future internet

MDPI

Article

EngraveChain: A Blockchain-Based Tamper-Proof Distributed Log System

Louis Shekhtman [1,*] and Erez Waisbard [2,*]

[1] Network Science Institute, Northeastern University, Boston, MA 02115, USA
[2] Nokia Bell Labs, Kfar Saba 4464321, Israel
* Correspondence: Lsheks@gmail.com (L.S.); erez.waisbard@nokia-bell-labs.com (E.W.)

Abstract: A reliable log system is a prerequisite for many applications. Financial systems need to have transactions logged in a precise manner, medical systems rely on having trusted medical records and security logs record system access requests in order to trace malicious attempts. Keeping multiple copies helps to achieve availability and reliability against such hackers. Unfortunately, maintaining redundant copies in a distributed manner in a byzantine setting has always been a challenging task, however it has recently become simpler given advances in blockchain technologies. In this work, we present a tamper-resistant log system through the use of a blockchain. We leverage the immutable write action and distributed storage provided by the blockchain as a basis to develop a secure log system, but we also add a privacy preserving layer that is essential for many applications. We detail the security and privacy aspects of our solution, as well as how they relate to performance needs in relevant settings. Finally, we implement our system over Hyperledger Fabric and demonstrate the system's value for several use cases. In addition, we provide a scalability analysis for applying our solution in a large-scale system.

Keywords: blockchain; computer security; intrusion detection; distributed ledger; access control; trust

Citation: Shekhtman, L.; Waisbard, E. EngraveChain: A Blockchain-Based Tamper-Proof Distributed Log System. *Future Internet* **2021**, *13*, 143. https://doi.org/10.3390/fi13060143

Academic Editor: Peter Kieseberg and Thomas Moser

Received: 30 April 2021
Accepted: 26 May 2021
Published: 29 May 2021

1. Introduction

The need for tamper-resistance log files is prevalent in many areas and is a concern raised by numerous regulators and standards, such as PCI [1], HIPAA [2], and GDPR [3]. Examples include medical records whose reliability can have life-or-death consequences, financial information that must be accurate, and IT security files that are essential for identifying security incidents and carrying out forensics. The above use cases share a common file structure in which records are only being added over time, e.g., individuals have more medical procedures, more financial transactions, and more security events are logged. In addition, these files require high availability allowing a user to examine the records on demand. The log files are used for assessing fault analysis [4], anomaly detection [5,6], forensics and audits [7,8], and many other purposes. During an attack, one of the first steps that hackers who manage to obta privileges take, is to clear the event logs from a compromised system in order to hide their tracks. As covering the attack traces has become standard practice for attackers, it is even integrated into the popular Metasploit meterpreter cyber attack tool and is invoked using the script 'clearev' [9].

As evidenced from the above use cases, the need to have a trustworthy copy of the log file is common to many organizations within the given fields. Naturally, trust between these organization is a major issue. Our primary focus here is on several actors in a semi-trusted environment seeking to improve the reliability and security of their logging information, ensuring that the logging information is tamper-resistant. In order to illustrate this, consider the case in which several banks are instructed by a regulator to secure their log files. These banks will likely be willing to collaborate in such a joint effort to secure their log files, but would not be willing to disclose the content of their files. Thus, we seek a solution that does not reveal the content of the files to the participating parties.

However, we can safely assume that the collaborating banks are unlikely to actively try to hack each other.

Ensuring consistency of the distributed records in the presence of multiple writers is not an easy task. Some parties may be offline, messages may be delayed or dropped due to network disruptions and as a result the different parties may not share the same system's view at every given point in time. Furthermore, providing accuracy and tamper-resistance in the presence of some malicious parties is even more complicated. Thus, a solution must allow for distributing the information among the banks in a way that makes it available and tamper-resistant, but at the same time provides mutual confidentiality.

We demonstrate that the blockchain [10], with appropriate modifications, optimally fulfills these requirements and that it can be used to facilitate distributed secure-logging among this set of actors. At its core, blockchain technology is designed to provide a replicated state over the Internet. We note that while blockchain technology assists in achieving the above requirements, decisions must be made about the type and reason for using the blockchain. First, blockchain technology requires the joint effort of the parties to guarantee security. Therefore the parties must have sufficient incentive to dedicate their resources. Furthermore, the amount of effort is dictated by the properties of the specific blockchain that is being implemented. For example, public blockchains, such as Bitcoin, require significant (and costly) computational effort from the miners who are motivated through receiving payment for their (successful) effort. Private blockchains, which only allow specific identities to interact with the blockchain can use much more efficient consensus algorithms such as PBFT [11], but still every participating party needs to contribute to the network by providing servers that participate in the protocol. Thus, trust and scalability issues must be considered before choosing the appropriate blockchain. Different use cases will also involve different levels of trust between the parties and thus have different requirements.

After deciding on which blockchain to use to minimize overhead, it is worth briefly reviewing how the blockchain meets these requirements. The requirement of consistency is fulfilled by the blockchain's consensus protocol and blocks are tamper-resistant since they are cryptographically sealed after writing. The distributed nature of the blockchain not only provides high availability and resilience, it also challenges the attacker to break into several security systems in different locations. However, the requirement of privacy is not typically fulfilled merely by using a blockchain . While a permissioned blockchain can limit the access to only authorized parties, it still grants full read-access to all of those who are authorized. We provide privacy between authorized parties by adding an additional encryption layer that encrypts the data prior to writing it to the blockchain.

A further issue is that blockchain technology is typically limited in the size of the data that can be written to it in a single transaction. This means that storing large log files cannot be done efficiently and requires many transactions. In addition to adding a privacy layer, we therefore also add a mechanism that enables storing data off-the-chain to ensure high throughput, while managing the integrity of this data on the blockchain. All of the above allows us to achieve a solution that provides Confidentiality, Integrity and Availability (the CIA triad).

After solving the technical issues of using the blockchain, the key contribution is that it allows an organization to make copies of their log files and distribute them among multiple sites. This way attackers that manage to break into one of these devices and attain privileges may be able to locally compromise the storage of that device, but they will not be able to affect the other copies residing on devices that they do not control. This ensures that the file availability and integrity is maintained. At some point in time the compromised system will likely detect the attacker and remove them. Nonetheless, during the time period where the victim was compromised, the local copies of the log files are likely to have been corrupted. Therefore a recovery process is required to restore the file from the other uncompromised parties who still maintain a copy of the previous uncorrupted file. Furthermore, even before removing an attacker, if may be possible to spot an attack as it

occurs (e.g., by using a security service that monitors the different copies of the blockchain). In this work, we assume that the number of parties the attackers can control at any given time is limited and therefore most of the copies of the log file on other servers remain accurate (see Figure 1).

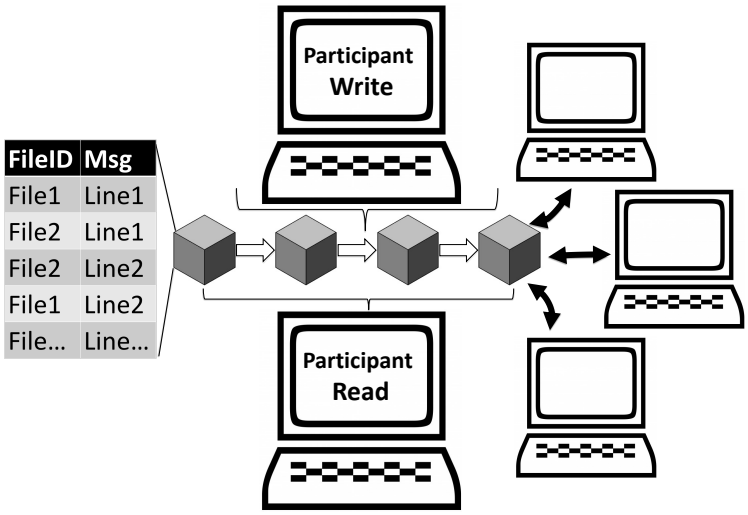

Figure 1. Log files are files in which information is appended over time. In each time period, new entries are added to some files. Our system treats these updates as transactions containing a FileID to which a given message (Msg) relates. These are combined into a block and stored on the blockchain. To read a file from the blockchain, the various transactions associated with a given FileID are collected and appended sequentially.

We envision our proposed solution to be most appropriate to the case where several organizations that wish to protect their log files are willing to assist each other towards achieving that mutual goal. In some use cases, parties may completely trust each other (for example, in the case of several branches of the *same* organization). However, in other use cases parties may only trust each other enough to cooperate for mutual benefit, but do not trust each other with the contents of their log files. The additional encryption layer provides the required privacy to support this and prevent legitimate parties from reading the content of each others' logs. This form of protection remains true [7,12] even against an attacker that has full control over a few compromised parties.

The remainder of the introduction, reviews prior related work. In Section 2, we describe the settings and requirements for our system and in Section 3 we describe our solution in detail. Section 4 analyzes the security of our solution and Section 5 explains our implementation. Lastly, we give our conclusions and an outlook for future work in Section 6.

Prior Work

Several solutions have been proposed to secure log files, such as write once read many (WORM) memory systems [13]. Another method combines the use of an untrusted machine with a trusted one in order to prevent an attacker from being able to read any log files generated prior to their penetration of the unstrusted machine [7,12]. Likewise, the system prevents the attacker from being able to modify the log files without detection. Holt improved on this work [14] by incorporating public key cryptography and thus allowing for the separation of log verification and modification, i.e., a user may be given the ability to verify logs but not to modify them. In any case, these solutions (except for WORM memory) only provide proof that logs were tampered with, but do not describe methods for recovering the logs after failure. Furthermore, WORM memory systems, which do allow for such a possibility, have so far been hardware-based involving, e.g., organic

semiconductors, which are difficult to scale at the same level as software-based solutions that can be operated on the cloud.

The blockchain as a shared ledger was originally implemented in the bitcoin protocol [10]. Essentially it is a set of chronological 'blocks' containing lists of transactions that are grouped together and sealed for writing using a cryptographic hash. Later transactions are appended to a second block that is then cryptographically linked to the first block, and so on for later blocks. The primary use of blockchain has been for cryptocurrencies, yet many other use cases have recently come to attention including supply chains [15], financial trade clearing [16], and others [17].

Several prior works have considered using the blockchain to store information. For example, Ref. [18] considered the possibility of sharing information over the blockchain and developed a protocol for giving access to private information stored on the blockchain. Similarly, Ref. [19] used the blockchain together with smart contracts to establish individuals' identities and assure accountability, while Ref. [20] used the blockchain in a similar way, but focused on information storage on the cloud. Others considered proving the information accuracy by saving a hash to the bitcoin blockchain (rather than creating a new blockchain). For example, Ref. [21] uploaded a hash to prove the timestamp of a video, e.g., recording a collision and [22] proposed to do so to verify voting records. In the medical realm several works [23,24], recently suggested encryption protocols and permissions that could be used to store medical records on the blockchain while still ensuring privacy and empowering patients. Another recent work explored how one can improve the querying of information from the blockchain as in many cases certain types of files must be queried regularly [25].

None of the above works focused on using the blockchain for system log files, addressed how organizations can mutually benefit from multiple copies of their information while not sharing that information, or considered how the information can be retrieved if it is corrupted. A recent work from [26] suggested using the blockchain to allow forensic investigators to detect deletion and modification of past records by a cloud provider. Their solution focuses on a single cloud provider and a single forensic investigator and does not address the many aspects of efficiency and trust between the different stakeholders as described in our use case. Closely related to our work here is [27], who considered methods of securing a distributed database on the blockchain. In contrast to our work, however, they focused on the case where other 'miners' (peer-to-peer nodes) are themselves the malicious attackers whereas we focus on securing the logs from an attacker outside the blockchain. Another closely related recent work [28] considered using a blockchain to store log files though there the authors used different frameworks and made alternative choices from those we make here [29].

2. Settings and Requirements

2.1. Settings

We envision two related, though distinct, settings where EngraveChain could be applied. Both settings involve multiple peer nodes, each containing a stored copy of the log information. Furthermore, there are multiple participants who write their information to the blockchain. In many cases, each peer node will be associated with one blockchain participant and vice versa, however this need not necessarily be the case as some participants may not want to serve as a node that stores the information.

In our first setting, the peer nodes and participants will correspond to different systems that are all part of a single organization. For example, a multinational corporation with multiple sites could take advantage of its replicated infrastructure at multiple sites by having each site operate a node and participant to commit important information to the blockchain. The organization would benefit such that even if one site is compromised, the log information remains available at the other sites and can be recovered.

A second setting consists of multiple organizations, possibly competitors, who agree to cooperate in order to improve security for all. Here one could envision several large

banks (or other organizations) who agree to help one another by preserving each others' log information on each of their infrastructures. Thus, even if one organization's system is compromised, it could recover its information from the copies of the blockchain stored on the infrastructure of the other organization. These two organizations would thus have an incentive to cooperate in order to insure that should one of them face a significant attack, they could in the end recover their systems from the other bank. Furthermore, our above example of protecting banks from cyberattacks has also taken on national security importance in many countries and thus regulators may choose to encourage banks to collaborate in forming such a blockchain in order to preserve their logs and increase their security.

Lastly, for some applications these two cases may be combined such that a large organization may want to replicate some of its information on multiple nodes, all of which are internal to that organization, whereas other information may be replicated on the nodes of other external cooperating organizations. Whether to replicate only internally or externally could depend on a variety of factors including the level of privacy of the information, the importance of it remaining secure, and other considerations.

2.2. Security

We now define the security requirements for our system. Loosely speaking, we want to maintain a distributed set of consistent copies of the log system despite the presence of corrupted parties and provide the ability for recovering parties to restore an accurate copy. Namely, an attacker that can compromise only a fraction of the parties is unable to corrupt the distributed log file and can neither disrupt its availability nor privacy. Finally, there is a process wherein the system can recover the log information from prior to when it was compromised.

More formally, we require the following three properties:

Security: An adversary that only controls a fraction smaller than a threshold T of the parties (for Practical Byzantine Fault Tolerance T is $1/3$ of the parties [11]), cannot modify the integrity of the distributed copies for the honest parties.

Privacy: We require that parties on the blockchain can only read the content of files that the origination file owner authorized them to read. In addition we require that the content on the blockchain is stored encrypted.

Availability: An adversary cannot make the distributed files inaccessible to uncompromised parties.

We note that the privacy requirements are particularly important for the use case in which the multiple machines actually belong to distinct organizations (or even different security levels within a single organization). In that case, it is required that other organizations are unable to actually read the content of the logs. Still, another desirable property would be the ability to selectively share/store information with distinct organizations. For example, perhaps some information can be shared with all other organizations, whereas other information should only be shared with a smaller subset of trusted organizations.

3. Our Solution

In this section, we describe EngraveChain—our secure distributed log system. We start with a permissioned blockchain in which all participants and peer nodes are identified. Not only does using the permissioned blockchain best fit our use cases, it also allows for efficient consensus protocols, such as Practical Byzantine Fault Tolerance (PBFT) [11]. We begin by detailing our solution in a context in which several organizations collaborate to secure each others' log files. We find it to be the more generic use case and emphasize that this solution also works for a single organization with multiple sites. The first layer of this solution is a peer-to-peer network consisting of k nodes each of which belongs to one of o distinct organizations. Each organization will have an administrator and certificate authority who generate credentials for its associated peer nodes, allowing them to access the peer-to-peer network. Each peer stores a copy of the blockchain and every write operation is written as

a transaction to the blockchain. Peer nodes arrive at consensus (e.g., by using PBFT) on the set and order of approved transactions. Aside from the peer-to-peer layer, there is also a layer consisting of n blockchain participants who write to the blockchain by submitting the 'transactions' to the peer nodes.

For our purposes, the transactions submitted to the blockchain will consist of the log information from one or more files belonging to the participants. In Figure 2, we demonstrate the writing of log files to the blockchain. To begin, a user must first initialize a log file on the blockchain by submitting an initial transaction that establishes a unique FileID for that log file and governs which other participants are allowed to submit transactions adding to the file. This transaction will also contain the first chunk of the file and the ID of the participant that created the file. Next, to write later parts of the file to the blockchain, either the creator or another approved participant can submit a transaction containing (i) the ID of the file being added to, (ii) the next chunk of the file, and (iii) the participant's ID. In the most common sense, it is expected that only the creator of the file will write to it. One way to do so is by explicitly requiring this in the architecture. Alternatively, one may add an 'open' operation in which the creator of the file specifies who are the authorized parties allows to write to the file.

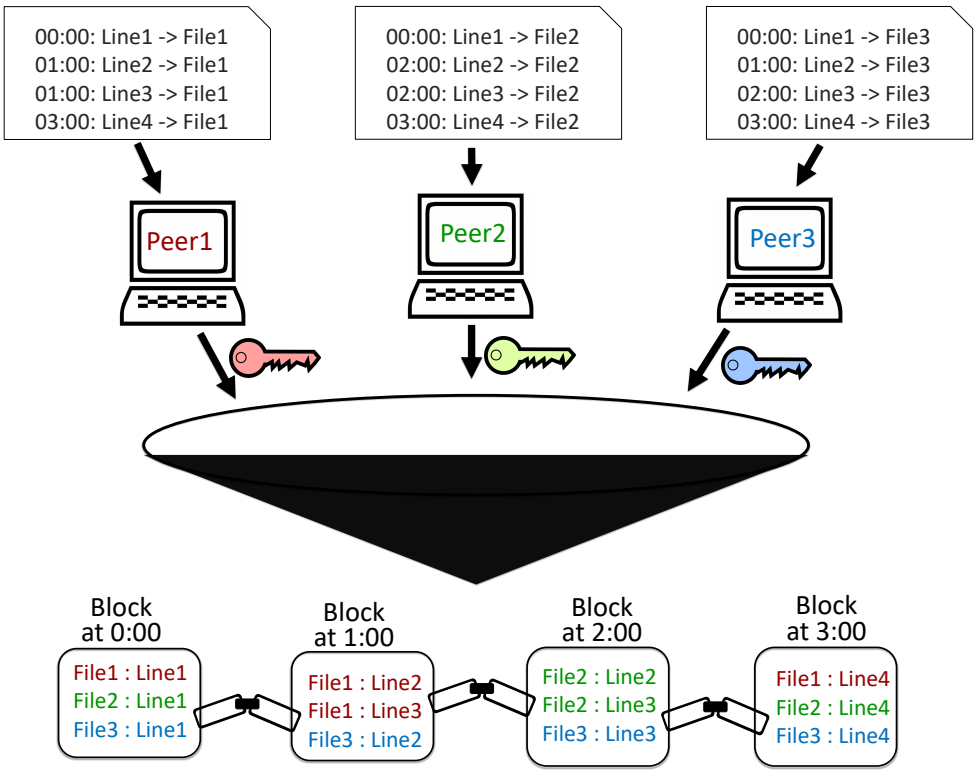

Figure 2. The process of writing logs to the blockchain: unencrypted files are encrypted by each peer with its own key. The encrypted lines of the file are then written to the blockchain.

An interesting question involves the optimal size of the chunks to be submitted to the blockchain, and the answer to this question is likely application specific. Larger chunks imply less transactions and presumably less overhead for the peers in approving and validating transactions. At the same time, larger chunks also mean that the writers are likely submitting their logs less frequently, which could be problematic since between write operations to the blockchain, if a system is penetrated by an attacker, all of its local copies of logs can be altered without being noticed. For systems constantly carrying out sensitive operations at short time scales, it is reasonable to accept the additional overhead involved in writing many transactions, while for systems carrying out infrequent, but also sensitive, operations, it may be acceptable to wait and submit larger chunks of data to the blockchain.

3.1. Secrecy

The secure and redundant copies that the blockchain provides means that the other participating parties with access to the blockchain have access to the information that is stored on it (we note that in permissioned blockchains, only authorized participants have access to the blockchain. The authentication of the authorized parties is enforced using public key infrastructure (PKI)). We assume that the parties with access to the blockchain trust each other enough to secure the data that is being written, but at the same time, they may not feel completely comfortable having their data visible to all other parties. Such situations represent many of our use cases and therefore we propose that the participants will first encrypt the chunks of their files and only then submit a transaction containing the encrypted version of the logs to the blockchain. Other participants who do not have the decryption key will not be able to read the clear content from the encrypted records, while the holder of the decryption key can decrypt it whenever it is needed. A question that then arises is which encryption scheme to use. Using a symmetric key encryption scheme, e.g., advanced encryption standard (AES) [30], where the encryption key is the same as the decryption key would mean that an attacker that compromises the participant could get this key and be able to decrypt past records stored on the blockchain. This conflicts with the basic idea behind EngraveChain and therefore a way to avoid this is by using a public key encryption scheme, e.g., Rivest–Shamir–Adleman encryption (RSA) [31], where the encryption and decryption keys are different. In such a scheme, one can use the encryption key for the frequent encryption operations, while keeping the decryption key securely stored away from the hacker's reach. In the (relatively rare) cases where access to the logged data is required, the owner of the decryption key can use it to decrypt the data. In order to maximize the security of this process against cyberattacks this private decryption key can be stored for safekeeping offline and only be used by a standalone log analysis system that can decrypt the files and provide them as needed. Thus, an attacker would not be able to get the private decryption key to decrypt records that were written by the participant.

3.2. Recovering the File

To read a log file previously written to the blockchain a user can gather all the transactions in the blockchain associated with a particular FileID, order the transactions by timestamp, decrypt the chunks using the secret key that was used to encrypt them and thereby recover the file (see Scheme 1 and Figure 3). We should note that while encryption provides the confidentiality of data, some information may still leak. For example, the varying rate in which the information is being written. An organization may take further actions to ensure that such information does not leak (e.g., writing at constant rate). Still, the participating organizations (e.g., respectable banks) are not expected to apply significant effort using advanced cryptanalysis towards decrypting log information of other banks. These entities already face high levels of oversight and regulation and would face significant backlash for carrying out a cyber-attack of that sort.

Write (filename, message):
//Participant P_i writes a message to a file
1. E_m = Encrypt(message, public_key$_i$) //Encrypt message with public key
2. Submit_Transaction_to_Blockchain(E_m, filename, P_i)

Read (filename):
//Retrieve all file parts from the blockchain
1. GET all m_j messages from the blockchain where FileID = filename
2. For each message m_j in filename:
   D_{m_j}=Decrypt(message, secret_key$_i$) //Decrypt using the secret_key$_i$
3. Combine(All D_{m_j}) //Append the retrieved message parts to a single logical file

Recover Process:
For compromised peer node k_i:
1. Remove attacker from infected system.
2. Submit blockchain transaction to revoke compromised peer node certificates.
3. Certificate Authority reissues blockchain certificates if they were compromised.
4. Sync blockchain state with uncompromised participants to obtain blockchain consensus state.

Scheme 1. Pseudocode for the three basic operations through the blockchain: Write, Read and Recover.

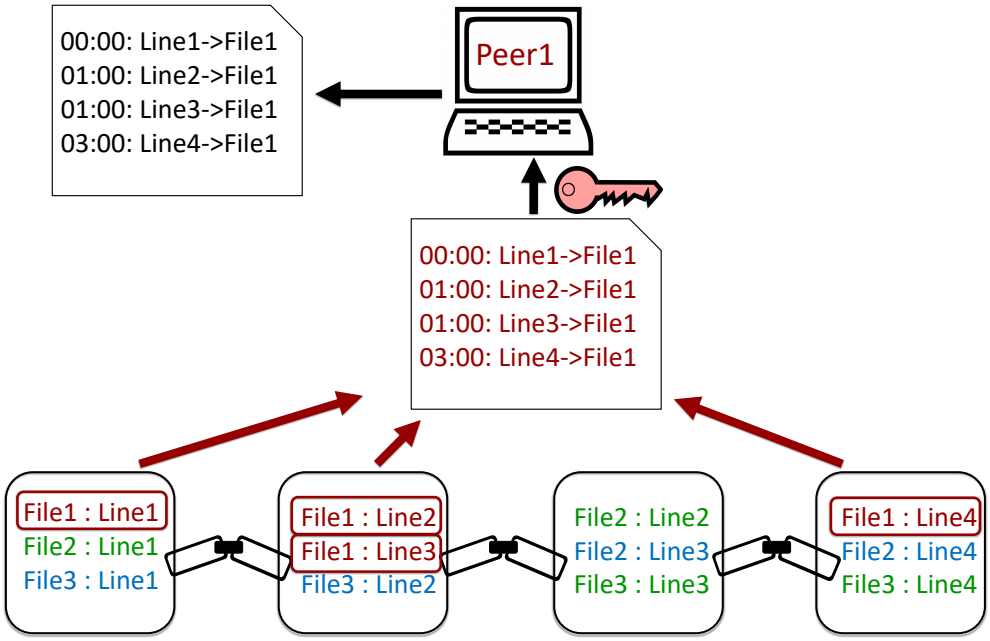

Figure 3. To reconstruct a log file, first the relevant lines of the file are gathered from the various blocks and ordered by their timestamps. The organization or peer that wrote the file will then use their unique private key to decrypt the file in order to recover the file.

For very large files, one could use a method similar to that of [21,22] where they uploaded a hash of a file to the bitcoin blockchain. In our case we could save the large file to cloud storage or a distributed storage system (e.g., storj [32]) and then only include a hash of the file on our blockchain (see Figure 4). This hash could then be compared to later versions of the file to ensure the authenticity of the file. Nonetheless, this method involves a tradeoff as if the file is corrupted, one cannot necessarily obtain an accurate version of the file unless there are other copies stored elsewhere. As such, we only suggest using this method for particularly large files where including them on the blockchain involves too much overhead or where knowledge of file integrity is more important than actually having an accurate copy of the file.

A further consideration that can be added to our system is the possibility of 'closing' a file and preventing any further logs from being written to it. While we did not include such a capability in our current implementation we note that it could be desirable for some use cases.

Compromised systems will often notice different system behaviors that lead them to suspect and discover the presence of an attacker. For example, a user that notices that their machine's performance has decreased significantly, will often choose to scan their machine for malware. Upon discovering an attacker, the system administrator can then initiate a process to remove the attacker. If the attacker deleted information from the infected computer, then a recovery process is also necessary to restore the previous data. In our solution, the administrator of the infected machine triggers the recovery process through which the previous logs are restored from the blockchain. In this way the previously infected participant can recover the blockchain state containing the deleted files from the uninfected nodes. If a node was infected, i.e., a machine that also blockchain protocol in addition to writing its log files, then the certificates and secret key used by the node to access the blockchain will have to be reissued. The certificate authority of the infected node can revoke the compromised node's blockchain access certificate [33] and secret key and then issue a new certificate and key for that node.

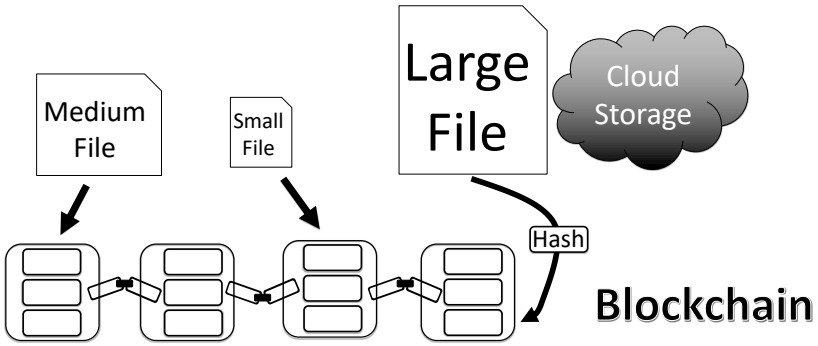

Figure 4. If the size of the files is a limiting factor for the system, large files can be uploaded to cloud storage and only a hash of the file can be saved to the blockchain. By doing so, the storage on the blockchain is minimal and large files take up no more space than smaller files. However, this comes with some sacrifices as only integrity will be guaranteed, whereas the file may not be available if it gets deleted in the cloud.

Permissioned blockchains also have access control mechanisms that allow for fine grained control of access to the information on the blockchain. This allows for creating multiple chains, referred to as channels, that only authorized parties can access. For example, let us consider a system that logs users' access to resources (e.g., entry to rooms using employee cards or login credentials to networks). We may have two types of resources—generic ones and restricted ones (e.g., entry to a building vs. accessing a restricted zone in the building). In such a case we may want to create two channels: one which logs all accesses to the public resources and another that logs access to the restricted ones. Some participants will have access to both channels while others will have access only to the public one (see Figure 5). The access rights of each participant and nodes are defined in the blockchain access certificates and enforced by the blockchain infrastructure (in our case through Hyperledger Fabric's 'channels' feature).

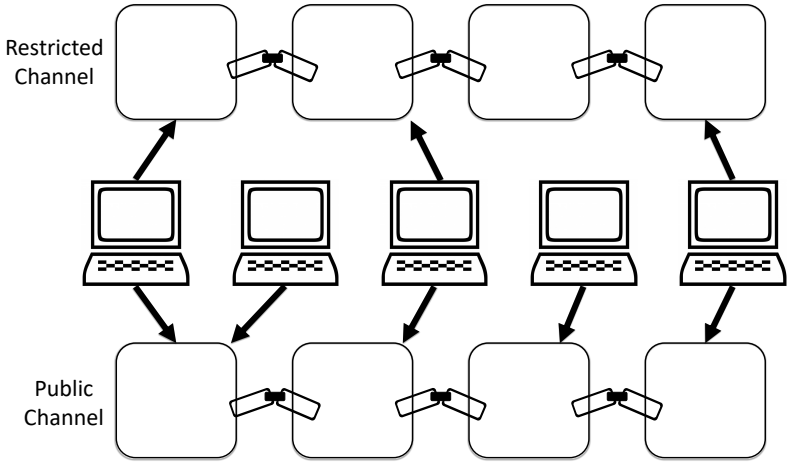

Figure 5. Separate channels with distinct blockchains. Here we demonstrate a set of peer nodes (who are also participants in this figure), all of whom have access to the public channel (**bottom**), yet with only some of the peer nodes having access to the restricted channel (**top**).

4. Security Analysis

In this section, we analyze the security of our solution. Much of the security is inherently derived from the blockchain security properties. In particular, an attacker that is able to compromise a limited subset of the participants is unable to tamper with the state of the blockchain. Naturally, if the attacker has full control over a participant's device, he can locally do as he or she pleases. However, at some point in time this attacker is likely to be detected and removed (e.g., using anti-virus software). Once the attacker is removed, the correct system state can be restored by syncing with the uncompromised nodes who maintain the blockchain state.

4.1. Security Assumptions

We rely on the following assumptions: First, that at any given time, the attacker can corrupt only a limited subset of size $t = (n-1)/3$ out of n participants/nodes in the blockchain system. Second, that an attacker that controls a participant/node can alter its entire state and behavior including deviating from the protocol in any way he or she chooses. Third, that there exists a recovery process in which a corrupted participant/node completely removes the adversary (e.g., by restoring a clean image). Under the above assumptions, our solution achieves security and privacy in the malicious model.

4.2. Security against Attacks

We now analyze the security of our solution against the following attack scenarios:

Modifying the records: The security of the blockchain guarantees that an attacker cannot modify the records in the log. As long as more than $\frac{2}{3}$ of the peer nodes are not compromised, the BFT consensus protocol ensures the integrity and consistency of the records of the honest nodes [11].

Privacy: The privacy of the information stored on the blockchain is achieved both by the access control mechanism of the blockchain (the separate Hyperledger channels in our implementation) and by encrypting the data before writing it to the log file. The channels in Hyperledger ensure that only authorized entities can access the data on the blockchain. Encrypting the data before writing it to the blockchain, ensures that even nodes that are authorized to read the data from the channel can only read the ciphertext. This preserves the privacy of the information between participants and nodes that wish to collaborate in order to better secure their data, but do not wish to share the data. While the privacy of the data is secured by strong encryption, the write pattern (i.e., data size and writing rate) is visible. Parties who are concerned that the writing pattern may leak some information to their competitors can modify their writing pattern. This can be done,

for example, by grouping logs or adding dummy records (as encrypted dummy records are indistinguishable from other encrypted records). We note that in the case where the writer uses a public key encryption scheme [34] (a.k.a. asymmetric encryption) to encrypt its data while safeguarding its private key off of the infected machine, the participant can also prevent an attacker from reading past records of the compromised system.

DoS: An attacker that controls a participant can try to clog the blockchain by sending far more log entries than expected. In order to prevent this, the peer nodes can specify a policy limiting the rate and sizes of entries from any single participant. We note that anomaly detection in writing patterns can be carried out by participants and peers and also trigger a temporary revocation of the attacking participant, e.g., by using the Hyperledger Fabric revoke method [33].

5. Implementation

For our secure log POC implementation, we used open source tools Hyperledger Fabric, Hyperledger Composer, and Hyperledger Blockchain Explorer [35]. We also designed a browser GUI to simplify interaction with EngraveChain. We carried out our implementation on a Ubuntu 16.04 Virtual Machine with 8GB RAM, 4 Virtual CPUs, and 80 GB storage. We emphasize that while we used Hyperledger, our system's components are sufficiently general that other frameworks could also be used, e.g., multichain or others. Our choice of Hyperledger was based on our experience with these frameworks and Hyperledger's compatibility with our requirements.

5.1. Review of Hyperledger Components

We begin by briefly reviewing the Hyperledger Project components [35]. Hyperledger Fabric is an open source tool that enables the creation of private, permissioned blockchains. An important aspect of permissioned blockchains for our use case is that the steep costs of mining through proof-of-work in public blockchains (like Bitcoin), can be significantly reduced. Furthermore, Hyperledger Fabric allows for creating distinct *channels* where only specific organizations may have access to each channel. On top of Hyperledger Fabric, one is able to create customized 'business logic' referred to as *chaincode*. Hyperledger Composer provides a basic structure for chaincode that includes assets, participants, and transactions (among others) on the blockchain. Finally, we use Hyperledger Blockchain Explorer, which is not an integral part of the system but was added to allow for examining transactions and blocks.

5.2. Our Hyperledger Implementation

In our POC implementation we created three organizations and two peers per organization in Fabric. We also created two channels: one with all three organizations and a second with only two of the organizations. In Hyperledger Composer we defined participants with a unique id and name. We defined transactions with a unique FileID, a chunk of saved text (presumably encrypted as detailed above), an associated participant who submitted the transaction, and a timestamp. We did not define 'assets' belonging to specific users, yet an alternative architecture could involve each log defined as an asset with the creator being the owner. This alternate architecture could raise issues if two users want to write to the same log file, but it would allow for defining a 'close' action, mentioned earlier, which would prevent any further writing to the file.

To simplify interaction with the Hyperledger components, we developed a GUI for creating, retrieving, and adding to log files (see Figure 6). To retrieve a log, one enters the FileID of the desired file and then a command is submitted to Hyperledger Composer to scan the blockchain for transactions with that FileID. Recall that we have two separate channels, so a member of the third organization is unable to retrieve files from the channel that is private to only the first two organizations. In fact, the third organization would not even know about the private channel. To create a new log, one enters the (unique) FileID to be associated with the log and the first string (chunk) of the file. Furthermore, while

FileIDs must be unique on the same channel, the same FileID can be used on each channel. Similarly, for initializing or adding to a log file, one must specify an appropriate FileID on the given channel. We also enabled Hyperledger Blockchain Explorer, which shows raw transaction data, raw blocks, and other information about the blockchain.

A video demonstrating the interaction with our implementation is available at: https://www.youtube.com/watch?v=i577L_nVmQA (accessed on 26 May 2021).

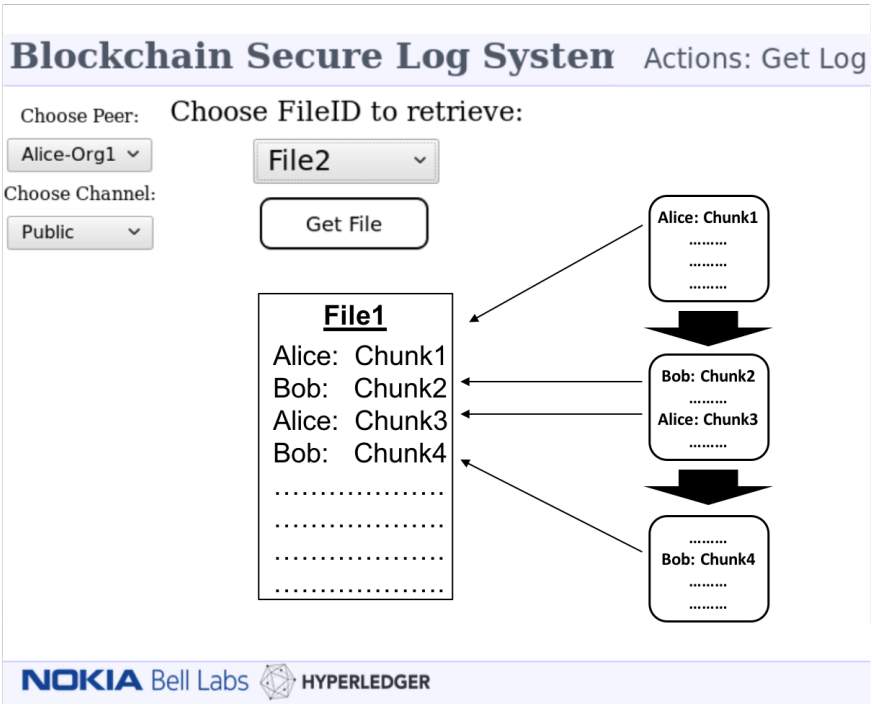

Figure 6. A screenshot taken from our proof-of-concept (POC) GUI demonstrating retrieval of a log file from the blockchain, where transactions stored in different blocks come together to create a single logical file.

5.3. Performance and Scaling

A key issue of any log system, is its ability to perform at scale and successfully process the potentially high rate of entries it receives [8,14]. As our implementation is built upon the Hyperledger Framework, its performance will largely depend on that of the underlying framework, as well as the number of peers and organizations, and the underlying consensus algorithm. Our proof-of-concept was built upon a simple architecture in which consensus was achieved using a single 'orderer' node (using the 'solo' consensus algorithm) which set the order of transactions with the necessary signatures and then distributed this information to the peers (see [36] for more on orderer nodes). In production one would use a more sophisticated consensus algorithm and thus we did not carry out a full performance analysis on our proof-of-concept. Rather, here we will review the relevant aspects of performance for the most recent version of the Hyperledger Framework and how they apply to our case. For a production system, the current Hyperledger recommendation is to use the RAFT consensus algorithm [37], which generally has better performance than PBFT. At the same time, it must be noted that RAFT is not robust against malicious attackers in a byzantine setting, in which case using PBFT would be necessary and there could be a performance penalty. A preliminary assessment [38] of the performance of the RAFT consensus found a throughput of several hundred TPS, which differed little from the performance using the solo orderer. For a fixed blockchain logic, the key performance factor was the endorsement policy, i.e., how many peers/organizations had to approve a transaction.

More explicitly regarding the performance, a published benchmark analysis [39] for a network consisting of two organizations and using a single orderer (as in our implementation) found that assets can be created at a rate between 60 and 400 transactions per second (TPS) on their system consisting of simple hardware with 8 CPUs and a 3.8 GHz Processor. Furthermore, the endorsement policy and asset size, as well as other parameters effect the performance as seen in Figure 7. A previous analysis of log files [40] found that an enterprise-scale production network using the Snort network intrusion detection system output 2.25 million alerts over 27 days, a rate of just under 1/s; and that HTTP server log data from a website received 109,481 http requests over a period of about 16 h for a rate of about 2/s. Of course, the overall rate of log files will also depend on the size of the system. One estimate [41] that took this into account suggesting an estimate of 500 events per day per user, suggesting that a rate of 100 TPS could accommodate around 17,000 users assuming the events were evenly distributed throughout the day. It is noteworthy that for our log system architecture, there is no need to create a Hyperledger Asset but rather simply write text to the blockchain, so even evaluating on similar hardware, we would expect our performance of write actions to be higher than their rate of asset creation. Likewise, improved hardware could be used to obtain even stronger performance.

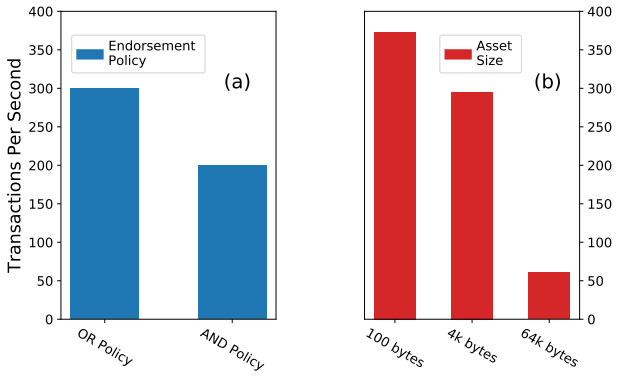

Figure 7. Performance references for hyperledger fabric. (**a**) The transactions per second based on an endorsement policy requiring either of the two organizations to approve transactions (OR Policy) and bath of the organizations to approve the transactions (AND Policy) [38]. (**b**) The variation in performance based on the size of the asset created in the transaction [39].

A key aspect of the 'write' performance will involve the nature of the logs being written and what level of detail and frequency is desired. Many of the common Windows Security Events tend to be somewhere between 600 and 2500 bytes [41] and the aforementioned throughput analysis [39] found that using a CouchDB database achieved a throughput of 179 TPS assuming a size of 2000 bytes and 164 TPS with a size of 4000 btes. While writing less frequently will result in longer logs, this will overall increase performance as larger transactions are written somewhat slower, but even with this tradeoff, there is greater throughput of log information for larger transaction sizes, i.e., greater amounts of logging information in our case [39]. Likewise, for particularly large files, only the hash of the file could be written to the blockchain with the contents being stored in the cloud (see Figure 4). This would reduce the amount of information being written to the blockchain and thus increase throughput, while involving other tradeoffs as discussed previously. We also note that our transactions are fairly simple, generally consisting only of text and some other basic metadata. For a basic architecture, there is no need for ownership of assets to update or other more complex processes during a transaction. This should allow us to obtain very close to the optimal performance allowed by the Hyperledger framework.

We now consider the above performance aspects in our example of a group of 5 banks collaborating to preserve log information. In this setting RAFT is acceptable as a consesnsus algorithm since we do not expect the banks to maliciously attempt to sabotage another particular bank/s and thus PBFT is not necessary. Furthermore, from requiring that

the originator is the one to submit the log, we need only a small number of additional confirmations to approve the transaction and thus we can have a fairly simple endorsement policy which should allow us to obtain strong performance. An example would be requiring any 3/5 organizations to approve a transaction. Lastly, depending on the nature of the logging information the banks can choose how often to write their logs and thus potentially compile more logging information into each transaction possibly improving the overall throughput as each write action becomes larger.

A final important aspect, is the ability to retrieve logs. In the benchmark study [39], just as for the submit actions, the performance of 'get' actions depended on the size of the assets and ranged from around 100–700 TPS with assets of larger size having slower times though still leading to an overall greater throughput of data. For our system, we would likewise expect the read actions to depend on the size of the logs written and involve similar tradeoffs. At the same time, since logs are likely assessed less frequently than they are written, it is reasonable to presume that 'write' performance is of greater importance than 'read' performance.

These benchmarks must be taken with some caution as they are based on a single orderer node, as in our implementation, however when combined with the work in Wang et al. [38], we can be optimistic that changing to the RAFT consensus algorithm is unlikely to significantly reduce the performance.

6. Conclusions

We introduced EngraveChain, a tamper proof distributed system for securing log files. We suggested use cases, presented and implemented the solution and showed its value for these cases. While we have introduced the concepts, there still remains work to integrate it with a real system. We note that each system has its own unique settings and challenges that need to be addressed including integration with related components that are particular to the system, e.g., integrating with an intrusion detection system to protect its security log files. Each of these specific cases would require addressing the scalability and performance issues discussed above. With respect to the trust issues, it would be interesting to explore for which cases assuming a semi-trusted environment is acceptable and when a more byzantine setting is required. As the need for securing log information in various use cases grows, we see this distributed solution as a novel approach to address the security and availability problem of the data. At a broader level, this technology can be a way to ensure the integrity and availability of critical data in many systems.

Author Contributions: Conceptualization, L.S. and E.W.; methodology, L.S. and E.W.; software, L.S.; writing—original draft preparation, L.S. and E.W.; writing—review and editing, L.S. and E.W.; visualization, L.S. and E.W.; supervision, E.W.; project administration, E.W.; funding acquisition, E.W. All authors have read and agreed to the published version of the manuscript.

Funding: The Research and APC was funded by Nokia Bell Labs.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. PCI Security Standards Council. Available online: https://zh.pcisecuritystandards.org/index.php (accessed on 26 May 2021).
2. Health Insurance Portability and Accountability Act. Available online: https://www11.anthem.com/networkupdate/articles/archive/oct2017_healthinsurance.html (accessed on 26 May 2021).
3. The General Data Protection Regulation. 2018. Available online: https://www.gdpr-info.eu/ (accessed on 26 May 2021).
4. Liang, Y.; Zhang, Y.; Sivasubramaniam, A.; Jette, M.; Sahoo, R. Bluegene/l failure analysis and prediction models. In Proceedings of the International Conference on Dependable Systems and Networks, Philadelphia, PA, USA, 25–28 June 2006; p. 425.
5. Frei, A.; Rennhard, M. Histogram Matrix: Log File Visualization for Anomaly Detection. In Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, Barcelona, Spain, 4–7 March 2008; pp. 610–617.
6. Goldstein, M.; Raz, D.; Segall, I. Experience Report: Log-Based Behavioral Differencing. In Proceedings of the 2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), Toulouse, France, 23–26 October 2017; pp. 282–293.

7. Schneier, B.; Kelsey, J. Secure audit logs to support computer forensics. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **1999**, *2*, 159–176. [CrossRef]
8. Zawoad, S.; Dutta, A.K.; Hasan, R. SecLaaS: secure logging-as-a-service for cloud forensics. In Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, Hangzhou, China, 8 May 2013; pp. 219–230.
9. Simmons, C.; Jones, D.; Simmons, L. A Framework and Demo for Preventing Anti-Computer Forensics. *Issues Inf. Syst.* **2011**, *12*, 366–372.
10. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 26 May 2021).
11. Castro, M.; Liskov, B. Practical Byzantine fault tolerance. *OSDI* **1999**, *99*, 173–186.
12. Schneier, B.; Kelsey, J. Cryptographic Support for Secure Logs on Untrusted Machines. *USENIX Symp.* **1998**, *98*, 53–62.
13. Möller, S.; Perlov, C.; Jackson, W.; Taussig, C.; Forrest, S. A polymer/semiconductor write-once read-many-times memory. *Nature* **2003**, *426*, 166. [CrossRef] [PubMed]
14. Holt, J. Logcrypt: Forward Security and Public Verification for Secure Audit Logs. Available online: https://eprint.iacr.org/2005/002.pdf (accessed on 26 May 2021).
15. Tian, F. An agri-food supply chain traceability system for China based on RFID & blockchain technology. In Proceedings of the International Conference on Service Systems and Service Management (ICSSSM), Kunming, China, 24–26 June 2016; pp. 1–6.
16. Tsai, W.T.; Deng, E.; Ding, X.; Li, J. Application of Blockchain to Trade Clearing. In Proceedings of the 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), Lisbon, Portugal, 16–20 July 2018; pp. 154–163.
17. Underwood, S. Blockchain beyond bitcoin. *Commun. ACM* **2016**, *59*, 15–17. [CrossRef]
18. Kokoris-Kogias, E.; Alp, E.; Siby, S.; Gailly, N.; Gasser, L.; Jovanovic, P.; Syta, E.; Ford, B. CALYPSO: Auditable Sharing of Private Data over Blockchains. Available online: https://eprint.iacr.org/eprint-bin/getfile.pl?entry=2018/209&version=20180806:124914&file=209.pdf (accessed on 26 May 2021).
19. Angiulli, F.; Fassetti, F.; Furfaro, A.; Piccolo, A.; Saccà, D. Achieving Service Accountability Through Blockchain and Digital Identity. In Proceedings of the International Conference on Advanced Information Systems Engineering, Tallinn, Estonia, 11–15 June 2018; pp. 16–23.
20. D'Angelo, G.; Ferretti, S.; Marzolla, M. A Blockchain-based Flight Data Recorder for Cloud Accountability. In Proceedings of the Workshop on Cryptocurrencies and Blockchains for Distributed Systems, Munich, Germany, 15 June 2018; pp. 93–98.
21. Gipp, B.; Kosti, J.; Breitinger, C. Securing Video Integrity Using Decentralized Trusted Timestamping on the Bitcoin Blockchain. In Proceedings of the Mediterranean Conference on Information Systems (MCIS), Guimarães, Portugal, 8–10 September 2016; p. 51.
22. Cucurull, J.; Puiggalí, J. Distributed immutabilization of secure logs. In *Security and Trust Management*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 122–137.
23. Sun, J.; Yao, X.; Wang, S.; Wu, Y. Blockchain-based secure storage and access scheme for electronic medical records in IPFS. *IEEE Access* **2020**, *8*, 59389–59401. [CrossRef]
24. Fatokun, T.; Nag, A.; Sharma, S. Towards a Blockchain Assisted Patient Owned System for Electronic Health Records. *Electronics* **2021**, *10*, 580. [CrossRef]
25. Ozdayi, M.S.; Kantarcioglu, M.; Malin, B. Leveraging blockchain for immutable logging and querying across multiple sites. *BMC Med. Genom.* **2020**, *13*, 1–7. [CrossRef] [PubMed]
26. Rane, S.; Dixit, A. BlockSLaaS: Blockchain Assisted Secure Logging-as-a-Service for Cloud Forensics. In Proceedings of the International Conference on Security & Privacy, Jaipur, India, 9–11 January 2019.
27. Aniello, L.; Baldoni, R.; Gaetani, E.; Lombardi, F.; Margheri, A.; Sassone, V. A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In Proceedings of the European Dependable Computing Conference (EDCC), Geneva, Switzerland, 4–8 September 2017; pp. 151–154.
28. Putz, B.; Menges, F.; Pernul, G. A secure and auditable logging infrastructure based on a permissioned blockchain. *Comput. Secur.* **2019**, *87*, 101602. [CrossRef]
29. Shekhtman, L.; Waisbard, E. EngraveChain: Tamper-proof distributed log system. In Proceedings of the 2nd Workshop on Blockchain-enabled Networked Sensor, New York, NY, USA, 10 November 2019; pp. 8–14.
30. Daemen, J.; Rijmen, V. *The Design of Rijndael: AES-the Advanced Encryption Standard*; Springer: Berlin/Heidelberg, Germany, 2013.
31. Rivest, R.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
32. Wilkinson, S.; Boshevski, T.; Brandoff, J.; Buterin, V. Storj a Peer-to-Peer Cloud Storage Network. 2014. Available online: https://www.storj.io/storj2014.pdf (accessed on 26 May 2021).
33. Fabric CA User's Guide. Available online: https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html#revoking-a-certificate-or-identity (accessed on 26 May 2021).
34. Bellare, M.; Desai, A.; Pointcheval, D.; Rogaway, P. Relations among notions of security for public-key encryption schemes. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 23–27 August 1998; pp. 26–45.
35. Cachin, C. Architecture of the Hyperledger blockchain fabric. In Proceedings of the Workshop on Distributed Cryptocurrencies and Consensus Ledgers, Chicago, IL, USA, 25 July 2016.

36. The Ordering Service. Available online: https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html (accessed on 26 May 2021).
37. Huang, D.; Ma, X.; Zhang, S. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *50*, 172–181. [CrossRef]
38. Wang, C.; Chu, X. Performance Characterization and Bottleneck Analysis of Hyperledger Fabric. *arXiv* **2020**, arXiv:2008.05946.
39. Hyperledger Fabric 1.4.0 Performance Information Report. Available online: https://hyperledger.github.io/caliper-benchmarks/fabric/resources/pdf/Fabric_1.4.0_javascript_node.pdf (accessed on 26 May 2021).
40. Mell, P.; Harang, R.E. Lightweight packing of log files for improved compression in mobile tactical networks. In Proceedings of the 2014 IEEE Military Communications Conference, Baltimore, MD, USA, 6–8 October 2014; pp. 192–197.
41. Todd, B. *Creating a Logging Infrastructure*; Technical Report; SANS Institute: Bethesda, MD, USA, 2017.