Proceedings of 14th IOE Graduate Conference
Peer Reviewed
Year: 2023 Month: December Volume: 14
ISSN: 2350-8914 (Online), 2350-8906 (Print)

Decentralized File Storage System for Web3 with IPFS

Sandesh Pokhrel [a], Nischal Bhattarai [b], Rupesh Shrestha [c], Kumar Tiwari [d], Binod Sapkota [e]

[a, b, c, d, e] *Department of Electronics and Computer Engineering, Thapathali Campus, IOE, Tribhuvan University, Nepal*

[a] sandesh.pokhrel33@gmail.com, [b] bhattarai.nischal010@gmail.com, [c] shrestharupesh02@gmail.com, [d] linux.user6969@gmail.com, [e] replybinod@gmail.com

Abstract

The internet is at the core of everything today. People use the internet in every aspect, exchanging crucial data for various services online. These services are free to use, but only after complying with data policies. Companies like Facebook, Google and Tiktok make billions from user data, and most do not realize how much data these services collect or what they use it for. Using Web3 [1] technologies such as blockchain and IPFS (InterPlanetary File System), a decentralized storage mechanism is created that is controlled exclusively by the user. The use of blockchain ensures that data is stored securely, with no single entity having control over it, using various mathematical functions to cryptographically secure and verify data among many users. IPFS provides a decentralized file storage system that facilitates the sharing of data without the need for a centralized server. With the combination of blockchain and IPFS, a user-controlled decentralized storage mechanism is created. The system provides the user with complete control over their data, ensuring that it is only accessible to those who have permission. The user has the option to share their data with others or keep it private; the choice is entirely up to the user. The system splits a file into smaller parts and generates unique hashes. These hashes are stored on the blockchain for later retrieval, ensuring file integrity. AES is used for encryption, with a symmetric key for private uploads and a public key for public uploads. Passing the content ID (CID) and encryption key to the recipient ensures that only the intended recipient can access the file. Performance evaluation was based on system responsiveness, data integrity, and confidentiality compared to centralized storage systems.

Keywords

Blockchain, Data policies, Decentralized storage, IPFS, Web3

1. Introduction

In recent decades, the control of personal data has been monopolized by major companies. These firms collect an immense amount of data through their cloud services, and customers are left in the dark about how their data is being utilized. Blockchain [2], instead of granting authority to a single person or organization, employs multiple mathematical functions to cryptographically encrypt and verify our data across numerous users. It enables a technique to not rely on a single centralized server but to utilize mathematics and a number of computers storing a copy of a ledger to verify transactions among them. Among several existing blockchains, Ethereum is a flexible, programmable blockchain that permits the development of advanced smart contract systems, enabling decentralized applications that go beyond simple financial transactions.

IPFS[1] is a decentralized file storage system that is well-suited for use in the Web 3.0 context. IPFS enables the storage and retrieval of massive files and data on a peer-to-peer network without the need for a centralized server. IPFS maintains data across numerous nodes in the network, making it resistant to censorship, data loss, and single points of failure. Files are identifiable by their content rather than by their location, which means that the same file is stored on numerous nodes and accessed from anywhere in the network.

The implemented system supports client-side file uploads that are initiated by users, who can choose between public and private modes. Files are transferred to IPFS via an IPFS node client in the event of public uploads, with vital file characteristics like name, size, and the resulting CID[2] securely recorded in a public upload contract placed on the testnet. User-uploaded files are encrypted in private mode using a randomly generated key, which is then encrypted with the user's public key and safely stored in the private upload contract. Sharing files is facilitated by sharing the CID directly; however, for private files, the owner uses key decryption and re-encryption for easy file access by shared users.

Our contributions are as follows:

- We have integrated Web 3 technologies to create a distributed file storage system where the user has full control over the data.
- We have analyzed encryption and decryption algorithms to ensure efficient file sharing and storage that is robust to data changes and loss.

2. Background and Related Works

2.1 Blockchain

Bitcoin, introduced by Satoshi Nakamoto [3], made its way into the mainstream as a peer-to-peer electronic transaction system. Bitcoin provided a way to not rely on anyone specific but on the nodes as a whole. It gave an explanation of how to initiate a transaction, how to verify it, how to add it to a block,

[1] InterPlanetary File System

[2] Content Identifier

etc. With the popularity of bitcoin came its limitations. The Bitcoin network was hard to program for new systems to be built upon it.

As a solution, the Ethereum [4] blockchain was created, which introduced the concept of programming the blockchain. EVM[3], which runs smart contracts (programs of EVM) after using a compiler named Solidity. With the use of smart contracts, many new crypto tokens were created and transferred among various accounts existing on the blockchain. With this, decentralized applications are created on top of the blockchain embedded with immutable contracts, storing things that cannot be changed by anyone or creating contracts that are carried out automatically when certain events are triggered.

2.2 Decentralized Storage

The concept of decentralized storage of data started with BitTorrent [5]. With BitTorrent, accessing a single file from different servers, which don't have any central point of control, is possible. For that, a torrent file, which consists of data about where the file is located, is needed. The file is broken down into pieces, and each node keeps a certain piece of it. And when someone wants to access the file, a torrent file is needed to know where that file is located. It surely paved the way for decentralized file storage. But still, the problem of that data being modified exists, and the privacy of that data is not maintained.

After nearly a decade of BitTorrent, IPFS [6] made its way into decentralized storage by solving the problem of data modification through the use of content-based addressing. IPFS is a new and innovative decentralized file storage system that has been designed to work on the Web 3.0 architecture. It is designed to provide a fast, efficient, and secure way of storing and sharing data online. The IPFS network allows users to store and access data in a peer-to-peer manner rather than relying on centralized servers. This means that data is distributed across a large number of nodes, making it much more resilient and secure than traditional centralized systems.

2.3 Advanced Encryption Standard

The AES[4] [7], introduced in 2001 and specified by the NIST[5], operates on a fixed block size of 128 bits and supports key sizes of 128, 192, and 256 bits. The algorithm works by dividing the plain text into blocks and then applying a series of transformations to each block using a secret key. The encrypted blocks are then concatenated to form the cipher text.

2.4 Elliptic Curve Cryptography

ECC[6], which uses the mathematics of elliptic curves to generate secure key pairs for public key encryption, has become popular due to its smaller key size and ability to maintain security, which are based on the algebraic structure of elliptic curves over finite fields, as shown by [8, 9]. The security of ECC is based on the hardness of solving the ECDLP[7], which is considered computationally infeasible for large values of the key size.

2.5 Prior Works on Decentralized Storage with Blockchain

V. Sarasvathi et al. [10] demonstrated a system in which the medical reports are saved on IPFS and the hashes of the files are saved on the blockchain to store patient medical reports for privacy issues. It reduced the usage of blockchain, which reduces costs, and since blockchain is immutable, the patient file can never be lost.

Shuije Cui et al. [11] have proposed to create a storage system that takes care of securing the files by using encryption that treats the storage system as malicious, and only the owner and shared user can access the data. The concept of a master key and secondary key was used; the master key remains with a proxy server, and the secondary key is shared with the user. This way, even if the proxy server wants to decrypt the file, it cannot. But files were stored on a centralized server, which only took care of privacy.

3. Methodology

Due to the remarkable advances in blockchain, a shared drive based on it offers secure alternatives to centralized servers. In contrast to the work of V. Sarasvathi et al. [10], our system provides a user-controlled, shareable drive. Shuije Cui et al. [11] were able to demonstrate a similar system only on paper without analyzing system performance, and this was based on centralized servers. Their work inspired the creation of a robust storage system that is truly decentralized and shareable. Our system demonstrates the integration of access control mechanisms where files are managed in the file hierarchy.

Using the underlying peer-to-peer file storage protocol embedded in the blockchain, a decentralized storage system that provides users with full control over their data was created. Users can publish files publicly or privately and share them among themselves. All file-related operations happen on the client side due to privacy issues. The CID of the file returned from IPFS was stored in a blockchain smart contract. A smart contract records each user's uploaded CID and file name in its state variable. And it makes an array of files for each user. It utilizes mapping types and structures to make it possible. Also, another smart contract gives the user the ability to store encrypted files in the IPFS and store the key and IV[8].

3.1 System Block diagram

After a file is uploaded, all operations are performed by the interface block, which is client-side, prior to storing it in IPFS. Only the minimum details for file pointers remain in the blockchain. On the blockchain, interaction is accomplished through two methods: making a transaction call or a view call. A transaction call is used to execute a function that changes the state of the blockchain, whereas a view call returns the

[3] Ethereum Virtual Machine
[4] Advanced Encryption Standard
[5] National Institute of Standards and Technology
[6] Elliptic Curve Cryptography
[7] Elliptic Curve Discrete Logarithm Problem
[8] Initialization Vector

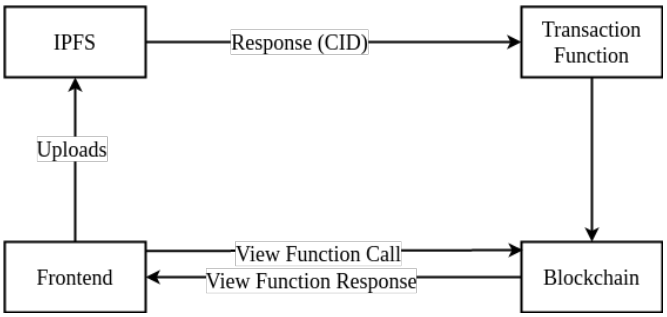

Figure 1: System Block Diagram

current state of the blockchain without modifying it. When making a transaction call, it is essential to note that it requires spending gas fees, which are paid in ether. The user can query data from the blockchain first, then retrieve the actual file from IPFS. The transaction function block is the actual call to the blockchain for storing the requisite details in a blockchain smart contract. View call observes the state of the blockchain to retrieve information from the blockchain, which doesn't require a gas fee.

3.2 File Encryption/Decryption

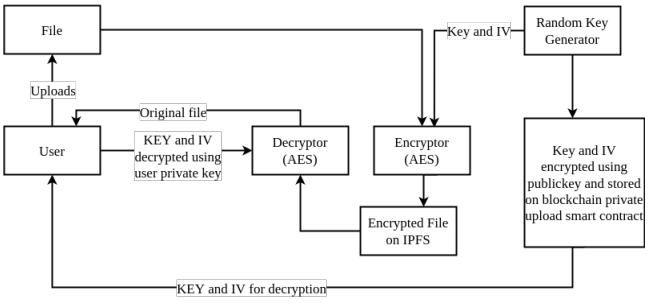

Figure 2: File Encryption/Decryption

A random key and IV are produced when a user uploads a file in order to encrypt it on the client side. The addition of an IV is crucial for the AES algorithm's implementation. Prior to encryption, a block of binary data known as the IV is added to the plaintext, increasing the security of the final encrypted communication. IV was used, which had a 12-byte size and was produced using a special function called getRandombytes.

The encryption procedure is initiated by generating an encryption key and the IV using a random number generator function. Subsequently, the target file was encrypted using the AES-GCM[9] algorithm, a capability efficiently provided by the Webcrypto API[10] incorporated within web browsers. The resulting encrypted file was then securely stored on IPFS, while the encryption key and IV were safeguarded within a smart contract. This was accomplished by encrypting these sensitive components using the ECC [12, 13] which derives the public key associated with the Ethereum private key.

Upon the user's request to download the encrypted file, the matching encryption key and IV were received from the smart contract, connected with the unique CID. Subsequently, these recovered components underwent decryption using the user's private key. The file of interest was then fetched from IPFS using the corresponding CID. The final stage included the decryption of the file using the IV and key using the Webcrypto API. The successfully decrypted file was then made accessible for download and restored to its original form within the user's file system. This technique, incorporating AES encryption, IPFS storage, and secure key management, protects the confidentiality and integrity of data during both storage and retrieval activities, thereby matching strong security procedures in current data handling situations.

[9] AES-Galois/Counter Mode
[10] Application Programming Interface

3.3 File Sharing among users

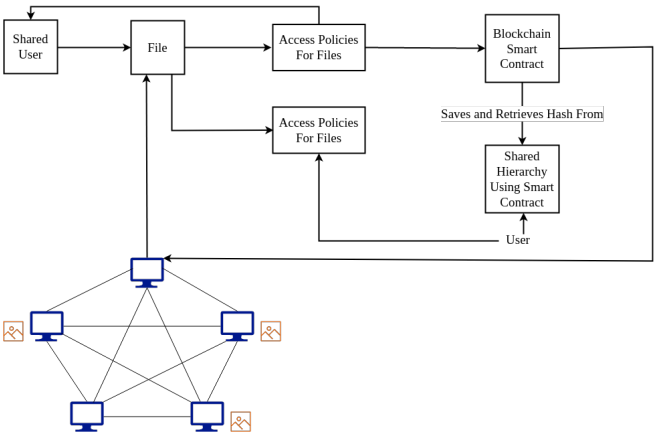

Figure 3: File Sharing among users

When a user intends to share a file, it's essential to determine whether the file is accessible privately or publicly. If it is a privately uploaded file, the process involves providing the CID along with the encryption key that was utilized to secure the file. To ensure the security of this key during transmission, it is encrypted using the recipient's unique public key. This cryptographic approach guarantees that only the intended recipient, holding the corresponding private key, can decrypt and obtain access to the key, thereby unlocking the file. Subsequently, the recipient can utilize their private key to decrypt the key and, in turn, access the contents of the file.

In contrast, for publicly uploaded files, only the CID associated with the file is shared. Consequently, anyone in possession of this CID has the ability to access the file without the need for additional encryption keys or authentication measures.

3.4 File Hierarchy

In a decentralized storage system, a file hierarchy system was established through a backend server. The files themselves are stored within IPFS, and their unique identifiers (CIDs) are retrieved from IPFS and employed by the backend to construct the file hierarchy. A database was implemented to monitor folder-CID associations, facilitating the creation of a multi-level file hierarchy.

Users initiate the process by uploading a file to IPFS, which returns a corresponding CID. This CID is subsequently linked to a folder within the backend. Users can also generate new folders and associate them with CIDs, expanding the

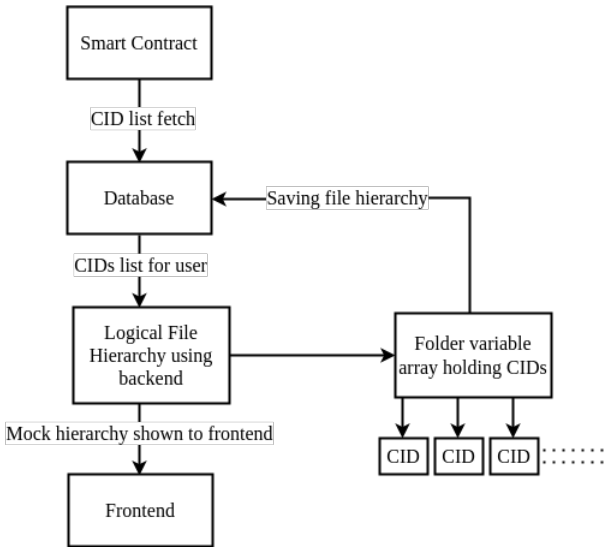

Figure 4: File Hierarchy

hierarchical structure. Logical hierarchy refers to the concept of constructing a file hierarchy without actually preserving the file. Arrays and CIDs were used to create folders that contain the files. When users desire to access a file within this system, the backend queries the database to retrieve the associated CID and uses it to access the file from IPFS.

A decentralized storage system employs a backend interface to construct and administer a file hierarchy stored in IPFS. CIDs from IPFS are pivotal in associating files with folders, thus facilitating a multi-level file hierarchy. Additionally, a database maintains folder-CID relationships, and user interaction with the file hierarchy is facilitated via a web-based frontend that communicates with the backend.

4. Experiments and Results

This section describes the detailed procedures for smart contract testing and deployment, key generation, encryption, decryption, uploading, and sharing of files using IPFS.

4.1 Smart contract deployment and testing

Two smart contracts were developed on the Ethereum testnet, and unit testing was done in the following ways:

a) PublicUpload Contract Testing

i. Adding CID to Contract

The transaction was tested to add the CID returned from IPFS to the contract. The test checked if the length of the userCID array changed and verified that the CID mapped to the filename.

ii. Removing CID from Contract

Testing involved the removal of CID from the contract. To optimize gas usage, only the CID and name were removed from the index, leaving a blank array entry. The test confirmed that the array at the deleted index contained empty string values, as expected.

iii. Sharing File with Other Users

In this test, two accounts were utilized: one as the owner and the other as the recipient. Both accounts were provided by brownie.accounts. The sharewithother function was called from accounts[0], passing accounts[1] as an argument. Afterward, it was asserted that the CID was successfully added to sharedwithuser, and the test passed.

b) PrivateUpload Contract Testing

i. Adding Public Key of User

For this test, the public key was extracted from MetaMask using the getEncryptionPublicKey function of the MetaMask API. Then, the addpublickey function was called with the public key as an argument to verify whether the public key had been successfully added for the user.

ii. Adding CID Along with Key and IV

For privately uploaded files, it was necessary to store the key and IV along with the CID. Initially, the file details were added, and then it was asserted that all details for the file were correctly transacted.

iii. Sharing File with Others

As this is a unit test, the file and key were added for sharing without decrypting and re-encrypting, which requires access to the MetaMask API provided by the browser window. After calling the sharewithother function, confirmation was sought that file details had been added to getsharedwithuser.

4.2 Key generation

a. Connecting MetaMask Wallet

Upon connection, the site gains access to the MetaMask account number and establishes a blockchain connection via the MetaMask interface.

b. File List

A table displays user-uploaded files, retrieved directly from the blockchain using the user's account value. The smart contract function for that is GetUserCID(user account address). It provides an array of files uploaded by the user. In the public upload contract, this function returns a file array with each

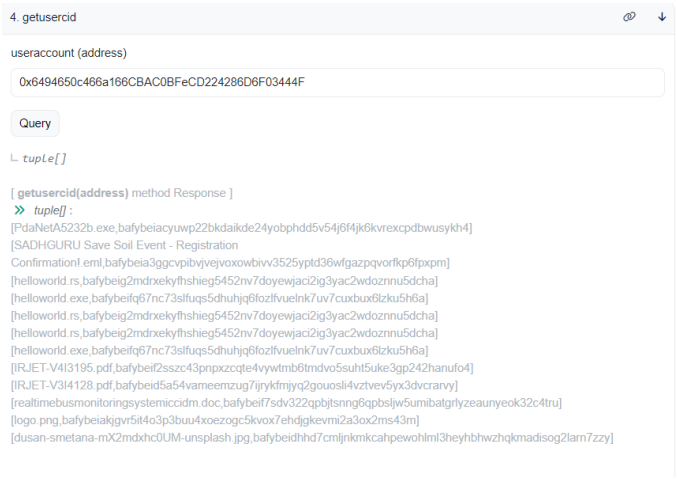

Figure 5: The GetUserCID function for public upload on Goerli Etherscan

element containing a name and CID data. In the private upload contract, it returns an array of file lists, with each element having a CID, name, key, and IV attribute.

c. File Deletion

An option allows for file deletion from the blockchain. RemoveCID(string CID): This function removes file details associated with the given CID from the blockchain contract.

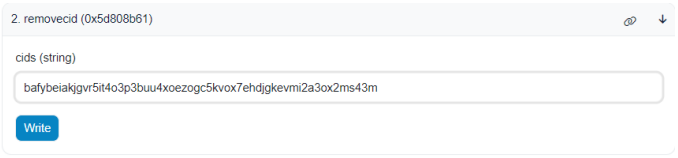

Figure 6: The RemoveCID function on Goerli Etherscan

4.3 Public and Private File Upload/Download

In public mode, users select and upload files from their systems using an IPFS client, preserving file integrity. The resulting CID is stored in the publicupload.sol contract for downloads.

AddCID(string CID, string name): This function takes a CID and the name of a file and writes them into the blockchain. It requires user fee approval.

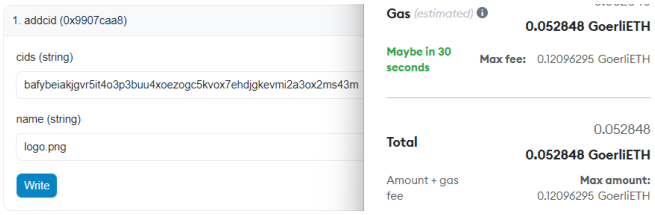

Figure 7: The AddCID function in the public contract on Goerli Etherscan

In private mode, users select files, upload them, and send them for encryption using AES with randomly generated keys and IVs. Encrypted files are uploaded via an IPFS client, and the CID, along with encryption parameters, is saved in the privateupload.sol contract for future downloads. Upon download requests, files are retrieved, decrypted using the corresponding key and IV, and made available for download.

AddCID(cids,name,key,IV): key and IV are encrypted using the user's public key and only passed as parameters in the AddCID function in privateupload contract.

4.4 File Sharing

The user can select a specific file for sharing; the following function will be utilized for sharing files:

ShareWithOther(user account address, CID): This function shares a file with the specified CID and a given user account address. In the public contract, only CID is required, but in the private contract, additional arguments, including key and IV, are needed.

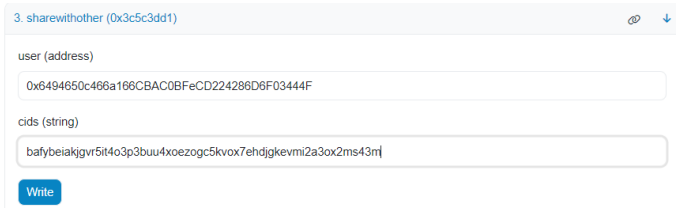

Figure 8: The ShareWithOther function on Goerli Etherscan

5. Result Analysis

For key and IV generation, the WebCrypto API is utilized, and since it is on the user side, it is manageable for small amounts of files that are uploaded by the user. For encryption and decryption, the system is able to handle about 1GB of files without issue since it is on the client side. But for larger files, chunking should be done to encrypt chunks of data and create the whole file in encrypted form so that the system doesn't have any issues due to the large file sizes in memory.

The system consists of more client-side operations for file encryption and decryption. Analysis was done for the required timing of encryption and decryption of the file using the AES algorithm on the client side. Following were the results that were found.

Based on file type

It is observed that specific file types exhibit varying durations in the encryption and decryption process. Audio files require the most time, whereas text files require the least. The graph below, extracted from the same paper, illustrates the encryption time while maintaining other parameters, such as file size, nearly constant.

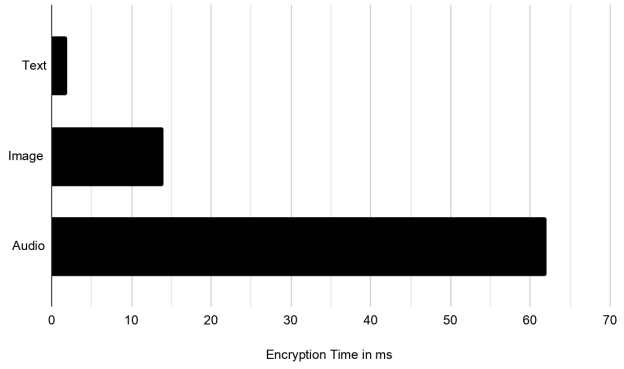

Figure 9: Encryption Time Comparison

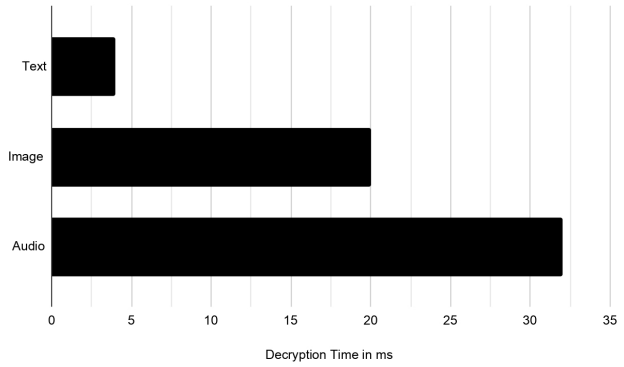

Figure 10: Decryption Time Comparison

Based on file size

The encryption time in AES increases proportionally with the key size employed. It is evident that encryption time experiences exponential growth with key sizes of 192 and 256, while for a key size of 128, the encryption time rises more gradually in comparison to the others [14]. Consequently, for larger files, a key size of 128 is the preferred choice. As the packet size (file size) increases, the encryption time of the file increases exponentially, making it obvious that large key sizes consume more energy. Cryptosystems have a problem with high energy consumption, and this trend is used to increase system responsiveness and efficient energy consumption.

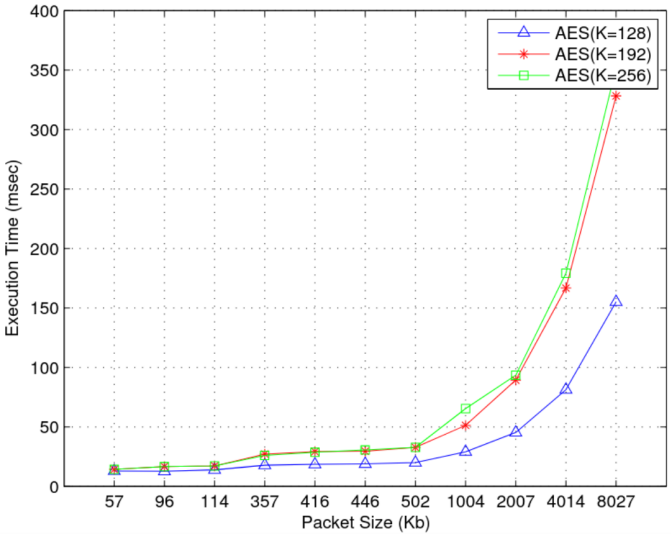

Figure 11: Encryption Time vs Packet size

AES vs other Symmetric algorithms

AES stands out in security due to its larger 128-bit block size, surpassing other symmetric encryptions like DES[11] and 3DES with their 64-bit blocks. This larger block size boosts AES's resilience against brute force and differential cryptanalysis attacks. In terms of speed, AES outperforms other symmetric encryption methods, particularly in software implementations. Its implementation complexity is simpler, thanks to well-documented processes and efficient software and hardware support [15]. AES excels in security, speed, and ease of implementation. This makes it a top choice for secure data transmission and storage in applications like internet banking, e-commerce, and online communication.

Comparison with Centralized Storage System

The system uses a smart contract for storing user file pointers and IPFS for file storage, and uses AES and public key cryptography for encryption and sharing. Centralized storage systems lack native encryption and decryption capabilities, allowing users to maintain control over their data storage and share files without third-party access. To address this problem, a system was developed that uses the AES algorithm for client-side encryption and decryption. The key was encrypted with the user's public key and the file was uploaded to IPFS, providing a unique identifier (CID). To move away from a centralized system, blockchain smart contracts were chosen to store the user's file CID and encrypted key. This ensures that users cannot decrypt files without the file owner's private key, a key issue in centralized systems. Additionally, the CID is generated based on the contents of the file to ensure that file integrity is not compromised. The immutable nature of user CIDs in smart contracts ensures that file integrity is not compromised by changes in file content.

[11] Data Encryption Standard

6. Conclusion

A file system developed using blockchain as a pointer to data stored using IPFS provides a platform for private or public data uploads into IPFS. The deployment of smart contracts, content-based addressing, and encryption techniques enabled a robust and secure alternative to centralized file storage systems, empowering users to manage their data on their own terms while ensuring data integrity and confidentiality. Implementation of access control mechanisms provided legitimate users with access to resources in a well-defined manner. Integrating the system with other blockchain networks can further enhance the security and transparency of the file storage system.

References

[1] Kravtsov. Semantic web and web 3.0. 21, 2019.

[2] Melanie Swan. *Blockchain: Blueprint for a New Economy*. O'Reilly Media, 2015.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009.

[4] Vitalik Buterin. Ethereum whitepaper. 2014.

[5] Jahn Arne Johnsen, Lars Erik Karlsen, S. S. Birkelan, and Sebjorn Saether Birkeland. Peer-to-peer networking with bittorrent. 1(2):1–22, 2005.

[6] Juan Benet. Ipfs - content addressed, versioned, p2p file system. 1(3), 2019.

[7] F. Pub. Advanced encryption standard (aes). 2001.

[8] Mohammed, Abdalbasit, and Nurhayat. Varol. A review paper on cryptography. 1(1), 2019.

[9] J.H. Silverman, Jill Pipher, and Jeffrey Hoffstein. *An Introduction to Mathematical Cryptography*. Springer, 2008.

[10] Pearl Alisha Lobo and V Sarasvathi. Distributed file storage using ipfs and blockchain. 1(1):1–6, 2021.

[11] S. Cui, Asghar, M. Rizwan, and Russello Giovanni. Towards blockchain-based scalable and trustworthy file sharing. *International Conference on Computer Communication and Networks (ICCCN)*, 1(1):1–2, 2018.

[12] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. 1(1), 2019.

[13] Ralph C. Merkle. A digital signature based on a conventional encryption function. 1987.

[14] Haque, M. Enamul, Zobaed, Sm I. M. Usama, and M.Areef Faaiza. Performance analysis of cryptographic algorithms for selecting better utilization on resource constraint devices. 1(1), 2018.

[15] Shraddha More and Rajesh Bansode. Implementation of aes with time complexity measurement for various input. *Global Journal of Computer Science and Technology*, 15(4), 2015.