Journal of Computer Science

Original Research Paper

A Blockchain-Enabled Scalable Network Log Management System

Mohammad Habibullah Rakib, Showkot Hossain, Mosarrat Jahan and Upama Kabir

Department of Computer Science and Engineering, University of Dhaka, Bangladesh

Article history
Received: 15-02-2022
Revised: 27-03-2022
Accepted: 06-04-2022

Corresponding Author:
Mosarrat Jahan
Department of Computer Science and Engineering, University of Dhaka, Bangladesh
Email: mosarratjahan@cse.du.ac.bd

Abstract: Log data is an essential tool to identify the footprint of unauthorized activities executed in a network system. Hence, a compact storage mechanism is required for the massive volume of log data to protect them from malicious tampering attacks. In this regard, Blockchain (BC) has been used to design tamper-proof storage of log records. However, the existing BC-based solutions cannot efficiently handle continuously growing massive log data, creating tremendous storage overhead on the participating BC nodes. Although some works address the storage scalability issue through separate off-chain storage, these works cannot support log data confidentiality and essential query mechanisms to manage log data are missing. Moreover, due to inadequate analysis of the real-time implementation, the performance gain obtained by these schemes is not clearly understood. To handle these deficiencies, we propose a BC-based network log data storage and management scheme that uses an InterPlanetary File System (IPFS) to outsource most of the log data to external off-chain storage. In addition, the proposed scheme performs query and audit operations to manage plaintext and encrypted log records efficiently. Besides, we present a theoretical analysis to show our scheme's scalability in storage gain. Extensive experiments on the prototype implementation of the proposed system show that storage gain increases exponentially with increasing log records per transaction. Moreover, our scheme attains nearly 93% storage reduction in supporting per day storage demand of log records. The experimental results also demonstrate that the proposed system can be realized with a low computational overhead.

Keywords: Blockchain, Distributed Computing, Off-Chaining, Log Management System

Introduction

Nowadays, the cyber-physical system has evolved into a reality due to the onset of several leading-edge technologies such as the cloud, Internet of Things (IoT), Software-Defined Network (SDN), 5G cellular technology and blockchain. It eases our life by automating almost every operation of our day-to-day life. Therefore, the accurate operation of a cyber-physical system is essential to ensure the safety and security of human life. Due to continuous internet connectivity, cyber-physical systems constantly face various security threats. Often patches are not readily available as newer attacks are emerging every day (Poolsappasit *et al*., 2011). Besides, incorporating patches into the running system is difficult (Poolsappasit *et al*., 2011). In this respect, log data analysis performs a vital role in guaranteeing the security of cyber-physical systems.

Different components of a cyber-physical system such as servers, firewalls, routers, switches and individual PCs create log records when executing various operations. These log records are a valuable source of information for tracking the activities conducted in a system (Accorsi, 2009). Hence, they can be used to audit different events in the network and detect and troubleshoot different improper activities that hinder the network's functionalities (Kent and Souppaya, 2006). Consequently, log data is often a target of various cyber attacks that tamper log files to hide the malicious activities conducted within a system. Besides, a lack of stable storage and management platform for massive log data can cause deficiencies in proper network logging and auditing, allowing attackers to hide their footprint and an attack may go unnoticed. Hence, designing a compact mechanism to store the enormous log data is essential to assure the security of the computing system.

Conventional log management systems utilize the memory capability of the most up-to-date storage technologies and take the support of cloud servers for storing and replicating the massive volume of log data. However, classical systems are a victim of a single point of failure (Ray *et al*., 2013), (Söderström and Moradian, 2013). Besides, ensuring consistency among the multiple copies of data stored in different storages is a problematic issue (Söderström and Moradian, 2013), (Ray *et al*., 2013). Recently Blockchain has received wide acceptance for data decentralization and tamper-proof storage of data (Zheng *et al*., 2018). However, storing a huge volume of system log records directly in the BC creates several performance issues. For example, each node of a BC network stores the complete copy of log records maintained by the BC, imposing significant storage and bandwidth overhead (Bhutta *et al*., 2021), (Zheng *et al*., 2018). Over time the size of BC increases, and the performance degrades significantly. Besides, a new node in the BC needs a significant amount of time to synchronize with the chain state (Miyachi and Mackey, 2021). In literature, the memory scalability issue of BC is handled using off-chain mechanisms (Wang *et al*., 2021), (Miyachi and Mackey, 2021), where most of the data is outsourced to off-chain storage while the digital fingerprint of data is maintained in the BC (on-chain).

Although few works (Pourmajidi and Miranskyy, 2018), (Kumar *et al*., 2018) presented a framework to store log data using BC, they did not address the storage scalability issue created due to storing huge volumes of log records. Besides, these works lack real-time implementation and hence it is not possible to get an indication about the performance of these systems. Moreover, they do not support query mechanisms, an indispensable tool for network log management and do not ensure log data confidentiality. More recently, Rakib *et al*. (2020) partly addressed these shortcomings and proposed efficient BC-based log storage and management scheme with data privacy, query and audit operations. In addition, this study presents a real-time performance analysis of the proposed BC-enabled log management scheme. Nevertheless, this scheme also falls short of managing the massive volume of log data due to the lack of an off-chain mechanism. Some recent works (Huang, 2019), (Rane *et al*., 2021), (Marangappanavar and Kiran, 2020) address the memory scalability issues using IPFS network (Benet, 2014) besides the BC network, where IPFS network plays the role of off-chain storage. However, these works cannot show the performance gain achieved through the use of an off-chain (IPFS) mechanism.

In this study, we supplement the work of (Rakib *et al*., 2020) to devise well-structured and practical log storage and management scheme that can efficiently handle the storage scalability issue and execute query and audit operations on both plaintext and encrypted data. Besides, we implement the proposed system and conduct extensive analysis to evaluate its performance. Specifically, we make the following contributions:

- A reliable storage scheme for network log data that addresses the memory scalability issue by incorporating IPFS network (off-chain) with the BC network
- A practical network log management scheme, supporting query and data auditing operations on the plaintext and encrypted log records
- A theoretical analysis of the storage gain of the proposed scheme over a non-off-chain BC-based log storage scheme
- A prototype implementation and analysis demonstrate that the proposed scheme can be accomplished with low processing and storage overhead. More specifically, our scheme exhibits exponential storage gain with more log records per transaction. Besides, it cuts down the per day log storage requirement by almost 93%.

Related Work

Log records are crucial for investigating malicious activities, detecting policy violations and analyzing system performance issues (Accorsi, 2009). Therefore, the design of a reliable storage and management scheme for log data is explored extensively in the literature. Olaf and Esmiralda (Söderström and Moradian, 2013) proposed a system where log data are collected through installed agents and stored in a centralized server. However, this scheme does not support a query mechanism and is subject to a single point of failure due to a centralized server. Besides, (Ray *et al*., 2013) explored the challenges of a cloud-based log management service and proposed a mechanism for storing log records in a cloud environment. However, this scheme does not guarantee data consistency and does not prevent illegal access as the cloud server is semi-trusted.

Recently BC has been used to store unblemished log data. For example, (Pourmajidi and Miranskyy, 2018) proposed Log chain, uncontaminated log storage for cloud service providers built upon BC. This scheme divides data into various segments and associates each piece with a block in a higher-level BC known as super chain. The authors also developed a preliminary prototype of their scheme, which is immature for production implementation as the APIs are not implemented.

Besides, Louis and Erez (Shekhtman and Waisbard, 2019) proposed Engrave Chain to secure log data using the natural properties of BC. The authors implemented a proof-of-concept using Hyperledger Fabric. However, this work does not support query operations and does not provide a performance analysis of the proposed scheme. In addition, (Liang *et al*., 2017) presented Prov Chain, a BC-based framework for accumulating and confirming cloud data provenance by mapping provenance data to BC transactions. Alongside, (Park *et al*., 2017) proposed a mechanism for data logging while ensuring the integrity of data through the use of BC. Although the authors presented a theoretical analysis of performance evaluation, the data collection interval is not practical and privacy of log data is not ensured. Moreover, (Ali *et al*., 2021) proposed a BC-enabled log management framework for a cloud computing environment that manages immutable log records of the fog networks. The gateway responsible for each fog network stores the filtered log records in the BC network. However, the compromise of the gateway node can damage the log data of an entire fog network. Moreover, this scheme is unsuitable for handling the enormous volume of log data generated by the edge devices due to the lack of off-chain mechanisms. Recently (Rakib *et al*., 2020) proposed a network log management scheme that supports data privacy, query and audit operations on the plaintext and encrypted data stored in the BC. However, this scheme suffers for storing a vast volume of data as it does not consider off-chaining log records. In our work, we enhance the work of (Rakib *et al*., 2020) to remove the shortcoming of storage issues by incorporating an off-chain mechanism.

The requirement of storing a large volume of data in BC creates memory scalability issues, hampering the extensive deployment of BC. To overcome this shortcoming, Jabarulla and Lee (2021) proposed a proof-of-concept design for a distributed patient-centric medical record management system that stores and shares medical images using BC and IPFS networks. This scheme uses the IPFS network as off-chain storage to outsource most BC data. Although the authors illustrated the performance of the proposed scheme using various parameters, the performance improvement compared to a non-off-chain BC-based system was not shown. Besides, this scheme does not support query operations on the stored data. Similarly, Kumar and Tripathi (2020) offered a platform for sharing patients' COVID-19 reports using consortium blockchain and IPFS. However, this scheme does not support data privacy and query mechanism and does not provide a comparative analysis between the BC-based system with an off-chaining mechanism and the BC-only system. Moreover, (Rane *et al*., 2021) proposed a BC-based model for tamper-proof storage of log records generated in a cloud service environment. The encrypted log records are stored in the IPFS network while BC keeps the IPFS hash of the log files. Although this scheme supports data confidentiality and audit operation, it does not support query operations. Also, this study did not show the performance gain over a BC-based scheme without an off-chain facility. Similar to the previous works, our scheme proposed a flexible network log management scheme that uses BC and IPFS networks to store log data efficiently. In contrast to the existing works, our scheme supports data confidentiality, query and audit operations. Moreover, we present both a theoretical and implementation-based detailed analysis of the performance gain of the proposed BC-based framework with the off-chain facility over the BC-based system without off-chain provision.

Terminology

In this section, we present a brief discussion on the jargon used in our proposed scheme.

Blockchain

Blockchain (BC) is a constantly expanding distributed database of records known as blocks (Zheng *et al*., 2018). It acts as a distributed ledger to permanently preserve transactions among several parties in a verifiable way. Each participating node in a BC network remains synchronized with other nodes and holds the same copy of BC. They acknowledge the validity of the transactions through the use of Consensus Algorithms such as Proof of Work (PoW), Proof of Stake (PoS) and Practical Byzantine Fault Tolerance (PBFT). A set of BC nodes acts as miners that assembles transactions into blocks and makes them permanent using a hash value. In a BC, the hash value of a previous block is included in the successive block. Hence, a modification in a block alters the hash value, promulgating to the blocks mined subsequently. Therefore, any node can detect data modification and prevent illegal stored data tampering. The first block in a BC is known as the genesis block. It does not have any parent block, so its previous block hash is set to 0. Fig. 1 shows the configuration of a BC.

A BC block consists of a block header and a block body as shown in Fig. 2. The block header contains various information such as version number of a block, timestamp, Merkle tree root hash, earlier block hash and nonce. In contrast, the block body contains a transaction counter and transactions.

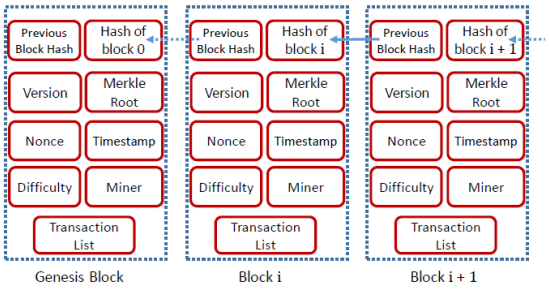

Fig. 1: Architecture of Blockchain

Field	Value
Block Version	003
Timestamp	1625116125
Hash	0472c36b12729a9e2abfba78a126436676c096e47eb07d414a1140f0dec5a4d3
Merkle Root	9b371d3c2dd590b0a060fc71104cd981a5189632aa65196e415f4640a873e8c5
Previous Block Hash	00309a291a5c4c5b12a63a6c46d342fbce4336abf38225fd73fc5a94a8484eb0
Nonce	65
Difficulty	5.96046447753906e-8
Miner	17CQL9CA58xmxejqW8QUqDJ4fCTaMRZQyA9vng
Confirmations	154
Transaction List	
TxID 1	TxID 2 ... ... TxID n-1 TxID n

Fig. 2: Structure of a block

InterPlanetary File System

InterPlanetary File System (IPFS) is a distributed peer to-peer (P2P) system for storing and sharing files (Benet, 2014). Each file stored in IPFS is associated with a unique Content Identifier (CID) used for locating that file in the network. The CID of a file is computed using the unique hash of that file. Often the content of a large file is split into multiple blocks for ease of hosting by peers. IPFS uses the Merkle Directed Acyclic Graphs (DAGs) to link the split content of a file and Distributed Hash Table (DHT) to discover the content in the P2P network.

A file in IPFS is represented by an *IPFS* object, a data structure consists of data and links. Data represents a BLOB of unstructured binary data less than 256 KB while links is an array of link structure containing name, hash and size. A file of size less than 256 KB is represented by an IPFS object where data field contains file content and links is an empty array. In contrast, a file larger than 256 KB is represented by multiple IPFS objects where links field contains a list of links to other IPFS objects into which the content of the file is divided. An empty IPFS object with no data is also created to hold links to all the IPFS objects representing the file. Figure 3 shows the procedure of storing a file greater than 256 KB in an IPFS network.

An IPFS network can be either public or private. In a public IPFS network, every node connected to the network can access data if it knows the CID of the data. In a private IPFS network, only peers with a shared secret key can access data.

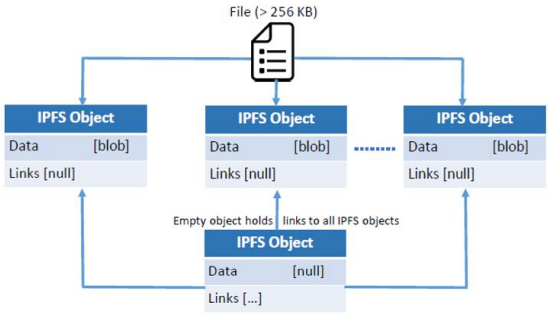

Fig. 3: IPFS file storage mechanism

The advantage of using IPFS as off-chain storage with BC is that it is distributed and decentralized in nature like BC. Besides, IPFS provides immutability and persistence of data along with garbage collection.

MultiChain

MultiChain (Greenspan, 2015a) is an open-source, private BC platform where participants need permission from the network members to connect to the BC. It is less decentralized than public BC and uses a lightweight consensus process to obtain faster transaction processing. This platform is based on a fork of Bitcoin Core, enabling the deployment of private BC within or between organizations (Greenspan, 2015b). We choose MultiChain to realize the BC network in our proposed scheme for the following reasons:

- Abstraction of storage, indexing and retrieval of data through data stream make data processing faster in BC
- Simple and thorough composition of BC parameters
- Easy management of permission for nodes and user friendly API

We utilize the data stream feature of MultiChain to store metadata related to log data in BC, where a *stream* is a group of items, each associated with a transaction. A single transaction can publish one or multiple items in BC, where an item contains data to be stored, publisher address, keys for faster retrieval, transaction ID, etc. A BC node subscribes to a stream to index its items to enable searching by keys. Besides, MultiChain provides a rich set of APIs to allow any modern programming language to interact with the BC.

System Model

Our proposed system model comprises four entities, the BC network, an IPFS private network, participating nodes and auditors. Fig. 4 shows a pictorial representation of our system model.

The BC network is a distributed peer-to-peer network for storing data in an unmodifiable form. It ensures transparency among the participating nodes through data verifiability. In our scheme, BC network is used to store metadata about the system log data.

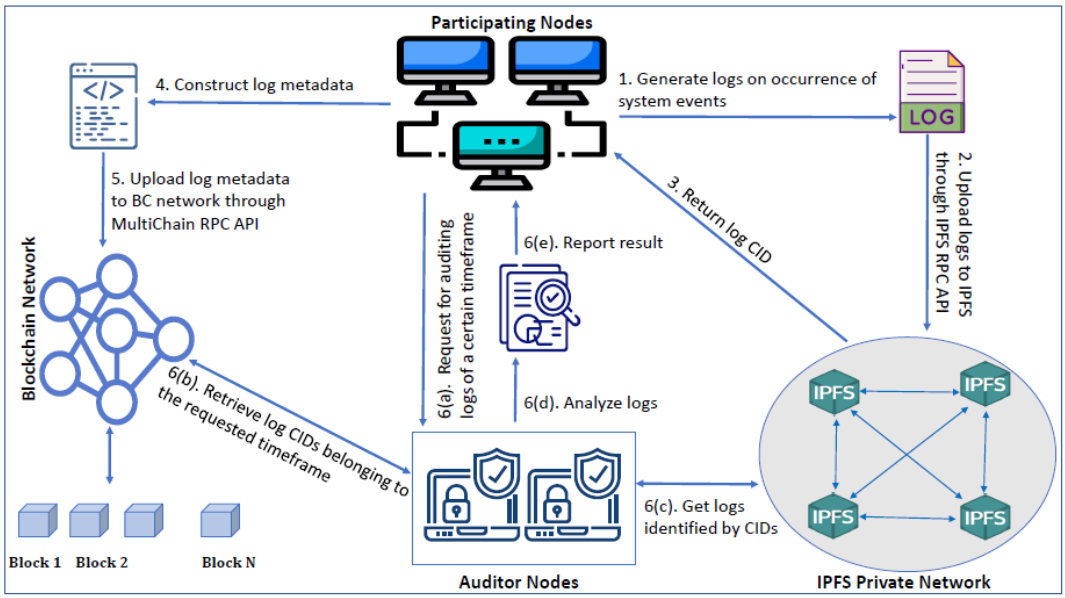

Fig. 4: System model of the proposed scheme

An IPFS private network is used as off-chain storage in our proposed scheme. It stores the actual log records in plaintext or an encrypted form. Data stored in this network are only accessible to the member nodes of this network.

Participating nodes are Personal Computers (PCs) that are part of the IT infrastructure of an organization. These PCs create log entries to preserve the details of each operation and Quality of Service (QoS) measures related to the operation of software applications, servers, network components, etc. They periodically upload encrypted or plaintext log data to the private IPFS network for storage. Besides, they interact with both the BC network and IPFS network using appropriate Remote Procedure Call (RPC) API. Each *participating node* maps to a unique blockchain address to track its published log records. When a node uploads a log file to the IPFS network, the CID of the file along with the blockchain peer address of that node is stored in BC. This ensures mapping between blockchain data and IPFS data for a particular node.

Auditors are responsible for auditing log files and reporting discovered inconsistencies. Upon receiving a request from a participating node, they retrieve log files from the *IPFS* network using CID obtained from the BC network. They analyze the retrieved log files using traditional log analysis tools such as Papertrail (2021), Loggly (2021) and Splunk (2021) to report any threat. In case of encrypted log files, encryption keys are shared with the auditors via the BC network.

Working Principle

This section presents the detailed working principle of the proposed scheme with an elaborate discussion on each functional feature.

Publication of Log Records

Participating nodes upload log records periodically and metadata to the IPFS private network and BC network, respectively, where both networks run in the cloud. Before interacting with both networks, each participating node must authenticate and map to an IPFS peer address and BC node address separately. At first, a participating node uploads a file to the IPFS network and receives the corresponding CID. It then immediately constructs the file metadata object and publishes it to BC. The BC network contains an entry for the BC node address, the CID and metadata to link with the corresponding log file in the IPFS network. The time interval between two successive publish operations is determined by the attack probability of the network (Hajizadeh *et al*., 2018). In reality, the time interval should be compact enough to reduce the possibility of attackers modifying the log data before being published to the IPFS private network. The content of metadata is Table 1.

Table 1: Metadata of Log Records

Field	Value
File_ID	Syslog
IPFS_CID	QmP3jiWmaViXJW1c7ZyQ6ntdcwADe5qAksVGUsdwbD3Gxo
Size	12917
Publisher	1Y8VTgooMfiAvgoQc51TpsRu7jCGoUEJsEVrR6
Start_UTC	1613274426
End_UTC	1613274431
Publish_time	1613274433
Encryption_status	Yes

In our proposed scheme, a participating node can choose to publish encrypted or plaintext log data to IPFS.

Retrieval of Log Records

Participating nodes retrieve log data from the IPFS network using a web interface. This process starts with inputting a time frame encompassing the desired log records from the participating node. Our scheme then uses appropriate BC API (e.g., *liststreampublishersitems*) to retrieve transactions containing log metadata of that particular participating node. If the *start* and *end* timestamp in metadata are within the desired time frame, the IPFS CID included in the metadata is added to the list of desired CIDs. Finally, our scheme fetches all original log records referenced by the list of CIDs from the IPFS private network using appropriate IPFS API.

Publication and Retrieval of Confidential Log Data

Storing plaintext log data in the IPFS network compromises privacy since data is visible to every peer in the network. A participating node may not share its data with its peers in the same or different organization. Hence, a participating node can encrypt its data before uploading it to the IPFS network and share the encryption key securely via the BC network with the suitable recipient nodes.

We use *Advanced Encryption Standard (AES)* (Daemen and Rijmen, 2001) to support encryption and decryption operation and public-key cryptography algorithm *RSA* (Rivest *et al*., 1978) to share keys. Our scheme maintains two separate streams in the BC network for sharing the RSA public key and AES secret key. The AES secret key and RSA key length are 256 bit and 2048 bit, respectively. Each participating node generates its RSA public-private key pair, stores the private key locally and publishes the public key on BC. The public key is labeled with the node's address for easy retrieval by other nodes. Whenever a node wants to share an encryption key, it retrieves the recipient's public key and encrypts the secret key using that public key. It then publishes the encrypted secret key on BC by labeling it with the IPFS CID of the file and the recipient node's address. The recipient participating node can now easily find the secret key on BC. After retrieval, it decrypts the encrypted secret key using its private key and decrypts a file using the retrieved secret key.

Consensus Process

Our proposed system uses permission-based mining supported by the MultiChain framework to add transactions into blocks. In this scheme, a set of nodes with mine permission granted by the admin nodes acts as miners. A mine permission is allocated when a specific portion (e.g., admin-consensus-mine parameter) of admin nodes reach consensus in doing so. Admin nodes use a BC parameter known *as* mining diversity, taking a value between 0 and 1, to prevent monopoly in the mining process. A node with mine permission must wait for $\lfloor$*mining diversity* × *active miners*$\rfloor$ blocks to be mined by other nodes before it gets a chance to mine another block.

Admin Node Selection[*]

Admin nodes are super users having every permission to control the entire BC network and need to be changed frequently to prevent a single authoritarian in the network and distribute workload equally to every node in the BC. Hence, we propose the Algorithm 1 (Rakib *et al*., 2020) to periodically rotate the admin permission among nodes in the BC network. Every admin node in the BC network executes this algorithm:

- There are $\lceil n \times p \rceil$ admin nodes where n is the total number of nodes in the BC network and p is a value between $0 < p \leq 1$
- *list* is a sorted list of admin nodes as reported by the time of being an admin. Besides, next denotes the succeeding admin node to enter list
- Each node in the list allocates admin permission to the next and abolishes admin permission from the first k nodes in the *list.* For example, $k = 1$ revokes admin permission from first node of *list* and grants that to the $(\lceil n \times p \rceil+1)$-th node after current admin nodes finish the execution of Algorithm 1.

Algorithm 1: Algorithm for selecting admin nodes

Require:
n: number of nodes
p: proportion of nodes that can be admins
k: number of admins to be swapped

$N \leftarrow \lceil n \times p \rceil$ ▷ N = number of admins
if *Clock*() == *ElectionTime* **then**
  list = sorted list of current admins based on time
  for $i \leftarrow 0$ to $k-1$ **do**
    remove admin permission from *list*[*i*]
    next = (*i*+*N*) mod *n*
    assign admin permission to node *next*
  end for
end if

In case one or more admin nodes do not grant *admin* permission to next, our scheme uses MultiChain's consensus process where *admin-consensus-admin*

[*]© 2020 IEEE. Reprinted, with permission, from Rakib *et al*. (2020).

parameter determines the number of admin nodes required to reach a consensus.

Query Mechanisms

Our proposed scheme performs query operation on both plaintext and encrypted log records (after decryption). We develop an integrated approach to support queries on both data types by linking MultiChain's metadata with the original log records stored in the IPFS private network.

Our scheme stores the timestamp of the earliest and latest log records in a log file in the metadata information associated with that file in the BC (Table 1). To retrieve log records of interest, our scheme obtains metadata from the BC using the timestamp associated with those log records that ultimately reveals the CID of the desired log records in the IPFS network. After retrieving log records from the IPFS network, a user decrypts the data in the local machine using the correct decryption key recovered from the BC if the data is encrypted. Here, a query is *single-phase* query if it uses only the timestamp, otherwise *two-phase* query if the user inquires on other fields of the log records recovered from the single-phase query.

Algorithm 2: Algorithm to execute timestamp point query

Require:
S: name of a stream
T: a particular date or timestamp
P: owner of log records
1: *items* ← *liststreamkeyitems(S, T)*
2: *cid_list* ←Ø
3: **for** *item in items* **do**
4: **if** *item["Publisher"] == P* **then**
5: *cid_list* ← *cid_list* ∪ *item["IPFS_CID"]*
6: **end if**
7: **end for**
8: *logs* ←Ø
9: **for** *cid in cid_list* **do**
10: *logs* ← *logs* ∪ *IPFS_API.get(cid)*
11: **end for**
12: **return** *logs*

Single-Phase Query

A single-phase query can be of two types. They are:

- **Timestamp Point Query:** Algorithm 2 describes the procedure of retrieving log records associated with a particular date or timestamp *T* for a specific publisher or owner *P*. Our scheme first fetches metadata associated with *T* from the BC network using *liststreamkeyitems* API of MultiChain, running in $\mathcal{O}(\log m)$ time, where *m* is the number of items fetched from BC (Greenspan, 2015b). Our scheme further retrieves CID associated with a particular owner *P* from the retrieved metadata. Finally, log records associated with *P* for a particular date or timestamp *T* is fetched from the IPFS network. The execution time (i) for plaintext log records turns into $\mathcal{O}(\log m) + r$ and (ii) for encrypted log records is $\mathcal{O}(\log m) + r + e$ where *r* is the file retrieval time from IPFS using *IPFS_API.get(cid)*, *e* is the time complexity for AES decryption operation measured following (Daemen and Rijmen, 2001). Here to note that running the loop in line 9 ~ 11 in parallel contributes *r* to the overall execution time, while introducing parallelism in the loop in line 3 ~ 7 generate a negligible constant overhead.
- **Timestamp Range Query*:** Algorithm 3 (Rakib *et al*., 2020) describes the procedure of searching records in a specified time range. Our scheme considers the timestamp range query as a collection of a timestamp point query for each timestamp between the start time and end time of that range. The execution time of the algorithm (i) for plaintext log records is $\mathcal{O}(R \log m) \approx \mathcal{O}(\log m)$ where *R* denotes the time range and different timestamp queries within the range *R* execute in parallel and (ii) for encrypted log records is $\mathcal{O}(R \log m) + e \approx \mathcal{O}(\log m) + e$.

Algorithm 3: Algorithm to execute timestamp range query

Require:
S: name of a stream
T_{start}: start time
T_{end}: end time
P: owner of log records
1: *logs* ←Ø
2: **for** *t in range* (T_{start}, T_{end}) **do**
3: *logs* → *logs* ∪ *timestamp_point_query*(*S, t, P*)
4: **end for**
5: **return** *logs*

Two-Phase Query*

Two-phase queries carry out further queries on the records fetched from a single-phase query based on different fields of log records. There are three types of queries. They are:

*© 2020 IEEE. Reprinted, with permission, from Rakib *et al*. (2020).

- **Single Key Query:** Our scheme considers each field of a log record as a key and searches the log records obtained from a single-phase query using getitems () as shown in Algorithm 4 (Rakib *et al*., 2020). The execution time of the two-phase query (i) for plaintext log records is $\mathcal{O}(log\ m) + \mathcal{O}(ql)$, where $\mathcal{O}(log\ m)$ is the time complexity of a single-phase query and $\mathcal{O}(ql)$ is the execution time of a single key query where q is the number of records in D and l is the average length of a log record and (ii) for encrypted log records is $\mathcal{O}(log\ m) + e + \mathcal{O}(ql)$

Algorithm 4: Algorithm to execute single key query

Require:
D: Log records fetched from single-phase query
K: Key
1: *list_rec* ← *getitems(D, K)*
2: **return** *list_rec*

- **Multiple Key AND Query:** In our scheme, a user can also perform query using multiple keys. Algorithm 5 (Rakib *et al*., 2020) shows that for each key in l_k, a single key query is executed. Finally, the query result is obtained by taking the intersection of every single key query performed using keys in l_k. The overall running time complexity of the two-phase query (i) for plaintext log records is $\mathcal{O}(log\ m) + \mathcal{O}(Lql) \approx \mathcal{O}(log\ m) + \mathcal{O}(ql)$ and (ii) for the encrypted log records is $\mathcal{O}(log\ m) + e + \mathcal{O}(ql)$, where $\mathcal{O}(ql)$ is the execution time of a single key query, L is the length of l_k and the queries using multiple keys L run in parallel.

Algorithm 5: Algorithm to execute multiple key AND query.

Require:
D: log records obtained from single-phase query
l_k: list of keys
1: $L \leftarrow |\ l_k\ |$
2: *list_rec* ← *single key query*(*D*, l_k[0])
3: **for** $i \leftarrow 1$ to *L-1* **do**
4: *list_rec* ← *single_key_query*(*list_rec*, $l_k[i]$)
5: **end for**
6: **return** *list_rec*

- **Multiple Key OR Query:** This query also allows a user to perform queries using different keys. Similar to the multiple key AND query, this query also performs a single key query on different keys. It differs from *multiple key AND query* by taking the union of the results of the single key query performed on all keys. The execution time of multiple key OR query is the same as multiple key AND query.

Audit Log Files

Any modification in a log file stored in the IPFS network changes the CID of that file. However, analysis of log files stored in the IPFS network is required for detecting threats in an IT infrastructure. In the proposed scheme, auditor nodes serve this purpose. They can access both the IPFS private network and the BC network. A participating node may request an auditor to verify log data stored in a specific timeframe. The auditor queries the BC using the timestamp range query to retrieve the CIDs of the log files and then fetches original logs from the IPFS network. If the log data is encrypted, AES secret key is shared with the auditor. Auditors analyze log files using traditional log analysis tools and report any threat to the participating nodes. In our scheme, several auditors work concurrently to decide the validity of log files. A decision is taken by a participating node when more than 51% auditors support that decision, similar to consensus algorithms in BC (Zheng *et al*., 2018).

Theoretical Analysis of Memory Scalability

Our proposed scheme distributes data between two networks where BC stores log metadata and the IPFS network stores actual log records. Here, the IPFS private network works as an off-chain storage layer to our system. With IPFS storage, we retain log records as long as we need and get them removed by garbage collector service in IPFS when data are not required anymore. This approach keeps storage requirements within the capacity; thus, reasonable usage of the storage is ensured and memory scalability is attained.

To perform the theoretical analysis on the memory scalability of the proposed scheme, we consider two scenarios, one with IPFS storage and the other without IPFS storage. We introduce some notations for this purpose in Table 2.

Table 2: List of notations

Symbol	Description
$\mathcal{N}$	Number of participating nodes
$\mathcal{B}$	Size of a block
$\mathcal{T}$	Time interval for periodic collection of log records
$\mathcal{S}_{tx}$	Size of a transaction
$\mathcal{L}_r$	Average length of a log record
$\mathcal{N}_r$	Average number of log records generated within $\mathcal{T}$
$\mathcal{S}_m$	Size of metadata for log records within $\mathcal{T}$
$\mathcal{N}_{tx}$	Number of transactions in a block

Blockchain size without IPFS (off-chain) storage:

It can be seen that the size of a transaction is

$$\mathcal{S}_{tx} = \mathcal{L}_r \times \mathcal{N}_r \quad (1)$$

where, $\mathscr{S}_r$ and $\mathscr{N}_r$ are the average length of a log record and the average number of log records generated in $\mathscr{T}$, respectively. We assume that all the log records generated by a participating node in $\mathscr{T}$ are accumulated in a single BC transaction.

Total amount of storage required per Transactions per Second (TPS) is

$$\mathscr{S}_{tps} = \frac{\mathscr{S}_{tx} \times \mathscr{N}}{\mathscr{T}} \quad (2)$$

where, $\mathscr{N}$ is the number of BC nodes.

Total size of a BC is

$$\mathscr{S}_{without\_off-chain} = \frac{\mathscr{S}_{tx} \times \mathscr{N}}{\mathscr{N}_{tx}} \times \mathscr{B} \quad (3)$$

Blockchain size with IPFS (off-chain) storage:

This scheme keeps only the metadata $\mathscr{S}_m$ in the BC and outsources the remaining data to the IPFS network. Hence, the size of a transaction is

$$\mathscr{S}_{tx} = \mathscr{S}_m \quad (4)$$

Total amount of storage required per Transaction per Second (TPS) is

$$\mathscr{S}_{tps} = \frac{\mathscr{S}_m \times \mathscr{N}}{\mathscr{T}} \quad (5)$$

Total size of a BC with an off-chain storage is

$$\mathscr{S}_{off-chain} = \frac{\mathscr{S}_m \times \mathscr{N}}{\mathscr{N}_{tx}} \times \mathscr{B} \quad (6)$$

Since, $\mathscr{S}_m$ is much smaller than $\mathscr{S}_{tx}$, which gives $\mathscr{S}_{off-chain} << \mathscr{S}_{without\_off-chain}$.

$$Storage\ gain = \frac{\mathscr{S}_{without\_off-chain} - \mathscr{S}_{off-chain}}{\mathscr{S}_{without\_off-chain}} \times 100\% \quad (7)$$

Experimental Evaluation

In this section, we discuss the experimental setup, followed by a detailed analysis of the performance of the proposed scheme.

Experimental Setup

We implemented the proposed system using MultiChain community edition, version 2.1.2 (Greenspan, 2015b) and IPFS version 0.7.0 (Benet, 2014). We set up both the BC network and IPFS private network on virtual machines running on Microsoft Azure cloud platform (Azure, 2021). Besides, we used Python IPFS HTTP Client (Python IPFS Client, 2021) and MultiChain JSON-RPC API (Python MultiChain Client, 2020) to interact with the IPFS private network and BC network, respectively. We also developed a web interface to enable easier interaction between the participating nodes and the proposed system.

We ran the experiments on computers possessing Intel Core i7 processor, 16 GB RAM and 1 terabytes HDD and running Ubuntu operating system 18.04. Besides, we used Syslog data (Gerhards and GmbH, 2009) to obtain log records for our experiments. In addition, to support encrypted publication of log data, we used AES-256 (Daemen and Rijmen, 2001) to perform encryption operation.

We compared the performance of the proposed BC-based log management scheme associated with an off-chain mechanism (BC with off-chain) with Rakib *et al*. (2020)'s BC-only log management scheme (BC-only) (Rakib *et al*., 2020).

Performance Evaluation

We considered processing time as the performance metric to depict the efficiency of various operations of the proposed scheme. Besides, we considered storage requirements to narrate the performance gain obtained through off-chain storage. The results shown in different graphs were taken as mean values over 50 samples.

Record Extraction and Encryption*

Before uploading data to the BC and IPFS network, the proposed scheme spends time locating and retrieving log records that are part of a given time frame. We denote this time as record extraction time. It depends on the log records' placement in the file and the size of log records' chunk, which relies on the amount of data collected between two subsequent publish operations (Rakib *et al*., 2020). To run the experiment, we consider chunk size up to 4 MB as it is the default maximum size of a transaction in MultiChain (Greenspan, 2015b).

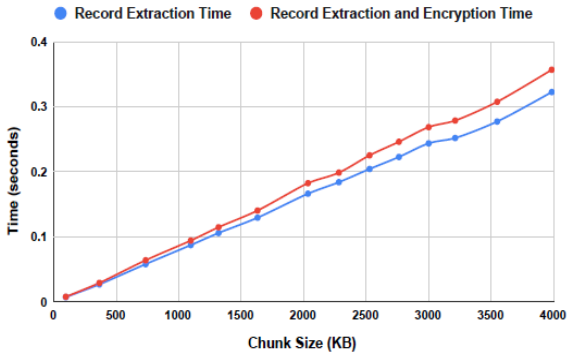

Fig. 5: Log records extraction and encryption time
© 2020 IEEE. Reprinted, with permission, from Rakib *et al*. (2020).

*© 2020 IEEE. Reprinted, with permission, from Rakib *et al*. (2020).

Figure 5 shows that log record extraction time grows linearly with the chunk size. The time to retrieve a 4 MB chunk is nearly 0.32 seconds while the time to fetch and encrypt the same size chunk is a bit more than 0.35 seconds due to encryption operation. Tiny discrepancies are noticed in the graph due to the position of chunks in the log files.

Publish and Retrieval Operation

Throughout a publish operation, a participating node uploads a chunk of log records to the IPFS private network and publishes metadata about the log records in the BC network. Later on, to retrieve the log records, a computing node first accesses the BC network to obtain the corresponding CIDs and uses those CIDs to access log files from the IPFS network. As shown in Fig. 6 and 7, publish and retrieval time increase almost linearly with chunk size. Besides, publish time is slightly higher than retrieval time as a publish operation consists of multiple steps such as IPFS nodes splitting data into multiple chunks, calculating their hash and publishing metadata (i.e., publisher, timestamp, etc.) to the BC network. The proposed scheme takes about 0.25 sec to publish and 0.125 sec to retrieve a chunk of 4 MB. We observed that the proposed BC-based off-chain scheme took more time than the BC-only (onchain) scheme for publish and retrieval operations. Although data uploading and retrieval from the IPFS network are faster than that of BC, overall performance slightly degrades compared to the BC-only approach for the construction and searching of metadata during publishing and retrieval operation, respectively. Our experiment found that the publish and retrieval operations had an average degradation of approximately 26% compared to the BC-only scheme because of internet connectivity. A faster and more stable internet connection can improve performance and reduce the performance gap with the BC-only scheme.

Query Operation

To perform the query operation, we first fetched the CIDs of the log files from the BC and then used those CIDs to retrieve data from the IPFS network. We then performed search operations on those data depending on various attributes to obtain the desired query results. Hence, query time considers the time to fetch log records, decryption time (if required) and search time on the obtained log records. Although an individual log record can possess various attributes such as appname, hostname, timestamp and PID, we selected the six most frequently used attributes K to perform query operations in our experiments.

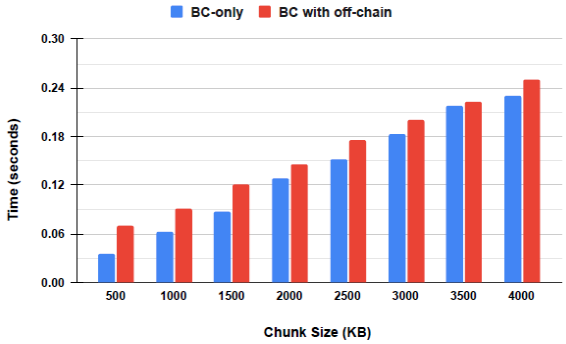

Fig. 6: Processing time of publish operation

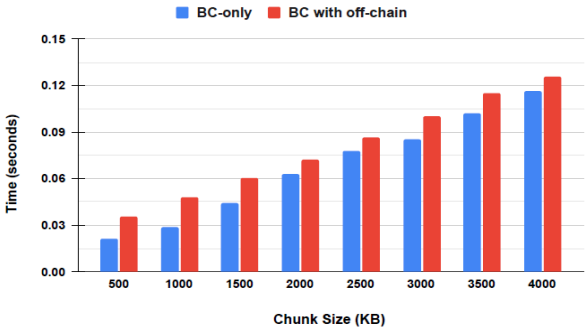

Fig. 7: Processing time of retrieval operation

Figure 8 shows the processing time of different queries on plaintext data for both the proposed BC with an off-chain mechanism and BC-only scheme. It is clear from the experimental data that the *multiple key AND query* is constant for different values of K in the proposed scheme and BC-only scheme. In the proposed scheme, the *multiple key AND query* attains nearly 19% more processing overhead compared to the BC-only scheme, mainly due to the retrieval operation. In contrast, the processing time of *multiple key OR query* grows with the higher values of K in both schemes. This is due to the OR query searches more records compared to the AND query for the inherent properties of query operations as discussed in Subsection Query Mechanisms. The *multiple key OR-query* incurs approximately 8% more processing time in the proposed scheme compared to the BC-only scheme.

Figure 9 shows that the processing time of a query on plaintext data relies on the number of log records N retrieved from the IPFS network when $K = 6$. It increases linearly as the retrieval, decryption and search time are proportional to N in both schemes. As shown in Fig. 9, the *multiple key AND query* and *multiple key OR query* incur nearly 11% and 12% more processing time, respectively in the proposed scheme compared to the BC-only scheme at $N = 300000$ and $K = 6$. The performance gap can be reduced if a faster and more stable internet connection is used.

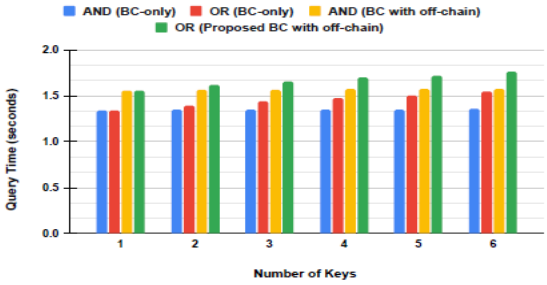

Fig. 8: Query time for different number of keys when N = 300000

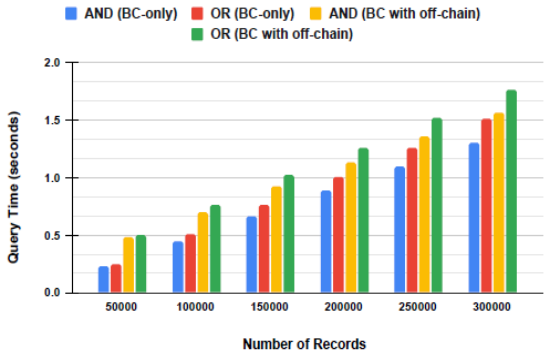

Fig. 9: Query time for different number of records when K = 6

The graphs for query operations on encrypted log records exhibit the same trend depicted in Fig. 8 and 9 except the fact that decryption operation adds approximately 20% more processing time. We do not show the graphs for query operations on encrypted data here for the compact presentation of the paper.

Storage Gain for IPFS Storage

This section highlights the storage gain of the proposed scheme for using IPFS (off-chain) storage. To evaluate the effect of off-chain storage, we varied the number of log records per transaction up to 100 and measured storage requirements for both our proposed and BC-only schemes. Then using the formula in Eq. 7, we measured the storage gain. The size of metadata in our system, as shown in Table 1 is approximately 250 bytes and we found the average size of a log record to be around 150 bytes. Here to mention that we consider storage gain for a single participating node for simplicity of computation. From Fig. 10 we find an exponential increase in the storage gain as we publish more records in a single transaction. It is noticeable that the storage gain increases rapidly for lower values of log records/transactions (<60). It approaches almost 100% when log records/transaction = 60 and after that, the storage gain maintains a constant value. The reason behind this performance trend is that the constant value of $\mathscr{S}_m$ does not change with the increasing number of records in a transaction. As a result, the storage gain increases with the higher number of log records, which finally reaches very close to the optimum value of 100%.

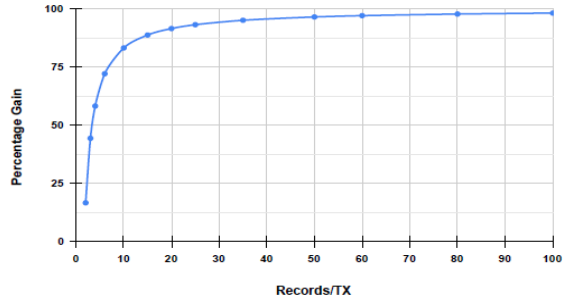

Fig. 10: Storage gain for publishing varying number of log records per transaction

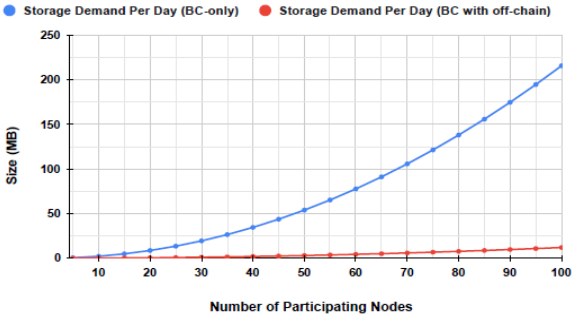

Fig. 11: Per day storage demand of BC network

We also compare per day storage demand of the BC network in both cases of our proposed scheme and the BC-only scheme. We obtained the graph shown in Fig. 11 by calculating the size of the BC network against varying the number of participating nodes in the system. We assume that log records are generated evenly throughout the day with a log generation rate of 30 log records per hour per participating node. We also consider transaction submission rate = 1 transaction per hour per participating node, the average size of a log record = 150 bytes, size of a metadata object = 250 bytes and the number of participant nodes mapped with a BC node = 5. Figure 11 shows that the proposed scheme takes approximately 93% less storage compared to the BC-only scheme to store the log records generated in a day for 100 participating nodes.

Conclusion

This study has addressed the storage scalability issue of BC-based log management schemes by incorporating an IPFS private network with a BC network. Besides, the proposed system preserves data privacy and supports query and audit mechanisms to administer the stored log records efficiently. Moreover, extensive analysis on the prototype implementation of the proposed scheme reveals that the storage gain due to the off-chaining facility increases exponentially with the increasing log records per transaction. Besides, the proposed scheme achieves

nearly 93% storage savings compared to the BC-based schemes without the off-chain facility to support daily storage demand for log records. Furthermore, our scheme executes the query, publish and retrieval operations with a low processing overhead. Hence, our research indicates that it is possible to build up a practical network log management scheme by utilizing the advantages of BC. In the future, we want to extend this platform with various machine learning and deep learning tools to analyze the patterns of log records for detecting different cyber attacks.

Acknowledgment

This study is funded by the ICT Innovation Fund' 2020-21, research grant no: 1280101-120008431-3631108 provided by the ICT division, Ministry of Post, Telecommunication and Information Technology of the People's Republic of Bangladesh.

Author's Contributions

Mohammad Habibullah Rakib: Investigation, problem formulation, methodology, formal analysis, software implementation, data analysis, original draft paper preparation.

Showkot Hossain: Problem formulation, methodology, software implementation, data curation, original draft paper preparation.

Mosarrat Jahan: Supervision, design research plan, research administration, problem formulation, methodology, resources, writing review, draft paper correction and editing.

Upama Kabir: Supervision, design research plan, research administration, problem formulation, methodology, resources, writing review, draft paper correction and editing.

Ethics

This article is an original research work. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Accorsi, R. (2009). Log data as digital evidence: What secure logging protocols have to offer? 2009 33rd Annual IEEE International Computer Software and Applications Conference, 2, 398–403. doi.org/10.1109/COMPSAC.2009.166

Ali, A., Khan, A., Ahmed, M., & Jeon, G. (2021). BCALS: Blockchain-based secure log management system for cloud computing. Transactions on Emerging Telecommunications Technologies, e4272. doi.org/10.1002/ett.4272

Azure. (2021). Microsoft Azure: Cloud computing services. https://azure.microsoft.com/en-us/

Benet, J. (2014). IPFS-content addressed, versioned, P2P file system. arXiv preprint arXiv:1407.3561. http://arxiv.org/abs/1407.3561

Bhutta, M. N. M., Khwaja, A. A., Nadeem, A., Ahmad, H. F., Khan, M. K., Hanif, M. A., Song, H., Alshamari, M., & Cao, Y. (2021). A survey on blockchain technology: Evolution, architecture and security. IEEE Access, 9, 61048-61073. doi.org/10.1109/ACCESS.2021.3072849

Daemen, J., & Rijmen, V. (2001). Advanced encryption standard (AES), FIPS PUB 197. National Institute of Standards and Technology, U.S. Department of Commerce. Accessed on 12 Jan. 2021. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Gerhards, R., & GmbH, A. (2009). The syslog protocol (pp. 1-38). RFC 5424. https://tools.ietf.org/html/rfc5424

Greenspan, G. (2015a). Multichain: Open source blockchain platform. https://www.multichain.com/

Greenspan, G. (2015b). Multichain private blockchain-white paper. https://www.multichain.com/download/MultiChain-White-Paper. pdf

Hajizadeh, M., Phan, T. V., & Bauschert, T. (2018). Probability analysis of successful cyber attacks in SDN-based networks. 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 1–6. doi.org/10.1109/NFV-SDN.2018.8725664

Huang, W. (2019). A Blockchain-based framework for secure log storage. 2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET), 96–100. doi.org/10.1109/CCET48361.2019.8989093

Jabarulla, M. Y., & Lee, H.-N. (2021). Blockchain-based distributed patient-centric image management system. Applied Sciences, 11(1), 1-20. doi.org/10.3390/app11010196

Kent, K. A., & Souppaya, M. (2006). Guide to computer security log management. National Institute of Standards and Technology. Accessed on 12 Jan. 2021. doi.org/10.6028/NIST.SP.800-92

Kumar, M., Singh, A. K., & Suresh Kumar, T. V. (2018). Secure log storage using blockchain and cloud infrastructure. 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 1–4. doi.org/10.1109/ICCCNT.2018.8494085

Kumar, R., & Tripathi, R. (2020). A secure and distributed framework for sharing COVID-19 patient reports using consortium blockchain and IPFS. 2020 6th International Conference on Parallel, Distributed and Grid Computing (PDGC), 231-236. doi.org/10.1109/PDGC50313.2020.9315755

Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K., & Njilla, L. (2017). ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 468–477. doi.org/10.1109/CCGRID.2017.8

Loggly. (2021). Loggly: Unified log management, monitoring and analysis. Accessed on 12 Jan. 2021. https://www.loggly.com/

Marangappanavar, R. K., & Kiran, M. (2020). Inter-planetary file system enabled blockchain solution for securing healthcare records. 2020 3rd ISEA Conference on Security and Privacy (ISEA-ISAP), 171–178. doi.org/10.1109/ISEA-ISAP49340.2020.235016

Miyachi, K., & Mackey, T. K. (2021). hOCBS: A privacy-preserving blockchain framework for healthcare data leveraging an on-chain and off-chain system design. Information Processing and Management, 58(3), 102535. doi.org/10.1016/j.ipm.2021.102535

Papertrail (2021). Papertrail: Cloud-hosted log management, live in seconds. Accessed on 12 Jan. 2021. https://www.papertrail.com/

Park, J. H., Park, J. Y., & Huh, E. N. (2017). Block chain based data logging and integrity management system for cloud forensics. Computer Science & Information Technology, 149. https://csitcp.net/paper/7/711csit12.pdf

Poolsappasit, N., Dewri, R., & Ray, I. (2011). Dynamic security risk management using bayesian attack graphs. IEEE Transactions on Dependable and Secure Computing, 9(1), 61-74. doi.org/10.1109/TDSC.2011.34

Pourmajidi, W., & Miranskyy, A. (2018). Logchain: Blockchain-assisted log storage. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 978–982. doi.org/10.1109/CLOUD.2018.00150

Python IPFS Client (2021). py-ipfs-http-client: A python client library for the IPFS API. Accesses on 12 Jan. 2021. https://github.com/ipfs-shipyard/py-ipfs-http-client

Python MultiChain Client. (2020). mcrpc: A python wrapper for MultiChain JSON-RPC API. Accessed on 12 Jan. 2021. https://github.com/coblo/mcrpc

Rakib, M. H., Hossain, S., Jahan, M., & Kabir, U. (2020). Towards blockchain-driven network log management system. 2020 IEEE 8th International Conference on Smart City and Informatization (iSCI), 73-80, © 2020 IEEE. doi.org/10.1109/iSCI50694.2020.00019

Rane, S., Wagh, S., & Dixit, A. (2021). Blockchain driven secure and efficient logging for cloud forensics. International Journal of Computing and Digital System. http://journal.uob.edu.bh/handle/123456789/4308

Ray, I., Belyaev, K., Strizhov, M., Mulamba, D., & Rajaram, M. (2013). Secure logging as a service-delegating log management to the cloud. IEEE systems journal, 7(2), 323-334. doi.org/10.1109/JSYST.2012.2221958

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126. doi.org/10.1145/359340.359342

Shekhtman, L., & Waisbard, E. (2019). EngraveChain: Tamper-proof distributed log system. 2019 2nd Workshop on Blockchain-Enabled Networked Sensor (BlockSys'19), 8–14. doi.org/10.1145/3362744.3363346

Söderström, O., & Moradian, E. (2013). Secure audit log management. Procedia Computer Science, 22, 1249-1258. doi.org/10.1016/j.procs.2013.09.212

Splunk. (2021). Splunk: Turn data into doing. Accessed on 12 Jan. 2021. https://www.splunk.com/

Wang, K., Yan, Y., Guo, S., Wei, X., & Shao, S. (2021). On-chain and off-chain collaborative management system based on consortium blockchain. 2021 7th International Conference on Advances in Artificial Intelligence and Security (ICAIS 2021), 172-187. doi.org/10.1007/978-3-030-78618-2_14

Zheng, Z., Xie, S., Dai, H.-N., Chen, X., & Wang, H. (2018). Blockchain challenges and opportunities: a survey. International Journal of Web and Grid Services, 14(4), 352–375. doi.org/10.1504/IJWGS.2018.095647