VERIFYING IMAGE AUTHENTICITY AND RECOVERY USING MERKLE TREE BASED MECHANISM

A Major Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

BACHELOR OF TECHNOLOGY
in
Information Technology
by

Akash Kumar (2007340130007)
Krishna Pratap Singh (2007340130031)
Prashant Sharma (2007340130042)
Sundaram Sharma (200734130060)

Under the Supervision of

Dr Gyan Singh
(Assistant Professor)
Rajkiya Engineering College, Banda

to the
Department of Information Technology
DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY
(Formerly Uttar Pradesh Technical University)
LUCKNOW

May, 2024

DECLARATION

We hereby declare that the project entitled “VERIFYING IMAGE AUTHENTICITY AND RECOVERY BY USING MERKLE TREE BASED MECHANISM", submitted by us in the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (IT) of Dr. A.P.J Abdul Kalam Technical University, is a record of our own work carried under our supervision and guidance of **Dr. Gyan Singh**.

To the best of our knowledge this project has not been submitted to Dr. A.P.J Abdul Kalam Technical University or any other University or Institute for the award of any other degree.

NAME: **Candidate’s Signature**

Akash Kumar (2007340130007)

Krishna Pratap Singh (2007340130031)

Prashant Sharma (2007340130042)

Sundram Sharma (2007340130060)

CERTIFICATE

Certified that **Akash Kumar** (2007340130007**), Krishna Pratap Singh** (2007340130031), **Prashant Sharma** (2007340130042), **Sundaram Sharma** (2007340130060) has carried out the work presented in this major project entitled **“Verifying Image Authenticity and Recovery by using Merkle Tree based Mechanism”** for the award of **Bachelor of Technology** from Dr. APJ Abdul Kalam Technical University, Lucknow under our supervision. The report embodies result of original work, and studies are carried out by the students himself and the contents of the report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University or Institution.

Signature:

Dr. Gyan Singh

(Assistant Professor)

(Rajkiya Engineering College Banda)

Date:

ABSTRACT

In today's digital world we see a lot of pictures that are real as well as fake, making it hard to trust their authenticity. There are several cases where we see the image copyrighting issues. This project is based on the authenticity and recovery of the image. And, by this project we can knows the part where our images being tampered. There are many images authentication techniques are used. But there are several drawbacks shown in different – different type of algorithm which are unable to give accurate solution in every field. e.g. Water Marking technique has one most user experience related drawback that is – Visible watermarks may degrade image quality; Cryptographic authentication technique has drawback that is hash function are very sensitive means the image is said to be manipulated even when one bit of this image is changed; Machine learning based authentication technique drawback is models may overfit to the training data and generalize poorly to new images. Our solution is to finding a better way make sure a picture is real and have not been tempered with. Merkle Tree is a data structure used for data validation. The Merkle tree root in the blockchain ensures a secure storage environment for image features. The verification process involves obtaining the hash value of the Merkle tree node on the path, enhancing image integrity. Additionally, the method incorporates the Inter-Planetary File System (IPFS) to boost image availability. The primary objective is to achieve reliable image integrity verification, allowing not only verification of image integrity but also restoration of tampered areas in case of image manipulation.

Keywords: Image Authentication, AES encryption technique, Merkle Tree, Hash Function, Distributed Storage.

ACKNOWLEDGEMENT

We would also like to express our sincere gratitude towards **Dr. Gyan Singh** who provided their guidance in every possible way whenever we were struck to help us out of the dilemma and thanks for helping us throughout the duration of the project and clearing our doubts.

We would like to express our vote of thanks towards the Head of Department of our branch **Dr. Vibhash Yadav** for his constant support and motivation during these difficult times.

We also thank all the staff members for their help rendered.

Signatures:
Akash Kumar (2007340130007)
Krishna Pratap Singh (2007340130031)
Prashant Sharma (2007340130042)
Sundaram Sharma (2007340130060)

TABLE OF CONTENT

	Page No.
Declaration	ii
Certificate	iii
Abstract	iv
Acknowledgement	v
Table of content	vi
List of Tables	viii
List of Figures	ix
List of Abbreviations	x
CHAPTER 1: INTRODUCTION	**1-3**
1.1 Problem Statement	2
1.2 Objective	2
1.3 Background	2
CHAPTER 2: LITERATURE SURVEY	**4-7**
CHAPTER 3: PROJECT METHODOLOGY	**8-19**
3.1 Project Execution Steps	8
3.1.1 Images Uploading by users on interface	8
3.1.2 Slicing of image	9
3.1.3 Encryption	9
3.1.4 Upload to IPFS	10
3.1.5 Repeat step (1-4) for damage image	13
3.1.6 Merkle Tree Generation	14
3.1.7 Merkle tree comparison (CIds of original and damaged image)	16
3.1.8 Downloading block of Image from IPFS	17
3.1.9 Decryption	17
3.1.10 Highlight the block of image (Red shade)	18

3.1.11 Merging of blocks 19

CHAPTER 4: RESULT AND DISCUSSION 20-21

CHAPTER 5: CONCLUSION AND FUTURE SCOPE 22

REFERENCES

APPENDIX

LIST OF TABLES

Table No	Caption	Page No
2	Research Papers	4-5
3.1.1	Difference in Centralized, Decentralized and Distributed File System	11
3.1.2	Advantage and Disadvantage of Decentralized File System	12

LIST OF FIGURES

Figure No	Caption	Page No
3.1.1.a	Image to block	8
3.1.2.a	Image to Slicing	9
3.1.3.a	Slicing to Image	9
3.1.4.a	IPFS	10
3.1.5.a	Upload image	13
3.1.5.b	Background process	13
3.1.6.a	Merkle Tree	14
3.1.6.b	Background	15
3.1.6.c	Damage Image	15
3.1.7.a	Compare Root	16
3.1.7.b	Process	16
3.1.7.c	Calculate manage	17
3.1.9.a	Decryption	17
3.1.11.a	Merging the Block	18
3.1.11.b	Background	19
3.1.11.c	Workflow Chart	19
4.1.a	Original Image 1	20
4.1.b	Damage Image 1	20
4.1.c	Output Image 1	20
4.2.a	Original Image 2	21
4.2.b	Damage Image 2	21
4.2.c	Output Image 2	21

LIST OF ABBREVIATIONS

Acronym	**Full Form**
IPFS	Inter Planetary File System
CID’S	Content Identifiers
AES	Advanced Encryption Standards
SHA	Secure Hash Algorithm
DES	Data Encryption Standards
RSA	Rivest Shamir Adleman
NFT	Non-Fungible Token

CHAPTER 1

INTRODUCTION

In recent years, the internet has become a crucial part of our daily lives, especially for sharing digital content like images. The creation of content such as digital images is getting easier than before. It can be produced by a digital camera, scanner, computer software, etc. However, this ease of sharing comes with risks, as images can be easily stolen, tampered with, or misused by unauthorized users. Ensuring the integrity of digital image content has become a significant area of research in digital image management. Digital images are easy to copy and transmit anywhere over public computer networks, and can be spread across the world very fast. Also, it could easily be tampered with and transformed by the unexpected user. Thus, the integrity of a digital image is determined by checking whether an image has been tampered with or not. Generally, the integrity is not only checking for differences between original and altered images; it is also about pinpointing exactly where changes have occurred.

We have developed sophisticated techniques, including information hiding, to embed image features for protection. Inspired by the secure principles of blockchain technology and the decentralized storage capabilities of the Inter-Planetary File System (IPFS), a new image management system has been proposed. IPFS, or the Inter-Planetary File System, is a peer-to-peer distributed storage system designed to link all computing devices to a unified file system, enabling large-scale data storage. Despite its capabilities, IPFS lacks built-in functionality for verifying the integrity of images. However, recognizing the potential of blockchain technology and IPFS, we were inspired to develop a novel image integrity verification scheme based on blockchain principles. This scheme harnesses the security features of blockchain and the decentralized storage capabilities of IPFS to authenticate the integrity of digital images, ensuring their trustworthiness and reliability in an interconnected digital landscape.

This project employs Merkle tree technology within the blockchain to generate image verification data, while IPFS handles the decentralized storage and management of this data. By utilizing IPFS, the system aims to overcome the limitations of centralized image management and ensure image authentication without relying on a trusted third party. Each node in the blockchain network performs image verification, enhancing security and trustworthiness in image sharing and management on the internet.

Image authentication and recovery play pivotal roles in maintaining the reliability and integrity of digital image content. Authentication ensures that images have not been tampered with or altered in any unauthorized manner, thereby build trust in the authenticity of shared content. Additionally, the ability to recover tampered areas within images is essential for rectifying any unauthorized alterations and restoring images to their original state. This capability not only safeguards against malicious manipulation but also enhances the credibility of image content across various online platforms, fostering a more secure and trustworthy digital environment for image sharing and management.

1.1 PROBLEM STATEMENT

Verifying Image Authenticity and Recovery Using Merkle Tree based Mechanism.

1.2 OBJECTIVE

- Generate a new image authentication method based on Merkle Tree
- Develop user friendly UI (user interface) so that user can easily upload their image
- Detect the damaged part of the image if altered
- Recovery of the damaged part

1.3 BACKGROUND

Background and History of Image Authentication and Recovery System:

The integrity of digital image was protected by several methods like cryptographic image authentication, fragile and semi fragile watermarking. In cryptographic image authentication, there are two operations used one is encryption and second is decryption. Traditional cryptography algorithms exhibit satisfactory results for image authentication with high tamper detection. Drawback of this technique is that hash function is very sensitive means the image is said to be manipulated even when one bit of this image is changed. In watermarking image authentication message or any code has been hidden in the image. The purpose of a watermarking method can vary, but commonly it is used to verify the authenticity or ownership of the image.

In our project we are trying to resolve the problem using Merkle Tree based Mechanism and IPFS (Inter-Planetary File System). The project suggests using technologies like the Inter-Planetary File System (IPFS) and blockchain, which is famous for cryptocurrencies like Bitcoin, along with Merkle trees. This combination aims to create a secure and decentralized system for managing and verifying digital image integrity. The proposed method does not rely on a single trusted party; instead, each node in the blockchain network can independently verify the integrity of images, making it a more reliable and tamper-resistant approach.

Image authentication and recovery utilizing a Merkle-based mechanism ensures the reliability and trustworthiness of digital images between the challenges of potential tampering and manipulation. Traditional authentication methods may fall short in providing robust protection or may impose significant computational overhead. Merkle trees offer a solution by structuring hash values hierarchically, facilitating efficient verification. In this context, a Merkle tree is generated from the original image's pixels or features, with each node representing a hash value computed from its child nodes. The root of the tree, known as the Merkle root, encapsulates the overall image integrity.

Should tampering occur, inconsistencies between the hash values of the modified image and the Merkle tree would be detected, aiding in pinpointing the altered regions. This mechanism not only ensures authentication but also facilitates image recovery by identifying and rectifying alterations. Its efficiency and scalability make it applicable across diverse domains such as forensics, medical imaging, satellite imagery, and digital archives, where maintaining the integrity and authenticity of visual data is paramount for trust and reliability

Validation of fake or real images is very important in various scenarios like:

Digital forensic: In criminal investigation images are used as evidence.

Medical Imaging: In the field of medicine, image authentication is critical for ensuring the integrity of medical images such as X-rays, MRIs, and CT scans. It helps maintain the accuracy of diagnostic information and ensures patient safety by preventing manipulation of medical images.

Legal Documentation: Image authentication is used in legal proceedings to validate the authenticity of digital documents, photographs, and evidence presented in court. This ensures the integrity of legal documentation and prevents fraudulent manipulation of visual evidence.

Historical Preservation: In cultural heritage preservation, image authentication ensures the integrity of digital copies of historical documents, artifacts, and artworks. This helps safeguard the authenticity and integrity of cultural heritage for future generations.

Digital Rights Management (DRM): Image authentication plays a crucial role in DRM systems by protecting digital images from unauthorized copying, distribution, and alteration. This helps content creators and publishers maintain control over their intellectual property and prevent piracy.

Remote Sensing and Satellite Imaging: In remote sensing applications, such as environmental monitoring and land use mapping, image authentication ensures the reliability of satellite images and aerial photographs used for scientific research and analysis.

Biometric Identification: Image authentication is used in biometric systems for verifying the authenticity of biometric images, such as fingerprints, iris scans, and facial recognition images. This ensures the accuracy and reliability of biometric identification systems in security and authentication applications.

Authentication in Online Platform: Image authentication can be useful in online platforms where users upload images, such as social media or e-commerce websites. This helps prevent the spread of fake or manipulated images.

Overall, image authentication plays a crucial role in ensuring the integrity, authenticity, and trustworthiness of digital images across a wide range of applications, from forensic investigations and medical imaging to legal documentation and cultural heritage preservation.

CHAPTER 2

LITERATURE SURVEY

In the pursuit of enhancing the image authentication and recovery, our project examines into a comprehensive literature survey to explore the existing knowledge and advancements of image authentication. This section critically examines studies, research articles, and technological frameworks related to image authentication methods and recovery and its applications in various domains.

Following table shows the list of research papers used to make this project successful:

Table 2 Research Papers

S. No.	**Name of Research Paper**	**Author Name**	**Technique Used**	**Source**	**Finding of the study**
1	A Study on Image Authentication Methods [1]	Marakumbi Prakash, Jayashree V. Khanapuri (2018)	Cryptographic Authentication, Watermarking Authentication	IRJET	Watermarking scheme constructed based on chaotic maps is more efficient compared to other method
2	Methods of Image Authentication : a survey [2]	Adil Hauzia, Rita Noumeir (2007)	Fragile Watermarking, Semi Fragile Watermarking Methods	Springer	Fragile watermarking detects tampering, semi-fragile tolerates changes, and content-based methods enhance Security

3	The Jpeg Blockchain Framework for Glam Services [3]	Dr Bhowmik, T. Feng, A. Natu, T. Ishikawa, C. Abhayaratne (2018)	Image Processing	IEEE	Verified information is stored in JPEG header and image is stored in multimedia server
4	Multipurpose Watermarking for Image Authentication and Protection [4]	Chun-Shien Lu	Watermarking Technique	IEEE	It gives a comprehensiv e solution for authenticatio n and protection of images
5	Digital Image Watermarking Techniques: A Review [5]	Mahbuba Begum	Watermarking Technique	IEEE	Digital image watermarking proves to be a vital tool for protecting images
6	Design and Evaluation of IPFS [6]	Aravindh Raman, Gareth Tyson (2022)	Blockchain	SIGCOM	P2P and Decentralized Storage System
7	IPFS P2P File System [7]	Juan Benet	Blockchain	IEEE	IPFS is a single BitTorrent swarm exchanging object within git repository
8	A Secure File Sharing System Based on IPFS and Blockchain [8]	Hsiao-Shan Huang	Blockchain	Research Gate	IPFS proxy tool is used so that to control who can access which file in group
9	A survey on performance analysis of DES, AES, and RSA [9]	B. Padmavathi, S. Ranjitha Kumari (2013)	Encryption algorithms	IJSR	AES is more powerful than DES. Hardware and Software implementati on is faster

10	Merkle tree traversal techniques [10]	Boris Ederov (2007)	Merkle tree data structure And SHA-256 hashing algorithms	Citeseerx	Implementation of Merkle tree and it is used for verification

After a comprehensive review of above research papers exploring different approaches to image authentication, we have arrived at a key conclusion. Each paper presented a unique method:

In paper 1 there are two main image authentication methods describes those are cryptographic authentication, and watermarking authentication. In cryptographic authentication there are two operations used encryption (used to secure a message) and decryption (used to read the secured message). Advantage of this method is high temper detection. And disadvantage of this method is that hash function is very sensitive means the image is said to be manipulated even when one bit of this image is changed. In watermarking image authentication, message or any code has been hidden in the image. The advantage of this method is image tampering detection and embedding watermarks is easy and identify the author of copyright work. The disadvantage of this technique is cannot locate where the images have been tampered and visible watermarks may degrade image quality.

Paper 2 advocated for fragile watermarking and semi fragile watermarking. Fragile watermarking is primarily designed for tamper detection and localization of modifications in the digital content. Advantage of fragile watermarking is that fragile watermarks can often provide information about the location and nature of the tampering, helping in the identification and correction of alterations. Disadvantage of fragile watermarking is that fragile watermarks may not withstand common signal processing operations or compression. Semi-fragile watermarking aims to strike a balance between robustness and sensitivity to intentional tampering. Advantage of semi-fragile methods is that it tolerates desired manipulations. Disadvantage of semi fragile watermarking is that there is often a trade-off between the robustness and fragility of the watermark.

In Paper 3 Dr Bhowmik proposed a blockchain framework for JPEG images that has its own block content. The image related information is hidden in the header of a JPEG image, and the image is stored in a multimedia server. The information for verifying copyright were stored in the blockchain framework. Storing image verification information on the blockchain is a good strategy, but the image is still stored in the centralized manner, or kept by the owner, which will affect the availability of image management. Thus, we proposed a method to solve this problem.

Paper 4 tells us about a clever way to protect images and check if they have been altered. We embed two types of watermarks into the image using a special technique. One watermark is tough and stays intact even if the image is attacked, ensuring copyright protection. The other helps spot any changes made to the image, ensuring its authenticity. Our method works well even if the image goes through some changes. Tests prove it is great at both protecting images and checking their authenticity.

Paper 5 explore ensuring the authenticity of digital images is crucial in today's digital era due to the ease of tampering. Researchers have developed various watermarking techniques to address this concern, tailored to different applications. However, balancing robustness and security remains a challenge. This paper outlines standard watermarking frameworks and requirements for different applications, reviews current trends in digital image watermarking, and discusses common attacks. Looking ahead, future research aims to enhance techniques against emerging threats.

Paper 6 and 7 explores the design, implementation, and deployment of the Inter Planetary File System (IPFS). Emphasizing widespread adoption, the study covers IPFS's global reach and its hybrid gateway design. In some ways, IPFS is similar to the web but IPFS could be seen as a single BitTorrent swarm exchanging object within git repository.

Paper 8 tells us about securely share files using a special kind of technology called blockchain and another one called IPFS. We use a tool called an IPFS proxy to control who can access which files in a group. Even though the basic technologies do not have built-in access control, our system manages it so that only authorized group members can see specific files.

Paper 9 emphasized on the encryption and decryption algorithm that is based on symmetric key algorithm (DES, AES) and asymmetric algorithm (RSA). Both algorithms are used for sharing the data with proper encryption and decryption algorithm so that no one in middle can detect what exact data is transferred. In symmetric key algorithm in this paper AES is more powerful than DES. While comparing with AES, DES, and RSA we found that AES encryption algorithm is more secure than other ones, power consumption is also low compare to RSA, Hardware and software implementation is faster and efficient compare to others.

Paper 10 introduced Merkle tree that is a data structure used for data verification and synchronization. Paper tells about the how Merkle tree works and traversals for classical Merkle tree.

CHAPTER 3

PROJECT METHODOLOGY

The major steps involved in verifying image authentication and recovery using Merkle tree-based mechanism are slicing of image, encryption of blocks, uploading encrypted block to IPFS, Comparison of Cid’s, fetching blocks, decryption, block highlighting, merging the blocks of image. After these steps user gets the highlighted damaged part of damaged image and this damaged image recovered with help of original image.

3.1 PROJECT EXECUTION STEPS

Image authentication and recovery system explores how Merkle Trees, a cryptographic data structure, can be employed to authenticate and recover images efficiently and securely.
Following are the steps how this system works:
Step 1: Images uploading by users on interface
Step 2: Slicing of image
Step 3: Encryption
Step 4: Upload to IPFS
Step 5: Repeat step (1-4) for damage image
Step 6: Merkle tree generation
Step 7: Merkle tree comparison (CIDs of original and damaged image)
Step 8: Decryption
Step 9: Highlighting blocks
Step 10: Merging of blocks
Step 11: User gets result

3.1.1 Images Uploading by users on interface

We developed an interface as shown below on which users upload their original image and damaged image in task 1 and task 2.

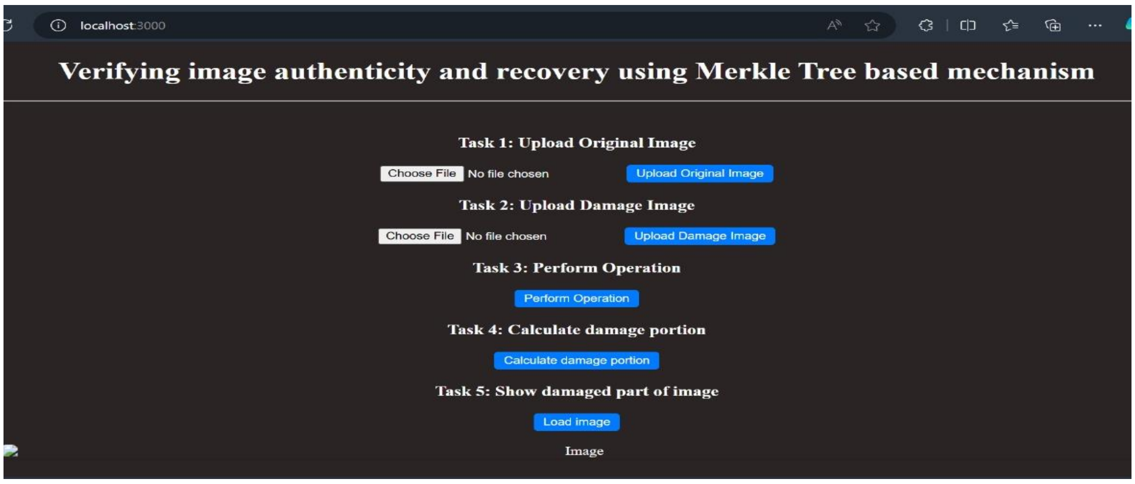

localhost:3000
Verifying image authenticity and recovery using Merkle Tree based mechanism
Task 1: Upload Original Image
Choose File
No file chosen
Upload Original Image
Task 2: Upload Damage Image
Choose File
No file chosen
Upload Damage Image
Task 3: Perform Operation
Perform Operation
Task 4: Calculate damage portion
Calculate damage portion
Task 5: Show damaged part of image
Load image
Image

Figure 3.1.1.a

3.1.2 Slicing of image

After uploading the image on interface, we divide the image into blocks with the help of canvas package. The canvas npm package is a powerful tool that provides a Canvas implementation for Node.js. This package allows developers to use the HTML5 Canvas API in a server-side environment. The canvas package is particularly useful for tasks such as creating and manipulating images, generating dynamic graphics, and even building games or data visualizations that can be rendered server-side. Key features of canvas such as drawing shapes as text, image manipulation, path drawing, transformations, advanced compositioning.

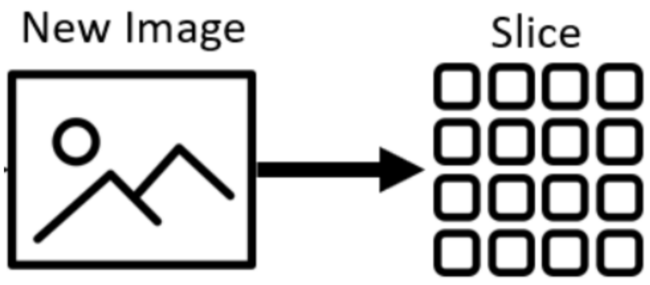


Figure 3.1.2.a

3.1.3 Encryption

Encryption of the block is required because of IPFS is a decentralized file system that anyone who knows the file address can browse to access the files, for file protecting purpose, any encryption system can be adopted to our method. There are various encryption methods of symmetric key encryption as (DES, AES) and asymmetric key encryption as RSA. We prefer to use AES because of faster than other encryption algorithm and more secure than DES.

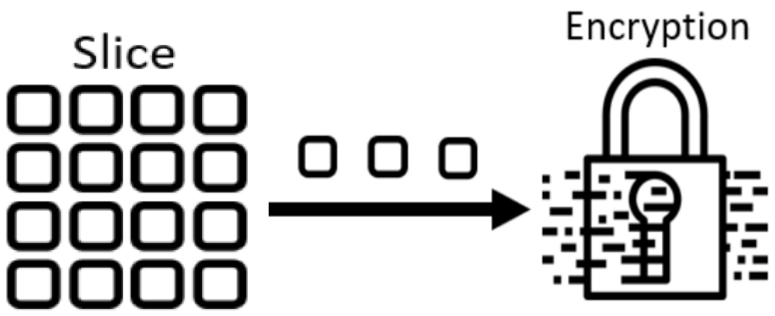


Figure 3.1.3.a

The crypto-js module is a popular JavaScript library for cryptographic algorithms, including AES (Advanced Encryption Standard). To use AES encryption and decryption in Node.js with the crypto-js module.

3.1.4 Upload to IPFS

After encryption of blocks of image, we have to finally upload into IPFS (Inter-Planetary file system) as distributed data storage. IPFS is open source, peer to peer distributed hypermedia protocol use for storing and sharing data in a distributed file system. IPFS uses content addressing to uniquely identify each file in a global namespace connecting IPFS hosts.

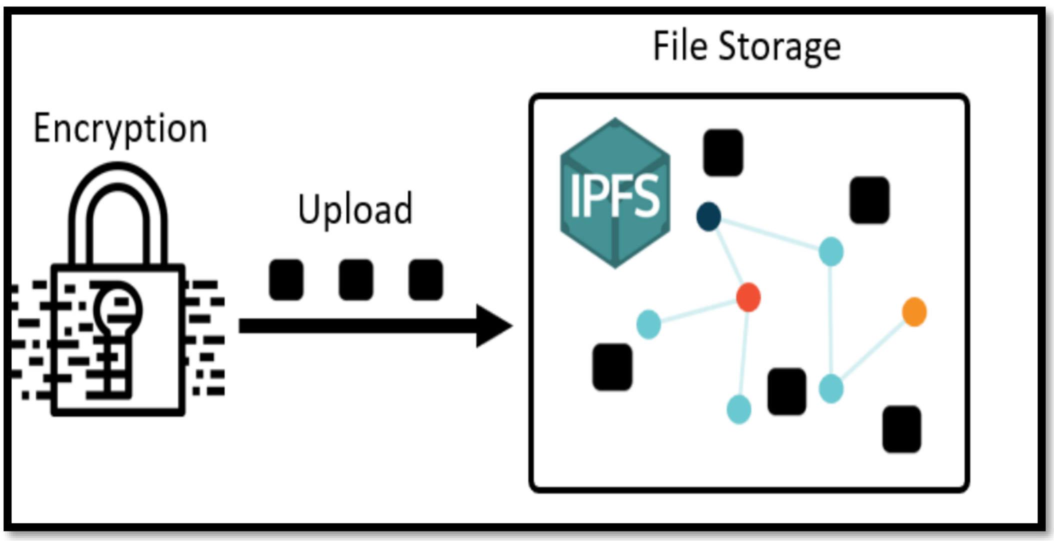


Figure 3.1.4.a

There are three file storage system are existing Centralized, Decentralized and Distributed File System. We used Decentralized file system in our project.

Decentralized File Storage System:
Decentralized file storage is a type of storage solution built on a decentralized network powered by blockchain technology rather than depending on a single centralized organization. Instead of being stored on a single server managed by a single authority, data is preserved on multiple nodes in a decentralized network. Decentralized file storage systems provide increased security by distributing data across multiple nodes, reducing

the risk of data loss or tampering due to a single point of failure; By removing reliance on a central server, decentralized file storage solutions offer greater accessibility to data.

Centralized File Storage System:
In centralized data storage, we store and maintain data in a single location as one central unit. It makes the data distribution simple from the chief unit to different teams/workspaces/business units for various purposes.

Distributed File Storage System:
A distributed file system (DFS) is a file system that spans across multiple file servers or multiple locations, such as file servers that are situated in different physical places. Files are accessible just as if they were stored locally, from any device and from anywhere on the network.

Here is the Difference between Centralized, Decentralized and Distributed File System:

Table No. 3.1.1 The Difference in Centralized, Decentralized and Distributed File System

Feature	Centralized	Decentralized	Distributed
Control	Single point	Multiple independent points	Coordinated, appears unified
Data Location	Central server	Multiple nodes, independent	Multiple nodes, coordinated
Resilience	Low, single point of failure	High, no single point of failure	High, with redundancy and fault tolerance
Management	Simplified	Complex	Complex but integrated
Scalability	Limited	Better than centralized but requires robust synchronization	High, designed for horizontal scaling
Security	Easier to implement and manage in one location	Challenges in managing multiple nodes	Complex but integrated security
Availability	Dependent on the central server	High, due to redundancy	High, due to coordinated redundancy
Backup and Recovery	Simplified due to centralized control	More complex due to multiple nodes	Managed across distributed nodes
Complexity	Lower	Higher due to lack of central control	Higher due to coordinated distribution

Pinata: A Platform for Uploading Images to IPFS

Pinata simplifies the process of uploading images to the Inter-Planetary File System (IPFS) by providing a user-friendly interface and robust API. It allows users to easily upload images, which are then assigned a unique cryptographic hash for identification. Pinata ensures the images remain available on IPFS by pinning them, preventing deletion and guaranteeing persistent access. Additionally, Pinata offers public gateway URLs for easy web access, making it a convenient solution for managing decentralized storage without the need to run an IPFS node.

The URL “https://api.pinata.cloud/pinning/pinFileToIPFS” is an endpoint provided by Pinata for uploading files to the Inter-Planetary File System (IPFS). This API endpoint allows users to programmatically pin files to IPFS through Pinata's infrastructure. When a file is uploaded via this endpoint, it receives a unique IPFS hash and is pinned to ensure its availability on the IPFS network.

For Authentication and authorization, we use pinata API keys. These keys are essential for interacting with the Pinata API to upload, pin, and manage files on IPFS.

After uploading blocks of an image to IPFS, we receive a CID as a response. This CID is crucial for downloading or retrieving blocks of the image from IPFS.

A CID is a unique string that IPFS uses to identify and access content. It is derived from the cryptographic hash of the content itself, ensuring that the CID is unique to that specific piece of data.

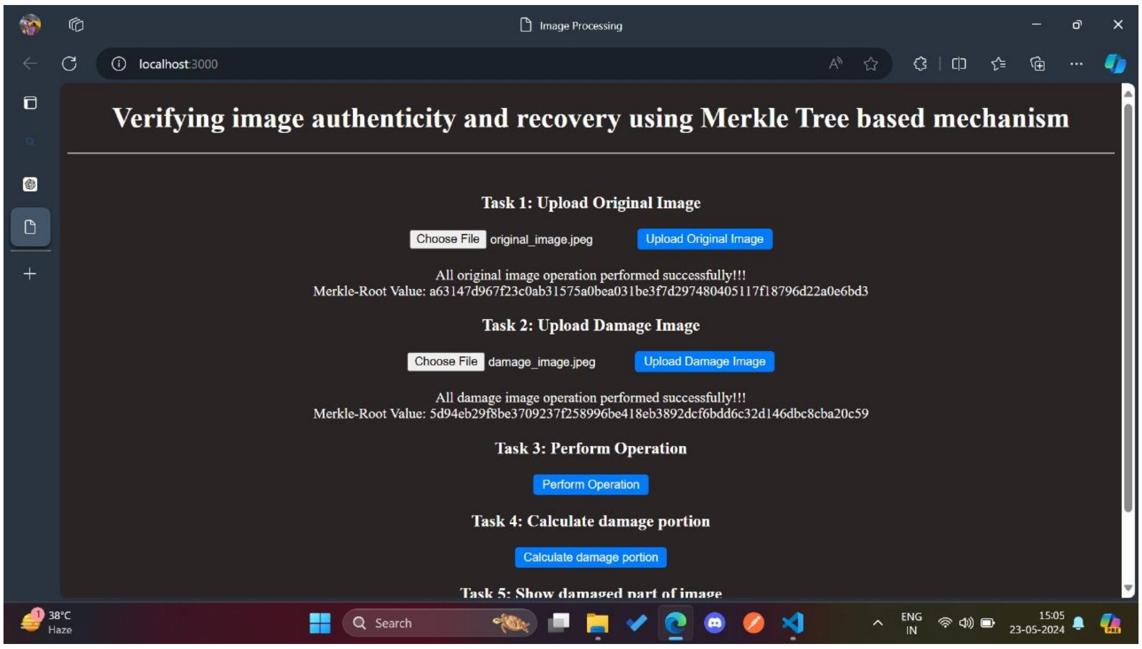
We used Decentralized file system So here are the following advantages and disadvantages of decentralized file system.

Table No. 3.1.2 Advantages and Disadvantages of Decentralized System

Aspect	Advantages	Disadvantages
Resilience	High, no single point of failure	Complex management of multiple nodes
Fault Tolerance	System continues to operate even if some nodes fail	Potential for data inconsistency without robust synchronization
Scalability	Better scalability compared to centralized systems	Coordination and synchronization challenges
Performance	Improved performance as load is distributed	Possible latency issues due to network communication
Redundancy	Data redundancy enhances availability and reliability	Increased storage and replication overhead
Security	Reduced risk of a single point of attack	More complex to implement consistent security policies
Maintenance	Localized issues can be resolved independently	Overall system maintenance is more complex
Network Congestion	Reduced congestion as load is distributed	Potential for higher bandwidth usage due to data replication

3.1.5 Repeat step (1-4) for damage image

From step 1 to step 4, the same tasks done for the original image should also be performed for the damaged image, such as converting the image into blocks, encrypting the blocks, and storing all the encrypted blocks in IPFS, and storing the CIDs.



Verifying image authenticity and recovery using Merkle Tree based mechanism

Task 1: Upload Original Image

original_image.jpeg

All original image operation performed successfully!!!
Merkle-Root Value: a63147d967f23c0ab31575a0bea031be3f7d297480405117f18796d22a0e6bd3

Task 2: Upload Damage Image

damage_image.jpeg

All damage image operation performed successfully!!!
Merkle-Root Value: 5d94eb29f8be3709237f258996be418eb3892dcf6bdd6c32d146dbc8cba20c59

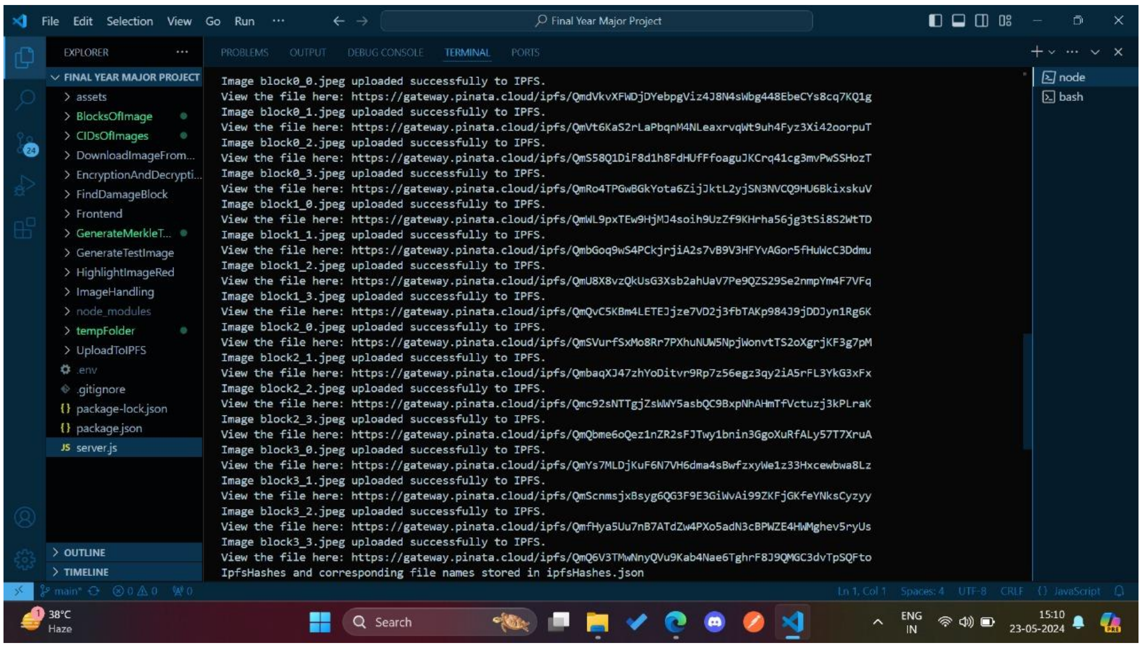
Task 3: Perform Operation

Task 4: Calculate damage portion

Task 5: Show damaged part of image

Figure 3.1.5.a

Background Process:



```
Image block0_0.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmdVkvXFWDjDYebpgViz4J8N4sWbg448EbeCYs8cq7KQ1g
Image block0_1.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmVt6KaS2rLaPbqnM4NLeaxrvqWt9uh4Fyz3Xi42oorpuT
Image block0_2.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmS58Q1DiF8d1h8FdHUfFfoaguJKCrq41cg3mvPwSSHozT
Image block0_3.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmRo4TPGwBGkYota6ZijJktL2yjSN3NVCQ9HU6BkixskuV
Image block1_0.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmWL9pxTEw9HjMJ4soih9UzZf9KHrha56jg3tSi8S2WtTD
Image block1_1.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmbGoq9wS4PCkjrjiA2s7vB9V3HFYvAGor5fHuWcC3Ddmu
Image block1_2.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmU8X8vzQkUsG3Xsb2ahUaV7Pe9QZS29Se2nmpYm4F7VFq
Image block1_3.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmQvC5KBm4LETEJjze7VD2j3fbTAKp984J9jDDJyn1Rg6K
Image block2_0.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmSVurfSxMo8Rr7PXhuNUW5NpjWonvtTS2oXgrjKF3g7pM
Image block2_1.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmbaqXJ47zhYoDitvr9Rp7z56egz3qy2iA5rFL3YkG3xFx
Image block2_2.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/Qmc92sNTTgjZsWWY5asbQC9BxpNhAHmTfVctuzj3kPLraK
Image block2_3.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmQbme6oQez1nZR2sFJTwy1bnin3GgoXuRfALy57T7XruA
Image block3_0.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmYs7MLDjKuF6N7VH6dma4sBwfzxyWe1z33Hxcewbwa8Lz
Image block3_1.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmScnmsjxBsyg6QG3F9E3GiWvAi99ZKFjGKfeYNksCyzyy
Image block3_2.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmfHya5Uu7nB7ATdZw4PXo5adN3cBPWZE4HWMghev5ryUs
Image block3_3.jpeg uploaded successfully to IPFS.
View the file here: https://gateway.pinata.cloud/ipfs/QmQ6V3TMwNnyQVu9Kab4Nae6TghrF8J9QMGC3dvTpSQFto
IpfsHashes and corresponding file names stored in ipfsHashes.json
```

Figure 3.1.5.b

3.1.6 Merkle Tree Generation

Cid (content identifier) are used to generate Merkle tree. A Merkle tree is a data structure used in computer science and cryptography for efficiently verifying the integrity and consistency of large datasets. It organizes data into a hierarchical tree structure, where each leaf node represents a small chunk of the original data, and each non-leaf node represents a cryptographic hash of its child nodes. By recursively hashing pairs of nodes up the tree until a single root hash is obtained, the Merkle tree enables quick verification of data integrity. If any part of the dataset is altered, it will result in changes to the leaf nodes, which in turn will affect the root hash, signalling potential tampering. Merkle trees are widely used in blockchain technology, distributed file systems like IPFS, and data synchronization protocols to ensure the security and reliability of data.

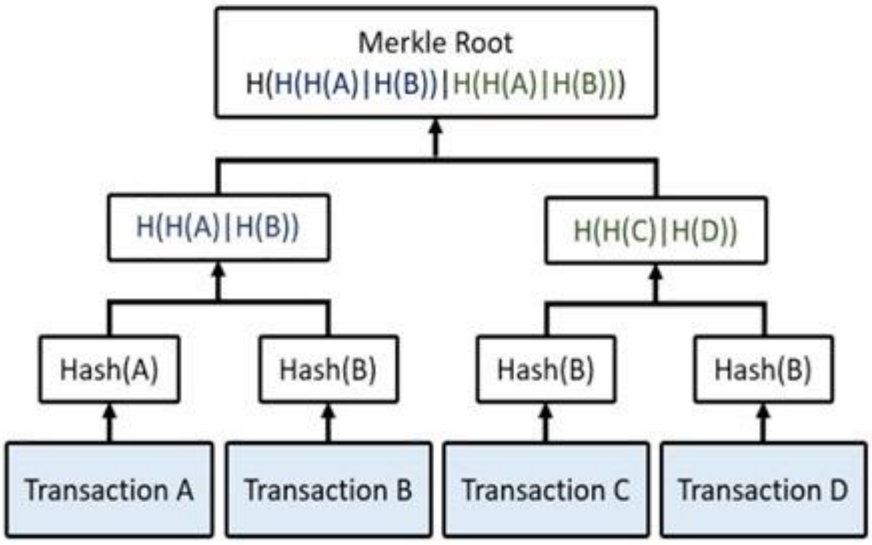
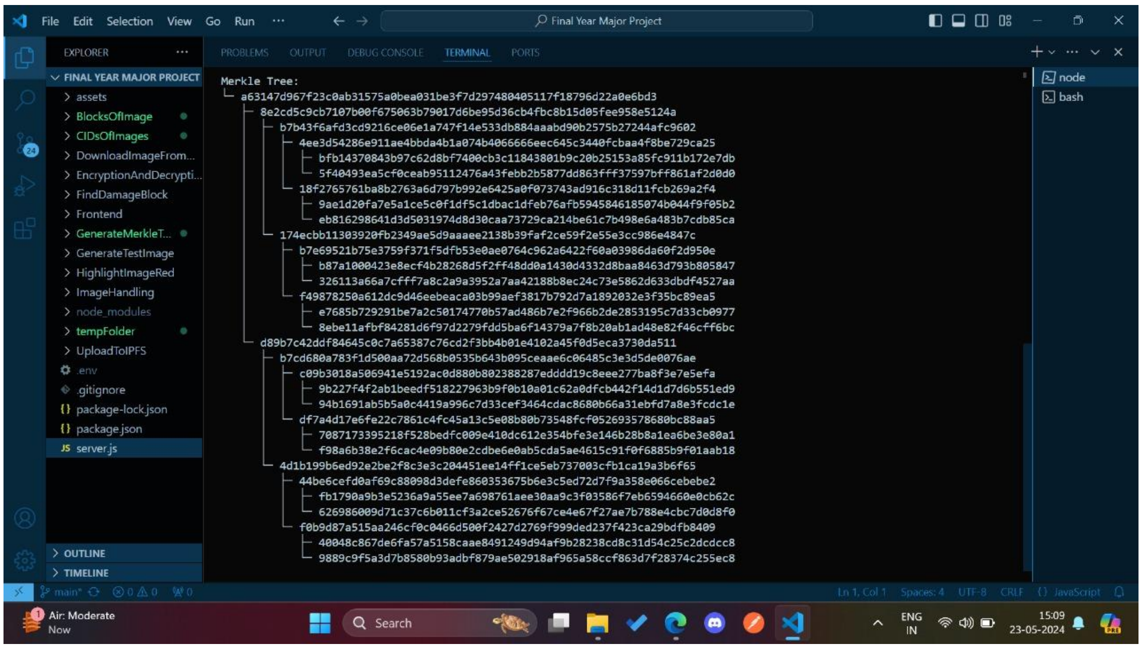


Figure 3.1.6.a

MerkleTree.js is a Node.js library that provides an easy-to-use implementation of Merkle trees for developers working with JavaScript. With MerkleTree.js, you can quickly create Merkle trees from datasets, verify data integrity efficiently, and generate Merkle proofs for specific elements in the tree. This library abstracts away the complexities of building and managing Merkle trees, offering simple and intuitive functions to interact with them. Whether you're working on blockchain applications, data verification systems, or decentralized storage solutions like IPFS, MerkleTree.js simplifies the process of implementing and leveraging Merkle trees in your projects.

Crypto-js/sha256 is a JavaScript library called CryptoJS that provides a convenient way to compute the SHA-256 hash function. SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that produces a 256-bit (32-byte) hash value from an input data of arbitrary size. It is commonly used in various applications, including digital signatures, password hashing, and blockchain technology. With crypto-js/sha256, you can easily calculate the SHA-256 hash of strings, files, or any other data in JavaScript environments, making it a versatile tool for ensuring data integrity and security in web applications and beyond.

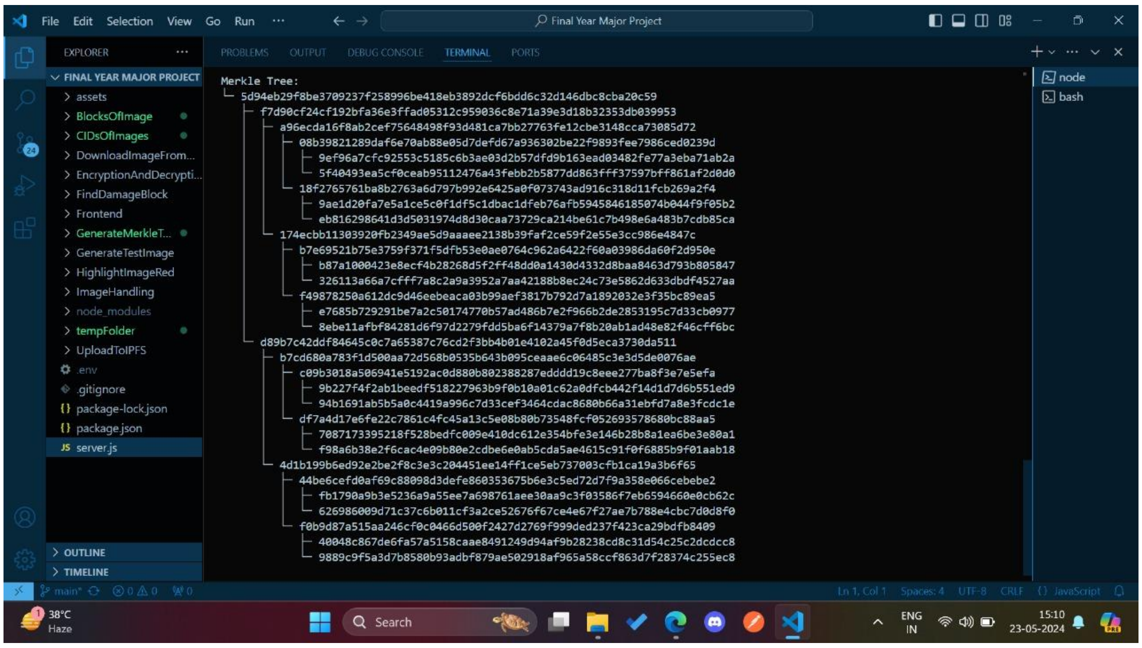
Background Merkle Tree generation process for original Image:



```
Merkle Tree:
└─ a63147d967f23c0ab31575a0bea031be3f7d297480405117f18796d22a0e6bd3
   ├─ 8e2cd5c9cb7107b00f675063b79017d6be95d36cb4fbc8b15d05fee958e5124a
   │  ├─ b7b43f6afd3cd9216ce06e1a747f14e533db884aaabd90b2575b27244afc9602
   │  │  ├─ 4ee3d54286e911ae4bbda4b1a074b4066666eec645c3440fcbaa4f8be729ca25
   │  │  │  ├─ bfb14370843b97c62d8bf7400cb3c11843801b9c20b25153a85fc911b172e7db
   │  │  │  └─ 5f40493ea5cf0ceab95112476a43febb2b5877dd863fff37597bff861af2d0d0
   │  │  └─ 18f2765761ba8b2763a6d797b992e6425a0f073743ad916c318d11fcb269a2f4
   │  │     ├─ 9ae1d20fa7e5a1ce5c0f1df5c1dbac1dfeb76afb5945846185074b044f9f05b2
   │  │     └─ eb816298641d3d5031974d8d30caa73729ca214be61c7b498e6a483b7cdb85ca
   │  └─ 174ecbb11303920fb2349ae5d9aaaee2138b39faf2ce59f2e55e3cc986e4847c
   │     ├─ b7e69521b75e3759f371f5dfb53e0ae0764c962a6422f60a03986da60f2d950e
   │     │  ├─ b87a1000423e8ecf4b28268d5f2ff48dd0a1430d4332d8baa8463d793b805847
   │     │  └─ 326113a66a7cfff7a8c2a9a3952a7aa42188b8ec24c73e5862d633dbdf4527aa
   │     └─ f49878250a612dc9d46eebeaca03b99aef3817b792d7a1892032e3f35bc89ea5
   │        ├─ e7685b729291be7a2c50174770b57ad486b7e2f966b2de2853195c7d33cb0977
   │        └─ 8ebe11afbf84281d6f97d2279fdd5ba6f14379a7f8b20ab1ad48e82f46cff6bc
   └─ d89b7c42ddf84645c0c7a65387c76cd2f3bb4b01e4102a45f0d5eca3730da511
      ├─ b7cd680a783f1d500aa72d568b0535b643b095ceaae6c06485c3e3d5de0076ae
      │  ├─ c09b3018a506941e5192ac0d880b802388287edddd19c8eee277ba8f3e7e5efa
      │  │  ├─ 9b227f4f2ab1beedf518227963b9f0b10a01c62a0dfcb442f14d1d7d6b551ed9
      │  │  └─ 94b1691ab5b5a0c4419a996c7d33cef3464cdac8680b66a31ebfd7a8e3fcdc1e
      │  └─ df7a4d17e6fe22c7861c4fc45a13c5e08b80b73548fcf052693578680bc88aa5
      │     ├─ 7087173395218f528bedfc009e410dc612e354bfe3e146b28b8a1ea6be3e80a1
      │     └─ f98a6b38e2f6cac4e09b80e2cdbe6e0ab5cda5ae4615c91f0f6885b9f01aab18
      └─ 4d1b199b6ed92e2be2f8c3e3c204451ee14ff1ce5eb737003cfb1ca19a3b6f65
         ├─ 44be6cefd0af69c88098d3defe860353675b6e3c5ed72d7f9a358e066cebebe2
         │  ├─ fb1790a9b3e5236a9a55ee7a698761aee30aa9c3f03586f7eb6594660e0cb62c
         │  └─ 626986009d71c37c6b011cf3a2ce52676f67ce4e67f27ae7b788e4cbc7d0d8f0
         └─ f0b9d87a515aa246cf0c0466d500f2427d2769f999ded237f423ca29bdfb8409
            ├─ 40048c867de6fa57a5158caae8491249d94af9b28238cd8c31d54c25c2dcdcc8
            └─ 9889c9f5a3d7b8580b93adbf879ae502918af965a58ccf863d7f28374c255ec8
```

Figure 3.1.6.b

Background Merkle root generation process for Damage Image:



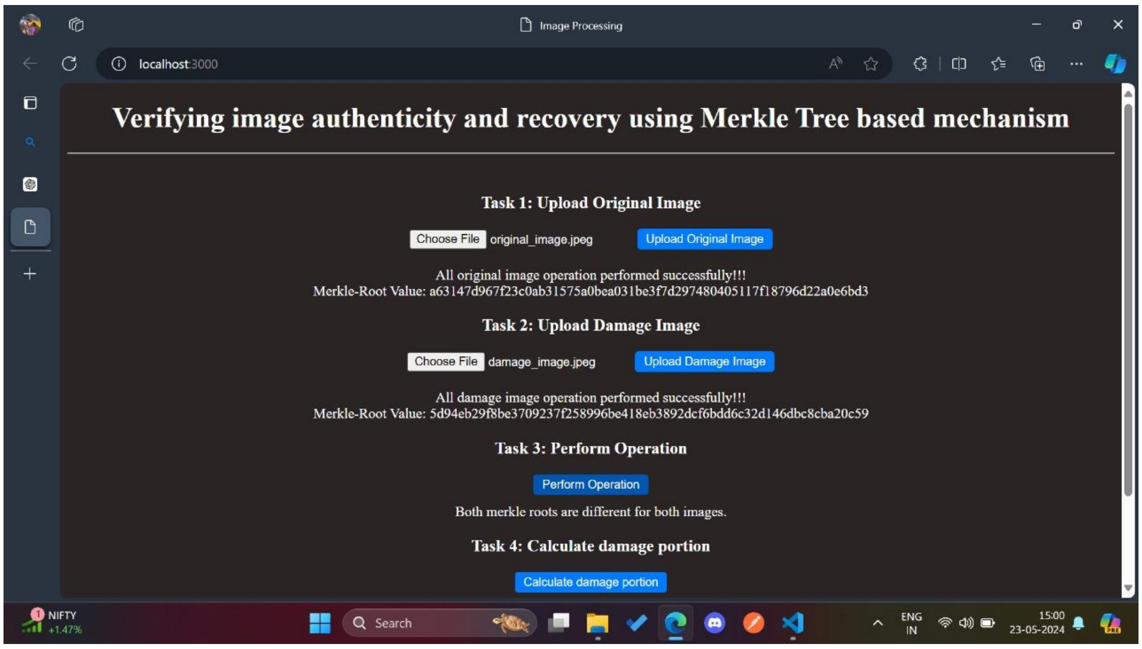
```
Merkle Tree:
└─ 5d94eb29f8be3709237f258996be418eb3892dcf6bdd6c32d146dbc8cba20c59
   ├─ f7d90cf24cf192bfa36e3ffad05312c959036c8e71a39e3d18b32353db039953
   │  ├─ a96ecda16f8ab2cef75648498f93d481ca7bb27763fe12cbe3148cca73085d72
   │  │  ├─ 08b39821289daf6e70ab88e05d7defd67a936302be22f9893fee7986ced0239d
   │  │  │  ├─ 9ef96a7cfc92553c5185c6b3ae03d2b57dfd9b163ead03482fe77a3eba71ab2a
   │  │  │  └─ 5f40493ea5cf0ceab95112476a43febb2b5877dd863fff37597bff861af2d0d0
   │  │  └─ 18f2765761ba8b2763a6d797b992e6425a0f073743ad916c318d11fcb269a2f4
   │  │     ├─ 9ae1d20fa7e5a1ce5c0f1df5c1dbac1dfeb76afb5945846185074b044f9f05b2
   │  │     └─ eb816298641d3d5031974d8d30caa73729ca214be61c7b498e6a483b7cdb85ca
   │  └─ 174ecbb11303920fb2349ae5d9aaaee2138b39faf2ce59f2e55e3cc986e4847c
   │     ├─ b7e69521b75e3759f371f5dfb53e0ae0764c962a6422f60a03986da60f2d950e
   │     │  ├─ b87a1000423e8ecf4b28268d5f2ff48dd0a1430d4332d8baa8463d793b805847
   │     │  └─ 326113a66a7cfff7a8c2a9a3952a7aa42188b8ec24c73e5862d633dbdf4527aa
   │     └─ f49878250a612dc9d46eebeaca03b99aef3817b792d7a1892032e3f35bc89ea5
   │        ├─ e7685b729291be7a2c50174770b57ad486b7e2f966b2de2853195c7d33cb0977
   │        └─ 8ebe11afbf84281d6f97d2279fdd5ba6f14379a7f8b20ab1ad48e82f46cff6bc
   └─ d89b7c42ddf84645c0c7a65387c76cd2f3bb4b01e4102a45f0d5eca3730da511
      ├─ b7cd680a783f1d500aa72d568b0535b643b095ceaae6c06485c3e3d5de0076ae
      │  ├─ c09b3018a506941e5192ac0d880b802388287edddd19c8eee277ba8f3e7e5efa
      │  │  ├─ 9b227f4f2ab1beedf518227963b9f0b10a01c62a0dfcb442f14d1d7d6b551ed9
      │  │  └─ 94b1691ab5b5a0c4419a996c7d33cef3464cdac8680b66a31ebfd7a8e3fcdc1e
      │  └─ df7a4d17e6fe22c7861c4fc45a13c5e08b80b73548fcf052693578680bc88aa5
      │     ├─ 7087173395218f528bedfc009e410dc612e354bfe3e146b28b8a1ea6be3e80a1
      │     └─ f98a6b38e2f6cac4e09b80e2cdbe6e0ab5cda5ae4615c91f0f6885b9f01aab18
      └─ 4d1b199b6ed92e2be2f8c3e3c204451ee14ff1ce5eb737003cfb1ca19a3b6f65
         ├─ 44be6cefd0af69c88098d3defe860353675b6e3c5ed72d7f9a358e066cebebe2
         │  ├─ fb1790a9b3e5236a9a55ee7a698761aee30aa9c3f03586f7eb6594660e0cb62c
         │  └─ 626986009d71c37c6b011cf3a2ce52676f67ce4e67f27ae7b788e4cbc7d0d8f0
         └─ f0b9d87a515aa246cf0c0466d500f2427d2769f999ded237f423ca29bdfb8409
            ├─ 40048c867de6fa57a5158caae8491249d94af9b28238cd8c31d54c25c2dcdcc8
            └─ 9889c9f5a3d7b8580b93adbf879ae502918af965a58ccf863d7f28374c255ec8
```

Figure 3.1.6.c

After generating the Merkle tree from the CIDs of the IPFS-stored original image, secure the Merkle root and by using this Merkle root, we determine whether the image is damaged or not.

3.1.7 Merkle tree comparison (CIDs of original and damaged image)

This part is done by comparison of Merkle tree of original image with Merkle tree of altered or damaged image recursively. Check left child hashes followed by right side child hashes. If the node does not matches point out that leaf node. By doing this we can find the damage encrypted part of the blocks of original image.



Verifying image authenticity and recovery using Merkle Tree based mechanism

Task 1: Upload Original Image

Choose File original_image.jpeg Upload Original Image

All original image operation performed successfully!!!
Merkle-Root Value: a63147d967f23c0ab31575a0bea031be3f7d297480405117f18796d22a0e6bd3

Task 2: Upload Damage Image

Choose File damage_image.jpeg Upload Damage Image

All damage image operation performed successfully!!!
Merkle-Root Value: 5d94eb29f8be3709237f258996be418eb3892dcf6bdd6c32d146dbc8cba20c59

Task 3: Perform Operation

Perform Operation

Both merkle roots are different for both images.

Task 4: Calculate damage portion

Calculate damage portion

Figure 3.1.7.a

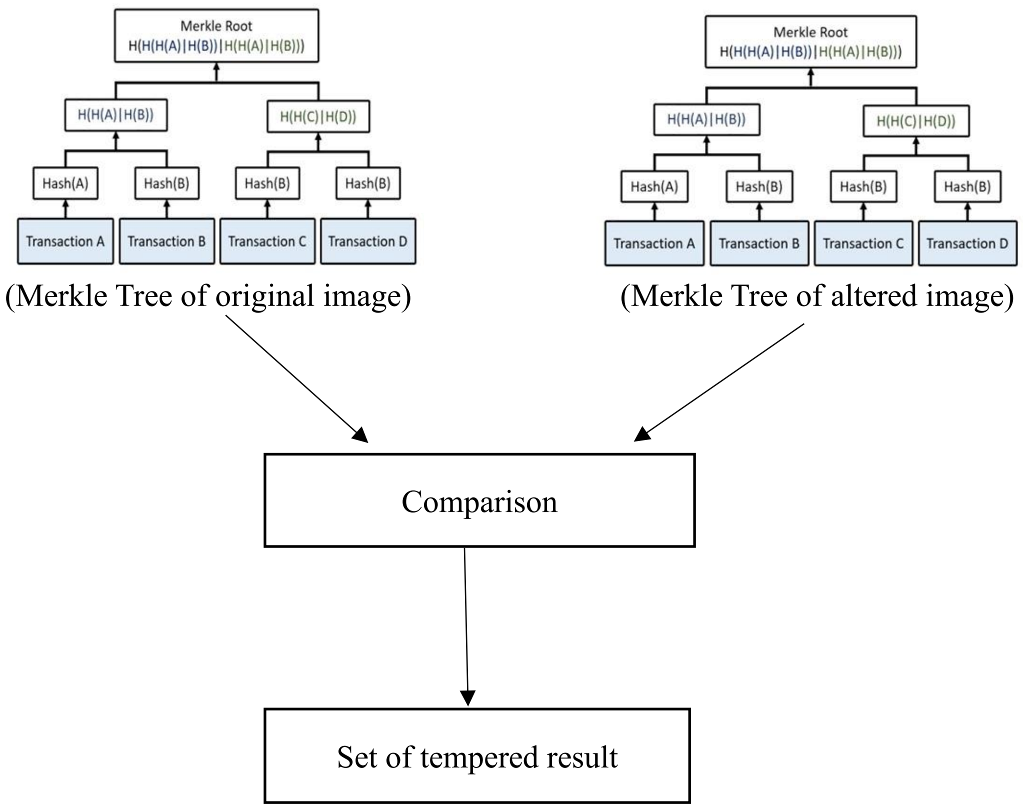
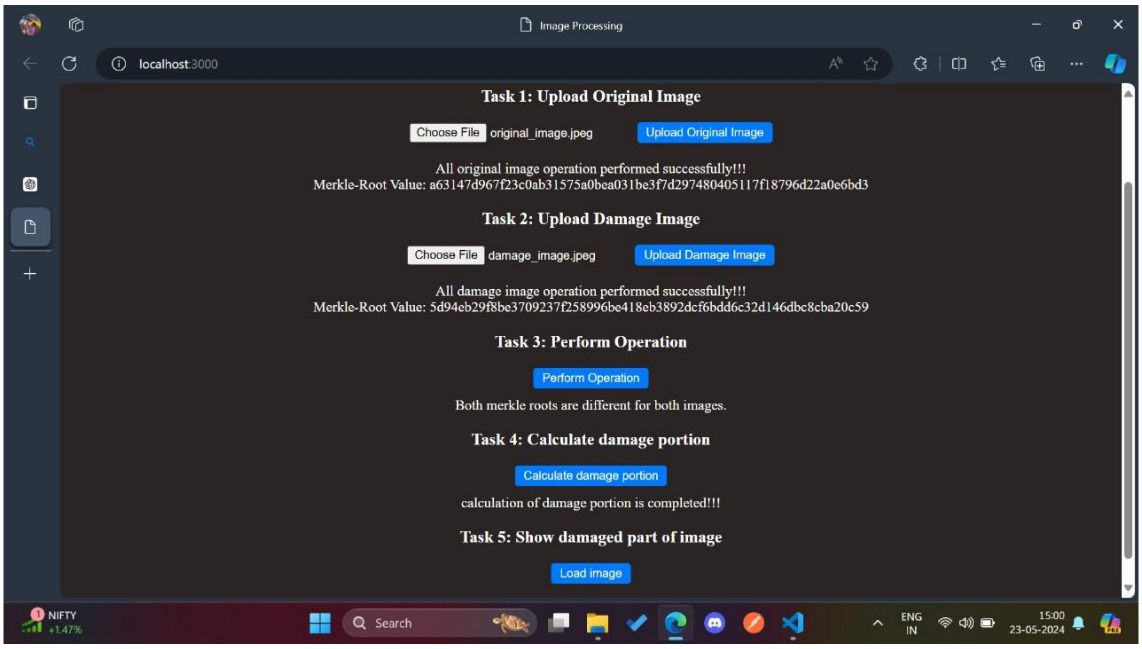


Figure 3.1.7.b

The Merkle tree JS module is used to generate Merkle trees and also to compare two Merkle trees.



Task 1: Upload Original Image

original_image.jpeg

All original image operation performed successfully!!!
Merkle-Root Value: a63147d967f23c0ab31575a0bea031be3f7d297480405117f18796d22a0e6bd3

Task 2: Upload Damage Image

damage_image.jpeg

All damage image operation performed successfully!!!
Merkle-Root Value: 5d94eb29f8be3709237f258996be418eb3892dcf6bdd6c32d146dbc8cba20c59

Task 3: Perform Operation

Both merkle roots are different for both images.

Task 4: Calculate damage portion

calculation of damage portion is completed!!!

Task 5: Show damaged part of image

Figure 3.1.7.c

3.1.8 Downloading block of Image from IPFS

After finding the changed CID that is calculated previously, we have to download that encrypted block from the IPFS.

The Pinata gateway is a public IPFS gateway provided by Pinata that allows you to easily access and download files, such as images, from the IPFS network using a standard web URL. When you upload an image to IPFS via Pinata, it generates a unique CID (Content Identifier) for that image. You can then use the Pinata gateway to retrieve and view the image by constructing a URL with the gateway address and the CID.

3.1.9 Decryption

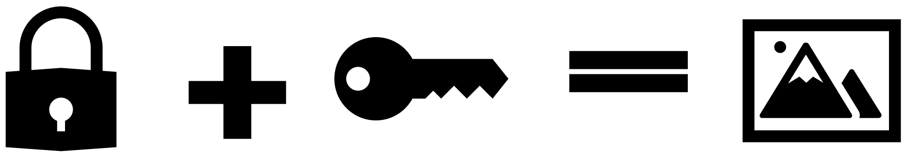


Figure 3.1.9.a

In this step we have to decrypt the downloaded encrypted block of image for this we have to use AES decryption algorithm to decrypt an image previously encrypted using AES encryption, the following steps are typically involved: First, the encrypted image data is loaded into memory. Then, the AES decryption algorithm is applied, which requires the decryption key and the encrypted image data. The decryption process involves reversing the encryption steps, including key expansion, initial round key addition, inverse cipher rounds (consisting of inverse Sub Bytes, inverse Shift Rows, inverse Mix Columns, and round key addition), and a final round. Once the decryption process is complete, the resulting plaintext represents the original image data. This plaintext image data can then be saved or displayed as desired.

3.1.10 Highlight the block of image (Red shade)

"After decryption, we need to convert the block into a red shade so that it can be easily identified. For doing this we use sharp npm package.

The sharp npm package is a high-performance image processing library for Node.js applications. It provides a simple and efficient API for resizing, cropping, rotating, and manipulating images in various formats, such as JPEG, PNG, WebP, TIFF, and GIF. sharp is known for its speed and memory efficiency, making it suitable for processing large volumes of images. Additionally, it supports advanced features like image compression, colour space conversion, and applying filters. Overall, sharp is a versatile tool for image processing tasks in Node.js applications, ranging from simple resizing operations to complex transformations.

3.1.11 Merging of blocks

To regenerate the image, we take all the parts of the image, including the unaltered parts and the altered parts. The altered blocks are marked with a reddish colour, while the unaltered blocks remain their original colour. We then construct a complete image using these blocks. For this, we use the Canvas library, which efficiently loads the image parts, combines them, and displays the full image

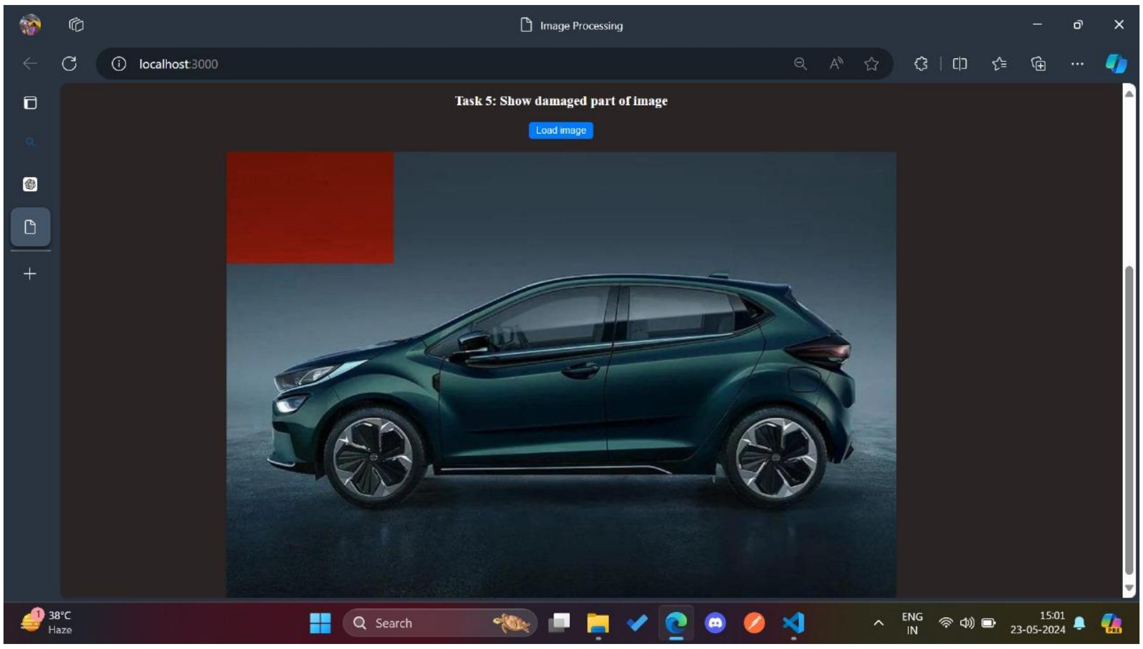
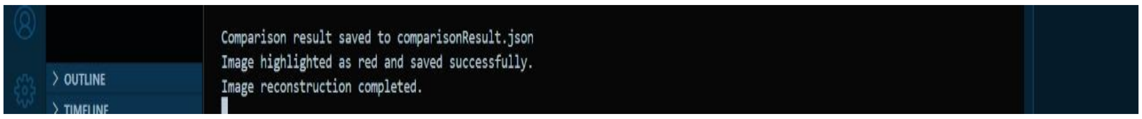


Figure 3.1.11.a

Background Task for regenerate image by image of blocks:



```
Comparison result saved to comparisonResult.json
Image highlighted as red and saved successfully.
Image reconstruction completed.
```

Figure 3.1.11.b

Merkle Root comparison Flow chart

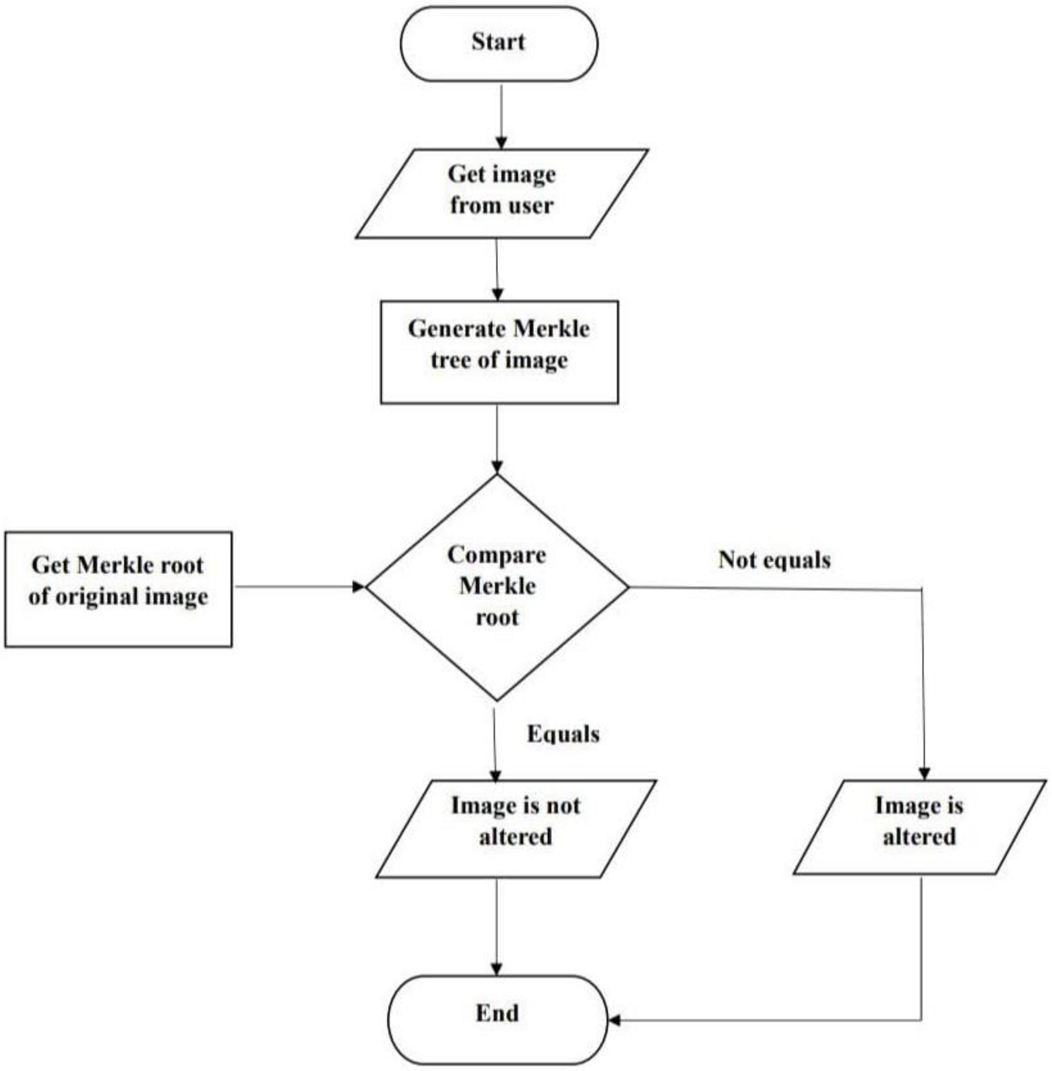


Figure 3.1.11.c

CHAPTER 4

RESULT AND DISCUSSION

When we test the multiple test case, then we found that our project showing the accepted result as we want. In the process of our project, first of all we passed our original image for securing it. Then we again passed the damaged image into our software, then we got the place where our damaged image is not matched with original image, and it showing us as the radish colour. So, by this method we can detect the minor change occurs inside the original image, and this will help to securing our document as well as the criminal record submitted before for securing it, and we used inside our project AES encryption algorithm, while, it is a symmetric key based algorithm, but it is the fast than our encryption algorithm as well as it is the most popular encryption algorithm in current time. In below there are present some test case which we are performing for testing our system.

Sample Test Image-1:

Original Image & Damage image

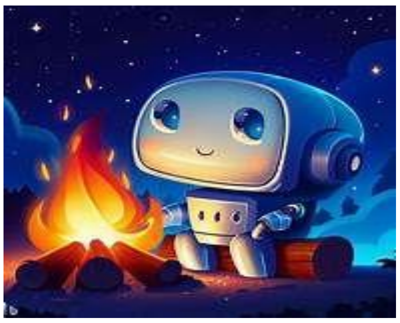


Figure 4.1.a

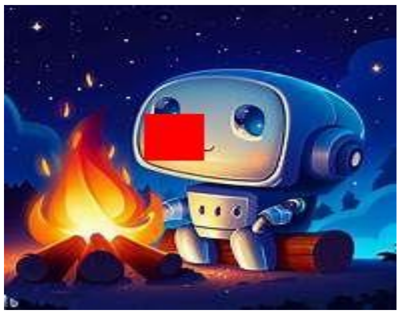


Figure 4.1.b

Output:

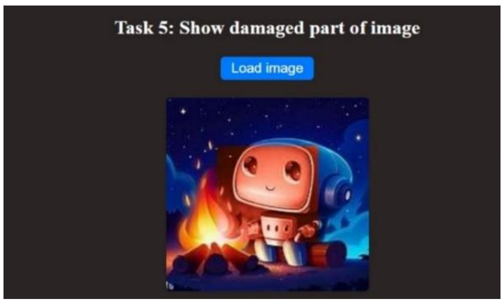


Figure 4.1.c

Sample Test Image-2:

Original Image & Damage image

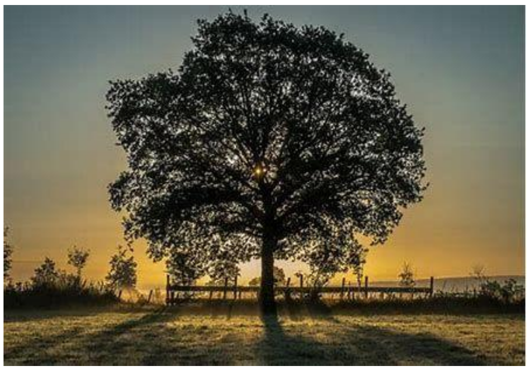


Figure 4.2.a

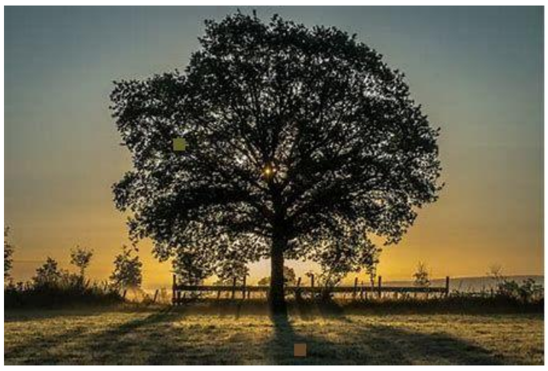


Figure 4.2.b

Output:

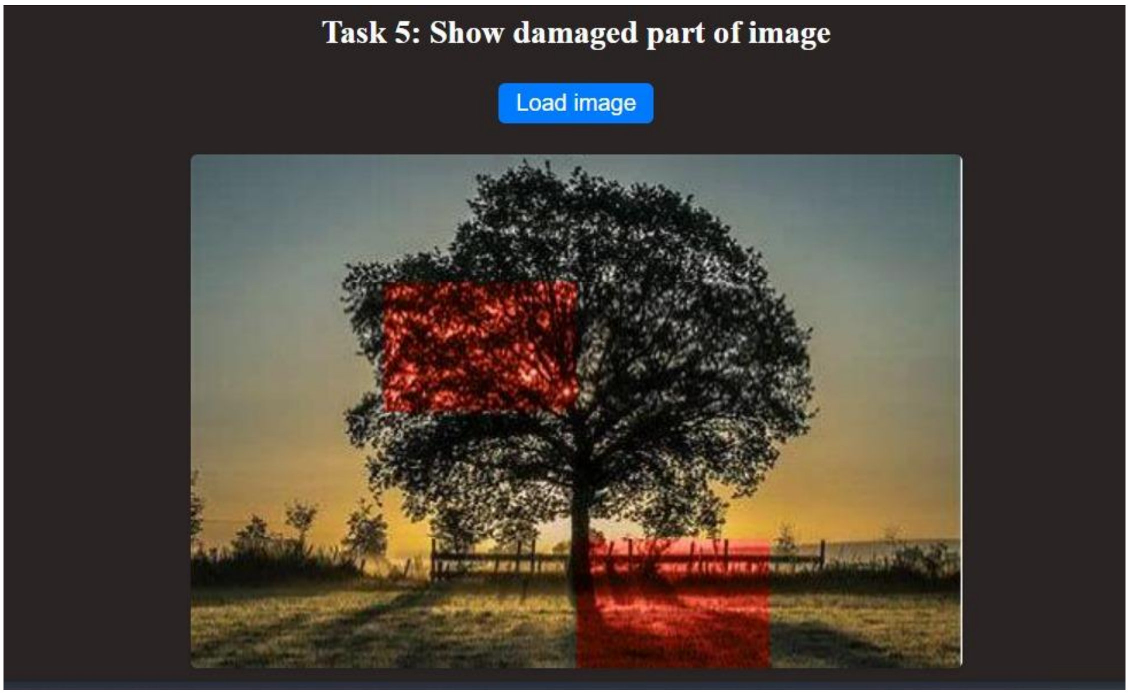


Figure 4.2.c

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

We are performing the multiple test cases - then we found that if we want to more accuracy, then we have need to breaks our image into extra number of block than current number of blocks which we are taking in our project, so we can find the more accurate altered place, but it can take more than extra time as compared to the current number of blocks.

Now for future scope, we can implement our system for the documents, video and many more places similar that aspects. And our current project is work well for the single image, but we can implement it easily for the multiple images. And one of the things, that current project is work well for the single type of image extension. We can do it easily for the multiple image extension, for this implementation we have need to one more function used before the storing images for the conversion of the desired image format.

REFERENCES

[1] J. V. K. Marakumbi Prakash, "A Study on Image Authentication Methods," *International Research Journal of Engineering and Technology (IRJET),* vol. 5, no. 12 Dec 2018, 2018.

[2] A. H. Adil Haouzia, "Methods for image authentication: a survey," *Springer Science + Business Media,* no. 1 Aug 2007, 2007.

[3] A. N. T. I. T. F. C. A. Deepayan Bhowmik, "THE JPEG-BLOCKCHAIN FRAMEWORK FOR GLAM SERVICES," *IEEE International Conference on Multimedia & Expo Workshops (ICMEW),* no. 2018, 2018.

[4] H.-Y. M. L. Chun-Shien Lu, "Multipurpose Watermarking for Image Authentication," *IEEE TRANSACTIONS ON IMAGE PROCESSING,* no. 10 Oct 2001, 2001.

[5] M. S. U. Mahbuba Begum, "Digital Image Watermarking Techniques: A Review," *Information,* no. 17 Feb 2020, 2020.

[6] A. R. G. T. I. C. W. S. M. S. B. G. Y. P. Dennis Trautwein, "Design and Evaluation of IPFS," *SIGCOMM,* no. 22 Aug 2022, 2022.

[7] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," 2014.

[8] T.-S. C. J.-Y. W. Hsiao-Shan Huang, "A Secure File Sharing System Based on IPFS and Blockchain," in *International Electronics Communication Conference*, New York, USA, 2020.

[9] S. R. K. B. Padmavathi, "A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique," *International Journal of Science and Research (IJSR),* no. 4 April 2013, 2013.

[10] B. Ederov, "Merkle Tree Traversal Techniques," Darmstadt University of Technology , 2007.

[11] "Pinata," [Online]. Available: https://www.pinata.cloud/.

APPENDIX

Frontend/index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Processing</title>
  <link rel="stylesheet" href="style.css">
</head>

<body>
    <h1>Verifying image authenticity and recovery using Merkle Tree based
mechanism</h1><hr><br>
  <h3><b>Task 1</b>: Upload Original Image</h3>
  <form id="originalForm" action="/uploadOriginal" method="POST"
enctype="multipart/form-data">
    <input type="file" name="original_image" accept="image/*" required>
    <button type="submit">Upload Original Image</button>
  </form>
  <div id="originalMessage"></div>
  <h3><b>Task 2</b>: Upload Damage Image</h3>
  <form id="damageForm" action="/uploadDamage" method="POST"
enctype="multipart/form-data" disabled>
    <input type="file" name="damage_image" accept="image/*" required>
    <button type="submit">Upload Damage Image</button>
  </form>
  <div id="damageMessage"></div>

  <!-- Add Task 3 Button and Result Display -->
<h3><b>Task 3</b>: Perform Operation</h3>
  <button id="performOperation">Perform Operation</button><br>
  <div id="operationResult"></div>

   <!-- Add Task 4 Button and Result Display -->
 <h3><b>Task 4</b>: Calculate damage portion</h3>
 <button id="CalculateDamagePortion">Calculate damage portion</button><br>  <div
id="damageOperationResult"></div>
<!-- Add Task 5 Button and Result Display -->
 <h3><b>Task 5</b>: Show damaged part of image</h3>
 <button id="imagePerformOperation">Load image</button><br><br>  <img
id="image" src="" alt="Image">
 <script>
   const originalForm = document.getElementById('originalForm');
const damageForm = document.getElementById('damageForm');
   const damageImageInput =
document.querySelector('input[name="damage_image"]');
const originalMessageDiv = document.getElementById('originalMessage');
const damageMessageDiv = document.getElementById('damageMessage');

   // Add Event Listener for Task 3 Button Click
const performOperationButton = document.getElementById('performOperation');
const operationResultDiv = document.getElementById('operationResult');

   // Add Event Listener for Task 4 Button Click
const CalculateDamagePortionButton =
document.getElementById('CalculateDamagePortion');
const damageOperationResultDiv =
document.getElementById('damageOperationResult');

   // Add Event Listener for Task 5 Button Click
const imagePerformOperationButton =
document.getElementById('imagePerformOperation');
const imageElement = document.getElementById('image');

   // Event listener for original image upload form submission
originalForm.addEventListener('submit', function (event) {
event.preventDefault(); // Prevent form submission
const formData = new FormData(originalForm);

 // Submit the form using fetch API
     fetch('/uploadOriginal', { method: 'POST', body: formData })
       .then(response => response.text())
       .then(message => {
         originalMessageDiv.innerText = message;
// Show the damage image upload form after successful original image upload
damageForm.style.display = 'block';
    })
    .catch(error => {       console.error('Error:', error);
   });
});
// Event listener for damage image upload form submission
damageForm.addEventListener('submit', function (event) {
    event.preventDefault(); // Prevent form submission
    const formData = new FormData(damageForm);
     // Submit the form using fetch API
     fetch('/uploadDamage', { method: 'POST', body: formData })
       .then(response => response.text())
       .then(message => {
         damageMessageDiv.innerText = message;
       })
       .catch(error => {
         console.error('Error:', error);
       });
   });
   performOperationButton.addEventListener('click', function () {
     // Send AJAX request to perform the operation
fetch('/performOperation')        .then(response => response.text())
       .then(result => {
         // Display the result in the div tag
         operationResultDiv.innerText = result;
       })
       .catch(error => {
         console.error('Error:', error);
       });
   });

   // task-4
   CalculateDamagePortionButton.addEventListener('click', function () {
     // Send AJAX request to perform the operation
    fetch('/CalculateDamagePortion')
    .then(response => response.text())
       .then(result => {
         // Display the result in the div tag
         damageOperationResultDiv.innerText = result;
    })
    .catch(error => {
      console.error('Error:', error);
   });
});
   // Event listener for Task 5 button click
   imagePerformOperationButton.addEventListener('click', function () {
    // Send a GET request to the server to fetch the image
fetch('/image')        .then(response => {
         // Check if the response is successful
if (!response.ok) {
   throw new Error('Network response was not ok');
         }
         // Return the response as a blob
         return response.blob();
       })
       .then(imageBlob => {
         // Create a URL for the blob object
         const imageUrl = URL.createObjectURL(imageBlob);
         // Set the src attribute of the image element to display the image
imageElement.src = imageUrl;
       })
       .catch(error => {
         console.error('Error fetching image:', error);
       });
   });
 </script>
</body>
</html>
```

Frontend/style.css

```
/* Resetting default button styles */
* {
   background-color: rgb(42, 37, 37);
   color: white;     text-align: center;
}

button {    padding: 4px 10px;    background-color: #007bff;
  color: #fff;
 border: none;   border-radius: 4px; cursor: pointer;
transition: background-color 0.3s ease;
}

button:hover {
   background-color: #0056b3;
}

/* Adjusting form layout */ h1 {     margin-top: 20px;
}

form {     margin-bottom: 20px;
}

/* Adjusting image styling */ img {     max-width: 100%;     height: auto;     display:
block;     margin: auto;     border-radius: 4px;     box-shadow: 0 2px 4px rgba(0, 0, 0,
0.1);
}

div {
   margin-top: 10px;
}
```

server.js

```
const fsAsync =
require('fs').promises;
 const fsSync = require('fs');
 const express = require('express');
 const multer = require('multer');
 const path = require('path');
const imgtoblock = require('./ImageHandling/imgtoblock');
const { encryptionMethod } = require('./EncryptionAndDecryption/encryption') const {
pinFilesToIPFS } = require('./UploadToIPFS/pinFileToIPFS') const {
generateMerkleTree } = require('./GenerateMerkleTree/merkletree') const {
findDamageBlock } = require('./FindDamageBlock/findDamageBlock') const {
decryptionMethod } = require('./EncryptionAndDecryption/decryption') const {
highlightImageRed } = require('./HighlightImageRed/highlightImageRed') const {
blocktoimg } = require('./ImageHandling/blocktoimg')
```

```
const app = express();
const port = 3000;

// Set up multer for handling file uploads
const storage = multer.diskStorage({
          destination: function (req, file, cb) {
          cb(null, path.join(__dirname, 'assets')); // Destination folder for uploaded images
   },
   filename: function (req, file, cb) {

// Rename the uploaded file to 'original_image' or 'damage_image' based on fieldname
          const fieldName = file.fieldname === 'original_image'
          ? 'original_image'
          : 'damage_image';
       cb(null, fieldName + path.extname(file.originalname));
   }
});
const upload = multer({ storage: storage });

// Serve static files from the frontend directory
app.use(express.static(path.join(__
dirname, './Frontend')));

// method for looping highlightimage
async function processImages() {
          try {
          // Read comparisonResult.json file
       const comparisonData = await
       fsAsync.readFile('./tempFolder/comparisonResult.json', 'utf8');
       const comparisonResult = JSON.parse(comparisonData);
       // console.log("hello in processImage")
// Loop through each entry in comparisonResult
for (const [inputPath, shouldHighlight] of Object.entries(comparisonResult)) {
          if (shouldHighlight) {
             // Call highlightImageRed method if boolean value is true
          Await highlightImageRed(
          `./BlocksOfImage/BlockOfOriginalImage/${inputPath}`
          );              //
          console.log(`./BlocksOfImage/BlockOfOriginalImage/${inputPath}`)
          }
       }
       // console.log("hello from processImages")
} catch (err) {
          console.error('Error processing images:', err);
          }
}

// testing function async function testing()
{
```

```
            console.log("hello in testing")
}

// Task 1: Handle original image upload
app.post(
            '/uploadOriginal', upload.single('original_image'),
            (req, res) => {
            ; (async () => {
// Task-1: Convert image into block of images.
            await imgtoblock(
                        './assets/original_image.jpeg',
                        './BlocksOfImage/BlockOfOriginalImage'
            );
// Task-2: Handling encrytion of all block of images.
        await encryptionMethod("./BlocksOfImage/BlockOfOriginalImage/block")
        console.log("Encryptrion on block of images is completed!!!!")
// Task-3: Have to upload all encrypted blocks into IPFS!!!!
        await pinFilesToIPFS(
            './BlocksOfImage/BlockOfOriginalImage',
            './CIDsOfImages/originalImageIpfsHashes.json'
            )
console.log("Upload of all encrypted block of images is completed!!!!")
//Task-4: Generation of Merkle tree and store Merkle root in a json file.....
await generateMerkleTree(
            './CIDsOfImages/originalImageIpfsHashes.json',
            './GenerateMerkleTree/MerkleTreeJSON/OriginalImageMerkleRoot.json',
            './GenerateMerkleTree/MerkleTreeJSON/OriginalImageMerkleTree.json', true
)
//Task-5: Send merkle-root to the user...

// Read the merkleroot.json file
fsSync.readFile(
            './GenerateMerkleTree/MerkleTreeJSON/OriginalImageMerkleRoot.json'
            , 'utf8', (err, data) => {
            if (err) {
                console.error('Error reading file:', err);
                res.status(500).send('Internal Server Error');
                return;
            }

            try {
                // Parse the JSON data
                const jsonData = JSON.parse(data);

    // Extract the root value
        const rootValue = jsonData.root;
            // Message to send along with root value
            const message = "All original image operation performed successfully!!!";
                // Construct the text response
                const responseText = `${message}\nMerkle-Root Value: ${rootValue}`;
                // Send the text response
            res.send(responseText);
 } catch (parseError) {
            console.error('Error parsing JSON:', parseError);
            res.status(500).send('Internal Server Error');
            }
        });

    })();
});

// Task 2: Handle damage image upload
app.post('/uploadDamage', upload.single('damage_image'), (req, res) => {
                    ; (async () => {
// Task-1: Convert image into block of iamges.
                    await imgtoblock('./assets/damage_image.jpeg',
                    './BlocksOfImage/BlockOfDamageImage');

// Task-2: Handling encrytion of all block of images.
            await encryptionMethod("./BlocksOfImage/BlockOfDamageImage/block")
            console.log("Encryptrion on block of images is completed!!!!")

// Task-3: Have to upload all encrypted blocks into IPFS!!!!
await pinFilesToIPFS(
            './BlocksOfImage/BlockOfDamageImage',
            './CIDsOfImages/damageImageIpfsHashes.json')
console.log("Upload of all encrypted block of images is completed!!!!")
//Task-4: Generation of Merkle tree and store Merkle root in a json file.....
await generateMerkleTree('./CIDsOfImages/damageImageIpfsHashes.json',
'./GenerateMerkleTree/MerkleTreeJSON/DamageImageMerkleRoot.json',
'./GenerateMerkleTree/MerkleTreeJSON/DamageImageMerkleTree.json', true)

//Task-5: Send merkle-root to the user...
// Read the merkleroot.json file

fsSync.readFile('./GenerateMerkleTree/MerkleTreeJSON/DamageImageMerkleRoot.jso
n'
, 'utf8', (err, data) => {
if (err) {
console.error('Error reading file:', err);
res.status(500).send('Internal Server Error');
return;
            }
            try {
// Parse the JSON data
const jsonData = JSON.parse(data);

// Extract the root value
const rootValue = jsonData.root;
```

```
// Message to send along with root value
const message = "All damage image operation performed successfully!!!";

// Construct the text response
const responseText = `${message}\nMerkle-Root Value: ${rootValue}`;

// Send the text response
res.send(responseText);            }
catch (parseError) {
console.error('Error parsing JSON:', parseError);
res.status(500).send('Internal Server Error');
          }
       });
   })();
});

// Task 3: Perform Operation
app.get('/performOperation', (req, res) => {

// Function to compare values in two JSON files     function
compareJSONFiles(file1Path, file2Path) {
// Read JSON data from the first file
const data1 = JSON.parse(fsSync.readFileSync(file1Path, 'utf8'));

// Read JSON data from the second file
const data2 = JSON.parse(fsSync.readFileSync(file2Path, 'utf8'));

// Compare values from both files
       if (JSON.stringify(data1) === JSON.stringify(data2)) { res.send('Values in both
JSON files are identical.')
} else {        res.send('Values in both JSON files are different.')
   }
}
// Example usage
const file1Path =
'./GenerateMerkleTree/MerkleTreeJSON/OriginalImageMerkleRoot.json';
const file2Path =
'./GenerateMerkleTree/MerkleTreeJSON/DamageImageMerkleRoot.json';
compareJSONFiles(file1Path, file2Path);
});

// Task 4: Perform Operation on Image
app.get('/CalculateDamagePortion', (req, res) => { ; (async () => {
// Task-4.1 : Comparision of two images CIDs save response...
await findDamageBlock();
// Task-4.2 : Decryption of all block of original images...
await decryptionMethod()
// Task4.3 : Highlighting damage part of image...
await processImages();
```

```
// Task-4.4 : Generate image from blocks of images
await blocktoimg('./BlocksOfImage/BlockOfOriginalImage/',
'./tempFolder/reconstructed_image.jpeg','./assets/original_image.jpeg')
res.send("calculation of damage portion is completed!!!")
   })()
});

// Task 5: Send Modified Image app.get('/image', (req, res) => {
// Replace 'path_to_image.jpg' with the actual path to your image file
   const imagePath = path.join(__dirname, 'tempFolder/reconstructed_image.jpeg');
   // Send the image file as the response
res.sendFile(imagePath);
});

// Start the server app.listen(port, () => {     console.log(`Server is running on
// http://localhost:${port}`); });
// const wait = (ms) => new Promise( resolve => setTimeout(resolve,ms));
// // Call the function with imagePath and outputDir arguments
```

ImageHandling/imgtoblock.js

```
const fs = require('fs');
const { createCanvas, loadImage } = require('canvas');

async function imgtoblock(inputImagePath , outputDir) {
try {
// Replace "inputImagePath" with the path to your input image
// const inputImagePath = './img1.jpeg';

// Load the image
const originalImage = await loadImage(inputImagePath);

// Split the image into 16 blocks
const rows = 4;
const cols = 4;
const blockWidth = originalImage.width / cols;
const blockHeight = originalImage.height / rows;

      // Create the output directory if it doesn't exist
      // const outputDir = './blocks';
if (!fs.existsSync(outputDir)) {
         fs.mkdirSync(outputDir);
      }

// Save each block as a separate image
      for (let i = 0; i < rows; i++) {
         for (let j = 0; j < cols; j++) {
const canvas = createCanvas(blockWidth, blockHeight);
const ctx = canvas.getContext('2d');
const x = j * blockWidth;
const y = i * blockHeight;
          ctx.drawImage(originalImage, -x, -y);

// Replace "outputPath" with the directory where you want to save the subimages

const outputPath = `${outputDir}/block${i}_${j}.jpeg`;

          // Save the subimage
const out = fs.createWriteStream(outputPath);
const stream = canvas.createJPEGStream();
stream.pipe(out);
          await new Promise((resolve, reject) => {
out.on('finish', resolve);
          out.on('error', reject);
        });
      }
   }
     console.log('Image splitting completed.');
   } catch (error) {
     console.error(error);
   }
}

module.exports = imgtoblock;
```

ImageHandling/blocktoimg.js

```
const fs = require('fs');
const { createCanvas, loadImage } = require('canvas');

async function blocktoimg(inputDirectory,outputImagePath,inputImagePath) {
try {
     // Load the image
     const originalImage = await loadImage(inputImagePath);

     // Set dimensions of the original image
const originalWidth = originalImage.width;
     const originalHeight = originalImage.height;

     // row and column which are used to combine the picture
const rows = 4;
     const cols = 4;

     // Create a canvas for the reconstructed image
const canvas = createCanvas(originalWidth, originalHeight);
const ctx = canvas.getContext('2d');

     // Disable image smoothing
     ctx.imageSmoothingEnabled = false;
```

```
        // Reconstruct the image by reading each block
        for (let i = 0; i < rows; i++) {
for (let j = 0; j < cols; j++) {
                // Construct the file path for each block
                const blockPath = inputDirectory +'block'+ i + '_' + j + '.jpeg';

                // Read the block image
                const blockImage = await loadImage(blockPath);

                // Calculate the position to place the block in the reconstructed image
const x = Math.floor(j * (originalWidth / cols));
const y = Math.floor(i * (originalHeight / rows));

                // Calculate the width and height of the block
const blockWidth = Math.ceil(originalWidth / cols);
const blockHeight = Math.ceil(originalHeight / rows);

                // Draw the block onto the canvas
                ctx.drawImage(blockImage, x, y, blockWidth, blockHeight);
            }
        }

// Replace "outputImagePath" with the desired path for the reconstructed image
// const outputImagePath = './reconstructed_image.jpeg';

// Save the reconstructed image
const out = fs.createWriteStream(outputImagePath);
const stream = canvas.createJPEGStream();
        stream.pipe(out);
        out.on('finish', () => console.log('Image reconstruction completed.'));
    } catch (error) {
        console.error(error);
    }
}

module.exports = {blocktoimg}
```

EncryptionAndDecryption/encryption.js

```
const crypto = require('crypto');
const fs = require('fs');

/* 1st run this for encryption run It, commect decrption code */
function generateAESKey() {
    return crypto.randomBytes(32); // Generating a 256-bit key (32 bytes) }

function encryptImage(key, iv, imageBytes) {
const cipher = crypto.createCipheriv('aes-256-cbc', key, iv); let encrypted =
cipher.update(imageBytes); encrypted = Buffer.concat([encrypted, cipher.final()]);
return encrypted;
}

const encryptionMethod = async (outputDirectory) => {
try {
      const keyAndIVData =
JSON.parse(fs.readFileSync('./EncryptionAndDecryption/key_and_iv.json'));
const aesKey = Buffer.from(keyAndIVData.key, 'hex');
const iv = Buffer.from(keyAndIVData.iv, 'hex');

// Set the number of rows and columns used for splitting
const rows = 4;
      const cols = 4;

      for (let i = 0; i < rows; i++) {           for (let j = 0; j < cols; j++) {
// Construct the file path for each block
const imagePath = `${outputDirectory}${i}_${j}.jpeg`;

// Read the block image
const imageBytes = fs.readFileSync(imagePath);

// Encrypt the image
const encryptedImageBytes = encryptImage(aesKey, iv, imageBytes);

// Save the encrypted image to a new file
// const encryptedImagePath = './encrypted_image.enc';
const encryptedImagePath = `${outputDirectory}${i}_${j}.jpeg`;
fs.writeFileSync(encryptedImagePath, encryptedImageBytes);
         }
      }
   } catch (error) {
      console.error(error);
   }
}

module.exports = {encryptionMethod}
```

EncyptionAndDecryption/decryption.js

```
const crypto = require('crypto');
const fs = require('fs');

function decryptImage(key, iv, encryptedImageBytes) {
const decipher = crypto.createDecipheriv('aes-256-cbc', key, iv);
let decrypted = decipher.update(encryptedImageBytes);
decrypted = Buffer.concat([decrypted, decipher.final()]);
return decrypted;
}

const decryptionMethod = async () => {
```

```
try {
// Read the AES key and IV from the file
const keyAndIVData =
JSON.parse(fs.readFileSync('./EncryptionAndDecryption/key_and_iv.json'));
const aesKey = Buffer.from(keyAndIVData.key, 'hex');
const iv = Buffer.from(keyAndIVData.iv, 'hex');
const outputDirectory = "./BlocksOfImage/BlockOfOriginalImage/block";

      // Set the number of rows and columns used for splitting
const rows = 4;
const cols = 4;

for (let i = 0; i < rows; i++) {
     for (let j = 0; j < cols; j++) {
             // Read the encrypted image file into a byte array
// const encryptedImagePath = './encrypted_image.enc';
const encryptedImagePath = `${outputDirectory}${i}_${j}.jpeg`;
const encryptedImageBytes = fs.readFileSync(encryptedImagePath);

 // Decrypt the encrypted image using the same key
    const decryptedImageBytes = decryptImage(aesKey, iv, encryptedImageBytes);

     // Save the decrypted image to a new file
             // const decryptedImagePath = './decrypted_image.jpeg';
const decryptedImagePath = `${outputDirectory}${i}_${j}.jpeg`;
            fs.writeFileSync(decryptedImagePath, decryptedImageBytes);
        }
   }
}
catch (error) {
console.error(error); }
 }

module.exports = {decryptionMethod}
```

UploadToIPFS/pinFileToIPFS.js

```
require('dotenv').config(); const axios = require('axios');
const FormData = require('form-data');
const fs = require('fs');

const pinFilesToIPFS = async (folderPath,originalImageIpfsHashes) => {
try {
   // const folderPath = './assets'; // Path to the folder containing the images
const files = fs.readdirSync(folderPath);
// Read all files in the folder
const ipfsHashes = {};
// Object to store IpfsHashes and their corresponding file names

   for (const file of files) {
```

```
if (file.endsWith('.png') || file.endsWith('.jpg') || file.endsWith('.jpeg')) {
// Check if the file is an image (you can add more extensions if needed)
const filePath = `${folderPath}/${file}`;
const data = new FormData();
data.append('file', fs.createReadStream(filePath));
data.append('pinataOptions', '{"cidVersion": 0}');
      data.append('pinataMetadata', `{"name": "${file.split('.')[0]}"}`);
const response = await axios.post('https://api.pinata.cloud/pinning/pinFileToIPFS', data,
{         headers: {
          'Authorization': `Bearer ${process.env.PINATA_JWT}`
        }
  });

      const ipfsHash = response.data.IpfsHash;
      ipfsHashes[file] = ipfsHash; // Store IpfsHash and its corresponding file name
console.log(`Image ${file} uploaded successfully to IPFS.`);
console.log(`View the file here: https://gateway.pinata.cloud/ipfs/${ipfsHash}`);
    }
  }
  // Write ipfsHashes object to JSON file // 'originalImageIpfsHashes.json'
fs.writeFileSync(`${originalImageIpfsHashes}`,
JSON.stringify(ipfsHashes, null, 2));
console.log('IpfsHashes and corresponding file names stored in ipfsHashes.json');
} catch (error) {
console.error('Error uploading images to IPFS:', error);  }
};

module.exports = { pinFilesToIPFS }
```

GenerateMerkleTree/merkletree.js

```
const fs = require('fs');
const { MerkleTree } = require('merkletreejs');
const SHA256 = require('crypto-js/sha256');
const {printMerkleTree} = require('./printmerkletree')

const generateMerkleTree = async (OriginalImageCIDPath, merkleRootFilePath,
merkleTreeFilePath,flag) => {

   // Function to generate Merkle tree from CIDs
function generateMerkleTree(cids) {
      // Convert CIDs to leaf nodes by hashing them
      const leaves = cids.map(cid => SHA256(cid));

      // Create Merkle tree
      const tree = new MerkleTree(leaves, SHA256);

      // Get Merkle root
      const root = tree.getRoot().toString('hex');
```

```
        // Convert Merkle tree to string
        const treeString = tree.toString();

        return { root, treeString };
    }

    // Read the JSON file containing CIDs
const jsonData = fs.readFileSync(OriginalImageCIDPath);
   const cids = JSON.parse(jsonData);
// Extract CIDs from JSON object
const cidsArray = Object.values(cids);
// Generate Merkle tree
const { root, treeString } = generateMerkleTree(cidsArray);
   // Save Merkle root to JSON file
fs.writeFileSync(merkleRootFilePath, JSON.stringify({ root }, null, 2));
console.log('Merkle root saved to', merkleRootFilePath);

    // Save Merkle tree as string to JSON file
    fs.writeFileSync(merkleTreeFilePath, JSON.stringify({ treeString }, null, 2));
console.log('Merkle tree saved to', merkleTreeFilePath);

    if(flag==true){
        printMerkleTree(merkleTreeFilePath)
    }
}

module.exports = { generateMerkleTree }
```

GenerateMerkleTree/printmerkletree.js

```
const fs = require('fs');
// Read the JSON file containing the Merkle tree
const printMerkleTree = async (merkleTreePath) => {     const merkleTreeData =
fs.readFileSync(merkleTreePath);     const { treeString } =
JSON.parse(merkleTreeData);

    // Print the Merkle tree
console.log('Merkle Tree:');
console.log(treeString);
}
module.exports = { printMerkleTree }
```

FindDamageBlock/findCamageBlock.js

```
 const fs = require('fs');
const path = require('path');
const findDamageBlock = () => {

   // Get the directory path of the current script
  const currentDirectory = __dirname;
  // Define the path to the directory containing the JSON files     const directoryPath =
path.join(currentDirectory, '../CIDsOfImages');     const directoryPath2 =
path.join(currentDirectory, '../tempFolder');

   // Read contents of originalImageIpfsHashes.json and damageImageIpfsHashes.json
const originalDataPath = path.join(directoryPath, 'originalImageIpfsHashes.json');
const damageDataPath = path.join(directoryPath, 'damageImageIpfsHashes.json');

   const originalData = JSON.parse(fs.readFileSync(originalDataPath, 'utf8'));     const
damageData = JSON.parse(fs.readFileSync(damageDataPath, 'utf8'));

   // // Function to compare two JSON objects
   function compareJSONObjects(obj1, obj2) {
for (let key in obj1) {
if (obj1[key] !== obj2[key]) {
            return false;
         }       }
      return true;
   }

   // // Iterate over keys in originalData and compare with damageData
const comparisonResult = {};     for (let key in originalData) {
const isMatch = compareJSONObjects(originalData[key], damageData[key]);
comparisonResult[key] = !isMatch;
   }

   // // Write comparison result to a new JSON file
   fs.writeFileSync(`${directoryPath2}/comparisonResult.json`,
JSON.stringify(comparisonResult, null, 2));
   console.log('Comparison result saved to comparisonResult.json');

}

module.exports = {findDamageBlock}
```