vol. 6, 227-238, 2011.
- [8] Wang, C., Wang, Q., Ren, K., Cao, N., and Lou, W, “Toward secure and dependable storage services in cloud computing”, IEEE Transactions on Services Computing, Vol.5, No.2, 220 – 232, 2012.
- [9] Cong Wang, Sherman S.M. Chow, Qian Wang, Kui Ren, and Wenjing Lou, “Privacy-Preserving Public Auditing for Secure Cloud Storage”, IEEE Transactions on Computers, Vol.62, No.2, 362-375, 2013.

- [10] Yang, K., and Jia, X, “An efficient and secure dynamic auditing protocol for data storage in cloud computing”, IEEE Transactions on Parallel and Distributed Systems, Vol.24, No.9, 1717 – 1726, 2013.
- [11] H. Wang, “Proxy Provable Data Possession in Public Clouds”, IEEE Transactions on Services Computing, Vol.6, No.4, 551 – 559, 2013.
- [12] Boyang Wang, Baochun Li, and Hui Li, “Oruta: Privacy-Preserving Public Auditing for Shared Data in the Cloud”, IEEE Transactions on Cloud Computing, Vol.2, No.1, 2014.