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

# Blockchain-based publishing layer for the Keyless Signing Infrastructure

Christopher Jämthagen
Dept. of Electrical and Information Technology
Lund University
P.O. Box 118, SE-221 00, Lund, Sweden
Email: christopher.jamthagen@eit.lth.se

Martin Hell
Dept. of Electrical and Information Technology
Lund University
P.O. Box 118, SE-221 00, Lund, Sweden
Email: martin.hell@eit.lth.se

***Abstract*—A Keyless Signing Infrastructure (KSI) provides users with a means to timestamp documents on a per-second basis. The KSI consists of a global infrastructure with several server layers and by using Merkle hash trees, the root hash can be used to verify the integrity and timestamp of a document. Regular publication of root hashes, e.g., once per month, allows a document to be verified without support of the servers, but the previously proposed publication channels have several limitations. In this paper we address these limitations and show how a blockchain can be used as an additional publication layer on top of a KSI. Using the scripting capabilities in transactions, new features are introduced to the publication of root hashes. This includes faster publication, proof of origin for the publisher and the possibility for third parties to explicitly verify root hashes before they are published. These improvements will allow the verification of documents to be simpler, more flexible and more secure.**

## I. Introduction

A Keyless Signing Infrastructure (KSI) was proposed in [1]. KSI provides a means to timestamp data on a large scale. The timestamps can be verified by anyone and the supporting infrastructure ensures that modifications of documents or timestamps are infeasible. In that sense, the timestamps can be seen as server-based signatures. Different from ordinary signatures, where the user can sign a document offline, here a server must participate in the computation and publication of the signature. Keyless, here, means that the signatures can be verified without relying on the secrecy of a private key. A reliable and unforgeable timestamp will ensure the integrity and issuance of documents.

The security of timestamps and the simplicity of implementation and verification relies on the periodic publication of root hashes in widely witnessed media. The widely witnessed media is proposed to be, among others, newspapers, public forums and micro-blogging platforms [2]. To avoid too frequent publication, for example in newspapers, a frequency of once per month has been proposed [1].

At the same time, blockchains have emerged as widely used databases of linked blocks. While they originally were used for securing and linking transactions in Bitcoin [3] and other cryptocurrencies, blockchains have developed into more general platforms, suitable for timestamping and signing information. Blocks are issued frequently, once per 10 minutes on average for Bitcoin, and by linking blocks together it is infeasible to rearrange them or change information in them without having access to a majority of the computational resources that secures the blockchain.

It is clear that the goals of KSI can be combined with the functionality of blockchains. In this paper we show how to use a blockchain in order to mitigate some of the limitations in the KSI. We add a publishing layer that utilizes the blockchain which allows more frequent publication of root hashes, as well as simplified methods for verifying these hashes. Using the scripting functionality of blockchain transactions, we also show how the KSI operator can strengthen its operational security by requiring multiple signatures to actually publish hashes. While the Bitcoin blockchain is the most widely used blockchain, and will be the targeted blockchain in this paper, our proposed solution is not necessarily limited to this blockchain. In order to gain the byzantine fault-tolerant properties we seek, a public blockchain that utilizes proof-of-work should be used.

The paper is organized as follows. In Section II we give a short overview of KSI and enumerate what limitations it suffers from. Section III will describe the basic functionality of blockchains and Section IV describes details of the design of our proposed solution for the publishing layer of the KSI. We consider some proposed enhancements to blockchains and how they affect our solution in Section V and discuss advantages and disadvantages of some of our design choices in Section VI. Related work is found in Section VII and we conclude in Section VIII.

## II. Keyless Signing Infrastructure

In this section we give a more detailed overview of KSI and its limitations.

The KSI is based an a set of servers, ultimately producing an unforgeable timestamp, i.e., a proof of existence. The timestamps, or signatures, are constructed through a globally distributed Merkle hash tree [4], or more specifically, a hierarchy of hash trees with roots propagating through several layers. Clients provide data, or hashes of data, that they want timestamped to a *gateway*. The data from multiple clients are aggregated in a hash tree and the root is sent to the next layer, an *aggregation layer*. Several servers, typically geographically distributed, serve as aggregation layer. The root hash in one

978-1-5090-2771-2/16 $31.00 © 2016 IEEE
DOI 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.30

layer serves as a leaf in the next layer, ultimately reaching a globally unique root hash in the *core layer* of the hierarchy. Such a root hash is generated regularly, e.g, once per second in the current implementation [1], building a calendar of root hashes.

When the root hash has been computed in the core layer, the hash path needed to compute the root hash from the submitted value is propagated down through the network. The calendar hash together with the hash path needed to compute the calendar hash from the document can be used to verify that the document has been correctly timestamped. The hash path can be stored with the document and anyone can query a particular calendar hash in order to verify the integrity and timestamp of a given document. By distributing the calendar hashes to all participants, a successful forgery by reordering or manipulation of calendar hashes will require cooperation between all participants (or at least those that are used when verifying timestamps). In order to obtain a trusted calendar hash, or sequence of hashes, this has to be obtained from several sources.

To simplify this procedure, periodically, a calendar root hash (CRH), which is the root of the Merkle tree consisting of calendar hashes, is published in widely witnessed media. This can be newspapers, public forums or micro blogs. The hash path from the calendar hash to the published CRH can be added to the document hash path. Now, assuming that enough participating servers verify the published CRH, a document can be verified only using the published value. Thus, the published value adds important aspects to KSI.

- The KSI infrastructure is no longer needed in order to verify a document
- The need to query several participating servers in order to obtain a trusted calendar hash is removed.

### A. Limitations of KSI

The hash paths used by KSI to verify timestamps rely on the root hashes being trusted. This is achieved by distributing them to all participants. Storing all in a central database would require this database to be fully trusted. Publishing CRHs in newspapers, forums or micro-blogging platforms once per month will simplify the procedure of verifying root hashes and it will also simplify the trust model since anyone can verify the published value and object if it looks manipulated.

However, publishing CRHs in newspapers is rather inefficient. First, the frequency of publication is at best once per day. Second, verifying a hash value printed on paper requires some form of manual process to digitize it. Using centralized micro-blogging platforms or public forums is prone to denial-of-service attacks [5] and could be the target of malicious attackers. With infrequent publications, a verifier must trust the subset of servers that are queried before a calendar hash path and CRH can be used for verification. An additional disadvantage of infrequent publishing is that the timestamp on the publishing medium, e.g., date in the newspaper, only guarantees that the data was available before that date, but will not give more granular guarantees about it. For more granular information, trust in a third party will be needed.

These limitations can be solved by publishing the CRHs in a blockchain. A blockchain provides a possibility for much more frequent publication as new blocks are typically added several times per hour. They also provide a standardized interface to allow simple verification and retrieval of values. Moreover, their integrity, once enough blocks have been added, is guaranteed by a distributed consensus involving a large amount of servers and miners.

Utilizing an open-access blockchain will reduce the trust requirements of the KSI operator and aggregators, as clients are able to independently verify the CRHs as they are included in the blockchain.

We aim to solve the following limitations in this paper:

1) No authentication of published CRHs.
2) Infrequent publication of CRHs.
3) Inconvenient to independently verify CRHs.
4) No possibility to verify CRH before publication.

We will propose solutions utilizing the blockchain to solve these limitations.

## III. Blockchain and proof-of-work

A blockchain can be viewed as a database where data entered into it has strong guarantees of immutability and where these guarantees grow stronger as time passes. These guarantees are what makes a blockchain ideal for timestamping applications, because they provide assurances that the data will not change in the future.

A blockchain consists of linked blocks, which in turn contain transactions. A block contains a block header and a set of transactions as its payload. The set of transactions are commited to in the block header via the Merkle root of all transactions as leaves in a Merkle tree. Additionally the header contains a Unix timestamp and the hash of the previous blocks' header. This makes sure that each new block commits to all previous ones as well.

The blockchain is secured with proof-of-work, where the block header is repeatedly hashed until the hash value fulfills some condition. The condition is that the value of the hash digest must be below some target value, also specified in the block header. This form of block signing allows anyone to try to create a block without discrimination, because all you need is access to hardware that can perform these hashing operations. The target value is regularly adjusted such that the average time between the issuance of two blocks is fixed.

Proof-of-work makes modification of data in existing blocks expensive since the hash digest of the block header have to be recalculated to satisfy the condition. Additionally, every subsequent block header will need to be recalculated since they depend on the previous block header, going back to the first block that was modified. Unless the attacker possesses more than half of the computational power of the entire network, he is unlikely to be able to catch up and create a longer chain. The deeper the data to be modified is, the more expensive it will be.

Since tokens are issued to the creator of a block, the incentives are aligned with extending the chain with most work attached to it. Otherwise there is a risk that the block will never be part of the chain, and the block creator cannot claim the tokens rewarded in that block. There are some exceptions to this rule where a single entity possessing a large enough share of the hashing power of the network can benefit from withholding valid blocks in an attack called selfish mining [6], [7].

The valid chain is considered the one with most work attached to it. Occasionally a different fork can overtake the previously longest chain and replace it, orphaning one or more blocks in the process. The process of rearranging the blockchain, undoing transactions in orphaned blocks and so on, is referred to as a chain reorganization. There are no guarantees that a transaction included in an orphaned block is included in the new fork.

Proof-of-work is not the only thing necessary to create valid blocks. Apart from the proof-of-work, several other consensus rules exist which must be followed for blocks to be considered valid. The most important consensus rule is arguably that a specific token must not be spent more than once. This rule against double-spending allows deterministic tracking of transaction outputs in a transaction chain, which will be useful to track CRHs embedded by the KSI.

Another important consensus rule for our use case is that the timestamp in the block header must not be ahead more than 2 hours of the time of the local machine that receives it, nor must it be earlier than the median timestamp of the previous 11 blocks. Should the timestamp not be within these boundaries, the block is deemed invalid. This provides rough assurances of what time the data in the block were commited at.

### A. Transactions

A transaction specifies how tokens in the blockchain get reallocated. A transaction must include one or more inputs and outputs. Each input will reference an output with unspent tokens and each output will specify conditions for how to spend the amount of tokens locked in it. The amount in the outputs referenced by the inputs must be equal to or more than the amount set in the outputs of the new transaction. If the outputs spend less than what is available, the difference can be collected as a fee by the miner that includes the transaction in a block. Fees are essential to get transactions included in a block.

To decide whether a transaction is elligible for inclusion in a block, inputs and outputs include scripts which are executed and must return `OP_TRUE` in order to be valid. When a new transaction is created, each input script will be paired with the referenced outputs script. Only transactions that can provide a valid input script will be able to spend that specific output. Additionally, there is a locktime field in the transaction, the value of which must be below the current time in order to be valid.

Scripts are stack-based with a list of instructions and data items processed from left to right. If at any point during execution a verify operation sees a `OP_FALSE` value at the top of the stack, the execution stops and the transaction is marked as invalid. If the transaction executes in full, the top stack value must be `OP_TRUE`. Below are some examples of some standardized ways scripts are currently being used to give the reader, both an idea of how scripts are executed, and what constructs we will use later on.

*1) Pay-to-Public-Key-Hash (P2PKH):* In a P2PKH type of transaction, the script in the output to be spent is formed to force the spender to supply the raw public key in the input script to be hashed and matched to the hash digest supplied in the output script. Should the hashes match, the script goes on to validate a signature of the transaction with that public key. The scripts look like this:

```
output script:  OP_DUP
                OP_HASH160
                <hash_of_public_key>
                OP_EQUALVERIFY
                OP_CHECKSIG

input script:   <signature>
                <public_key>
```

The following happens when this script executes.

- The `<signature>` and `<public_key>` are pushed to the stack from the input script
- The `OP_DUP` instruction duplicates the top stack value, i.e., `public_key`, so there are now two instances of `public_key` on the top of the stack
- `OP_HASH` pops the top stack value, hashes it and pushes the hash to the stack
- `<hash_of_public_key>` is pushed to the stack
- The two top values of the stack are compared to make sure they are equal via the `OP_EQUALVERIFY` instruction
- If they are not equal, execution stops and the transaction is deemed invalid
- If they are equal, execution continues and `OP_CHECKSIG` will try to verify that `<public_key>` validates `<signature>` correctly
- If it is valid, the transaction is valid and it may be included in a block

*2) Pay-to-Script-Hash (P2SH):* In the same way that a P2PKH requires a public key to hash to a specific hash digest, P2SH requires an entire serialized script to be hashed to match the specific digest in the output. This provides much more flexibility for the receiver to specify the conditions he wants in his script without burdening the sender with the details. The serialized script that must be provided to spend a P2SH output is called a redeem script and is executed in full if the hashes match. It also has the advantage of script confidentiality until it is spent.

Scripts presented later will be embedded in P2SH transactions.

*3) Multisignature transactions:* Multisignature encumbered outputs forces the spender to supply M valid signatures from N given public keys, where M $\leq$ N. This allows the security of that output to be distributed among many keys, where loss or theft of one of the keys does not necessarilly mean that the output is compromised. Below is an example multisignature script.

```
output script:  OP_2
                <public_key1>
                <public_key2>
                <public_key3>
                OP_3
                OP_CHECKMULTISIG

input script:   OP_0
                <signature1>
                <signature3>
```

The following happens when this script executes

- The data items from the input script and the output script are pushed to the stack
- `OP_CHECKMULTISIG` is then executed which takes the top stack value, `OP_3`, to know how many public keys to pop
- After the public keys are popped off the stack, the value `OP_2` is popped and tells the instruction how many signatures it will need
- `OP_CHECKMULTISIG` tries all combinations of the public keys to verify all signatures and will push `OP_TRUE` to the stack if it finds enough valid signatures, otherwise it will push `OP_FALSE`
- `OP_0` is there due to a bug in `OP_CHECKMULTISIG`, where it always pops one extra value from the stack

*4) OP_CHECKLOCKTIMEVERIFY:* `OP_CHECKLOCKTIMEVERIFY` can be used to make the locktime field of transactions much more powerful. Instead of just pre-signing transactions with the locktime requirements, an output can include a locktime requirement on the stack together with the instruction `OP_CHECKLOCKTIMEVERIFY`, which will only mark a transaction as valid if the locktime field of the transaction trying to spend the output is above the locktime requirement specified in the output script. We will utilize this instruction to create spending conditions that prioritizes some groups in Section IV-B.

*5) OP_RETURN:* `OP_RETURN` is an opcode that fails the script execution immediately and makes any attempt to spend such an output invalidate the transaction. In practice it makes the specific output unspendable, and this is used when you want a hash value embedded in the blockchain, or for some reason burn tokens. The reason for using `OP_RETURN` is that it marks that output as unspendable and allows clients to stop tracking it, thus not requiring unnecessary resources for running a client. These types of outputs can also be pruned and deleted from permanent storage in some modes of operation to save disk space. Since it is unspendable, the associated value with the output should be zero.

`OP_RETURN` is the way we will embedd CRHs in the blockchain.

## IV. Design

The naive approach of embedding hashes in the blockchain is to simply create a transaction with the commitment value included in an `OP_RETURN` output and have that transaction included in a block. KSI clients are then notified of which block and transaction the CRH is included in to allow them to independently verify that their data has been included. This approach has two notable drawbacks. First, nothing stops the KSI operator from including multiple variants of CRHs and reporting different values to different clients. This would break consistency of having all hashes collected under the same tree and may open up attack vectors. Secondly it forces clients to always keep references to the specific block and transaction that include the hash for any future verification needs of their data. Additionally there is no way to authenticate that the KSI operator actually was the entity that included the hash in the blockchain. Each CRH would have to be accompanied by authentication data, which would be cumbersome to maintain. Authentication is important for many reasons. In the case of a chain reorganization we want to make sure that the KSI operator does not fraudulently change any of the values. If only they can sign transactions with the embedded CRHs, we can provide proofs of fraud in an event where they try to alter the CRHs.

A better solution is for the KSI operator, at the start of its operations, to signal one transaction that it has control over. This transaction will be dubbed the genesis transaction. This transaction will mark the beginning of a transaction chain that the KSI operator will have sole control over. When a CRH should be embedded in the blockchain, the KSI operator creates a new transaction that spends the first output of the genesis transaction, adds a new output, which will be the next output to track, and one additional output with the CRH itself. For each new CRH to be added, the KSI operator creates a new transaction that spends the first output from the last transaction in the transaction chain that begins with the genesis transaction.

Figure 1 illustrates these concepts. Transaction y is the genesis transaction. Transaction y+1 spends the first output via input 0 and includes the first CRH in output 1. When output 0 of transaction y+1 is spent, a new transaction with a new CRH and a new tracking output is created. Note how it is not necessary for a transaction to be included in every block. The KSI operator can make as many or as few transactions as they deem necessary. The tracking of the transaction chain is unaffected. If there is a need to include more funds in the transaction for fees, it is possible to include more inputs. It does not matter which input spends the tracking output, as long as one of them does. It is also possible to include more outputs after the first two. The first output in each transaction of the transaction chain is used for tracking the chain, while all other outputs are ignored.

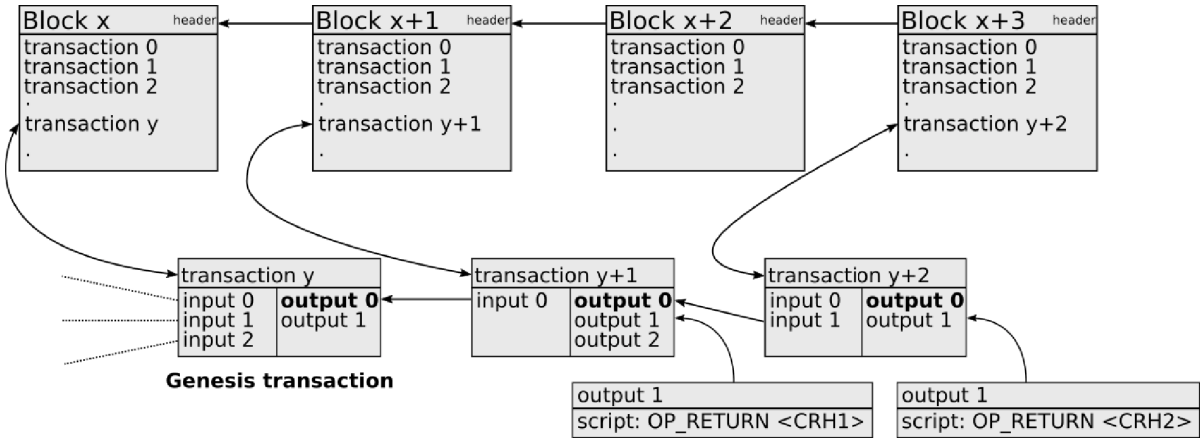


Fig. 1. Clients track the first output in each transaction to retrieve a deterministic and ordered chain of transactions with CRHs in the second output of each transaction.

With this design clients in a KSI infrastructure can deterministically track CRHs with the only prerequisite knowledge being the genesis transaction. Timestamps in the block headers provide a rough idea of the time when the CRH was included, and clients can be assured, with strong guarantees, that if a chain reorganization occurs and the KSI operator attempts to tamper with orphaned CRHs, they can also be held accountable by providing proof of the orphaned chain with differing values of the commited CRH.

If there are a lot of clients utilizing the services provided by the KSI, there will be wide consensus on which transaction is the genesis transaction. Verification of CRHs in a future where the KSI may be out of business could retain the trust in the per-second granularity it provided during its time. It should be easy to know and trust which transaction marked the first in the KSIs transaction chain. If the CRHs were commited and reported individually, outside a transaction chain, the assurances would be weaker because only the clients that commited data in that individual block would keep track of it. There could also be alternative commitments. Commitments in a transaction chain has presumably been validated by all its participants, can be authenticated to originating from the genesis transaction and can thus be more trustworthy.

### A. Broadcasting CRHs

One question is when to broadcast transactions with CRHs to be added to the blockchain. We dicuss two options here. The simplest, cheapest and most manageable seems to be for the publisher to continue to add calendar hashes as leaves to a tree until the previous CRH has been included in a block, and only after that broadcast a transaction with the new CRH. When this transaction has been broadcast to the blockchain network, the publisher continues to work on a new hash tree, the root hash of which will be broadcast when the previous one has been included in a block. This approach has the advantage of only requiring one transaction and transaction fee per CRH and a simple implementation. The disadvantages though is because block creation is a random process, it could take a long time before the current hashes are commited to the blockchain. For some clients it may be important to have the CRH that commits their data included in a block as soon as possible.

To accomodate this, the KSI operator could perform transaction replacement. Transaction replacement is the act of replacing a previously broadcasted transaction, that has not been included in a block yet, with a new version. The new version will need a higher fee attached to it compared to the previous version in order to be reliably forwarded by nodes on the network and eventually reach miners.

Another approach would be to spend the unconfirmed transaction over and over, thus creating a chain of unconfirmed transactions each with their own CRH attached to it. When a block is created the whole chain of transactions could be included. However, this approach bloats the blockchain which is a limited resource, and the replace-by-fee [8] approach should be preferred.

Figure 2 illustrates how the publishing layer collects calendar hashes to create CRHs for publishing on the blockchain.

### B. Complex spending conditions

In its most simple form, the condition to spend an output will only require a signature of the transaction from the private key related to the specified public key and the public key itself. These scripts can be arbitrarily complex though to cover various failover modes like key-loss or key-theft. Most common is to make use of multi-signature scripts, as described in Section III-A3, to require a threshold of signatures for an output to be spent. This provides the KSI operator with the ability to strengthen its operational security. Should one of the keys be lost or stolen, the compromised key cannot by itself spend the output. The remaining, uncompromised, keys can still spend the output and specify a new spending condition that excludes the compromised key.

Spending conditions can be even more complex. We can assign multiple groups, each with a set of signers, with different conditionals for when and how they may make a spend. Should something happen to the top priority group, the second

group of signers can take over operations after some amount of time has passed without the output having been spent. Such a spending condition could be a 2-of-3 multisignature script from the primary group or 3-of-4 multisignature script from a failover group, with the condition that the output has not been spent until a specific time. The failover group would not be able to spend the output unless the primary group fails to make a spend until that time. Any number of groups with different priorities and timeouts can be set to cover multiple failures.

Below follows an example redeem script of how three groups (A,B,C) have spending permissions at different times. This redeem script will be supplied in an input together with appropriate signatures to succeed in spending it. Group A will always be able to spend the output, and will require 2 valid signatures from any 3 specified public keys. Group B will be able to spend the output only after time X, and require 3 valid signatures given from any 4 specified public keys. Group C will be able to spend the output only after time Y, where Y occurs after X, and requires only 1 valid signature from any of the 5 specified public keys.

```
OP_2
<A1.pubkey>
<A2.pubkey>
<A3.pubkey>
OP_3
OP_CHECKMULTISIG
OP_NOTIF
    OP_3
    <B1.pubkey>
    <B2.pubkey>
    <B3.pubkey>
    <B4.pubkey>
    OP_4
    OP_CHECKMULTISIG
    OP_NOTIF
        OP_1
        <C1.pubkey>
        <C2.pubkey>
        <C3.pubkey>
        <C4.pubkey>
        <C5.pubkey>
        OP_5
        OP_CHECKMULTISIG
        <time_Y>
        OP_CHECKLOCKTIMEVERIFY
        OP_DROP
    OP_ELSE
        <time_X>
        OP_CHECKLOCKTIMEVERIFY
        OP_DROP
        OP_TRUE
    OP_ENDIF
OP_ELSE
    OP_TRUE
OP_ENDIF
```

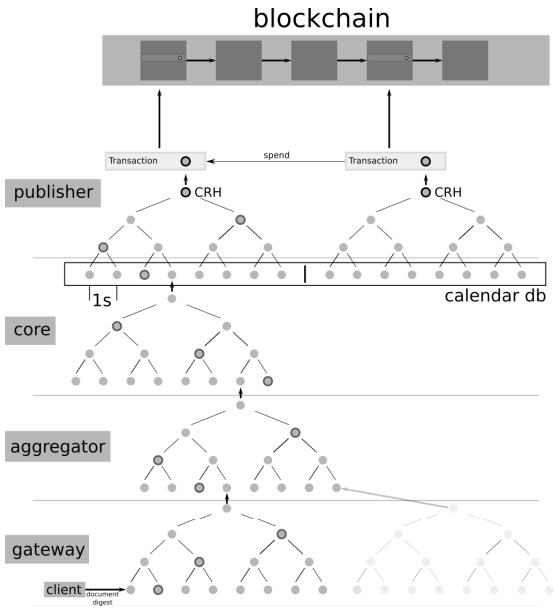


Fig. 2. Overview of KSI and the proposed publishing layer. Publisher collects calendar hashes and eventually tries to publish root hashes with the calendar hashes as leaves in a Merkle tree. The marked nodes in the trees is everything the client needs to verify that their document digest is part of the CRH

The publishing layer of the KSI could be merged with other stakeholders of the KSI. Group A could, for example, be the core cluster of the KSI, while group B includes participants from the aggregator layer, ready to step in if there is a failure. Group C could be a set of trusted functionaries that will reboot the transaction chain if and when Groups A and B fail.

An alternative setup is that group A's conditions could require a signature from a core cluster node and a threshold signature from several aggregator nodes. This would increase the trust of the CRH before it is included in a block, given that the core cluster and various aggregator nodes are independent of each other, thus minimizing the risk of collusion. These types of scripts addresses the limitation of the KSIs centralized trust model to become more decentralized by including independent parties to sign transactions.

## V. Further considerations

Blockchain technology is far from mature, and a number of development efforts exist to extend and enhance them. Several proposed features could be adopted to improve the proposed design described in this paper.

### A. Merkleized Abstract Syntax Trees

Improvements to the scripting system, such as Merkleized Abstract Syntax Trees (MAST) [9], can be used to enhance our use case, including confidentiality of unused spending conditions and reduced on-chain footprint. A MAST encodes mutually exclusive spending conditions of a script as seperate branches of a Merkle tree where only the branch used will be revealed.

Currently the entire redeem script will have to be included in order for the transaction to be verified correctly, exposing unused spending conditions in the process. Confidentiality of unused spending conditions will allow the KSI operator to keep critical operational security information hidden from potential attackers that will try to harvest as much information as possible. It will also reduce the size of large redeem scripts, since only the condition used to make the spend will have to be revealed, including intermediate hash values to correctly verify the Merkle root of the tree. Smaller redeem scripts lowers transaction fees and minimizes blockchain bloat. Furthermore, confidentiality of spending conditions can be kept even between the different groups that have spending permissions, since they only need to be aware of their own condition. One party must obviously be aware of all conditions since someone must generate the script.

One additional improvement that MASTs can provide for our use case is that the CRH commitment can be bundled in the tracking output, hiding the fact that hash commitments are occuring in that transaction for non-participants, as well as enabling additional reduction of the transaction footprint on the blockchain.

### B. Relative locktime

The `OP_CHECKLOCKTIMEVERIFY` enables spending conditions to be valid at some absolute time in the future. A relative locktime, where an output becomes spendable some specific amount of time after it has entered the blockchain can improve our proposed solution.

With an absolute locktime, such as with `OP_CHECKLOCKTIMEVERIFY`, we have to calculate a new absolute time to use in all newly created transactions. With a relative locktime variant we can select an appropriate relative time for the specific conditions and reuse them without having to recalculate absolute times for each new transaction.

Relative locktime has been implemented [10], [11], but not deployed on the Bitcoin network at the time of writing.

## VI. Discussion

The functionality provided by the proposed publishing layer of the KSI could be merged with the core cluster and other parts of the KSI. Since the core cluster consists of many servers that runs their own agreement protocol to decide what value to commit to, each of them could hold one signing key and sign transactions including these CRHs directly, rather than outsourcing this work to some other set of servers. It can also be merged with aggregators and gateways and form complex spending conditions to assure clients further that no single independent organization has control over the values embedded on the blockchain. There are incentives for the KSI operator and other parties to not try tampering with values in circumstances such as chain reorganizations. Since transactions are basically flooded to the blockchain network, and a spend from a tracking output enumerates exactly which spending condition was utilized, the responsible party/parties can be held accountable. Such an attempt at fraud could make people question the integrity of the operators.

There are some disadvantages to using a chain of transactions which includes the CRHs. Since the transaction chain can be uniquely identified, miners could choose to force the KSI operator to pay larger fees, or even outright try to censor transactions that is trying to spend from these outputs. Such a scenario could seriously degrade the reliability of the system. If the computational power of the network is sufficiently decentralized, censorship by some of the miners should not be a problem. There is also a potential bootstrapping attack where the KSI operator could signal multiple genesis transactions to different clients. As time passes and as more clients connect to the system, this should be less of an issue. Clients can verify the genesis transaction with other peers if necessary.

Blockchain technology is still in its infancy, and it is an open question as to how such a system will be secured in the future when block rewards diminish. An idea is that transaction fees will provide the incentive for miners to continue building blocks. Since the KSI want to make many transactions per day, it could pose an economical issue in the future if transaction fees become more expensive than they are today.

Using the approach of transaction replacement for CRH propagation on the blockchain solves the problem of having to let clients wait unnecessarily long for a block to be found to send out a new transaction. It does however come with some its own set of drawbacks. The cost is greater since more transaction fees need to be added for each transaction replacement. This could be mitigated by allowing clients to pay premiums to have their data embedded earlier. Additionally, at the time of writing the replace-by-fee [8] policy is somewhat controversial, and should it not be widely adopted, this approach could prove to be unreliable to use. Should an old version of a transaction be included instead of the latest one, the publisher should start the next tree beginning with the excluded values.

The tracking of CRHs on the blockchain can be implemented utilizing low resources for client-side implementations. If a client is not using the blockchain for something other than verifying CRHs, they only need to download the block headers in order to verify the proof-of-work, and the specific transactions they are interested in. The transactions are provided with Merkle branches to prove that they in fact are commited in a specific block header and all other transactions in that block can be ignored. We can be sure that all transactions have been provided since missing transactions will be evident due to breaks in the tracking outputs in the transaction chain.

## VII. Related work

With the invention of Bitcoin and the blockchain [3], it has been proposed as a solution for many of todays problems pertaining to trust. From reforming the financial system by taking advantage of the immutability and transparency provided by blockchains, to tracking ownership of digital assets [12] as well as physical property, so called smart property [13]. There have been proposals for protocols to operate directly on top

of existing blockchains for entirely new purposes than what they were orignally intended for, such as Counterparty [14], which is a platform for P2P financial applications on top of the Bitcoin blockchain, and Colored Coins [15] which is a technique for associating assets with addresses, also on the Bitcoin blockchain. Brand new blockchains have also been bootstrapped to fill the need of these new solutions, like Namecoin [16] for a censorship-resistant alternative to the DNS system, and Ethereum [17] which provides powerful and feature-rich scripting capabilities to do smart contracting. Decentralizing prediction markets with the help of blockchains have also been a topic of research [18].

Timestamping documents with minimal trust requirements, including security from back-dating and forward-dating, using Merkle trees to prove inclusion of a document was studied in [19], [20], [21]. New signature schemes utilizing quantum-immune primitives for KSI clients is discussed in [22]. It provides extensions where clients can authenticate themselves via one-time keys using hash-chains. Revealing such OTKs could be problematic if the signed data is not sufficiently secure from tampering. Commitcoin [23] presents the use of the Bitcoin blockchain as a means to publish proof of a document's existence at a specific point in time.

If KSI clients have no use for the per-second timestamping that KSI provides, they could utilize the blockchain by themselves and timestamp what they need without involving any third parties. If everyone did this, however, it would bloat the blockchain to the point where either fees will be too high for individual pieces of data to be timestamped, or the blockchain grows so large that few people validate its contents and it becomes more dependant on third parties for validation. There is software available [24] for anyone to run a federated timestaping service on the blockchain which includes document digests from all interested participants and aggregates them into a hash tree, of which the root is the only value included in the blockchain, and proofs are sent back to all clients at that time. This saves blockchain space, as well as transaction fees.

## VIII. Conclusion

Several limitations related to the previously proposed publishing channels for calendar root hashes (CRHs) in the keyless signing infrastructure have been identified. We have shown that by using a blockchain as a publication channel for these root hashes, these limitations have been mitigated and the proposed solution additionally introduces new features to the KSI. The time window in which trust in calendar hashes, and queries of these, are required can be lowered and we can, by using the scripting functionality of transactions, make sure that only one entity is able to publish new CRHs. Moreover, by using multisignatures, it is possible to require explicit verification of a CRH prior to its inclusion in a block. It also adds a fallback if the primary signing group fails.

The use of a blockchain has its own limitations, such as transaction fees, and possible attacks on the system in itself, but our proposed publication layer constitutes a viable alternative to newspapers, forums and micro-blogging platforms. It adds simplicity, flexibility and security to KSI and from the perspective of the emerging blockchain technology, it shows yet another application which can take advantage of its properties and power.

## References


[1] Ahto Buldas, Andres Kroonmaa, and Risto Laanoja. *Secure IT Systems: 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, chapter Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees, pages 313–320. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] Ahto Buldas and Märt Saarepera. Document verification with distributed calendar infrastructure, May 6 2014. US Patent 8,719,576.

[3] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[4] Ralph C. Merkle. *Advances in Cryptology — CRYPTO '87: Proceedings*, chapter A Digital Signature Based on a Conventional Encryption Function, pages 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[5] Eliot van Buskirk. Denial-of-service attack knocks twitter offline. http://www.wired.com/2009/08/twitter-apparently-down/, 2009.

[6] Ittay Eyal and Emin Gün Sirer. *Financial Cryptography and Data Security: 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, chapter Majority Is Not Enough: Bitcoin Mining Is Vulnerable, pages 436–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[7] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *CoRR*, abs/1507.06183, 2015.

[8] David A. Harding and Peter Todd. Opt-in full replace-by-fee signaling. https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki, 2015.

[9] Jeremy Rubin, Manali Naik, and Nitya Subramanian. Merkelized abstract syntax trees.

[10] Nicolas Dorier Mark Friedenbach, BtcDrak and kinoshitajona. Relative lock-time using consensus-enforced sequence numbers. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki, 2015.

[11] Mark Friedenbach BtcDrak and Eric Lombrozo. Op_checksequenceverify. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki, 2015.

[12] Bitfury Group. Digital assets on public blockchains. http://bitfury.com/content/5-white-papers-research/bitfury-digital_assets_on_public_blockchains-1.pdf, 2016.

[13] The idea of smart contracts. http://szabo.best.vwh.net/idea.html, 1997.

[14] Counterparty. http://counterparty.io/.

[15] Meni Rosenfeld. Overview of colored coins. *White paper, bitcoil. co. il*, 2012.

[16] Namecoin. https://namecoin.info/.

[17] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[18] Jeremy Clark, Joseph Bonneau, Edward W Felten, Joshua A Kroll, Andrew Miller, and Arvind Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security, State College, Pennsylvania*, 2014.

[19] Henri Massias, Xavier Serret Avila, and Jean-Jacques Quisquater. Design of a secure timestamping service with minimal trust requirement. In *the 20th Symposium on Information Theory in the Benelux*, 1999.

[20] Stuart Haber and W Scott Stornetta. *How to time-stamp a digital document*. Springer, 1990.

[21] Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334, 1993.

[22] Ahto Buldas, Risto Laanoja, and Ahto Truu. Efficient quantum-immune keyless signatures with identity. *IACR Cryptology ePrint Archive*, 2014:321, 2014.

[23] Jeremy Clark and Aleksander Essex. Commitcoin: Carbon dating commitments with bitcoin. In *Financial Cryptography and Data Security*, pages 390–398. Springer, 2012.

[24] Chainpoint. https://github.com/chainpoint/chainpoint/.