Distributed immutabilization of secure logs

Jordi Cucurull and Jordi Puiggalí

Scytl Secure Electronic Voting,
Pl. Gal·la Placídia, 1-3, 1st floor,
08006 Barcelona - Spain,
{jordi.cucurull,jordi.puiggali}@scytl.com

Abstract. Several applications require robust and tamper-proof logging systems, e.g. electronic voting or bank information systems. At Scytl we use a technology, called immutable logs, that we deploy in our electronic voting solutions. This technology ensures the integrity, authenticity and non-repudiation of the generated logs, thus in case of any event the auditors can use them to investigate the issue. As a security recommendation it is advisable to store and/or replicate the information logged in a location where the logger has no writing or modification permissions. Otherwise, if the logger gets compromised, the data previously generated could be truncated or altered using the same private keys. This approach is costly and does not protect against collusion between the logger and the entities that hold the replicated data. In order to tackle these issues, in this article we present a proposal and implementation to immutabilize integrity proofs of the secure logs within the Bitcoin's blockchain. Due to the properties of the proposal, the integrity of the immutabilized logs is guaranteed without performing log data replication and even in case the logger gets latterly compromised.

Keywords: secure logging, blockchain, distributed immutabilization, integrity, trust [1]

1 Introduction

There are several applications that require a robust and tamper-proof logging system, e.g. electronic voting systems [15] or bank information systems. At Scytl we have applied a logging solution, the immutable logs [4], in our electronic voting systems [7] that ensures the integrity, authenticity and non-repudiation of the generated logs.

In the event the logger or its private key gets compromised, the data logged before this point could be truncated or altered without being detected if no additional measures are applied. The most common solutions to this problem are 1) storage and/or replication of the information in a location where the logger only has write-only permissions [1], 2) implementation of third party notary services [16] or 3) usage of advanced cryptographic mechanisms based on aggregated signatures that are implemented as one-way functions [10]. The first solution guarantees that a manipulation of the log is not

[1] The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-319-46598-2_9

possible, due to the medium used to store the logs, or that it will be detected, due to their replication. However, the log replication requires the deployment and management of a specific infrastructure, composed of one or more servers, for this purpose. This solution does not protect against colluding entities with the logger. Thus, if a large enough distributed infrastructure needs to be deployed to prevent that a sufficient number of malicious nodes collude, this infrastructure can be expensive and difficult to maintain. The second solution relies on the usage of a trusted service which stores integrity proofs. This service could also collude with the attacker to manipulate the logs. And the third solution ensures that a log truncation will be detected because it is not possible to restore an existing aggregated signature at the point at which the log is truncated (see the reference for more details). This is presented as a two-options solution, one more efficient and dependent on a trusted party based on the usage of Message Authentication Codes (MACs), and one more computationally expensive based on public key signature and a PKI. However, in this article we explore an alternative solution based on the use of an existing efficient secure logging technology, which combines MACs and Digital Signatures (DAs), and the Bitcoin blockchain, which provides off-the-shelf distributed immutabilization. The solution consists of publishing log integrity proofs within the blockchain. Thus any manipulation of the logs is detected and collusion is not possible due to the distributed nature of the mechanism.

The blockchain is the underlying technology used by the crypto-currency Bitcoin [11] as a public ledger of all the economic transactions performed. The transactions are immutabilized, within a sequence of blocks, in a distributed manner by a set of nodes, called miners. The miners compete to perform this operation and obtain rewards and fees for each block generated. The main advantage of the blockchain is that it guarantees the integrity and non-repudiability of all the transactions registered without the need of a trusted entity.

In this article we take advantage of both technologies, the secure logs and the blockchain, to make a proposal and implementation to immutabilize integrity proofs of the secure logs within the blockchain. This guarantees the integrity of the immutabilized logs, when no replication is enabled, even in case the logger gets latterly compromised. There are other proposals that also take advantage of the off-the-shelf immutabilization capacity provided by the blockchain. For example, Zyskind et al. [18] propose a system to protect personal data, using Distributed Hash Tables (DHTs) to store the data and the blockchain to keep the access rights to it. There is also a specification, the Colored Coins [3], that extends the Bitcoin by the possibility of tagging coins and associate metadata to the transactions which may include a link to a digital asset in the BitTorrent network. There are simple services, such as Proof of existence [2], that provide immutabilization of single documents in the blockchain for a fee. And there are more complex services, such as Factom [17], which provides immutabilization and chaining of any type of data for a certain amount of fees. In this case the service is implemented as a permissioned blockchain [8] with public access, that contains references to external documents. This blockchain is connected to the main Bitcoin blockchain for enhanced verifiability. However, the work we have performed, which also uses the blockchain to immutabilize data, is different from the previous ones since: 1) it is specifically devoted

[2] https://proofofexistence.com

to immutabilize logs, thus it presents several challenges: a) the information must be kept chained in the same order as it is generated and b) the system must support the speed at which the logs are generated given the speed constraints of the Bitcoin blockchain. And 2) our proposal does not require additional infrastructure, such as intermediate databases or private and/or permissioned blockchains.

The article is structured in six sections: Section 2 explains the immutable logs and the blockchain; Section 3 explains the proposal; Section 4 presents the implementation and tests performed; Section 5 discusses some issues dealt with during the definition of the proposal and its implementation; and, finally, Section 6 concludes the article with the conclusions and future work.

2 Background

This section presents the two technologies used throughout the proposal of distributed logs immutabilization.

2.1 Secure logs

The secure logs are an event logging technology called immutable logs [4]. This technology implements cryptographic measures to preserve the integrity and authenticity of logs, without compromising the performance of the system. The system is based on chaining the log entries using a combination of Message Authentication Codes (MACs) and Digital Signatures (DAs). Each logger has a pair of signing keys, thus the log authenticity and non-repudiation is guaranteed.

The logging process comprises two types of log entries, the regular ones (see Equation 1) and the checkpoints (see Equation 2). Each log entry (L_i) is chained with the previous one using a MAC cryptographic function (specifically a HMAC [12] in the implementation used). The input of the MAC is the log entry text ($LogInfo_i$) concatenated with the integrity proof of the previous entry (h_{i-1}). A different random session key (K_j) is used for a number of consecutive entries. This scheme prevents any modification, deletion or addition of intermediate entries.

$$L_i = (LogInfo_i, h_i) \qquad where \qquad h_i = HMAC(K_j, (h_{i-1}|LogInfo_i)) \tag{1}$$

The checkpoints are a special type of entry that are used to guarantee the authenticity and non-repudiation of the last block (j) of entries. A checkpoint (Chk_j) is issued every a certain number of lines or given time, depending of the logger configuration. In each checkpoint the MAC session key used to chain the last block of entries is disclosed and a new one, kept secret by public key encryption (P_{enc}), is generated. Finally, a digital signature of the entry is also created with the signing key (S_{sig}). Thus, any log entry manipulation or deletion is detected during the verification process.

$$\begin{aligned} &Chk_j = L_i = (LogInfo_i, K_{j-1}, E(P_{enc}, K_j), Sig_j, h_{i-1}, h_i) \qquad where \\ &h_i = HMAC(K_b, (h_{i-1}|K_{j-1}|LogInfo_i)) \\ &Sig_j = S(S_{sig}, (h_{i-1}|K_{j-1}|E(P_{enc}, K_j)|h_i|Loginfo_i)) \end{aligned} \tag{2}$$

The main advantage of the secure logs, compared with only digitally signed logs, is that they allow to detect the exact location of any manipulation attempt and isolate it, from the rest of the log entries that remain intact, while keeping a good performance. These logs can also be replicated by sending them to a central log server which centralizes all the log information, to ensure logs availability and provide log monitoring. The log replication also ensures that a compromised logger cannot modify former entries, e.g. by truncating and regenerating the chain of log entries.

2.2 Bitcoin blockchain

Bitcoin [11] is a well-known decentralized cryptocurrency system. The scheme is based on a consensus network where all the nodes agree on the state of the system according to a certain set of rules. No central authority operates it neither has control of the money. A set of nodes called miners are in charge of the system operation and, to be more specific, to the generation of the blockchain that is used as a public ledger.

The operation of the system, at a high level, consists of users generating transactions that represent cryptocurrency transfers, and miners that validate and immutabilize them. Each transaction (see Figure 1) contains a reference to one or more former transactions, the inputs (I_k), which prove the user has the money he/she is spending, and the quantity and receivers of the money to spend, the outputs (O_l). The inputs are a list of tuples that include a transaction identifier (Tx_{id}) and the index of an output (ϕ_p) in that transaction. Several types of transaction outputs exist, but the most common (Pay-to-PubkeyHash) refers to a Bitcoin address ($@_l$), which is the hash of a public key, and the amount of money (θ_l) to give to that address. This type of output can only be spent by the entity that has the matching private key. Thus, each transaction contains the public key and signatures required to redeem the money of the former transactions referred. The ECDSA signing algorithm [14] with the secp256k1 curve is used. The amount of money transferred by a transaction is the sum of all its inputs. However, a small quantity of it, the difference between inputs and outputs, is taken as a publishing fee by the miners. The transactions are sent and validated by the miners through a distributed network that supports the system. After this, they remain publicly available thus all the money exchanges are publicly auditable. Despite the initial functionality of the blockchain is to be a Bitcoin ledger, it is also possible to publish a small quantity of information within each transaction output. This is used in the proposal presented in the following section and it is possible by creating a special type of output, called non-spendable output, by using the op-code OP_RETURN. After this op-code they can be included up to 75 bytes of information that are published within the transaction.

The Bitcoin transactions are registered and immutabilized within the blockchain, which provides integrity, public auditability and non-repudiation. The blockchain is a sequence of blocks that are chained by cryptographic means and that contain transactions. The blocks are generated by the miners in a competitive process that requires to solve a Proof of Work (PoW). Each block contains a set of transactions and a header with a reference to the previous block, a nonce used by the PoW and a hash that is the root of a Merkle tree that groups all the transactions to be immutabilized by the block. The transactions are immutabilized in a certain order as leaves of the Merkle tree, however this order depends of the miners and is not chosen by their issuers. The

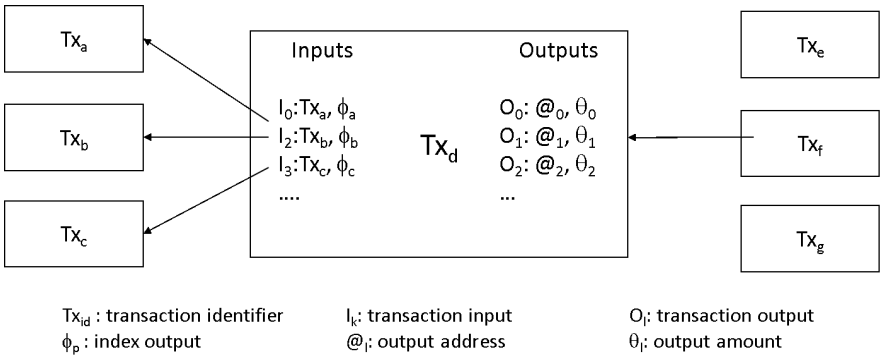

Fig. 1: Transaction detail

generated PoW consists of finding a cryptographic hash of the block, by trying different nonces, that starts with a certain number of zeroes. The complexity to solve the PoW is adjusted every two weeks in order to produce one block every ten minutes on average. When a miner solves a PoW and publishes a block, it receives a certain amount of Bitcoins both as a reward and from the fees of each transaction included in the block. The PoW, assuming that no entity has more than 51% of the mining power, is used as a mechanism to 1) prevent that a deliberately chosen entity can publish a certain block in the blockchain, and 2) prevent that formerly published blocks can be modified. Other mechanisms that are not based on the computational performance are also available [2], but they are implemented in alternative coins.

3 Distributed logs immutabilization

As explained in Section 2.1, the secure logs provide integrity and authenticity to the logs generated. In addition, the logs can be sent to a central server in order to ensure their availability and provide monitoring. But the replication of logs is also a desirable security feature. It ensures that, in the event of a logger compromise, a logger signing key compromise or a log truncation, the manipulation of previously logged entries could be detected by comparing them with the replicated information. However, this solution requires an infrastructure of servers where the information will be replicated, a mechanism to export the logs and a protocol to determine which information is valid when discrepancies are observed among the different copies.

In this section we propose a distributed log immutabilization solution that combines the use of the secure logs [4] with the blockchain. The solution does not require log replication to detect the mentioned manipulations (despite it still supports it if needed for durability and fault tolerance). The proposal takes advantage of the blockchain to distributedly publish and immutabilize log integrity proofs.

3.1 Distributed immutabilization proposal

The secure logs periodically issue a special type of log entry called checkpoint. The information included in this entry allows the verification of the block of entries present between the current and the previous checkpoint. Thus our proposal for distributed log immutabilization consists of publishing an integrity proof, i.e. a hash, into the blockchain for every issued checkpoint. These integrity proofs can later be validated with the actual log files in order to see that they were not manipulated. Each log that is immutabilized in the blockchain will have a log identifier to distinguish it from other logs immutabilized by the same entity, thus it is not globally unique.

The integrity proofs registered to the blockchain are SHA256 hashes [13] of the checkpoints (Chk_j) present in the log (see Equation 2 in Section 2.1):

$$H_j = SHA256(Chk_j) \tag{3}$$

One or more checkpoint hashes can be included within a single transaction (Tx_{id}). This accommodates the solution to the blockchain scalability constrains (see Section 5) and can reduce the total amount of transaction fees. Each hash is included within a non-spendable output entry (O_l where $l > 0$) of the transaction. The following information is published (see Table 1 for description of the fields):

$$O_l = (prefix, version, logId, H_j) \tag{4}$$

Name	Bytes	Content	Description
$prefix$	2	"SL"	OP_RETURN prefix
$version$	1	1	Version of the data structure
$logId$	16	-	Unique log identifier
H_j	32	-	Hash of the checkpoint

Table 1: Data immutabilized within the transaction output

In order to properly validate the logs against the checkpoint hashes published in the blockchain, it is needed to maintain, and enforce, the order of them. Since several checkpoint hashes can be included into a single transaction, the order has to be guaranteed at the level of transaction outputs. The order has also to be kept when multiple transactions are present. Thus, for the first case, the hashes are just included in the transaction as outputs (O_l) in the same order as they appear in the log file, as the order is kept by design of the Bitcoin. And, for the second case, each transaction is linked to the previous one using the transaction input (I_k). Hence, the transactions generated by the application are linked forming a chain of transactions (see Figure 2). As it can be seen, each transaction contains one or more inputs from transactions with spendable outputs used to pay current and future transaction fees, as well as a single output with spendable outputs that will be linked with the next transaction used to register checkpoint hashes. In addition, the transaction contains all the non-spendable outputs required in order to

register the checkpoint hashes. It is important to clarify that, the transaction issued, is not directly linked to the next one via its spendable output. This output points to an address, which is different for each transaction issued [3], that belongs to the logger or registrar application. However, the transaction can only be spent by the logger application, thus only the logger can actually link it with the input of the next transaction with checkpoint hashes.

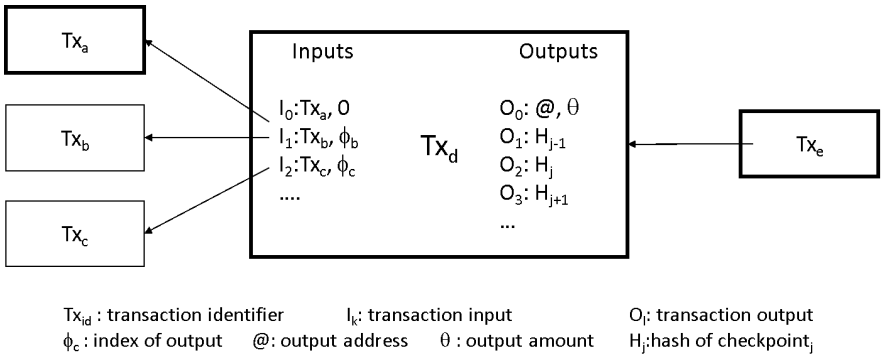

Fig. 2: Transaction link

In order to go through the generated chain of transactions and retrieve the integrity proofs of a given log, it is required to make public the first transaction of the chain and the log identifier, e.g. in a newspaper or by a trusted entity in the blockchain.

3.2 Validation proposal

The validation of the information published consists of retrieving all the published checkpoint hashes, recompute them using the log files and compare they match in content and order. We assume that the content of the logs files and the content of the blockchain is validated following the regular logger and blockchain validation mechanisms. The first transaction issued by the logger and the log identifier must be publicly well-known.

The algorithm used to perform the validation is depicted in Algorithm 1. The algorithm starts by going through all the transactions issued and registered in the blockchain by the logger (the first while in the algorithm). The first transaction (*firstTransaction*) is taken as a trusted beginning of the chain and the log identifier (*logId*) is used to select the appropriate entries (in case checkpoint hashes of different logs were included in the same transaction). The checkpoint hashes are temporally stored (`WriteHashes`), for later comparison, in a file (*fileHashesBC*). The next transaction is searched and processed (`GetNextTransaction`). As it can be appreciated in Algorithm 2, finding

[3] Although it is not enforced by the implementation of Bitcoin, it is recommended by the community not to reuse Bitcoin addresses. Thus, we decided to regenerate the reception address for each transaction generated.

the next transaction requires to go through the current and following blocks of the blockchain (outer while of the algorithm) until a transaction that has the current one as its input is found (inner while of the algorithm). After the checkpoint hashes are collected from the blockchain (again in Algorithm 1), the checkpoint hashes are recalculated (`ComputeHash`) from the actual log to validate (second while in the algorithm) and they are temporally stored (`WriteHashes`) in a file (*fileHashesSL*). Finally the files with the two lists of checkpoint hashes (*fileHashesBC* and *fileHashesSL*) are checked to be identical (`CompareFiles`). If this is the case the validation is passed.

```
Data: firstTransaction, logId, logs, blockchain, fileHashesBC, fileHashesSL
Result: boolean result
transaction = firstTransaction;
while there is transaction do
    hashes = GetHashes(transaction, logId, blockchain);
    WriteHashes(hashes, fileHashesBC);
    transaction = GetNextTransaction(transaction, blockchain);
end
while There is a log file do
    logFile = GetNextLogFile();
    while There is a log line to read do
        line = ReadLine(logFile);
        if line is a checkpoint then
            hash = ComputeHash(line);
            WriteHashes(hash, fileHashesSL);
        end
    end
end
CompareFiles(fileHashesBC, fileHashesSL);
```

Algorithm 1: Validation algorithm

```
Data: blockchain, currentTransaction
Result: nextTransaction
blockHash = GetBlockTransaction(currentTransaction);
while there is a blockHash do
    block = GetBlock(blockHash);
    while there are pending transactions in block and transaction not found do
        nextTransaction = GetTransaction(block);
        inputs = GetTransactionInputs(nextTransaction);
        if ListContains(inputs, currentTransaction) then
            return nextTransaction;
        end
    end
    blockHash = GetNextBlock(block);
end
```

Algorithm 2: GetNextTransaction algorithm

If the registration of the checkpoint hashes in the blockchain is performed within a certain timeframe after the secure log checkpoints are generated, then it is also possible to validate the timing is correct. In this case, during validation it is required to gather and compare the timestamp of the log checkpoints, this is part of the log basic information, and the timestamp of the blockchain blocks at which each checkpoint hashes are linked (via a transaction). The timestamps, given a certain margin defined by the validator, must match. The margin depends on the delay between the checkpoint hashes generation and their immutabilization in the blockchain.

4 Implementation and test

The proposal described in the previous section has been implemented as two real Java applications: an immutabilizer and a validator. These two applications interact with the logs generated by a secure logger and with the Bitcoin Core software.

4.1 Distributed immutabilizer application

The immutabilizer application has been built as a standalone Java application that periodically reads the secure log files in order to register integrity proofs of the new checkpoints to the blockchain. The application takes advantage of the btcd-cli4j library [4] as a Java wrapper of the RPC-JSON calls provided by the Bitcoin Core version 0.12.1 [5] in order to access and operate the blockchain.

When the application is started, if it is the first time, it generates a new random log identifier and reads all the available log files. Then it calculates the hashes of the checkpoints, creates a transaction that includes them and signs it. From this point, the application periodically reads the log files to detect new lines to register. The period to register checkpoint hashes is configurable, but it should not be higher than the time required for the transaction to get immutabilized and confirmed (usually a transaction is considered confirmed after the sixth block of its inclusion in the blockchain, although it is a configurable parameter too). This means that in production the period should be at least one hour. Each time a transaction is published the state is stored, thus if the application is stopped it can be resumed from the same point and with the same identifier.

The transactions created by the application have as many outputs as hashes plus an additional one that is used as a change address. The fees to pay for the transaction are calculated using the RPC-JSON EstimateFee call that estimates the relative fees to publish a transaction within a certain number of blocks (set to one in our application). If the estimation is not available, a hard-coded value of 0.0002 BTC/kB is selected. The size considered to calculate the fee is 200 bytes for the basic transaction and 60 additional bytes for each output. The idea behind this calculation is to get the transaction included in a block in a timeliness manner without spending too much fees on it. When the fee is calculated the transaction is linked with the previous one, and with additional spendable transactions if not enough Bitcoins where available to pay the fees.

[4] https://github.com/priiduneemre/btcd-cli4j
[5] https://bitcoin.org/en/bitcoin-core/

It is also worth commenting that the Bitcoin Core does not provide RPC-JSON calls to use transactions to immutabilize other data than Bitcoin payments. However, there are a set of calls that permit to operate with transactions at binary level. Thus, the approach followed has consisted of creating a transaction dummy output for each data entry to register using the CreateRawTransaction RPC-JSON primitive. The transaction, encoded in hexadecimal, is returned and each dummy output is replaced with the manually tailored output that contains the OP_RETURN code and the hash to register.

4.2 Validator application

The validator application has also been built as an standalone Java application that takes advantage of the btcd-cli4j library as a Java wrapper of the RPC-JSON calls of Bitcoin Core 0.12.1. In this case the application downloads all the checkpoint hashes published to the blockchain, starting from a given initial transaction and for a particular log identifier, creates the checkpoint hashes from the log files and compares them. No timing checks have been implemented.

The application requires the identifier of the first well-known transaction registered by the immutabilizer and the assigned log identifier. Then it follows the algorithm described in Algorithm 1 in order to perform the validation.

4.3 Testing

The testing of the immutabilizer and the validator has been performed with a generator of secure logs and the Bitcoin Core 0.12.1 running as a full node in the Bitcoin testnet.

The generator has been enabled together with the immutabilizer in order to generate logs. The log generator has been set to generate one log every 5 seconds, one checkpoint every 5 minutes and to rotate the log files every hour. The log immutabilizer has been setup to generate one transaction every hour linked to confirmed transactions for at least 3 blocks. The log checkpoints integrity were successfully validated against the information published in the blockchain when the validator was executed (currently it is not implemented as a periodic mechanism).

One of the tests we performed comprised the immutabilization of 101 checkpoints generated during around 8 hours. The log identifier was:

$$8e4b943f45506c876a93ff4b113da68f$$

and the first transaction generated was:

$$59f5167416ecf96c8101d1cafce92075f4504b37548b92b6157ac966fee102d9$$

which can be seen with a blockchain viewer such as Block Trail [6]. From this transaction the trace of the logs can be followed during 9 transactions, containing the 101 checkpoints generated.

[6] https://www.blocktrail.com/

5 Discussion

In this section we discuss the design decisions taken during the design and implementation of the solution, the scalability and costs of the proposal and the security of it.

5.1 Design decisions

As explained in Section 2.2, the blockchain is a continuously-growing distributed database composed of blocks and transactions. The blocks are organized as a sequence, each of them linked with the previous one through a hash. The transactions are linked to a block using their identifier, which is a hash computed from its elements. This organization is appropriate to keep the integrity of the information as well as the order of it. However, it has the disadvantage that finding a specific transaction requires to download and index all the data, that currently accounts around 64 GBs, unless we know in which block is contained.

Thus, one of the first questions we had was about which was the most appropriate manner to organize the integrity proofs, i.e. checkpoint hashes, of our log immutabilizer while keeping their order. The approach chosen was to include the integrity proofs in its natural order within the transactions and to link these transactions as a sequence using the input transaction field. Another option considered was to enforce the order at application level, i.e. including in each entry the current and previous checkpoint hash. However, this was discarded for two reasons: 1) it required to read, process and sort the data included within all the transaction outputs from the first block containing one of our transactions, 2) it was oblivious to the Bitcoin mechanisms to keep the order of the data, and 3) it complicated the protection of the data authenticity when different pairs of keys were used for each transaction (see next paragraphs).

Another question was how the authenticity of the integrity proofs could be kept. This can be done by cryptographically signing the transactions issued. In Bitcoin the addresses where a transaction is sent are associated to a pair of cryptographic keys. However, the standard behavior is to automatically generate a new address for each transaction received. Thus a certain entity, e.g. the logger, will have multiple addresses as new transactions are sent to it. The transactions issued are signed with the keys required by the transactions being spent, which belong to the owner of them. This renders the possibility to use a single well-known address for the immutabilizer not advisable. Another possibility, would be to publish all the pairs of keys generated and associated to the logger. However, this does not guarantee the non-repudiability property. Given a certain transaction, the immutabilizer could claim the pair of keys are not their ones, as there is no PKI infrastructure. Hence, the approach taken was the following: 1) the first transaction with log integrity proof entries is made public in a trusted place, and 2) the immutabilizer will only generate transactions with a unique spendable output sent to an address controlled by itself (it is the only one that can spend it). Thus if the first transaction, with the associated address, is trusted, the rest of the transactions ahead can also be trusted. This approach does not allow to return the change of the last transaction when the immutabilization is finished to another entity. Otherwise, the other entity could continue the chain and create additional fake entries. However, we assume the immutabilizer can continue its task with other logs or just manage small quantities of

money just to pay a few fees ahead. If this was required it is possible to add a field with op-codes that indicate the beginning and end of the log immutabilization. Another possibility could be to create a transaction to give away the remaining funds without any checkpoint hash published, considering this as a signal that the log immutabilization activity is finished.

Another question was how we could facilitate the visibility of the published data. The chosen solution, as already mentioned, was to make public the very first transaction ID of the integrity proofs published. Hence, the rest of the entries can be found following the mentioned chain. A log identifier was also included in order to discriminate the integrity proofs from the ones of other logs immutabilized. However, the log identifier cannot be solely used since any entity is free to register transactions in the blockchain with any log identifier. Another option was to use a single well-known address as the change address of all the issued transactions. However, due to the regeneration of addresses explained before this option was discarded.

5.2 Scalability and costs

Another important aspect of the solution is its scalability. The blockchain of Bitcoin is currently setup to support up to 7 transactions per second in total [5]. There is a lot of discussion on how to increase this value, since it is very small for a globally used system. Considering this limitation, we decided to register the log integrity proofs in a periodic manner, at the level of log checkpoint, and including one or more entries per transaction. The logs must also be adjusted to generate a number of checkpoints that do not end up in too large transactions. The larger a transaction is, the more expensive are the fees and the less probability to be included in a block. Thus, this approach allows to adjust the system to have a trade-off between its scalability, security and transaction fees. As example, in our tests we created one transaction every hour of 1KB on average. Considering the mentioned size and the current 1 MB block size limit, a maximum of 6000 simultaneous logs could be immutabilized at a global scale. This is not a large value, specially considering that the current blocks are already used at almost its maximum capacity.

Another aspect to consider is the operational cost of the solution. Bitcoin has an associated cost related to the publication of transactions. The average fees have a high variance, but if no delays are desired in the confirmation of transactions a rate of 0.0006 BTC/KB has to be paid [7] (checked at 5th of August of 2016). As an example, if we setup our logger to generate 12 checkpoints/hour, 1 Bitcoin transaction/hour, during a period of one month, the cost is around 203 €/month. In the example each transaction has a size of almost 1KB with a cost of 0.0006 BTC that is approximately 0.31 €. This is an approximated cost, which depends on the system setup, that is obtained from the cost model we detail in Equation 5. The model considers the number of checkpoints per unit of time (c), number of transactions per unit of time (tx), transaction fees (f) given a maximum publication delay accepted, and time of operation (t). The model considers each transaction with a base size of 0.2 KBs plus 0.06 KBs for each checkpoint included. The selected unit of time is the hour. The final cost obtained may increase

[7] https://bitcoinfees.21.co/

due to the foreseeable increase of the transaction fees (f). Currently most of the miners profits come from the reward when a block PoW is solved, but this reward is halved every 210.000 blocks (approximately every 4 years). Thus in the future the reward will be lower and the transaction fees are likely to increase.

$$cost = ((tx * 0.2) + (c * 0.06)) * t * f \tag{5}$$

5.3 Security

The solution proposed enhances the data integrity, stream integrity (order of the data), forward integrity (ensures pre-compromised data cannot be manipulated), and non-repudiation properties provided by the secure logs for the case of insider attacks. Thus, the threat model addressed is focused on attackers that compromise the logger or with access to the logs and/or private key of it. The threats in this case are: 1) forging the full log, 2) truncation of the log, and 3) forging past log entries.

The forge or replacement of all the logs, as described in threat 1, is not possible since it would require the replacement or modification of the first transaction to refer to an alternative handcrafted chain of immutabilized logs. The first transaction cannot be replaced if its identifier has been conveniently made public and broadcasted at the beginning of the system operation (e.g. the identifier could be published to newspapers, etc.). Even in case this first transaction identifier is replaced, the timestamps of the blocks will not match the expected times, unless the genuine and replaced logs are created and registered to the blockchain at the same time. Furthermore, if the first transaction identifier is not replaced, the modification of this transaction is not an option. The data published in the blockchain is almost impossible to modify after it has been confirmed (usually when six blocks are chained after the one considered confirmed).

Any manipulation of the logs, as mentioned in threats 2 and 3, would be detected as long as it affected the log blocks commited in the blockchain. The reason is that the hashes of the log checkpoints, which cover the integrity of each block of the log, would not match the ones published. It is worth to note that if the manipulation affected the current block or an unpublished block, the attacker still would have the chance to manipulate it. Thus, the more frequently the checkpoint hashes are published in the blockchain, the smaller is the opportunity window to manipulate the data from unpublished blocks of the log.

For more security, it is also possible to publish the identifier of the first transaction in one of the first entries of the logs, thus creating a commitment of the logs with the information published in the blockchain.

Other possible attacks could be tried at the level of Bitcoin. Common attacks, such as the double spending attacks [9] do not apply in this case since the logger issues the transactions against itself, thus there is no threat model that fits with this attack. Another possibility would be the logger to collude with a miner to perform selfish mining [6]. In this case the logger would try to fork and construct a parallel chain of transactions to be immutabilized in a parallel branch of the blockchain. However, this would be too difficult to sustain during the whole live of the logs, since it requires to own at least 1/4th of the total mining power.

6 Conclusions

In this article we have presented a proposal, implemented on top of the Bitcoin's blockchain, that enhances the security of the immutable logs [4]. It provides additional integrity and non-repudiation security properties resilient to log truncation and log regeneration in cases in which the logger or its signing key gets compromised. The proposal is based on publishing log integrity proofs in the blockchain. This protection can also be given by creating a mechanism of log replication at different selected servers operated by independent entities or within a distributed network. However with the blockchain the mentioned security properties can be given off-the-shelf without log replication and scaling up the immutabilization at a global level. In this case the immutabilizers are the Bitcoin miners, entities that are not globally susceptible to respond at the particular interests of a potential compromised logger or attacker. Due to the blockchain design, the information immutabilized cannot be modified by anybody unless it has more than 50% of the system's mining capacity.

With the proposal and implementation presented it is shown that the blockchain can be used for the purpose described. However, we have also discussed the limitations that currently exist. If the number of transactions and block size supported by the blockchain do not increase, the number of immutabilized logs at a global scale cannot be large and the frequency of immutabilization and number of checkpoints of the log must be kept very low. As a side effect of the limited capacity, the transaction fees are also high in order to guarantee there are no delays in the publication of transactions. Thus, in order to commercially use this solution, the blockchain should offer more capacity than the current one or the user must be willing to pay high operational costs.

Further work can be done in order to improve our specification, for example to support the finalization of logs. Currently, our specification does not comprise the transfer of accumulated funds from one of the transactions used for registering assets to an address non controlled by the logger. Otherwise, the owner of this address could continue the chain with fake integrity proofs for a given log identifier. If a transaction included a signal to indicate the finalization of a log immutabilization, any other integrity proof in the following transactions could be ignored. In addition it would be more efficient by the validator to validate the logs, since it could stop when this sign was found. Regarding the implementation, further work can be done in order to support distributed immutabilization of multiple logs, since currently only a single node is supported. Finally, the validator can be extended to check the timing of the integrity proofs published in the blockchain match the secure log entries within a given time frame.

References

1. Mihir Bellare and Bennet S. Yee. Forward integrity for secure audit logs. Technical report, 1997.
2. Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016.
3. Colu. Colored Coins Protocol Specification. Accessed June 2016.
4. A.O. Cornet and J.M.B. Bosch. Method and system of generating immutable audit logs, January 15 2009. US Patent App. 12/096,048.

5. Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gün. On scaling decentralized blockchains. In *Proc. 3rd Workshop on Bitcoin and Blockchain Research*, 2016.
6. Ittay Eyal and Emin Gün Sirer. *Majority Is Not Enough: Bitcoin Mining Is Vulnerable*, pages 436–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
7. David Galindo, Sandra Guasch, and Jordi Puiggalí. 2015 Neuchâtel's cast-as-intended verification mechanism. In Rolf Haenni, Reto E. Koenig, and Douglas Wikstroḿ, editors, *E-Voting and Identity*, volume 9269 of *Lecture Notes in Computer Science*, pages 3–18. Springer International Publishing, 2015.
8. BitFury Group and Jeff Garzik. Public versus private blockchains. part 1: Permissioned blockchains. Technical report, BitFury Group, October 2015.
9. Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. Misbehavior in bitcoin: A study of double-spending and accountability. *ACM Trans. Inf. Syst. Secur.*, 18(1):2:1–2:32, May 2015.
10. Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5(1):2:1–2:21, March 2009.
11. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
12. National Institute of Standards and Technology. FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standard (FIPS), Publication 198-1. Technical report, U.S. Department Of Commerce, July 2008.
13. National Institute of Standards and Technology. FIPS 180-4, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-4. Technical report, U.S. Department Of Commerce, March 2012.
14. National Institute of Standards and Technology. FIPS 186-4, Digital Signature Standard (DSS), Federal Information Processing Standard (FIPS), Publication 186-4. Technical report, U.S. Department Of Commerce, July 2013.
15. Jordi Puiggalí, Jesús Chóliz, and Sandra Guasch. Best practices in Internet Voting. In *NIST: Workshop on UOCAVA Remote Voting Systems. Washington DC, August 2010.*
16. Richard T. Snodgrass, Shilong Stanley Yao, and Christian Collberg. Tamper detection in audit logs. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 504–515. VLDB Endowment, 2004.
17. Paul Snow, Brian Deery, Jack Lu, David Johnston, and Peter Kirby. Factom: Business processes secured by immutable audit trails on the blockchain. Whitepaper, Factom, November 2014.
18. G. Zyskind, O. Nathan, and A. S. Pentland. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 180–184, May 2015.