Log Storage System - Block Chain

Hitesh K
Student, Dept. of CSE,
Global Academy of Technology, Bengaluru

Koushik N S
Student, Dept. of CSE,
Global Academy of Technology, Bengaluru

Shivaraja B R
Student, Dept. of CSE,
Global Academy of Technology, Bengaluru

Puneeth K
Student, Dept. of CSE,
Global Academy of Technology, Bengaluru

***Abstract*—Logs are critical data, which can help us to troubleshoot and identify the person in charge of an unexpected accident. Log systems have been widely used for log storage. However, the traditional log system can't prevent the log from being tampered. Centralized servers are more vulnerable to be attacked. Those who have permission to operate records can easily tamper with logs. Blockchain is a disruptive technology in recent years, which has the advantages of decentralization, tamper-proof and traceability. Given its decentralization and tamper-proof properties, the paper proposed a blockchain-based framework for secure log storage. However, the cost of storing big files in the blockchain is very high. Thus, the paper utilized the InterPlanetary File System(IPFS) to store log files instead of a blockchain. Besides, the paper adopted Ethereum blockchain to store the hash of log files and a smart contract to create an index for log files. The solution is not only applicable to log but also other scenarios requiring secure data storage and efficient retrieval.**

Keywords-Log Storage; Blockchain; IPFS; Ethereum; Smart Contract

I. INTRODUCTION

Log data is digital evidence in disputes [1]. That is, we can learn about the Quality of Service(QoS) trough log data in the age of cloud computing. Nowadays, more and more enterprises and users choose to purchase services provided by cloud service providers. Since the server is hosted by a cloud service provider, it is necessary to utilize logs for problem investigation and accountability when some problems arise. Records can be used to draw conclusions that may affect the credibility of the service provider. There are many forms of log tampering. However, different types of cloud have different tampering motivations. We classify people with tampering motivation into two categories according to the cloud category: cloud service providers and IT departments. They may hide the truth by adding, modifying or deleting the log content. Next, we will explore some tampering situations according to the type of cloud.

In a public cloud, customers deploy applications directly to elastic cloud servers. To save resources and costs, they often choose elastic scaling schemes [2]. Usually, a threshold is set for CPU, memory and disk. When the usage rate reaches the threshold, additional resources are automatically allocated. For example, a rule is defined. When the CPU usage rate exceeds 90%, the cloud service provider should allocate 20% of the CPU to this application. Imagine that the customer received a complaint about application performance. Considering that the application has been able to respond to user's requests promptly, the customer may suspect that the elastic expansion has some problems, causing the application to fail to process the request in time when the request volume is high. Therefore, customers may ask the cloud service provider to provide a complete resource allocation report. By checking the logs, the cloud service provider found that it was indeed an automatic scaling function that intermittently worked, failing to allocate CPU in time. To escape liability and potential litigation, the cloud service provider is likely to tamper with the logs before sending the logs to the customer.

In a private cloud, all participants belong to the same company, but there may be a particular type of tampering motivation [2]. Now, we can imagine that an internet company has established a private cloud. The technical department was asked to back up data regularly. However, the primary storage of this company had failed so that critical data was not backed up. The technical department found that the backup stopped working some days ago and sent several alerts to them, but they had not checked these alerts. This technical department may tamper with logs, delete alert messages, and show reports generated based on tampered records to avoid being punished by the leader.

The above is just a brief list of possible log tampering in cloud solutions. However, we already can conclude that how to avoid log tampering in cloud solutions is very important through the above examples. Therefore, it is of high significance to provide a traceable, verifiable and tamper-resistant log system for cloud solutions. Although there are many solutions for log tampering detection, the traditional log system is not friendly to avoid log tampering. So, it is significant to provide a tamper-proof log system for cloud solutions.

The purpose of this paper is to solve the current log system's shortcomings by utilizing the tamper-proof, decentralization and traceability of blockchain technology to improve the trustworthiness, transparency of cloud solutions. The main contributions of this paper are summarized as follows:

1) We propose a blockchain-based framework to achievesecure log storage.

2)We utilize IPFS to solve the shortage of blockchain in storing big files.

1) We provide a complete code of smart contract and discuss critical implementation and test details to illustrate the crucial functionality of our prototype system.

The rest of this paper is organized as follows. Section II introduces the background and related work. Section III presents the overall system architecture of our proposed solution. Section IV describes the essential aspects of the implementation and illustrates the testing and validation of the smart contract code. Finally, we conclude the article and outline the future work in Section V.

II. Background and Related Work

A. Background

1) Blockchain

Blockchain is a distributed immutable ledger which consists of a continuous growing list of blocks [3], the structure of blockchain as shown in Figure 1. The blockchain records all transactions between participants in a blockchain network using blocks. Blockchain is able to prevent data tampering by nature. The information recorded in the block cannot be modified as it will cause the chain of blocks to be broken. Bitcoin introduced the concept of blockchain for the first time in 2008 [4][5]. Bitcoin is a first cryptocurrency that does not depend on a trusted third-party. Blockchain is the foundation of Bitcoin, but the blockchain is not only Bitcoin. It can also be used in many scenarios [6][7].

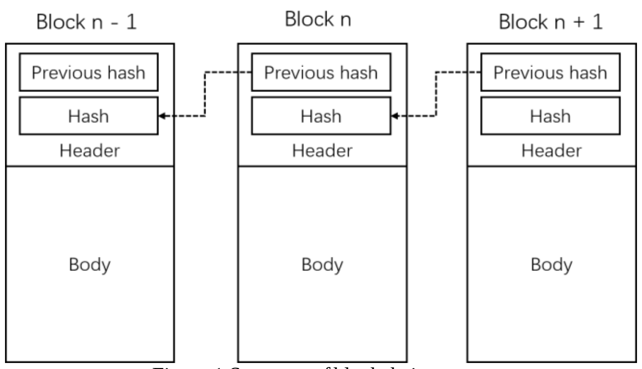


Figure 1 Structure of blockchain

2) Smart Contract

A smart contract is a computer program deployed onto a blockchain network [3]. The smart contract will automatically execute business logic of a service that all participants agreed when specific conditions are met [8]. As an open mutual agreement, the items of smart contract can be accessed by all participants [9]. Smart contract is a form of decentralized automation that validates and enforces agreements through transactions and records all state changes into blockchain [3].

III. Proposed Framework

3) IPFS

IPFS is a novel P2P distributed file system. It aims to build a file system that consists of same files using all computing devices [10]. IPFS is different from HTTP because it is based on content addressing. It generates a unique hash for every file uploaded to IPFS so that user can get file content according to its hash. Additionally, the hash will change when the file content is modified.

B. Related Work

The tampering of log file has existed for a long time. In order to reduce this phenomenon, researchers have proposed many schemes to detect document tampering. At present, there are various file verification technologies to detect whether the files have been tampered with. Peterson [11] introduced that verifys the authenticity of a given array of bits using cyclic codes. Similarly, checksum [12][13] is a common method to validate the integrity of files. Secure Hash Algorithm(SHA) is a popular hash algorithm [14] which can verify digital artifact. For example, git is designed as a distributed version control system that generates SHA-1 signature for source code [14]. But we also can edit the commit and then generate a new SHA-1 signature. Additionally, arXiv is a platform for storing digital documents and ensure the integrity of these digital documents. Although some services such as arXiv can help us prevent files from being tampered with, we must rely on the third-party service provider.

Many of traditional methods are also applicable for cloud solutions. However, the cloud environment is very complex and generates much logs, which brings more challenges in terms of the storage, retrieval and validation of logs. Sharma

[15] pointed out that a large-scale cloud environment is complex and suggested utilizing a variety of digital signatures and encryption algorithms to make sure the high level integrity of the key information stored in the cloud. Bharath and Rajashree [16] recommend using a third-party verification service to verify data integrity and send the integrity report to user. However, this solution also requires trusting third parties or central agencies.

As we know, blockchain is an append-only data structure that data cannot be forged. Smart contract is also a programmable contract that runs on a blockchain. It will be executed automatically when the predefined conditions are met. IPFS is a content-addressed distributed file system. We proposed a novel solution that combines blockchain and IPFS to achieve secure log storage.

Figure 2 System architecture of the blockchain-based solution for secure log storage.

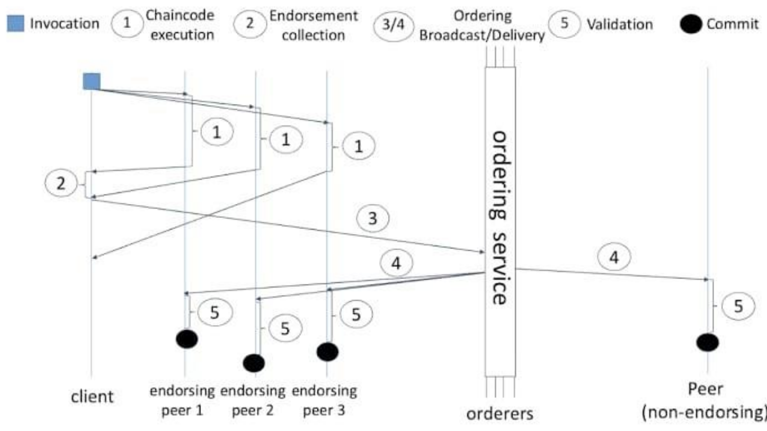


Figure 2 presents the overall system architecture we proposed. The system mainly consists of six parts: Database, Backend, IPFS, Smart Contract, Blockchain and Client. The function of each part is illustrated as follows:

1) *Database*

The database is responsible for storing logs that have not been uploaded to IPFS. If we already have a traditional log system, we can seamlessly migrate the traditional log system's database to the blockchain-based log system instead of using a new database.

2) *Backend*

The backend has two modules: Anchor and Query. Anchor is responsible for uploading logs from a database to IPFS, and then storing the hashes of log files to the blockchain. Query is responsible for receiving the user's query request and obtaining log indexes through the smart contract. Query is also responsible for obtaining the hash of log file from blockchain according to concrete log indexes and then getting the content of log file in IPFS using concrete file hash. Finally, Query responds log file content to client.

3) *IPFS*

IPFS is used to store concrete log files and then generates one hash for every log file. Finally, we can get content of a log file according to a specific hash.

4) *Smart Contract*

The smart contract is primarily used to receive transactions and then create an index for log file. What is more, the smart contract is also responsible for receiving query requests of getting log file indexes. The smart contract will return all block numbers containing log file hash within the time range, then the backend's Query module will get file hashes in the blockchain utilizing specific block number. Finally, the backend's Query module obtains log file content from IPFS based on log file hash.

5) *Blockchain*

Blockchain is used to store the hash of log file and to store smart contract code. Additionally, blockchain also provides a running environment for smart contract.

6) *Client*

The client is mainly responsible for sending a query request to the backend and obtaining the query result from the backend.

IV. IMPLEMENTATION AND TESTING

The solution we proposed is for secure log storage and efficient retrieval. To achieve our target, we deployed private IPFS cluster and private Ethereum blockchain firstly. We also tested our deployed IPFS and Ethereum blockchain to make sure they were working well. Additionally, considering the speed at which consensus algorithms reach consensus, we had neither adopted the PoW nor PoS algorithm but took the PoA algorithm as our consensus algorithm. There are a lot of tools for developing Ethereum smart contracts, but we chose the official recommended Remix IDE to develop and test our smart contract when everything is ready. Because Remix IDE provides rich features that make it easy to develop and debug smart contract code before deploying them. Considering the limited number of pages, we will mainly introduce our implementation details and focus on testing the functionality of the smart contract.

A. *Implementation Details*

Figure 3 shows the index structure we designed for the log file. Figure 4 shows that the log file hash is recorded into a concrete block of the blockchain by a transaction. Algorithm 1 shows how to create an index for log file according to timestamp. Algorithm 2 shows how to get an index utilizing timestamp and then get a specific block based on the index. We only show the essential algorithms of our smart contract code in this paper. But we provide the complete smart contract code at github.

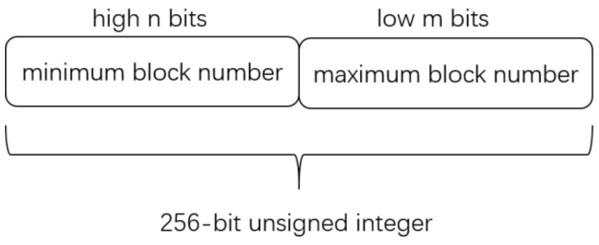


Figure 3 Structure of index

We used a 256-bit unsigned integer array to store log indexes. Every index is a 256-bit unsigned integer which represents the range of block numbers for a particular day. The highest n bits represent the smallest block number of the day. The lowest m bits represent the largest block number of the day. The structure of the index is as shown in Figure 3.

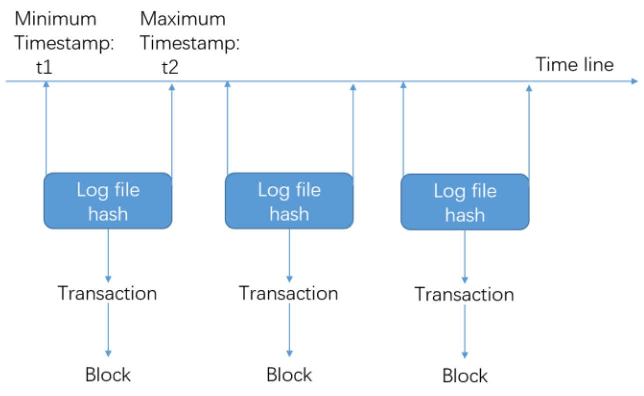


Figure 4 The process of file hash to blockchain.

Algorithm 1 Add a new log file

```
Require:
    The minimum timestamp of log file is $minTimestamp$;
    The maximum timestamp of log file is $maxTimestamp$;
    The hash of log file is $ipfsHash$, which is stored in
    transaction instead of smart contract;
Ensure: $maxTimestamp > minTimestamp$
 1: $minNum = (minTimestamp - startTimestamp)/dt$
 2: $maxNum = (maxTimestamp - startTimestamp/dt$
 3: $currentNum = indexArrayLength - 1$
 4: if $maxNum > currentNum$ then
 4:     Add a new data index.
 5: end if
 6: for each $i \in [minNum, maxNum]$ do
 7:     $maxBlockNum = indexArray[i] >> m$
 8:     if $currentBlockNum > maxBlockNum$ then
 8:         Expand the range of this data index.
 9:     end if
10: end for
```

Algorithm 1 is the core of storing hashes. The cost of changing the world state of the blockchain is very high. Besides, changes in the value of smart contract attributes can lead to changes in the world state of the blockchain, so it is appropriate to store hashes directly with the smart contract. Thus, we use the hash value as a parameter when sending a transaction, so that the hash value will be packaged into the block together with the transaction. The smart contract mainly provides the function of index and the ability to accept the parameter of hash value.

The backend's Anchor module regularly gets the latest logs that haven't been uploaded to IPFS from the database, and then packs these logs into a big log file. If there are some newly generated log files, Anchor will upload these log files to IPFS. Anchor obtains log file hashes generated by IPFS. Then, Anchor call the smart contract's algorithm 1 to create an index for log file and store file hash into blockchain by sending a transaction. The process of log file hash to blockchain as shown in Figure 4.

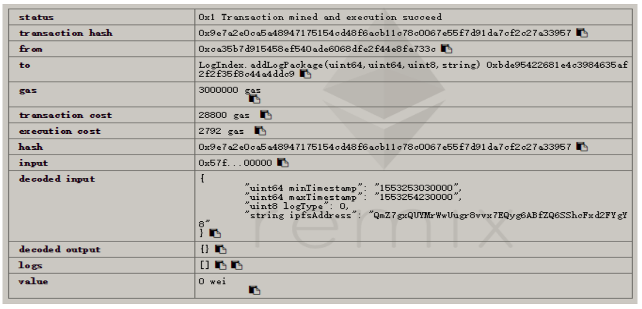
Algorithm 2 Get data index accordint to timestamp

```
Require:
    The timestamp of log file that user want to search is
    $logTimestamp$;
Ensure: $logTimestamp > startTimestamp$
 1: $indexNum = (logTimestamp - startTimestamp)/dt$
 2: if $indexNum > indexArrayLength$ then
 3:     return -1
 4: end if
 5: return indexArray[indexNum]
```

The backend's Query is responsible for receiving the client's query request and returning the result to client. Query call Algorithm 2 to get log index. When getting log index, it uses the index to get concrete file hash and then get content of file according to hash.

B. Testing and Validation

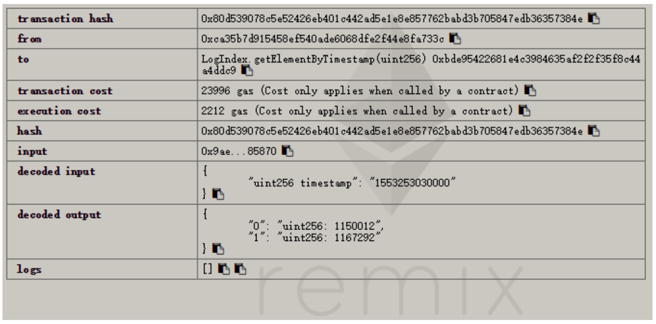
We utilized Remix IDE to deploy and test our developed smart contract code. We could easily verify the correctness of the smart contract we developed through testing. The hash of the log file was stored in blockchain by sending a transaction. Thus, getting the numbers of block which contains those transactions is very important. In this section, we mainly introduce the test results of creating indexes and getting block numbers.



status	0x1 Transaction mined and execution succeed
transaction hash	0x9e7a2e0ca5a48947175154cd48f6acb11c78c0067e55f7d91da7cf2c27a33957
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	LogIndex.addLogPackage(uint64,uint64,uint8,string) 0xbde95422681e4c3984635af2f2f35f8c44a4ddc9
gas	3000000 gas
transaction cost	28800 gas
execution cost	2792 gas
hash	0x9e7a2e0ca5a48947175154cd48f6acb11c78c0067e55f7d91da7cf2c27a33957
input	0x57f...00000
decoded input	{ "uint64 minTimestamp": "1553253030000", "uint64 maxTimestamp": "1553254230000", "uint8 logType": 0, "string ipfsAddress": "QmZ7gxQUYMrWwUugr8vvx7EQyg6ABfZQ6SShcFxd2FYgY8" }
decoded output	{}
logs	[]
value	0 wei

Figure 5 The result of adding new log file.

We called algorithm 1 by sending a transaction. We can see the timestamp and hash of log file as input to the transaction, which will be packaged into the blockchain with this transaction, and the transaction will trigger the smart contract to create an index for log file. The result of adding a new log file is as shown in Figure 5.



transaction hash	0x80d539078c5e52426eb401c442ad5e1e8e857762babd3b705847edb36357384e
from	0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to	LogIndex.getElementByTimestamp(uint256) 0xbde95422681e4c3984635af2f2f35f8c44a4ddc9
transaction cost	23996 gas (Cost only applies when called by a contract)
execution cost	2212 gas (Cost only applies when called by a contract)
hash	0x80d539078c5e52426eb401c442ad5e1e8e857762babd3b705847edb36357384e
input	0x9ae...85870
decoded input	{ "uint256 timestamp": "1553253030000" }
decoded output	{ "0": "uint256: 1150012", "1": "uint256: 1167292" }
logs	[]

Figure 6 The result of getting index.

Algorithm 2 are called by sending a transaction. And then we can get the minimum block number and the maximum block number according to the timestamp. If we get block numbers, we can quickly get log file hashes stored in blocks through reading specified blocks. Finally, we can get the content of log files using concrete hashes. The result of getting an index is as shown in Figure 6.

TABLE I. COMPARISON BETWEEN TRADITIONAL LOG SYSTEM AND THE PROPOSED SOLUTION

Items	Traditional System	Our Blockchain-based System
Data Security	Low	High
Data Authenticity	Low	High
Query Speed	High	High

Due to the page length limit, we only presented partial experimental results. Finally, we also compared our solution to the traditional log system. Table I showed the comparison between conventional log system and our proposed blockchain-based solution. Since we used IPFS to store log files and the blockchain to store file hashes, we could take advantage of the tamper-proof nature of IPFS and blockchain to ensure data security and integrity. Moreover, we used smart contracts to create indexes for log files to improve query efficiency, so our query speed is also fast.

V. CONCLUSION

In this paper, we proposed a blockchain-based framework for secure log storage. As we know, the cost of storing data in a blockchain is very high, so the paper adopted IPFS to store log files, the hash of log file generated by IPFS was stored in the blockchain. After comparing various consensus algorithms, the paper adopted the PoA algorithm as our consensus algorithm. Then, the paper utilized the IPFS and Ethereum smart contract to implement prototype. What is more, the paper gave a detailed introduction in terms of system architecture, algorithm, smart contract and testing. Finally, the paper also provided complete smart contract code. As future work, we plan to encrypt log files using Elliptic Curve Digital Signature Algorithm(ECDSA) before uploading to IPFS. Thus, even if the log file on IPFS is leaked, the log content can't be obtained. Our proposed solution can securely store large amounts of log files and provide efficient retrieval, so we think that our proposed solution can be used in more scenarios such as product traceability.

REFERENCES

[1] Accorsi R. Log data as digital evidence: What secure logging protocols have to offer?[C]//2009 33rd Annual IEEE International Computer Software and Applications Conference. IEEE, 2009, 2: 398-403.

[2] Pourmajidi W, Miranskyy A. Logchain: Blockchain-assisted Log Storage[C]//2018 IEEE 11th International Conference on Cloud Computing (CLOUD). IEEE, 2018: 978-982.

[3] Truong N B, Sun K, Lee G M, et al. GDPR-Compliant Personal Data Management: A Blockchain-based Solution[J]. arXiv preprint arXiv:1904.03038, 2019.

[4] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. 2008.

[5] Rabah K. Convergence of AI, IoT, big data and blockchain: a review[J]. The Lake Institute Journal, 2018, 1(1): 1-18.

[6] Crosby M, Pattanayak P, Verma S, et al. Blockchain technology: Beyond bitcoin[J]. Applied Innovation, 2016, 2(6-10): 71.

[7] Truong N B, Um T W, Zhou B, et al. Strengthening the blockchain-based internet of value with trust[C]//2018 IEEE International Conference on Communications (ICC). IEEE, 2018: 1-7.

[8] Buterin V. A next-generation smart contract and decentralized application platform[J]. white paper, 2014, 3: 37.

[9] Kosba A, Miller A, Shi E, et al. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts[C]//2016 IEEE symposium on security and privacy (SP). IEEE, 2016: 839-858.

[10] Benet J. Ipfs-content addressed, versioned, p2p file system[J]. arXiv preprint arXiv:1407.3561, 2014.

[11] Peterson W W, Brown D T. Cyclic codes for error detection[J]. Proceedings of the IRE, 1961, 49(1): 228-235.

[12] Cohen F. A cryptographic checksum for integrity protection[J]. Computers & Security, 1987, 6(6): 505-510.

[13] Sivathanu G, Wright C P, Zadok E. Enhancing file system integrity through checksums[R]. Technical Report FSL-04-04, Computer Science Department, Stony Brook University, 2004.

[14] Eastlake D, Jones P. US secure hash algorithm 1 (SHA1)[J]. 2001.

[15] Sharma S. A strongly trusted integrity preservance based security framework for critical information storage over cloud platform[J]. databases, 2016, 11(6).

[16] Bharathi P, Rajashree S. Secure file access solution for public cloud storage[C]//International Conference on Information Communication and Embedded Systems (ICICES2014). IEEE, 2014: 1-5.