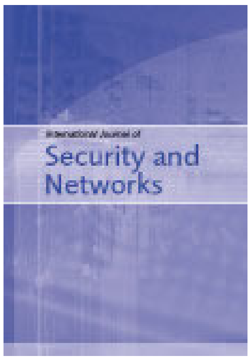


International Journal of Security and Networks

ISSN online: 1747-8413 - ISSN print: 1747-8405
https://www.inderscience.com/ijsn

A slice-based encryption scheme for IPFS

Changsong Zhou, Guozi Sun, Xuan You, Yu Gu

DOI: 10.1504/IJSN.2022.10047182

Article History:

Received:	02 January 2022
Accepted:	20 January 2022
Published online:	03 April 2023

A slice-based encryption scheme for IPFS

Changsong Zhou

School of Computer Science,
Nanjing University of Posts and Telecommunications,
Nanjing, 210023, China
Email: eric_zhou97@163.com

Guozi Sun*

School of Computer Science,
Nanjing University of Posts and Telecommunications,
Nanjing, 210023, China
and
Key Laboratory of Urban Land Resources Monitoring and Simulation,
MNR, Shenzhen 518000, China
Email: sun@njupt.edu.cn
*Corresponding author

Xuan You and Yu Gu

School of Computer Science,
Nanjing University of Posts and Telecommunications,
Nanjing, 210023, China
Email: you3xuan@163.com
Email: 243477384@qq.com

Abstract: The interplanetary file system (IPFS) has been used more and more widely because of its advantages of smooth integration with the current blockchain platform and its advantages as a distributed file system. However, the authors found that IPFS has some privacy issues; it cannot completely avoid unauthorised access to data by malicious nodes. In response to this problem, the authors propose a lightweight encryption scheme based on the characteristics of IPFS file slicing combined with AES256 and SHA256, which can be smoothly integrated into IPFS. During the upload process, this scheme encrypts some sliced file blocks according to the strategy formulated by the user. During the download process, the encrypted block is identified and decrypted according to a special encryption method. Through this scheme, the system can increase file security without affecting the performance of IPFS itself and retain the deduplication effect of IPFS to the utmost extent.

Keywords: advanced encryption standard; AES; SHA256; blockchain; distributed storage; security; Merkle DAG; interplanetary file system; IPFS; slice; deduplication.

Reference to this paper should be made as follows: Zhou, C., Sun, G., You, X. and Gu, Y. (2023) 'A slice-based encryption scheme for IPFS', *Int. J. Security and Networks*, Vol. 18, No. 1, pp.42–51.

Biographical notes: Changsong Zhou is a post-graduate student in the School of Computer Science, Nanjing University of Posts and Telecommunications. His research interests include blockchain, IPFS, distributed storage, and cybersecurity. He participated in the development of blockchain file management system, DFPP laboratory and other projects. Currently, he is engaged in IPFS cluster technical support and research.

Guozi Sun is a Professor at the School of Computer Science, Nanjing University of Posts and Telecommunications. He received his PhD from Nanjing University of Aeronautics and Astronautics in 2002, entered the post-doctoral station of Tsinghua University in 2003, and returned to Nanjing University of Posts and Telecommunications after leaving the post-doctoral station in 2005. His research interests include computer communication networks and security, blockchain forensics, digital forensics and digital investigation. He has published more than 60 scientific papers and has co-authored two monographs on digital forensics.

Xuan You is a post-graduate student in the School of Computer Science, Nanjing University of Posts and Telecommunications. His research interests include blockchain infrastructure, blockchain security, distributed storage, cybersecurity and cloud native. He participated in the development of blockchain file management system, DFPP laboratory and other projects. Currently, he works on fabric applications.

Yu Gu is a post-graduate student in the School of Computer Science, Nanjing University of Posts and Telecommunications. His research interests include blockchain forensics, cybersecurity and digital forensics. He participated in the development of blockchain file management system, DFPP laboratory and other projects. He is familiar with electronic forensics and the underlying technology of blockchain and has participated in the translation of many related technical books.

1 Introduction

With the further expansion of network scale, the problems brought by centralised system become increasingly serious. Centralised systems are vulnerable to single points of failure or external attacks from malicious actors, who may disclose data or confidential information. The situation becomes worse when user privacy is involved, which may lead to adverse effects. Besides, excessive server and bandwidth costs also make people more inclined to use distributed data storage systems. Therefore, the related research directions such as distribution and decentralisation have attracted extensive attention and research.

The interplanetary file system (IPFS) (Benet, 2014) is one of the most representative distributed data storage systems. However, the reason that makes IPFS a real research hotspot is the blockchain. The blockchain has to upload data to the public blockchain, which may cause scalability and privacy issues (Politou et al., 2019). In fact, it is not advisable to put a large amount of data in a blockchain transaction because of the high cost involved. In this case, when the node needs to download the entire ledger, there will be a delay problem. Moreover, the original advantages of the blockchain such as immutability and transparency will also cause problems when it comes to private data. Therefore, the combination with IPFS is currently a relatively complete solution.

IPFS uses many mature technologies, including distributed hash table (DHT) (Maymounkov and Mazières, 2002), BitTorrent (Cohen, 2003), GIT and SFS (Mazières, 2000), and simplifies and modifies them into a single cohesive system, but not just the sum of its parts. Due to the smooth integration of IPFS with the current blockchain platform, it is widely used for extended storage outside the chain, which has led to its widespread adoption by many blockchain projects to solve the problem of insufficient storage space on the chain.

However, IPFS has security problems (Politou et al., 2020). It does not provide access control at the connection level to restrict untrusted peers from obtaining unauthorised data, which means anyone holding the CID can obtain the corresponding content from the network.

Most of the work on IPFS has been dedicated to using IPFS as distributed data storage to develop distributed applications (DAPP), but the security of private data has not been considered seriously. The few papers (Hoffman et al., 2020; Ali et al., 2017; Xu et al., 2018 Li, 2018; Lin and Zhang, 2021) on security issues mostly increase the security of files from the direction of access control, and only a few papers (Sun et al., 2020; Pham et al., 2020) try to study from the perspective of encrypting file objects. However, these solutions still have problems such as heavy workload and complicated implementation. In addition, this also makes IPFS lose the characteristic attribute of deduplication. Therefore, the authors attempt to improve IPFS while retaining all the functions of the original IPFS without affecting the deduplication effect and without affecting the network performance, and provide users with the option of increasing file security at the expense of slightly increasing the workload.

The rest of this article is structured as follows. In the Section 2, the authors will review related works. Section 3 proves that there are data security issues on IPFS *storage nodes*. Section 4 introduces methods to protect data security. The specific methods and results of the experiment are given in Section 5, followed by conclusions and prospects for the future.

2 Background

IPFS (Benet, 2014) is a network transfer protocol that can create persistent and distributed storage and file sharing. Its goal is to replace hypertext transfer protocol (HTTP) to provide users with better network services. Above the routing and switching layer, IPFS uses Merkle DAG and hash to achieve deduplication of the entire network (including fragmentation) to prevent tampering and because the entire node maintains a DHT, the entire network can be realised without a server and connect all the idle storage devices together to make full use of hardware resources. At the same time, since that it is a distributed system, it is easy to dynamically expand the storage space. With reference to the P2P protocol, at the exchange layer, once a node establishes a connection, IPFS will use the BitSwap protocol to control data transmission. Combining the idea of git version control, IPFS constructs an encrypted authentication data structure to support file version control and efficient distribution. It uses the SFS self-verifying file system to verify the server (to achieve a private network) and establish a secure communication channel to the remote file system.

Recently, there are many applications about IPFS. In 2020, Hoffman et al. designed a DAPP, which uses the smart contract system based on Ethereum and IPFS storage mode for testers. The smart contract model provides a secure and transparent platform for the bug bounty program. Testers will submit the bug they find through the blockchain, and the company will accept or reject the bug through the blockchain. In 2017, Ali et al. proposed a modular alliance architecture based on blockchain and IPFS to solve the problem of data privacy on the Internet of things, which not only solves the disadvantage that the traditional blockchain network cannot store massive amounts of data, but also avoids the centralised management mode of IOT data. In 2018, Xu et al. proposed a social media application based on Ethereum and IPFs, in which Ethereum is used to save user data and IPFS is adopted to save large file data. This not only ensures the integrity and authenticity of user data, but also solves the redundancy problem caused by a large scale of file storage. In 2021, Lin and Zhang proposed a privacy protection method based on blockchain and improved IPFS. This method uses blockchain to store file information and user permissions. The improved IPFS can control file data sharing according to user permissions. At the same time, they expand the user group and file directory management function based on smart contract. Through this method, users can centrally manage the shared users in the system, organise file directories efficiently, and protect private files at the same time.

Many of the above applications use IPFS, and some applications provide access control in order to solve the security problem. However, only providing access control, the file still has security problems, which will be proved and described in the next chapter.

In 2020, Sun et al. combined blockchain technology to build an attribute-based encryption scheme for secure storage and efficient sharing of electronic medical records in IPFS storage environment. The scheme is based on ciphertext policy attribute encryption, which can effectively control the access of electronic medical data without affecting the efficient retrieval. It is feasible to use attribute-based encryption scheme to deal with small and high privacy files such as user cases, but this method is not suitable for large files.

In order to enhance the security and transparency of distributed storage system, in 2020, Pham et al. proposed the combination of IPFS, attribute-based encryption (ABE), multi-authority ABE (MA-ABE) and Ethereum blockchain. In this case, it may be necessary to encrypt large files, which will complicate the system and make the IPFS lose its deduplication effect. As more and more files need to be stored, the encryption time will be too long, which will seriously affect the use experience.

3 Problem description and verification

This section will explain why only access control is not enough, that the file object must be encrypted. First of all, it is necessary to understand where the file exists in the IPFS network, which will be demonstrated as the *storage node* mentioned below. Then, this chapter will separately elaborate on the files that are pinned and those that are not pinned. Finally, the authors will summarise the issues in conjunction with the work of others.

3.1 Definition of storage node

In practical application, IPFS network can be divided into private network and common network. The IPFS network used in most enterprise projects are private network.

IPFS realises the division of private network and public network through SFS, and only the nodes holding the corresponding key can access the corresponding private network. In the private network, since there is no upper incentive layer, all nodes are equal for storage. However, in actual application, the size of the storage space that these nodes can provide for IPFS network, which means the upper limit of warehouse of each node must be different. In this paper, those nodes that can provide a large amount of storage space are called *storage nodes*.

In the public network, IPFS is combined with its upper incentive layer called FileCoin, and miners provide huge storage space to ensure the backup redundancy of data storage and reorganisation in the entire network. The miner nodes here are also similar to the storage nodes defined in the article.

Storage nodes are necessary for IPFS. For a single node, if it does not join the IPFS network, it is meaningless to use IPFS for storage. In 2021, Abdullah et al. proved that IPFS cannot compare with FTP in terms of single read-write performance in small-scale private networks. The application advantage of IPFS in private network lies in the efficient utilisation of free storage space in the whole internal network and the data security brought by decentralisation, as well as reducing the server load through P2P acceleration after reaching a certain scale of network traffic. Therefore, only a medium-sized IPFS private network can give full play to the performance advantages of IPFS. In this case, in order to ensure that data is not lost, the IPFS network needs to have sufficient backup redundancy, which means that a file must be stored on multiple nodes. Consequently, no matter what kind of usage, if the users want to make IPFS meaningful and give full play to its performance advantages, you must have some nodes that provide a large amount of storage space, that is, the *storage nodes*.

3.2 Unauthorised access to pinned file

The experiments are aimed at the private network used by most of the above applications, but because FileCoin is only an incentive layer built on top of IPFS, most experiments can also be replicated on the *storage node* of the public network.

In IPFS, a file has two storage states in a node. One is pinned, which means that the file is locked in the node, and the other is not pinned, which means that the file is just in the cache. After a file is stored in the IPFS network, as the demand increases, there will be increasing nodes for caching the file, and the speed of downloading the file will become continuously faster. However, after this demand reaches its peak, it will become less and less over time, so it is necessary to remove redundant backups that are no longer needed in the IPFS network. In this regard, IPFS provides a garbage collector mechanism, which can run on schedule or manually to delete the cache files in the node. Therefore, in order to ensure that the file will not be garbage collected, a file needs to be pinned to multiple nodes. This has led to a large number of files being pinned by *storage nodes* that provide a large amount of storage space for the IPFS network.

Enter ‘ipfs pin ls’ directly in the console of the *storage node*, and the authors will get the *CID* of all the blocks pinned to this node. The *CID* with the suffix ‘recursive’ is the CID of a complete file, also known as the root. Then use ‘ipfs get *CID*’ to get this file, the file name is the above *CID*, but there is no suffix. However, by successively trying to replace various suffixes or using a file analyser, you can easily read the data in the file.

3.3 Unauthorised access to unpinned file

However, if the file is not pinned to other nodes, but only pinned to the user’s own node, there is also a problem if the file is cached on other nodes which are necessary in this situation. This may lead to the possibility of file loss. Also, the cache cannot ensure the security of the file on other nodes. Since the *storage nodes* can view the list of all local blocks by using ‘ipfs refs local’, through the comparison of the two lists, all newly added blocks can be obtained, and this is likely to include all the contents of the newly added cache file. The authors make an experiment with a 1.2 M size picture and input the command ‘ipfs refs local’ twice before and after caching the file on the *storage node*. By comparing the two storage lists of local warehouses, the paper found that all the new blocks’ CID and then use the API provided by ipfs-go-api to call the ‘cat’ method to obtain each block, determine whether it is a block storing file data, and finally try to splice those blocks storing file data in the form of data stream. After several attempts in the sequence, the original picture was finally obtained.

This article uses an 876 KB TXT file to further analyse the slicing method of IPFS. The authors figure out that the size of the local block is not the expected 256 KB (262,144 bytes), but 257 KB (262,158 bytes). After opening this block with txt, the paper found that this block has a garbled code at the beginning and the end. After deleting it, the file size is restored to the expected 256 KB (262,144 bytes). Considering that IPFS slices files in the form of a data stream, the authors open the file in binary mode and found that all the blocks have ‘0A 8A 80 10 08 02 12 80 80 10’ at the beginning and ‘18 80 80 10’ at the end. This means that although files are stored in file blocks in IPFS, each file block is almost in plaintext.

3.4 Description of related issues in other paper

In addition, because IPFS uses DHT technology, this means that when an IPFS node introduces new content, it will advertise the content it owns to all nodes connected to it. The more nodes the original node publishes content announcements, the more likely it is to encounter malicious nodes, leading to private data leakage. In 2021, Balduf et al.’s system can reveal which *CID* is requested by which node ID and IP address at which timestamp.

Therefore, if the file itself is not encrypted, the file is insecure. If the entire file is encrypted, the encrypted ciphertext is completely different due to the use of different keys, so the effect of IPFS network-wide deduplication will no longer exist. Even attribute-based encryption will invalidate IPFS deduplication when encrypting the same data in the face of multiple users. In addition, the aforementioned encryption will also complicate the system and require a huge amount of calculation when processing large files.

4 Embedded encryption methods based on slice

4.1 Overview

From the descriptions in the above sections, it can be concluded that when using IPFS, if the object is not encrypted, the data security cannot be ensured. Therefore, the authors propose a new scheme to increase the security of IPFS without affecting the deduplication effect and performance.

In the original version of IPFS, when a user adds a file to the IPFS network, the file is divided into multiple data blocks, and each data block has a unique identifier (*CID*) through which the user can search for data in the IPFS network. Besides to data blocks containing data content, some data blocks as parent nodes also contain the *CID* of lower-level data blocks, forming a directed acyclic graph. The file uploaded to IPFS will eventually return a unique file identifier to the user, which is also called the *CID* of the whole file or the root of the file in the DAG. Anyone can retrieve all data blocks of the file from IPFS through the file identifier and integrate them into one file.

From the perspective of usage, the changes to IPFS is to add an optional parameter to the upload and download functions (as shown in Figure 1), that is, to add an optional parameter *key* to the upload function to indicate whether encryption is required. If encryption is required, IPFS will not only return the parameter *CID* after uploading the file to the IPFS network, but also return a parameter *k*, as the symmetric encryption key. The download function adds an optional parameter *k*. If the file to be obtained this time is uploaded with encryption, the user needs to provide the correct parameter *k* to successfully download the file.

Regarding the management of key *k*, there are already many schemes for key management in distributed systems, so the specific key management scheme will not be discussed in depth in this article. For example, in 2021, Tang et al. well handled the problem of key distribution through DH key exchange technology. DH key exchange is a key exchange algorithm, which was proposed by Whitfield Diffie and Martin Hellman in 1976. The algorithm generates the same encryption key for two roles in a completely public environment, which is widely used in various data transmission protocols. According to the algorithm, two users only need to generate public content and private content respectively, and the same key can be generated by exchanging the public content of the other party (Joux, 2004). However, Diffie-Hellman key exchange cannot prevent man-in-the-middle attacks (Kocher, 1996). Due to the data traceability and tamper resistance of the blockchain, if the Diffie-Hellman key exchange process is implemented in the blockchain environment, this problem can be effectively avoided. Coincidentally, the most widespread application of IPFS is the combination with blockchain.

Figure 1 Schematic diagram of the distribution state before and after differential data processing/IPFS network (see online version for colours)

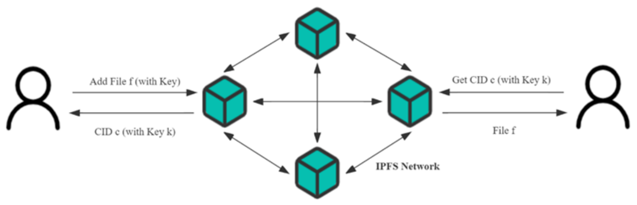


Figure 2 Chunker and DAG

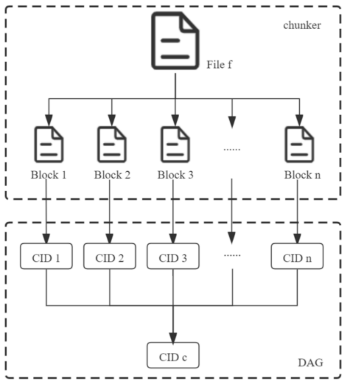


Figure 3 Upload (see online version for colours)

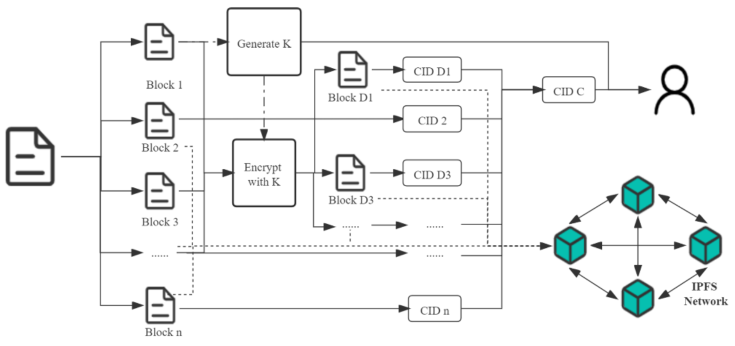
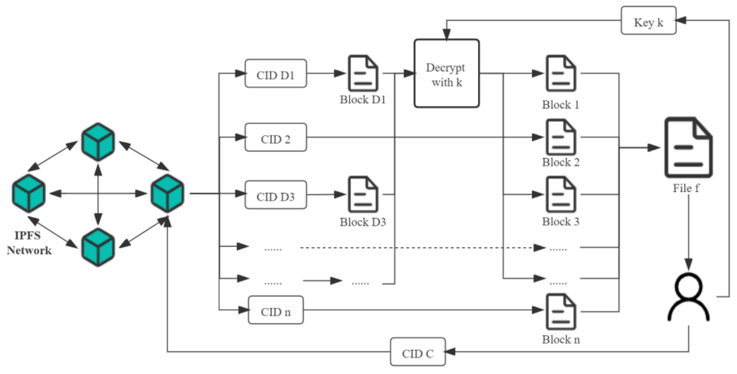


Figure 4 Download (see online version for colours)



Moreover, many of the schemes in background section also have feasible key management methods. Finally, the user can also handle the key *k* like the CID before.

4.2 Chunker and Merkle directed acyclic graph

If the authors want to modify the upload and download functions of IPFS, you must have a detailed understanding of the upload and download process. Among them, slicer and directed acyclic graph are the two most important parts.

Chunker, as shown in Figure 2, in the process of uploading files, IPFS first calls the slicer called chunker to slice the imported files in the form of data streams. The file is cut into blocks of a specified size (256 KB by default), and then these blocks are simply processed to become the most basic unit block of IPFS storage.

The Merkle directed acyclic graph is constructed on the basis of Merkle tree. The hash tree is composed of content blocks, and each content block is identified by its *CID*. The authors can refer to any of these blocks using its *CID*, which allows us to build *CIDs* that use these sub-blocks or refer to a tree with these sub-blocks as the root node. This means that our files are broken down into blocks and then arranged in a tree structure using 'link nodes' to connect them together. The *CID* of a given file is actually an encrypted hash of the root node (the uppermost layer) in the DAG. This brings the following three important advantages to IPFS. First, content addressing: use multiple hashes to uniquely identify the content of a data block. Second, anti-tampering: It is easy to check the Hash value to confirm whether the data has been tampered with. Third, deduplication: Since the hash values of data blocks with the same content are equal, it is easy to remove duplicate data and save storage space. During the upload process, those same data can be filtered out by Merkle DAG, and only need to add a file reference without occupying storage space.

The object format of Merkle DAG is defined in IPFS. IPFS object is a storage structure. There are two parts stored in the IPFS object: one is link, which is used to save references of other block data; the other is data, which is the content of this object. Link mainly includes three parts: link name, hash and size. If Merkle DAG is used to store the modification of the source file, the modified content may be only a small part. The system no longer needs to back up the whole modified file, which is the reason why IPFS saves storage space.

The blocks cut out by the chunker are sequentially input into the DAGBuilder module, which stores the entire file in the Merkle DAG structure, obtains the CID of each node in turn, and finally returns the CID of the root node of the entire file in the DAG.

4.3 File upload

In the upload part, the changes are mainly concentrated in the DAGBuilder module. As introduced in the previous section, during the upload process, the file is first sliced into blocks of a specified size by chunker and then input into DAGBuilder.

As shown in Figure 3, when the user adds the optional parameter *K*, the improved IPFS of the article will use the hash of the first cut out block as the key *k*. The hash method uses SHA256, which means that a 256-bit hash will be generated, which is also the method adopted by Bitcoin (Nakamoto, 2008). Then use the 256-bits key *k* as a symmetric encryption key to encrypt the file block that needs to be encrypted. The encryption method also uses the same AES-256 (Murphy, 1999) as Bitcoin, which is still sufficient to ensure data security at present.

Which file blocks need to be encrypted is mainly determined by two aspects, and this is also the result of considering both large and small files. The authors add two attributes to the IPFS configuration file for users to modify, encryption header length and encryption density. The encryption header length indicates how many consecutive blocks at the beginning of the file must be encrypted. Considering that some small files are still stored in clear code even after being cut, such as txt files, the default value is set to 4, which will ensure the security of most small files. The encryption density represents how many of the remaining blocks need to be encrypted except the encryption header. Considering the existence of large files, the default value is set to 0.01 (threshold), indicating that one block is encrypted for an average of 100 blocks.

Considering the problem of identifying encrypted blocks during decryption, there is a higher priority rule for encryption. Except for the first block, other blocks that are less than 256 KB will not be encrypted. This means that the system will only encrypt the first block and 256 KB blocks. The ZeroPadding method is adopted for the encryption of the first block. For the encryption of the remaining blocks full of 256 KB, the method of filling the key *k* at the end and then encrypting is adopted to ensure that except for the first block, the remaining encrypted blocks are larger than 256 KB to facilitate identification. And in this case, even if the uploaded user and the accepted user profile are inconsistent, the function will not be affected.

4.4 File download

In the download part, the changes are mainly concentrated in the NewDagReader module. Since files are stored in the form of file blocks in IPFS, after the user uses the download command, IPFS will obtain each file block in turn according to the DAG recording file slice information and the DHT recording storage node information, and finally Integrate these blocks into a complete file locally.

However, after the upload function is changed, the original download function is no longer applicable, and the file blocks stored in the IPFS network are no longer just plaintext blocks that are less than or equal to 256 KB. Therefore, as shown in Figure 4, it is necessary to identify the file block during the download process to determine whether it is a ciphertext block. If it is, it needs to be decrypted to obtain the plaintext block.

According to the encryption strategy mentioned in the previous section, there are two main rules for detecting whether a file block is a ciphertext block. First, the blocks larger than 256 KB are all ciphertext blocks. Second, if the user adds the optional parameter key *k*, the first block must be a ciphertext block.

In addition, due to the possibility of malicious users, it is necessary to consider the situation in which malicious users download encrypted files without providing the key or providing the wrong key. The detection methods for these two cases are as follows. First, the ciphertext block is detected and the user does not provide the key *k*. Second, the user provides the key *k*, but it is not equal to the hash after the first block is decrypted.

5 Implementation and performance evaluation

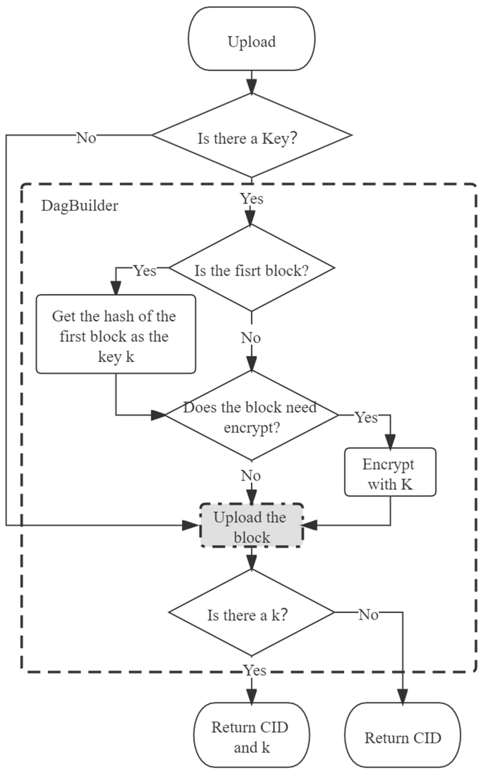
5.1 Environment configuration

The experiment environment used in this article is as follows: the operating system is Windows 10 Pro, Inter(R) Core (TM) i7-9750 CPU @ 2.60GHz, mechanical hard disk, 16 GB memory. The IPFS version is v0.6.0, and the go language version is go1.15 windows/amd64.

5.2 Upload implementation

IPFS brings together a large number of distributed ideas and opens all functions to users in the form of command lines. Therefore, the paper's improvements are also provided to users in the form of command lines and do not affect the original functions of IPFS.

Figure 5 Upload implementation



The DagBuilder module constructs the received blocks into DAG form for structured storage. This part of the content is the original function of IPFS, which is not described in detail in the article and is partially omitted in Figure 5. In the figure, there is only 'upload the block' is the original content of IPFS.

As shown in Figure 5, the authors add several steps at the beginning and end of the DagBuilder module to achieve the functions mentioned in previous chapter. The main code changes are concentrated in the 'importer/helpers/dagbuilder.go' file of the 'go-unixfs' sub-project. The entire implementation has a number of steps as follows:

Step 1 After the file is sliced by chunker, it will first judge whether there is an optional parameter *key*. If so, go to step 2. If not, return to the original process.

Step 2 It will first determine whether the current block is the first block. If so, obtain the hash of the block as the encryption key *k*.

Step 3 Determine whether the current block needs to be encrypted according to the encryption strategy set by the user. If encryption is needed, use the key *k* to encrypt, and then take the encrypted or unencrypted file block in the original way to construct the DAG and upload it to IPFS network.

Step 4 After uploading, determine whether the returned parameter contains the key *k* according to whether the key *k* is empty.

5.3 Download implementation

The NewDagReader module will recursively obtain each block according to the DAG, and combine them into a complete file in order. This part of the content is the original function of IPFS, which is not described in detail in the article and is partially omitted in Figure 6. Only the 'Combine file' in the picture is the original content of IPFS.

As shown in Figure 6, this article adds several steps to the NewDagReader module, mainly for processing the acquired blocks to achieve the functions mentioned in previous chapter. The main code changes are concentrated in the 'io/dagreader.go' file of the 'go-unixfs' sub-project. The entire implementation has a number of steps as follows:

Step 1 Determine whether there is an optional parameter *k*. If so, the system will decrypt the first block and obtain the hash of the decrypted block. If the hash is not equal to the key *k* provided by the user, it is proved that the user has not provided the correct key; the system terminates the process, deletes the downloaded data, and feeds the result back to the user. If there is no optional parameter *k*, enter the NewDagReader module as usual.

Figure 6 Download implementation

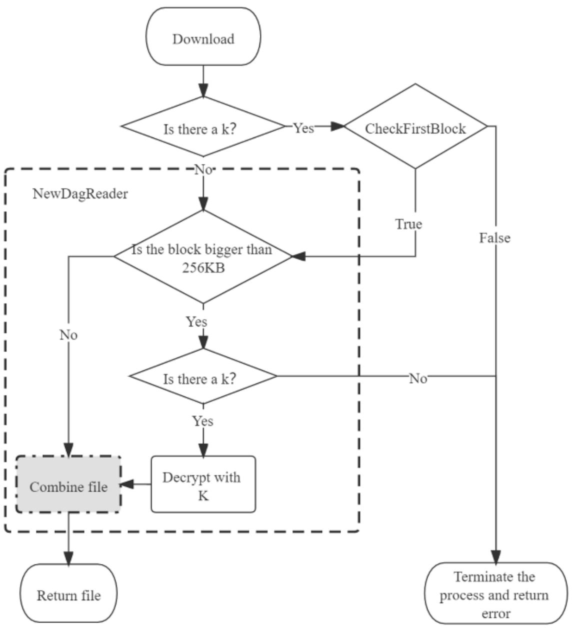


Figure 7 Block reuse

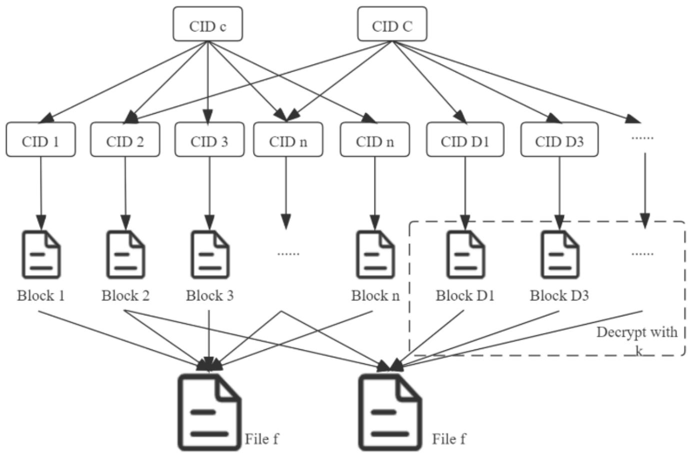


Figure 8 Upload performance

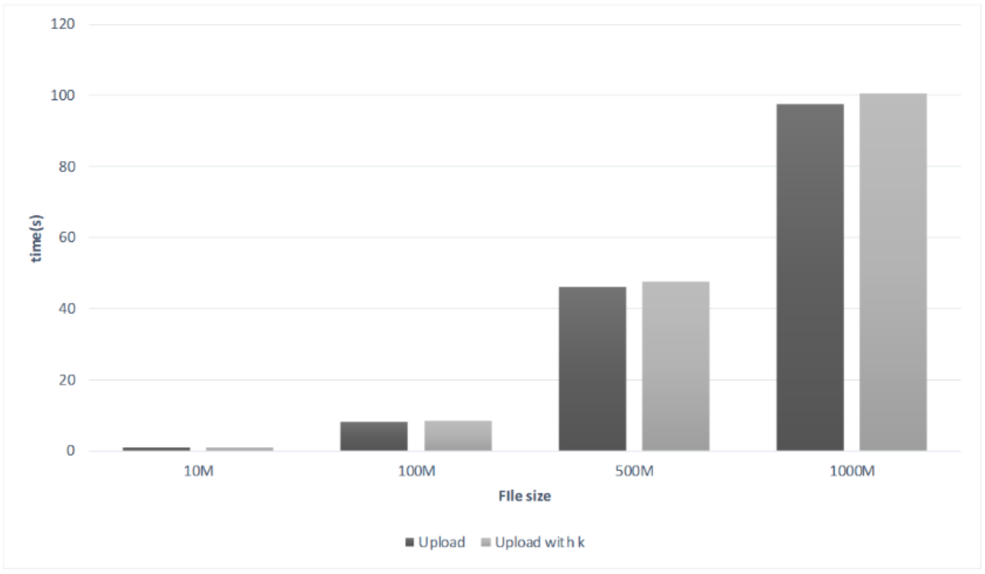
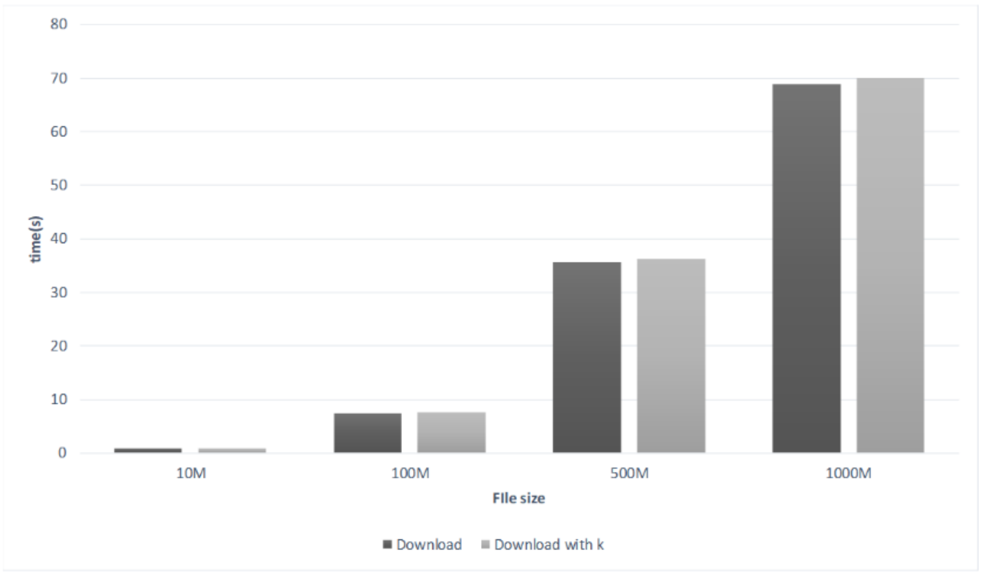


Figure 9 Download performance



Step 2 Determine whether there are blocks larger than 256 KB in each block obtained in NewDagReader, if yes, determine whether there is a key *k*, if yes, perform decryption to obtain a plaintext block with a size of 256 KB. If not, the process will be terminated and a prompt of wrong password will be returned. Blocks not larger than 256 KB are combined files normally.

Step 3 Return the final combined file to the user.

5.4 Deduplication effect

The article does a certain test and analysis on the improved IFPS deduplication feature, taking a file stored on the same node or the same IPFS network by multiple users as an example.

According to Figure 7, it can be seen that most of the blocks can be reused, which greatly saves space. In addition, since the encryption key uses the hash of the first file block, the keys of the same file are also the same, which makes the encrypted block also reusable. Because only those who have the same file will get the same key, this method will not cause leakage problems. Besides, because the encryption keys are the same, no matter how many users encrypt and store the same file, the space occupied by the file in the IPFS network is at most twice the original size.

5.5 Performance evaluation

The biggest negative impact of the solution on the system is the time to download and upload files. Therefore, the authors conduct two rounds of experiments to test the time-consuming uploading and downloading of files of 10 M, 100, 500, and 1,000 M size between the changed IPFS and the original IPFS to determine whether the impact of the changes on the original IPFS is negligible. The experimental data is the average duration of ten operations using go-ipfs-api, and the memory clearing operation is performed before each operation to prevent it from affecting the experimental results (the configuration file uses the default value, which means the encryption header length is 4 and the encryption density is 0.01).

Figure 8 shows the time it takes for the two types of IPFS to add files of different sizes. It can be seen from the experimental data that the improved IPFS has less impact on the time required to upload files. The authors use symmetric encryption technology, and the time complexity is much smaller than that of asymmetric encryption. Secondly, according to the encryption strategy in this article, the actual amount of encrypted data is still far smaller than other methods, and this method does increase the security of the file.

The bar on the left of Figure 9 shows the time taken by the original IPFS to obtain shared files through *CID*. In contrast, the improved IPFS displayed by the bar on the right has no significant difference in the time to download files of different sizes.

6 Conclusions

This article introduces a method to enhance the privacy of file data in IPFS at the cost of a slight increasement in workload. Compared with other methods, the workload brought by this schema is almost negligible. Moreover, it hardly affects the various advantages of IPFS itself. In view of the simple coded data stream file such as txt file, the unencrypted part is still equivalent to plaintext storage, and the information can be directly obtained. Therefore, in the future, the authors will consider optimising the encryption strategy, and formulate a more complete encryption strategy according to different file types, such as the 0.7 z format. Files in this format only need to encrypt a few blocks containing the tail file to greatly enhance the security of the file, while the txt format is just the opposite. Furthermore, because the objects of encryption and decryption are both blocks, it is possible to use multithreading to reduce time-consuming.

Acknowledgements

The authors would like to thank the anonymous reviewers for their elaborate reviews and feedback. This work was supported by the National Natural Science Foundation of China (No. 61906099), the Open Fund of Key Laboratory of Urban Land Resources Monitoring and Simulation, Ministry of Natural Resources (No. KF-2019-04-065).

References

Abdullah Lajam, O. and Ahmed Helmy, T. (2021) 'Performance evaluation of IPFS in private networks', in *2021 4th International Conference on Data Storage and Data Engineering*, February, pp.77–84.

Ali, M.S., Dolui, K. and Antonelli, F. (2017) 'IoT data privacy via blockchains and IPFS', in *Proceedings of the Seventh International Conference on the Internet of Things*, October, pp.1–7.

Balduf, L., Henningsen, S., Florian, M., Rust, S. and Scheuermann, B. (2021) *Monitoring Data Requests in Decentralized Data Storage Systems: A Case Study of IPFS*, arXiv preprint arXiv: 2104.09202.

Benet, J. (2014) *IPFS-Content Addressed, Versioned, P2P File System*, arXiv preprint arXiv: 1407.3561.

Cohen, B. (2003) 'Incentives build robustness in BitTorrent', in *Workshop on Economics of Peer-to-Peer systems*, June, Vol. 6, pp.68–72.

Hoffman, A., Becerril-Blas, E., Moreno, K. and Kim, Y. (2020) 'Decentralized security bounty management on blockchain and IPFS', in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, IEEE, January, pp.0241–0247.

Joux, A. (2004) 'A one round protocol for tripartite Diffie-Hellman', *Journal of Cryptology*, Vol. 17, No. 4, pp.263–276.

Kocher, P.C. (1996) 'Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems', in *Annual International Cryptology Conference*, Springer, Berlin, Heidelberg, August, pp.104–113.

Lin, Y. and Zhang, C. (2021) 'A method for protecting private data in IPFS', in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, IEEE, May, pp.404–409.

Maymounkov, P. and Mazieres, D. (2002) 'Kademlia: a peer-to-peer information system based on the XOR metric', in *International Workshop on Peer-to-Peer Systems*, Springer, Berlin, Heidelberg, March, pp.53–65.

Mazières, D.D.F. (2000) *Self-Certifying File System*, Doctoral dissertation, Massachusetts Institute of Technology.

Murphy, S. (1999) 'The advanced encryption standard (AES)', *Information Security Technical Report*, Vol. 4, No. 4, pp.12–17.

Nakamoto, S. (2008) 'Bitcoin: a peer-to-peer electronic cash system', *Decentralized Business Review*, p.21260.

Pham, V.D., Tran, C.T., Nguyen, T., Nguyen, T.T., Do, B.L., Dao, T.C. and Nguyen, B.M. (2020) 'B-Box – a decentralized storage system using IPFS, attributed-based encryption, and blockchain', in *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, IEEE, October, pp.1–6.

Politou, E., Alepis, E., Patsakis, C., Casino, F. and Alazab, M. (2020) 'Delegated content erasure in IPFS', *Future Generation Computer Systems*, Vol. 112, No. 32, pp.956–964.

Politou, E., Casino, F., Alepis, E. and Patsakis, C. (2019) 'Blockchain mutability: challenges and proposed solutions', *IEEE Transactions on Emerging Topics in Computing*, Vol. 9, No. 4, pp.1972–1986.

Sun, J., Yao, X., Wang, S. and Wu, Y. (2020) 'Blockchain-based secure storage and access scheme for electronic medical records in IPFS', *IEEE Access*, Vol. 8, No. 5096, pp.59389–59401.

Tang, X., Guo, H., Li, H., Yuan, Y., Wang, J. and Cheng, J. (2021) 'A DAPP business data storage model based on blockchain and IPFS', in *International Conference on Artificial Intelligence and Security*, Springer, Cham, July, pp.219–230.

Xu, Q., Song, Z., Goh, R.S.M. and Li, Y. (2018) 'Building an ethereum and IPFS-based decentralized social network system', in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, December, pp.1–6.