A secure and auditable logging infrastructure based on a permissioned blockchain

Benedikt Putz, Florian Menges, Günther Pernul

University of Regensburg, Universitätsstrasse 31, D-93053 Regensburg, Germany

Abstract

Information systems in organizations are regularly subject to cyber attacks targeting confidential data or threatening the availability of the infrastructure. In case of a successful attack it is crucial to maintain integrity of the evidence for later use in court. Existing solutions to preserve integrity of log records remain cost-intensive or hard to implement in practice. In this work we present a new infrastructure for log integrity preservation which does not depend upon trusted third parties or specialized hardware. The system uses a blockchain to store non-repudiable proofs of existence for all generated log records. An open-source prototype of the resulting log auditing service is developed and deployed, followed by a security and performance evaluation. The infrastructure represents a novel software-based solution to the secure logging problem, which unlike existing approaches does not rely on specialized hardware, trusted third parties or modifications to the logging source.

Keywords: Log management, secure logging, log auditing, permissioned blockchain, digital forensics

Email addresses: `benedikt.putz@wiwi.uni-regensburg.de` (Benedikt Putz), `florian.menges@wiwi.uni-regensburg.de` (Florian Menges), `guenther.pernul@wiwi.uni-regensburg.de` (Günther Pernul)

1. Introduction

Log data is produced today by most information systems used in organizations. It provides information about regular events occurring on these systems, but may also contain indicators for malicious behavior or attacks such as denial of service attacks, malware activities and other types of attacks on an organization's infrastructure. Analysis of these logs helps prevent security breaches, or enables detection and subsequent damage control when an incident has taken place [1].

In case a breach is successful, it's desirable to identify the perpetrator in a forensic investigation and bring the responsible person to court. In practice however intruders may attempt to alter or delete log entries documenting the intrusion [2]. Besides being exposed to malicious modification, log records are also often processed during analysis, for example by SIEM systems [3]. To be successful in a trial, the organization must be able to provide an indisputable proof of integrity for the log evidence. This proof must guarantee that no modification occurred during processing, so that the evidence remains admissible in court.

The primary requirements for legally admissible digital evidence are *relevance* and *authenticity* [4, p. 658ff]. In order for a piece of evidence to be relevant, there should be a persistent chain of custody. Reliable and verifiable evidence generation, transmission and storage are part of this chain of custody and prerequisites for authentication of evidence in court [5]. As a result, verifiable generation procedures constitute a key requirement for auditable logging infrastructures.

Prior research has already developed various approaches to create and protect secure logs from intruders (see Section 2). A key aspect of these works is integrity preservation of evidence using write-only or access-protected storage. Recently developed blockchain technology provides a novel way to achieve these goals. Blockchain systems are highly redundant data stores with the purpose of maintaining an append-only log of transactions. Since data is shared with other independent organizations based on distributed consensus, it is tamper-resistant. If a majority of participants are honest, availability and integrity of stored data is maintained. Of particular interest to enterprise applications are permissioned blockchains, where the set of participants is authenticated.

Based on a permissioned blockchain, we develop a secure infrastructure to ensure integrity and non-repudiation of log events without a trusted service provider. It is designed to prove the existence of a log entry at the time of generation by using integrity proofs stored in a distributed auditing layer. The blockchain network storing the proofs is maintained by a consortium of independent operators. Auditors can verify the integrity of previously submitted evidence by contacting any node in the network. Automated signing, storage and integrity proof generation for each log event provide the necessary authentication and non-repudiation. For evaluation, we create a prototype as part of the DINGfest project [3]. DINGfest aims to create an open-source SIEM infrastructure and currently consists of three main components: *data acquisition*, *data analysis* and *forensics & incident reporting*. This work adds a fourth *data auditing* component to ensure forensic auditability.

The paper is structured as follows: After explaining prior work and some of its shortcomings, we propose a general design for secure logging based on a permissioned blockchain. For evaluation, we then build the prototype within the DINGfest infrastructure and describe our results regarding security and performance.

2. Related Work

Specialized hardware or software is required to achieve the aforementioned security goals of secure logging systems. Prior work on secure logging systems can be grouped into three categories: append-only storage systems, forward-secure evolving signatures and trusted third party (TTP) notary services [6]. Hardware-based write-only devices are a cost-intensive solution, especially when there is a large amount of continuously generated log data. For this reason we focus on software-based techniques hereafter.

Software-based approaches use cryptographic mechanisms to detect modifications of log files. One of the earliest works on secure logging for forensic investigations was published by Schneier and Kelsey [2]. Their proposed algorithm is used for concatenating the sequence of log events and provides forward security and verifiability. Ma and Tsudik [7] claim that the Schneier and Kelsey scheme is vulnerable to a truncation attack, where an attacker may delete entries starting with the most recent. To eliminate these flaws, they introduce a new public-verifiable forward-secure aggregation scheme. It removes the need for an additional server and reduces the storage overhead introduced by the MAC and hash chain. While such forward-secure logging schemes are tamper-evident regarding modification by intruders, they cannot prevent deletion of the evidence. Accorsi [8] provides an overview of existing software-based secure logging protocols and highlights this weakness. The work favorably emphasizes the author's "BBox" secure logging approach, which is claimed to be the only protocol to fulfill all security requirements for transmission and storage. It relies on trusted computing modules [9], which require special hardware and thus introduce additional cost. Finally, all aforementioned software-based techniques require additional logging software adjustments beyond transmission to a server, which may not be possible in all organizational scenarios.

An example for a TTP-based timestamping solution is the Keyless Signing Infrastructure operated by the Estonian security firm Guardtime [10]. It is based on generating integrity proofs without cryptographic keys by aggregating hashes from different sources [11]. Through several aggregation layers, hashes of log records are aggregated in a binary Merkle tree. The system operates in fixed time intervals and produces one aggregation tree per round. The tree root hashes are stored in a custom data structure referred to as a hash calendar. The calendar's root hash is regularly published in Estonian newspapers for public verifiability [12]. The KSI method is less prone to attacks from quantum-computing based algorithms since it only relies on hashing and uses no public-key cryptography [12]. Thanks to usage of several aggregation layers the system is also highly scalable. However, besides being closed-source and fee-based, the platform also has the disadvantage of relying on the security of Guardtime's infrastructure.

Other researchers have used blockchain to assure log integrity and auditability. For example, a recent work proposed publishing the KSI root hashes to the Bitcoin blockchain [13]. Cucurull et al. [6] developed a secure logging approach that uses the permissionless Bitcoin network to record checkpoints of local log chains. Sutton and Samavi [14] use a local graph database to store logs and submit integrity proof digests to Bitcoin for auditability. Since all these approaches rely on the Bitcoin blockchain, the scalability is limited by its block size and throughput. Besides the Bitcoin transaction fees, costs are also incurred through the requirement to maintain a full copy of the blockchain.

Other approaches use the permissioned blockchain framework Hyperledger Fabric[1]. Ahmad et al. [15] focus on tracking changes made to database entries by storing change excerpts on Hyperledger Fabric. Shekhtman and

[1]`https://www.hyperledger.org/projects/fabric`

Waisbard [16] store the contents of log files directly on Hyperledger Fabric. They demonstrate the feasibility of auditable logging based on a permissioned blockchain, but it is not clear whether their approaches are scalable, as no throughput and storage scalability benchmarks are presented. However, scalability is required to manage large volumes of logging data in practical secure logging deployments.

In contrast to the works described above, our proposed approach does not rely on specialized hardware, modification of the logging source or trusted third parties. Instead it uses a combination of replicated local storage and a permissioned blockchain to ensure availability and integrity. Using a permissioned blockchain over a permissionless one comes with several advantages. Permissioned blockchain systems allow for higher throughput on the order of hundreds to thousands of transactions per second [17]. Additionally, transaction costs can be avoided due to the restricted set of participants, which allows using deterministic consensus algorithms.

To summarize, we contribute to the literature by reducing cost and alleviating performance limitations of prior blockchain-based secure logging approaches. To achieve this, we rely on a high-performance and low-latency permissioned blockchain, with enhanced security provided by anchoring to a permissionless blockchain.

3. System design

Following the design science research methodology by Peffers et al. [18], we begin by elaborating requirements and objectives for the system. Based on these requirements, we consider the available options for storing data when using a blockchain system and describe a general architecture for a blockchain-based auditable logging system. We also discuss two options for operation of the blockchain network in practice.

3.1. Requirements and preliminaries

The overall objective of a secure logging system is maintaining availability and integrity of log files. Records created by the system should also have the property of non-repudiation, meaning that records verifiably correspond to an event that occurred on a specific system. Onieva et al. [19] define five phases of a non-repudiation service. We use four of these phases (shown in Figure 1) to guide system design in the following chapter. In secure logging, verification occurs during dispute resolution, since tampered log records may also contain valuable information about system compromise.

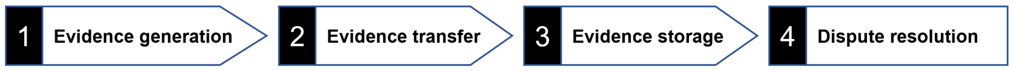


Figure 1: Phases of a non-repudiation service, adapted from Onieva et al. [19].

A prerequisite for all phases is the definition of what actually constitutes evidence. For information system logs, any record may be possibly relevant evidence. To separate availability and integrity preservation, we split log records into of *evidence data* and an *integrity proof*. The evidence data consists of the actual log event and associated metadata. Specifically, it consists of log event information, a generating system identifier & signature and a timestamp. The signature is generated by the source system or the hypervisor, if the data was extracted using Virtual Machine Introspection. The integrity proof is represented by a timestamped hash of the evidence data that confirms the existence of the log event at a specified time. If this information is stored immutably, it can later be used to prove that evidence data has not been changed since creation of the proof.

Evidence generation takes place on log sources, for example systems that are connected to external networks and vulnerable to intrusion. The log files should include digital signatures signed with the private key of the generating system. In our work, a unique public/private key pair is generated for each system. Certificates issued by a public key management authority enable attribution of log events to sources.

During the **evidence transfer** phase, the log event is then transferred from its source to the storage system. Using an encrypted connection is imperative to maintain the chain of custody. In our solution, the transmission is subject to a number of requirements established in prior work: confidentiality, origin authentication, integrity, uniqueness and reliable delivery [8].

Evidence storage is the main focus of this work. Any log data ingested by the service should be stored in a way that preserves availability and integrity for later use. This includes preventing deletion or modification by an attacker seeking to erase traces. Confidentiality is also a concern due to potentially sensitive information contained in log records. Prior solutions described in Section 2 use specialized append-only hardware, third-party providers or local storage combined with external notary systems. Our approach achieves immutability by storing integrity proofs on a blockchain network consisting of independent nodes.

Dispute resolution occurs when an intrusion has taken place and has been recorded in log files. Both intruder and victim may attempt to deny the authenticity of the evidence. An auditor must then be able to verify that the log was not modified by anyone since generation. This evidence verification consists of signature and integrity proof verification. For signature verification, the auditor must have access to the corresponding public key certificates of the generating sources. The integrity proof is represented by a hash and a corresponding timestamp in our work and must be stored in provably unmodifiable storage. It ensures that the signed file already existed at the claimed time.

3.2. Storage design considerations

To alleviate shortcomings of prior secure logging approaches discussed in Section 2, we consider an architecture using a permissioned blockchain network. The advantage of using a permissioned network is that it does not rely on a paid third party service provider. Instead, the blockchain node operators provide the immutability service for each other. Since the operational cost is evenly shared by all members, no additional costs arise besides operating the network.

We now discuss how a blockchain-based solution could help to store log files in an integrity-preserving way. Data can be stored either on-chain or off-chain in blockchain-based solutions. On-chain storage would imply replicating the full log data across all blockchain nodes. Logging infrastructures deal with considerable data volumes, so full replication to each node would be inefficient and lead to high storage costs. An additional concern with on-chain storage is loss of privacy. Log data may inadvertently contain sensitive data like usernames or even password hashes. Since blockchain node operators are independent, they should not share potentially sensitive log data. Off-chain storage maintains data in a separate local database to uphold privacy and confidentiality requirements. Off-chain data can be linked to the corresponding on-chain transaction through its hash, provided that the data has not been modified since its hash was included in the blockchain.

To accommodate the storage limitations of blockchains, we split the log storage into on-chain and off-chain parts as suggested by Barger et al. [20]. Evidence data is stored in a local storage cluster and protected from unauthorized access to maintain confidentiality. Availability protection for the off-chain data is achieved through local replication. Commodity hardware can be used to cheaply store log data while avoiding data loss. Apache Kafka[2] is one example for a suitable publish-subscribe system that maintains a locally replicated and crash-tolerant log, while allowing other applications to interact with the data [21]. To be able to detect potential corruption or modification of the off-chain data, integrity proofs are stored in transactions on the permissioned blockchain. The network is maintained by independent operators to ensure the proofs cannot be modified by any one participant. Since these operators are only semi-trusted, the network should be able to tolerate some amount of arbitrary, even malicious, behavior. These types of faults in distributed systems are also referred to as byzantine faults [22]. A byzantine-fault tolerant (BFT) consensus algorithm is used in our solution to tolerate up to f byzantine failures in a network of $3f + 1$ nodes.

BFT state machine replication can be implemented without a blockchain data structure to gain some throughput performance. However, secure logging requires additional authenticity and integrity guarantees as mentioned in Section 3.1. Permissioned blockchain frameworks provide these guarantees and come with other beneficial features such as audit-only nodes and APIs for external applications, so we build on them in our system architecture.

3.3. System architecture

The full architecture is shown in Figure 2. Evidence is generated and signed by different sources like containerized applications, firewalls or intrusion detection systems. The generated evidence data is securely transferred to a log verification system. Existing protocols based on reliable syslog (IETF RFC 3195 [23]) fulfill the requirements for secure transmission [8] and can be used for this purpose.

Newly arriving data in the storage cluster is monitored by a separate application. For each new entry, a transaction is generated containing a hash and the current timestamp. The transaction is included in a block together with transactions arriving from other nodes in the distributed system. A new block is appended to the blockchain as soon as enough transactions are available, or when a timeout is reached. The timeout ensures that the delay between log generation and inclusion in the blockchain remains low. The other participating organizations also add transactions containing hashes

[2]`https://kafka.apache.org/`

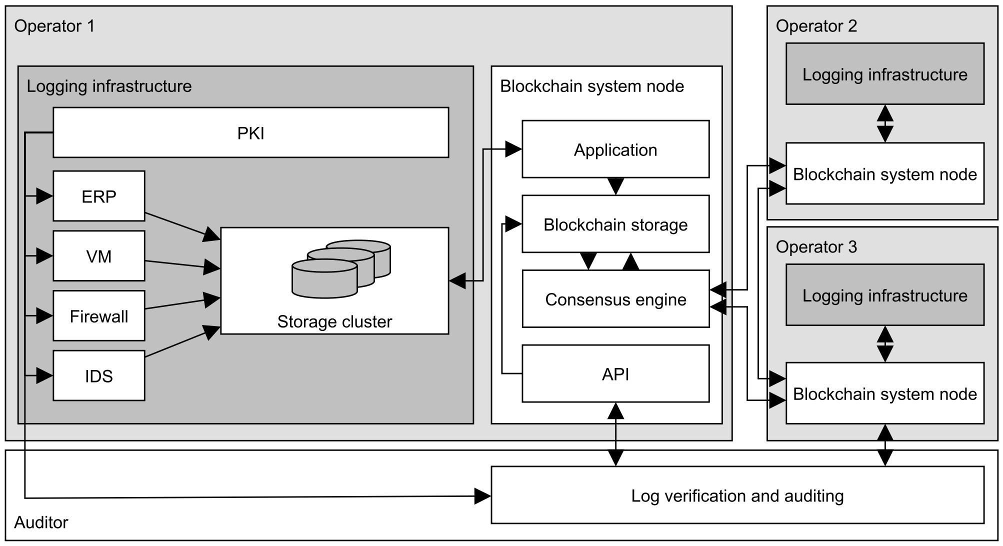


Figure 2: Proposed design for a secure and auditable logging infrastructure.

from their own logging infrastructures.

Any auditor is able to verify evidence at a later date. The actual evidence data remains in the replicated local storage and can be provided to the auditor at request. To verify the integrity of the evidence data, the auditor first verifies its signature against valid public key certificates from the PKI. This step ensures the log file can be mapped to its source. Afterwards the data's hash is submitted to a blockchain node for verification. The server then compares the hash values stored in blockchain transactions with the hash value of the proposed evidence. If an identical hash is found, there is non-repudiable proof that identical data was submitted at an earlier time. To tolerate possibly corrupted blockchain nodes, the proof can also be requested from multiple blockchain nodes independently.

3.4. Formal logging procedure

Figure 3 details the data flows that occur when each log entry is processed. Two processing steps **S1, S2** occur, with an optional verification step **S3**. To describe these steps, we define the following formal notation. Each organization $j \in \{1..n\}$ has an identifier ID_j, a private signing key Z_j and a corresponding public key K_j.

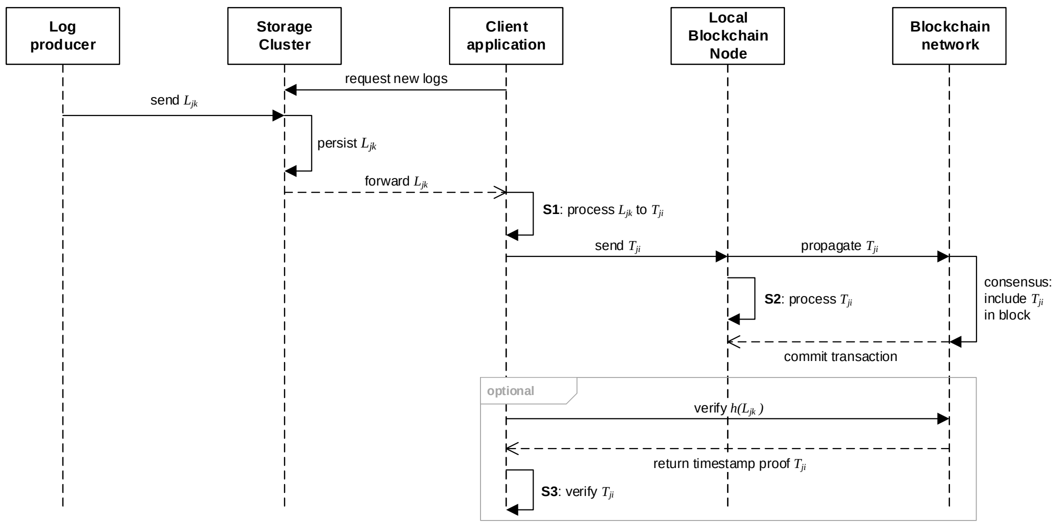


Figure 3: Sequence diagram of log entry processing data flows.

S1: Client application processing. We first define a log counter $k \in 1..l$ and a transaction counter $i \in 1..m$. Each newly arriving log entry L_{jk} is parsed to create a transaction payload P_{ji}, which is included in a new transaction T_{ji}. Initially, $i = k$, as one transaction is generated per log entry. $hash()$ refers to a preimage-resistant hash function, while $sign()$ is shorthand for a public-key signature function. It is assumed that T_{ji} is transmitted from client application to blockchain node via an encrypted channel, preventing man-in-the-middle attacks.

$$T_{ji} = (P_{ji}, H_{ji}, S_{ji}) \quad where$$
$$P_{ji} = (ID_j, K_j, L_{jk}),$$
$$H_{ji} = hash(L_{jk}),$$
$$S_{ji} = sign(Z_j, P_{ji})$$

S2: Blockchain node processing. After the transaction was propagated to the network and proposed as part of a block by the current consensus leader, the actual processing takes place on each node. A timestamp based on distributed consensus is included with each transaction and the uniqueness of the log entry is verified by each node:

$$i > 1 : H_{ji} \notin \{H_{j1}..H_{j(i-1)}\} \quad \forall j \in 1..n$$

If this condition fails, no changes to the persistent state occur as a result of the transaction. Two identical log entries with the same hash value can not be part of the system. If the condition passes and the entry is unique, a mapping of content hash to blockchain transaction hash is added to a dictionary: $(H_{ji}, hash(T_{ji}))$. This permits efficient lookups during verification. After more than two-thirds of all nodes have successfully processed the transaction, it is irreversibly committed to the ledger. We omit consensus protocol message exchanges for clarity in Figure 3.

S3: Verification. To verify, a user may optionally submit a log entry to the client application, which requests a proof from the blockchain network by sending $hash(L_{jk})$. If the corresponding transaction is found and returned, only its signature and hash must be verified:

$$verify(K_j, P_{ji}, S_{ji}) \quad \wedge \quad K_j \in \{K_1..K_n\} \quad \wedge \quad H_{ji} \overset{!}{=} hash(L_{jk})$$

$verify()$ is the verification function corresponding to $sign()$, returning 1 for a correct signature and 0 otherwise.

3.5. Blockchain network operation

For illustrative purposes only three nodes are shown in Figure 2. An actual deployment in practice must consist of $n \geq 4$ nodes to be able to tolerate at least one byzantine node (see Section 5.2 for optimal node counts). Nodes should be operated by separate organizations to make it harder for both the organization and attackers to modify data on the blockchain. If a single party controls the supermajority of nodes, it could simply replace and fabricate data, forfeiting the immutability benefits of a blockchain system. The blockchain network consortium could be coordinated by industry associations. In that case independent organizations within the association would jointly maintain a network of blockchain nodes, as illustrated in Figure 2.

Another alternative is operating the network within a single organization and spreading the nodes across multiple locations or organizational units. In that case the organization is in control of all nodes and an intruder would have to subvert nodes in multiple locations to remove traces. The organization itself could however initiate a coordinated replacement of blockchain data. This might be a concern in investigations where organizations try to hide a breach of their infrastructure. Adding an anchoring mechanism to the permissioned blockchain mitigates this possibility. Anchoring includes the latest block hash of the permissioned blockchain network in a transaction on a permissionless system like Bitcoin. These checkpoint transactions are submitted in regular intervals and cost a small fee. This allows external auditors to publicly verify the state of the private blockchain at the time of anchoring.

The advantage of this approach is that the entire logging infrastructure remains within the organization. At the same time it also incurs transaction fees similar to the permissionless blockchain approach [6]. Depending on the anchoring frequency this cost can add up to a substantial amount. While lowering the frequency decreases cost, it also widens the time window for possible replacement of blockchain data by the organization. To prevent this entirely, the time between checkpoints must be lower than the time an internally coordinated blockchain replacement would take.

4. Prototype

For demonstration and evaluation purposes, we create a prototype based on a SIEM reference architecture. We build on the DINGfest infrastructure created in prior work, which implements some parts of the design described above, like the storage cluster. It however currently lacks a way to ensure end-to-end integrity, auditability and non-repudiation of the original log evidence. The prototype adds this capability in the form of a blockchain-based distributed log auditing service. The service is then evaluated for security concerns and throughput/storage scalability.

The extended architecture is shown in Figure 4. A hashing application fetches log records from the data stream and computes the SHA256 hash for each record. The input data for the hash includes the log data and its signature. The hash is then submitted to the blockchain system in a signed transaction. The receiving blockchain node validates the hash against existing hashes in the database to prevent duplicates. A timestamp is obtained in consensus with other validators and included with the transaction. Finally, a separate proof application provides a web interface, where auditors can submit evidence for validation (see also Figure 5).

We evaluate various open source permissioned blockchain frameworks for our prototype. The evaluation criteria we deem essential for secure logging are shown in Table 1. First, we examine whether the framework currently offers a production-ready BFT consensus implementation. For performance reasons, we also want to avoid the virtualization overhead of smart contracts

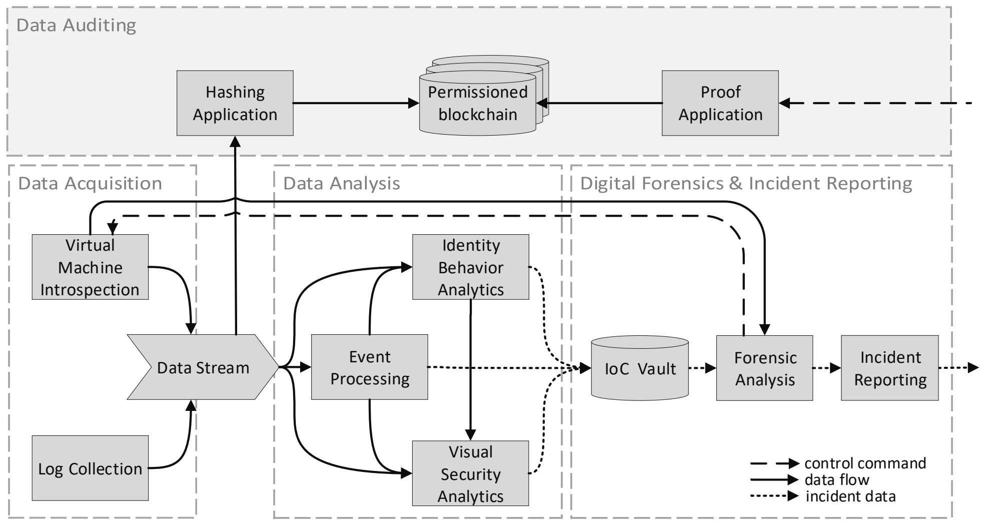


Figure 4: The DINGfest SIEM architecture extended with an auditing layer (based on Menges et al. [3]).

Table 1: Comparison of open source blockchain framework properties.

	Fabric	Sawtooth	Corda	Ethereum	Exonum
BFT consensus	(✓)	(✓)	(✓)	(✓)	✓
Native contract execution	-	✓	-	-	✓
Consensus-based anchoring	-	-	-	(✓)	✓
Deployment effort	high	medium	high	low	low

✓ built-in (✓) custom implementation required/experimental - not available

and thus look for frameworks offering native execution of custom logic. To protect against collusion manipulation of the permissioned blockchain, we include consensus-based anchoring to a permissionless blockchain as a criterion. To ensure a low barrier to entry, deployment effort is assessed by analyzing the number of different services and containers needed for running the framework.

We conclude that there are several advantages to using Exonum[3] over other permissioned frameworks. Firstly, other frameworks do not yet offer

[3] https://exonum.com

ready-for-use BFT consensus implementations or consensus-based anchoring. Additionally, Exonum does not use conventional smart contracts running in a virtualized execution environment. Instead, it uses natively executed *services* to implement custom logic. Similar to smart contracts, services are invoked by transactions and executed on every blockchain node, but service code must be final at compile-time and cannot be deployed dynamically at runtime. Services include the Exonum framework as a dependency and are compiled to a single binary containing both application and framework. For this reason there is no performance overhead due to virtualization. The binary can also be deployed easily without the need to maintain multiple separate containers (like in Fabric/Sawtooth/Corda).

The Exonum framework and log auditing service are implemented in Rust, a functional systems programming language focused on memory safety, concurrency and performance [24]. The framework uses a byzantine fault-tolerant consensus algorithm based on PBFT (see [22]) and supports throughput rates of up to 7,000 transactions per second [25]. High throughput is important to ensure timely inclusion of each log record's integrity proof in the blockchain. The Exonum framework also provides built-in services for distributed timestamps and Bitcoin anchoring. Anchoring to the permissionless Bitcoin blockchain increases security by providing publicly verifiable checkpoints (also discussed in Section 5.1). By using a permissioned blockchain, the process of deciding on the next candidate block for anchoring is based on consensus and avoids a single points of failure [25]. Since BFT consensus is the key argument for using blockchain in our architecture, we also choose Exonum for its low overhead approach to blockchain. The features it adds apart from PBFT consensus are all useful to the proposed solution. Encrypted node connections maintain confidentiality, consensus-based distributed timestamps ensure non-repudiation of log creation time and block grouping of transactions improves consensus performance.

The prototype consists of an Exonum backend **service** and a light **client** web application, as shown in Figure 5. The backend service runs on version 0.11 of the Exonum framework[4].

The **frontend client** provides interfaces for blockchain inspection, monitoring and verification of log data. It retrieves data from the blockchain using the server-side backend, which redirects the read requests to the Exonum

[4] `https://github.com/exonum/exonum/releases/tag/v0.11.0`

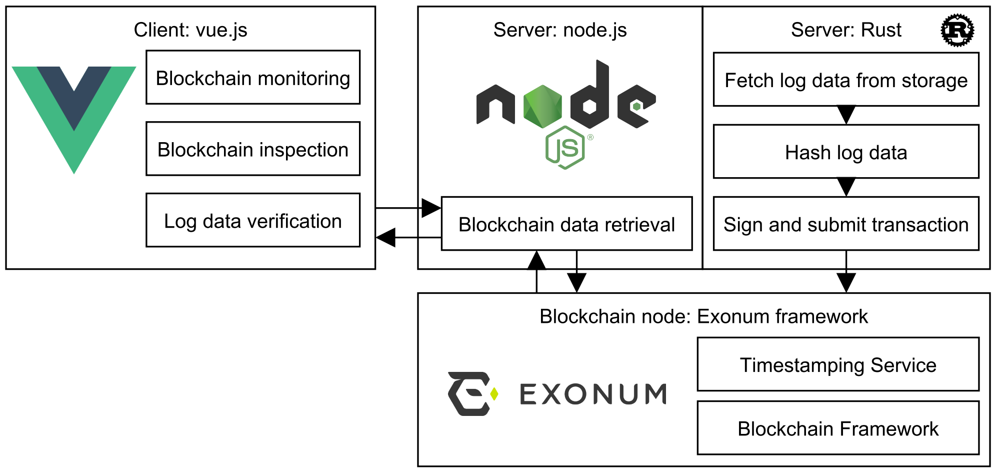


Figure 5: Client-server architecture of the application prototype.

blockchain API. The server also runs a separate background application written in Rust, which continuously receives new log events from the distributed storage cluster based on Apache Kafka. The signed log data is hashed and submitted to the blockchain in a transaction. The blockchain's log auditing service verifies that the hash does not already exist in the blockchain and groups arriving transactions into blocks. New blocks are appended to the blockchain by the framework after establishing consensus with the other nodes.

The **backend service** runs on Exonum blockchain nodes and specifies the transaction data model and the available API endpoints for the light client to interact with. Our prototype reuses the timestamping service example provided by the framework, which meets all requirements of the system design and formal specification.

In practice, each network participant would deploy an instance of the blockchain node and client application on a local server. To deploy the blockchain node, participants must agree on a shared configuration file, which includes parameters like block proposal timeout and transactions per block (further described in Section 5.2). Each node's individual configuration file is then generated locally based on its private key and IP address and public key of the other nodes. The node binary, which includes both the framework and the log auditing service, is then started and ready to receive transactions.

Finally, the client application must be configured to continuously receive log records from the local storage cluster. After every participant has finished the setup phase, the system may begin operation.

5. Evaluation

Crucial aspects of the design that should be evaluated are *security* and *performance*. System security is important since vulnerabilities to attacks may question the very purpose of the infrastructure. Performance considerations are important as well to ensure scalability for larger organizations, especially since blockchain systems are known for their scalability limitations. The security evaluation is based on a structured analysis of threats, while the performance evaluation focuses on throughput and storage metrics in a cloud-based deployment of the prototype.

5.1. Security

There are four fundamental security threats to consider in a distributed system: *interception*, *interruption*, *modification* and *fabrication* [26].

Interception could lead to delays in committing timestamps to the blockchain. Intercepting a node's connection is impeded by setting up authenticated and encrypted communication channels between nodes.

Interruption could be achieved for example with a denial of service attack. By preventing the node on the target network from communicating with other nodes, an attacker can delay outstanding log transactions. Alternatively, an intruder could attempt to gain control of blockchain nodes. Once enough nodes are compromised, it becomes possible to stall consensus or fully control the network. The exact number of nodes depends on the number of validators in the network. With a byzantine fault-tolerant (BFT) consensus algorithm, a minimum of $1/3$ of all nodes is required to stall consensus and $> 2/3$ to gain network control by adding/removing validator nodes. These bounds are based on the fact that more than two-thirds of validators must agree to commit a transaction in BFT consensus [22]). Anchoring to a permissionless blockchain would prevent this type of attack entirely. The permissioned blockchain's contents can then be verified against checkpoints published on the public blockchain to detect manipulations.

Data modification is the most significant threat to log evidence. An intruder could attempt to modify the original log record to obscure traces. As noted by Schneier and Kelsey [2], once an attacker has control of the log source, the integrity of new logs cannot be protected. The goal of secure logging is therefore to protect log data generated prior to intrusion. Our proposed two-layer approach protects availability using the replicated storage cluster and integrity with proofs on the blockchain. To void availability of the original log records stored on the local storage cluster, the attacker would have to corrupt or delete all copies. To modify the integrity proofs on the blockchain, the attacker has to subvert or convince more than two-thirds of all nodes due to the properties of BFT consensus [22]. Both scenarios require significant penetration of the organization's infrastructure and are rather unlikely in practice.

Fabrication of log entries is a concern and may invalidate authentication of the evidence. An intruder could fabricate false log entries to lead investigators astray. To that end it is necessary to gain control of the system first and compromise its private key for log generation. As recognized in Schneier et al. [2], no security measure can protect log files generated after a system has been compromised. Log files generated prior to intrusion should allow to identify the time of compromise, after which logs can no longer be considered authentic. Additionally, the organization maintaining the log infrastructure could also attempt to fabricate entries to falsely blame an adversary. This cannot be prevented, since the organization is in control of its private keys and can submit arbitrary data at any point in time signed with these keys. This possibility of organizational fabrication instead has to be excluded by legal means.

Regarding correctness of the verification, there is a non-zero probability that a fake log submitted for verification has the same hash as another valid log entry included in the blockchain. This type of hash collision can be neglected in practice however. There are 2^{256} possible hash outputs for the SHA256 algorithm used in the prototype, for which no computationally feasible collision has been found [27].

From a privacy perspective, the log contents remain confidential since only a hash of the data is stored on the blockchain. For preimage resistant hash functions like SHA256 it is (by current standards) impossible to find the content of the original log file.

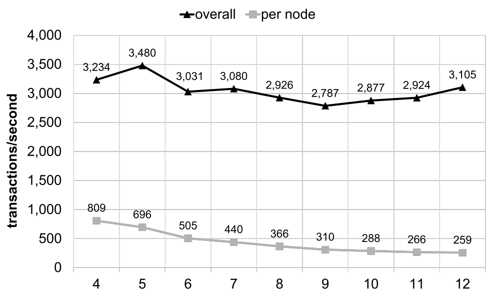
Additionally there are some operational security concerns regarding the prototype's blockchain framework. Exonum has not received any security audits, so independent validation of framework security is not yet available. Another limitation is the lack of built-in authentication and authorization for service endpoints. By default, anyone can query both private and public endpoints. This is not a critical problem for our infrastructure. If the hashing application is deployed to a separate system or container, it can be whitelisted as the only external system to use the port of the public blockchain API. Similarly, the system administrator's IP should be the only IP allowed to access the private port, since it could be used to add new peers or shut down the node. Whitelisting can be done by using the AWS security groups, iptables on Linux or a different firewall solution.

5.2. Performance

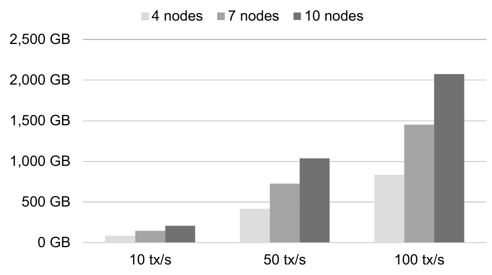
The three main concerns regarding the performance of the logging infrastructure are transaction throughput, node scalability and storage requirements. The bottleneck of the system is clearly the blockchain-based auditing layer due to its limited transaction throughput spread across all participants. Thus we set up blockchain networks of varying sizes on the DigitalOcean cloud to benchmark the prototype. We used droplet instances with 4 dedicated virtual CPU cores and 8GB of RAM. All virtual machines are set up within the same data center location. We deploy both the blockchain node and the hashing application on the same server. Consequentially, the overhead from signing and submitting transactions to the blockchain is included in the performance measurements. After some experimentation with consensus parameters we settled on a default configuration to a maximum of 10,000 transactions per block and a proposal timeout of 1 second. These parameters seemed to alleviate any bottlenecks related to system configuration. For stress testing we submit 5,000 log transactions per second for 20, 30 and 40 seconds at a time. The transaction load is evenly distributed across all nodes. Higher loads did not increase throughput and only resulted in increased processing time after the end of transaction submission. The average observed throughput rates are shown in Figure 6a.

The results show that overall system performance varies between 3,000 and 3,500 transactions per second on average, yielding between 250 and 800 processed transactions per node and second. Per-node performance continually decreases, while overall performance remains fairly stable even for a large number of nodes. The per-node performance can be interpreted as limits on the average transaction load that the system should be exposed to in each organization.

Given the per-node performance thresholds between 250 and 800 transactions per second, it is not unlikely that the log event rate exceeds system



(a) Transactions per second for varying node counts.



(b) Storage requirements for varying average throughput and node counts.

Figure 6: Prototype performance benchmark results.

capacity. In that case a possible solution is to group log records in transactions. In step **S1** from Section 3.4, the hash H_{ji} is then computed using r concatenated log records L_{jk} as input:

$$H_{ji} = hash(L_{j[(i-1)r+1]} \mid .. \mid L_{j[(i-1)r+r]})$$

The resulting load on the blockchain is reduced by a factor of $\frac{1}{r}$. However, hash verification in step **S3** changes accordingly to also require r records as input. The sequence number of the log entry may not be known to the verifier, requiring r verification requests and $2(r-1)$ additional log entries for verification (before and after the log at hand). This highlights the disadvantage to this solution: grouped records are now treated as one integrity unit. If one of the records in the unit is corrupted or no longer available, it becomes impossible to prove the integrity of all the records in the unit.

Fault tolerance maxima are achieved when the node count is equal to $3f + 1$, tolerating f byzantine or failing nodes. As a result, node counts of 4/7/10 nodes can be considered efficient in Figure 6a, tolerating 1/2/3 byzantine nodes respectively.

Figure 6b illustrates blockchain database growth per month, based on the measured average size of 800 bytes per transaction. Since the size of the data fields is fixed, all transactions have approximately the same size. The database grows linearly with both throughput and node count. To avoid excessive storage usage, log record transactions on the blockchain may be discarded after d days, as defined by the organizational log retention policy. The ideal technical solution would be a rolling blockchain, configured to discard blocks once they are d days old. The log retention period must then be agreed upon by all node operators in advance. A rolling blockchain system has been proposed in Dennis et al. [28], but no open source implementation is currently available. For this reason we leave this issue for future research.

5.3. Comparison with other blockchain-based secure logging approaches

Table 2: Comparison with other blockchain-based approaches.

Paper	Blockchain	Permissioned	BFT	Benchmarks	Proof/data sep.	Per-entry imm.
Cucurull and Puigalli [6]	Bitcoin	-	n.a.	-	✓	-
Sutton and Samavi [14]	Bitcoin	-	n.a.	-	✓	✓
Ahmad et al. [15]	Fabric	✓	-	-	-	✓
Shekthman and Waisbard [16]	Fabric	✓	-	-	-	✓
Present work	Exonum	✓	✓	✓	✓	(✓) [a]

[a]without log grouping from Section 5.2

To validate our results, we compare them with similar blockchain-based logging approaches. Prior works on blockchain-based secure logging have used both permissionless (Bitcoin) and permissioned (Fabric) blockchains, but none of them have proposed an integrated approach based on anchoring. While our solution is primarily based on the permissioned blockchain Exonum, it also provides the option for consensus-based anchoring to the Bitcoin blockchain to increase tamper-resistance.

With regard to evaluation, none of the other works thus far have included benchmarks for throughput and storage constraints. Since these constraints are crucial for high frequency logging infrastructures, we have addressed them in Section 5.2. Besides conducting benchmarks, we also proposed avenues for increasing scalability by grouping log records or using a rolling blockchain.

Due to limited storage space in Bitcoin transactions, both Bitcoin-based approaches separate the integrity proof and the log data. The works based on Hyperledger Fabric include partial [15] or full [16] log data on the blockchain, leading to limited scalability. Our approach is also based on a permissioned blockchain and focuses on storing a minimal amount of data on-chain.

Per-entry immutability refers to generating one blockchain transaction per log event. Sutton and Samavi [14] follow this approach, but use the Bitcoin blockchain, which leads to very high transaction costs and throughput limitations. These cost constraints lead Cucurull and Puigalli [6] to publish checkpoints instead of individual log entries. However, between checkpoints logs exist only on the local machine, which enables truncation attacks during the checkpoint interval. Mitigation is possible by reducing the checkpoint interval, but at the same time increases transaction costs. Permissioned systems do not suffer from this constraint and offer per-entry immutability, as evident from the works using Hyperledger Fabric [15, 16]. Our solution combines the advantages of both solutions: it gains per-entry immutability by using a permissioned blockchain and public non-repudiation by publishing checkpoints to a permissionless blockchain, witnessed by hundreds of independent nodes.

6. Conclusion

This paper presents an infrastructure for log auditing using a permissioned blockchain to store integrity proofs. It is based on legal requirements for admissible evidence and represents an on-premise alternative to third-party solutions and specialized write-only hardware. Even without a third-party service provider, the solution achieves immutability through cooperation and data sharing between independent nodes. It permits processing of evidence for security analytics purposes while ensuring auditability of the original log record.

The security analysis shows that the system withstands attempts to intercept, interrupt or modify its processed log data. While fabrication is a concern, it cannot be completely ruled out with technical measures. Performance benchmarks show that the blockchain implementation is able to cope with very high log event frequencies of 3,000 to 3,500 transactions per second, depending on the number of nodes. Storage requirements are substantial however due to full replication and retention of all historical data.

In future research, an enhanced prototype could implement log rotation to reduce storage costs, as outlined in Section 5.2. While we firmly believe that Exonum currently offers the best performance for our secure logging approach, this might change in the future. In that case the prototype could also be implemented in other frameworks to see if scalability can be further improved. Practical deployments by organizations would be useful to assess adequacy of system throughput in practice. Use case studies to compare the cost structure of the proposed logging infrastructure with third-party solutions like the KSI [10] would also be valuable.

Acknowledgement This research was supported by the Federal Ministry of Education and Research, Germany, as part of the DINGfest project (`https://dingfest.ur.de`).

References

[1] H. S. Venter, J. H. P. Eloff, A taxonomy for information security technologies, Computers and Security (2003). doi:10.1016/S0167-4048(03)00406-1.

[2] B. Schneier, J. Kelsey, Secure audit logs to support computer forensics, ACM Transactions on Information and System Security 2 (1999) 159–176. doi:10.1145/317087.317089.

[3] F. Menges, F. Böhm, M. Vielberth, A. Puchta, Introducing DINGfest: An architecture for next generation SIEM systems, in: SICHERHEIT 2018, Gesellschaft für Informatik e.V., 2018, pp. 257–260.

[4] H. Bidgoli, Handbook of information security, John Wiley, Hoboken, NJ, 2006.

[5] E. Casey, Digital evidence and computer crime: forensic science, computers and the Internet, Academic Press, Waltham, MA, 2011.

[6] J. Cucurull, J. Puiggalí, Distributed immutabilization of secure logs, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 9871 LNCS (2016) 122–137.

[7] D. Ma, G. Tsudik, A new approach to secure logging, ACM Transactions on Storage 5 (2009) 1–21. doi:10.1145/1502777.1502779.

[8] R. Accorsi, Log data as digital evidence: What secure logging protocols have to offer?, Proc. of the International Computer Software and Applications Conference 2 (2009) 398–403. doi:10.1109/COMPSAC.2009.166.

[9] R. Accorsi, A secure log architecture to support remote auditing, Mathematical and Computer Modelling 57 (2013) 1578–1591. doi:10.1016/j.mcm.2012.06.035.

[10] Guardtime, KSI Technology — Industrial Scale Blockchain — Guardtime, 2018. URL: https://guardtime.com/technology.

[11] A. Buldas, A. Kroonmaa, R. Laanoja, Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees, in: Proc. of the

Nordic Conference on Secure IT Systems, 2013, pp. 313–320. URL: https://eprint.iacr.org/2013/834.pdf.

[12] A. Buldas, R. Laanoja, A. Truu, Efficient Quantum-Immune Keyless Signatures with Identity., IACR Cryptology ePrint Archive (2014) 321.

[13] C. Jämthagen, M. Hell, Blockchain-Based Publishing Layer for the Keyless Signing Infrastructure, in: 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016, pp. 374–381. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0072.

[14] A. Sutton, R. Samavi, Blockchain Enabled Privacy Audit Logs, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2017, pp. 645–660. URL: http://link.springer.com/10.1007/978-3-319-68288-4_38. doi:10.1007/978-3-319-68288-4_38.

[15] A. Ahmad, M. Saad, M. Bassiouni, A. Mohaisen, Towards Blockchain-Driven, Secure and Transparent Audit Logs, in: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services, MobiQuitous '18, ACM, New York, NY, USA, 2018, pp. 443–448. doi:10.1145/3286978.3286985.

[16] L. M. Shekhtman, E. Waisbard, Securing Log Files Through Blockchain Technology, in: Proceedings of the 11th ACM International Systems and Storage Conference, SYSTOR '18, ACM, New York, NY, USA, 2018, p. 131. doi:10.1145/3211890.3211921.

[17] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, G. Danezis, Consensus in the Age of Blockchains, CoRR abs/1711.0 (2017). URL: http://arxiv.org/abs/1711.03936. arXiv:1711.03936.

[18] K. Peffers, T. Tuunanen, M. A. Rothenberger, S. Chatterjee, A Design Science Research Methodology for Information Systems Research, Journal of Management Information Systems 24 (2007) 45–77. doi:10.2753/MIS0742-1222240302.

[19] J. A. Onieva, J. Lopez, J. Zhou, Secure multi-party non-repudiation protocols and applications, Advances in information security, Springer, New York, NY, 2009.

[20] A. Barger, Y. Manevich, V. Bortnikov, Y. Tock, M. Factor, M. Malka, Shared Cloud Object Store, governed by permissioned blockchain, in: Proceedings of the 11th ACM International Systems and Storage Conference on - SYSTOR '18, ACM Press, New York, New York, USA, 2018, pp. 114–114. URL: http://dl.acm.org/citation.cfm?doid=3211890.3211915. doi:10.1145/3211890.3211915.

[21] G. Wang, J. Koshy, S. Subramanian, K. Paramasivam, M. Zadeh, N. Narkhede, J. Rao, J. Kreps, J. Stein, Building a Replicated Logging System with Apache Kafka, Proceedings of the VLDB Endowment 8 (2015) 1654–1655. doi:10.14778/2824032.2824063.

[22] M. Castro, B. Liskov, Practical byzantine fault tolerance and proactive recovery, ACM Transactions on Computer Systems 20 (2002) 398–461. doi:10.1145/571637.571640.

[23] D. New, M. Rose, Reliable Delivery for syslog, Technical Report, IETF, 2001. URL: https://www.ietf.org/rfc/rfc3195.txt.

[24] Mozilla Corporation, The Rust Programming Language, 2018. URL: https://www.rust-lang.org/.

[25] BitFury Group, Exonum Documentation, 2018. URL: https://exonum.com/doc/get-started/what-is-exonum.

[26] A. S. Tanenbaum, M. Van Steen, Distributed Systems: Principles and Paradigms, Prentice-Hall, 2014.

[27] Z. E. Rasjid, B. Soewito, G. Witjaksono, E. Abdurachman, A review of collisions in cryptographic hash function used in digital forensic tools, in: Procedia Computer Science, 2017, pp. 381–392. doi:10.1016/j.procs.2017.10.072.

[28] R. Dennis, G. Owenson, B. Aziz, A temporal blockchain: A formal analysis, in: Proceedings - 2016 International Conference on Collaboration Technologies and Systems, CTS 2016, 2016, pp. 430–437. doi:10.1109/CTS.2016.80.