Reinforcing Immutability of Permissioned Blockchains with Keyless Signatures Infrastructure

Conference Paper · January 2017

DOI: 10.1145/3007748.3018280

2 authors:

Harika Narumanchi
Tata Consultancy Services Limited
6 PUBLICATIONS 17 CITATIONS
SEE PROFILE

Nitesh Emmadi
Tata Consultancy Services Limited
6 PUBLICATIONS 17 CITATIONS
SEE PROFILE

Reinforcing Immutability of Permissioned Blockchains with Keyless Signatures' Infrastructure

Nitesh Emmadi and Harika Narumanchi
TCS Innovation Labs
Hyderabad, India
{nitesh.e,harika.n}@atc.tcs.com

ABSTRACT

With the emergence of Bitcoin, businesses are focusing on leveraging Bitcoin's blockchain technology to non-cryptocurrency based applications to improve efficiency of the operations. These business applications operate in environments where participants have verified identities; these are called permissioned environments. Blockchain is an immutable and append only distributed ledger that can be utilized for record keeping applications. We observe that when blockchain technology is adapted from permissionless environments to permissioned environments the immutability of blockchain becomes questionable as the end-users may not monitor or store a copy of the blockchain. We propose the use of Keyless Signatures' Infrastructure as an additional mechanism to ensure irreversible and irrefutable proof of block confirmations which acts as a global proof thus preserving long term immutability of blockchain in permissioned environments.

Keywords

Bitcoin; Blockchain; Permissioned Environments; Permissionless Environments; Keyless Signatures' Infrastructure; Merkle Tree; Hashchain

1. Introduction

In 2008, Satoshi Nakamota proposed a type of peer-to-peer decentralized digital currency termed as Bitcoin [1] based on a mathematical proof, created and maintained electronically. Bitcoins are mined by a group of individuals called *miners* through a process called *mining.* The difficulty in mining is introduced by the *proof of work* [2], an operation that is expensive and time consuming to execute but easy to verify. Miners validate the pending transactions in the Bitcoin network - submitted during a time period - into a *block* and confirm those transactions into a public ledger called the *blockchain.* Blockchain is an immutable and append only distributed ledger predominantly used for record keeping. Most recently, the progress in blockchain technology for use in applications other than cryptocurrencies has been promising due to advantages such as anonymity, decentralization, speed, non-repudiability, transparency and difficulty to forge giving them a unique edge over alternative solutions. For example, blockchain can be utilized to create and transfer digital assets in an efficient manner. It can also be used to build infrastructure for trade finance.

Several computational platforms were proposed on top of blockchain such as Enigma [3] and HAWK [4] that operate in open environments where the identity of the participants is anonymous. Enigma is a decentralized computational platform that ensures privacy. Its computational model is based on secure multi-party computation guaranteed by verifiable secret sharing schemes. HAWK is a decentralized model for privacy preserving smart contracts that ensures on-chain privacy and contractual security.

Most recently, there has been ongoing research on designing blockchain platforms that operate in permissioned environments where the identity of the involved parties is known. The transaction validators in open environments are anonymous outsider entities whereas permissioned environments have validators whose identities are required to be verified before joining the network. Organizations are developing their own blockchains solutions to support a wide range of applications. For example, Ethereum [5] is one of the decentralized smart contract platforms that can execute peer-to-peer contracts using cryptocurrency called *ether.* The Linux Foundation announced the creation of Hyperledger project [6] which is a cross-industry collaborative effort with more than hundred contributors to create an open standard for blockchain. Chain.com [7] supports asset transfer that has automated rules for issuance and transfer with role-based permissions for accessing the network. Eris blockchain platform [8] focuses on building financial applications leveraging smart contracts.

In this paper we observe that adapting blockchain to permissioned environments has an impact on the immutability property of blockchain. We propose the use of Keyless Signatures' Infrastructure (KSI) [9] signatures to provide irrefutable proof of block confirmations thus strengthening the immutability of blockchain. KSI provides a hash-tree based time-stamping service that provides a proof that the data existed at a particular time.

1.1 Our Contribution

We observe that adapting blockchain to permissioned en-

ICDCN '17, January 04-07, 2017, Hyderabad, India
© 2017 ACM. ISBN 978-1-4503-4839-3/17/01...$15.00
DOI: http://dx.doi.org/10.1145/3007748.3018280

vironments affects the immutability of blockchain with higher probability compared to permissionless environments. The validating parties can collude at some point to change the ledger (include or exclude transactions) and reconstruct it. The customers may not store the blockchain due to large number of transactions and memory constraints, or due to implicit trust in the validating parties. As the ledger is maintained by the controlled entities, we cannot ensure the immutability of the ledger. We propose the use of KSI signatures as an additional mechanism to ensure immutability wherein the security of the system does not depend on long-term secrecy of private keys as there are no keys involved in signature generation. The KSI signature acts as a receipt of the block hash, which includes user's transactions, and can be used to verify the integrity of the blockchain. This mechanism provides a way of verifying the integrity of the blockchain ledger even to the users who do not want to monitor the blockchain continuously. The past signatures in the KSI hashchain cannot be modified or generated. As KSI hash tree is public, the difficulty of forging a signature, is much higher and can be easily detected. KSI maintains a global proof that the authorities of the blockchain has no control over, thus ensuring that the ledger has not been tampered with.

1.2 Preliminaries

In this paper public key is represented by pk and private key is represented by sk. The n^{th} block in a blockchain is represented by b_n. Hash of a statement x is represented by $\mathcal{H}(x)$. A hash of a block b verified by both KSI signature and digital signature is represented by $\mathcal{H}_v(b)$. The n^{th} transaction is represented by t_n. Proof of work is denoted by $\mathtt{P}_w$.

The paper is organized as follows: Section II describes the blockchain in Bitcoin and how immutability is achieved in Bitcoin. Section III covers the challenges involved in adapting blockchain from permissionless to permissioned environments. Section IV illustrates KSI and how it can be used to strengthen the immutability of blockchain in permissioned environments.

2. Blockchain in Bitcoin

A hashchain is formed by recursive application of a hash function to a stream of data. In hashchain, previously computed hash is included in the current hash. The interesting property in hashchains is once the final hash is computed and confirmed, it is infeasible to alter the previous data without modifying the final hash. The confirmed final hash acts as a validator for the entire chain. When a hashchain becomes very long, the verification process takes lot of time. To overcome this disadvantage, we group some hashes before using them in the main tree to expedite the verification process, termed as the Merkle tree [10]. A Merkle tree is an optimized hashchain and is constructed by hashing pairs of leaves until a single hash remains, the Merkle root. Merkle tree is an efficient way to verify large data structures. Merkle trees are used in different types of softwares like file sharing protocols such as BitTorrent, distributed version control systems such as Git and blockchain in Bitcoins.

Blockchain is a chain of a continuously growing list of transaction records. It is essentially tamper-proof and provides integrity and transparency along with efficient availability [1]. The content of the blockchain can be accessed by anyone and the transactions of the blockchain can be verified anytime. The blockchain has to be updated with the recent changes since the last update. Each block is computationally infeasible to modify once it has been in the chain for sometime. To modify a block, every block after it has to be regenerated and the proof-of-work computation has to be redone. We refer to the computed hash, using the current hash and previous hash, as the *Final hash* (Figure 1). When a final hash is processed by the network and is highly unlikely to be reversed, we refer to it as the *confirmed hash*. The risk of reversing a confirmed transaction exponentially decreases with each subsequent confirmations.

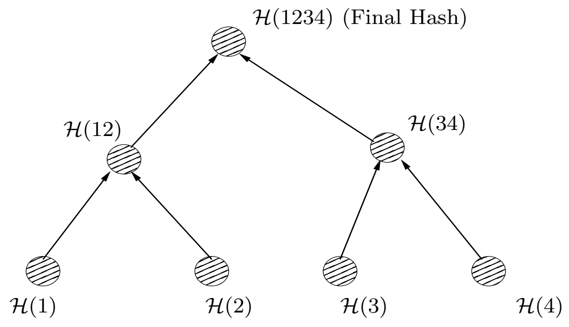

Figure 1: Merkle Tree

Bitcoin network uses proof-of-work to ensure consensus among the Bitcoin miners. Proof-of-work is used in trustless environments where there is no central authority and mutual trust. The first proof of work system was invented in 1997 called *Hashcash* [2]. It was used to limit spam emails and denial of service attacks. The proof of work function used as a mining algorithm in Bitcoin, uses essentially the same idea of Hashcash. The authenticity of proof of work is based on computational difficulty and honest majority. This protocol involves a challenge, say c and the prover of the protocol comes up with a response, say r corresponding to the given challenge c that satisfies a definite mathematical property. The challenge c should be such that the prover takes significant amounts of CPU time or resources to compute the response, but takes a lot less time to verify. The proof-of-work mechanism is highly inefficient both in terms of energy consumption as well as transaction throughput.

Immutability in Bitcoin is due to the following reasons:

- Proof-of-work requires expensive and time consuming operation satisfying a certain challenge that requires enormous amount of computational power but is easy to verify. The adversary who tries to overwrite the response with a fake value would spend enormous amounts of CPU time or resources to successfully overwrite the legitimate value.

- The copies of the blockchain are distributed and maintained by several independent parties. Therefore, it is not feasible to modify the blockchain without the knowledge of all the parties.

- Blocks are appended periodically after every ten minutes. Therefore, to modify a particular block all blocks after it should be regenerated.

Figure 2 represents the working of blockchain in Bitcoin environment. Miners run the proof of work for the k^{th} block represented by $\mathtt{P}_w(b_k)$ to find the nonce that satisfies the given condition (decided by the Bitcoin network).

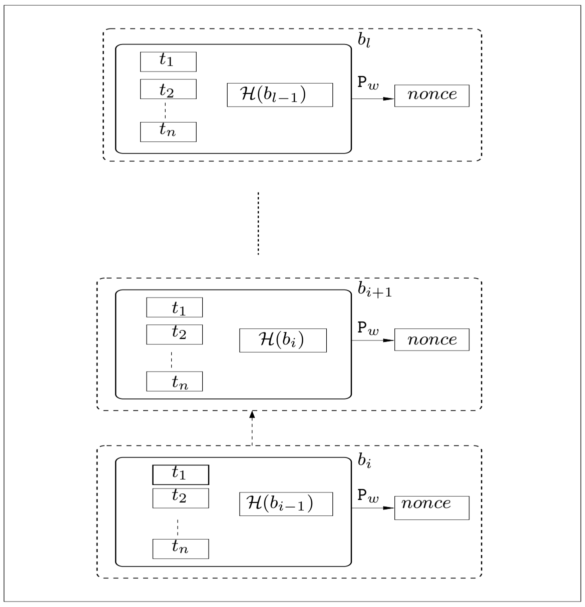

Figure 2: Blockchain with proof of work

3. Adapting Blockchain to Permissioned Environments

Having seen how the immutability is achieved in the permissionless environment such as Bitcoin [Section 2], this section describes how the permissioned environments differ and how the immutability is affected.

While adapting the blockchain to permissioned environments, difficulty in the consensus protocols in these environments can be, and often need to be, relaxed. There are challenges involved in adapting the consensus protocols used in permissionless environments to the permissioned environments. For example, the proof-of-work is not suitable in controlled environments due to its high energy consumption and low transaction throughput. Also, there is sufficient time for the attacker to reconstruct at-least the recent blocks in the blockchain during public holidays or weekends or auditing periods where new transactions are prohibited. In the case of Bitcoins this problem does not arise because even if a block has no transactions, miners mine a block every 10 minutes to collect rewards from it. Therefore, it is more difficult for the attacker to reconstruct the blocks as more blocks are periodically appended to the blockchain. However, in permissioned environments, as the frequency of the transactions is irregular, proof of work is not a suitable choice for consensus. It might seem a simpler solution to consider use of digital signatures to sign the verified transactions to reach consensus as the validating parties have verified identities.

However, as the copies of the blockchain are maintained by controlling parties, they can collude to modify the blockchain and reconstruct it. As the end-users may not monitor or store a copy of the blockchain, we cannot ensure that the blockchain is not tampered with.

4. Blockchain and Keyless Signatures' Infrastructure

Considering the challenges in adapting blockchain from permissionless environments to permissioned environments, we need a mechanism that the controlling party does not have control over to ensure the immutability of the ledger. To facilitate this we use a global time-stamping mechanism to provide a global proof that the data exists at a point of time and has not been tampered with. We use KSI signature to generate the global proof. The hashtree in KSI hashchain grows linearly with time. This property of the KSI hashchain makes it infeasible to modify or generate the predated signatures facilitating its use in permissioned environments.

4.1 Keyless Signatures' Infrastructure

KSI [9] provides scalable signature based authentication. KSI provides strong data integrity, long term immutability and tamper evident protection for digital assets with proof of time, identity and authenticity guarantee that the data has not been tampered with since it was signed. The integrity of the signatures is protected using one-way collision resistant hash functions and do not involve any secret keys. KSI hashchain is designed to provide scalable blockchains that scale linearly with time and independent of the number of transactions, unlike the traditional blockchains that scale linearly with the number of transactions. Hash-tree time-stamping technique uses one-way collision resistant hash functions to convert a list of hashes into a fixed length hash associated with time. The end-user sends a hash to the service provider and receives a signature token. A signature token, which is essentially a path to the Merkle root, is the proof that the data existed at a specific point of time and the request was received through a particular access point. KSI aggregates all the received requests into a hash-tree and top hash values are retained for each second. These top hash values are linked together in a globally unique hash-tree called the hash calendar. In Figure 3, to verify a signature token y in the place of b, we concatenate y with a that is retained as a part of the signature token and compute hash value $y_1 = h(y|b)$. y_1 is used as input for the next hash computation step until we reach the top hash y_2. If $top = y_2$, then it is risk-less to believe that y was in the original hash-tree. The size of the signature token depends on the number of aggregation levels. As the number of client requests increase, the number of aggregation levels increase logarithmically. Consequently, the number of hashes to be stored to construct the verification path to the root hash increases.

KSI works in three stages namely, hashing, aggregation and publication (Figure 5). The statements to be signed by KSI are hashed and the hash values represent the transactions. The user sends the hash of a statement through the KSI gateway to obtain signature in response to the hash. A global per-round hash is created using the aggregators to represent all the transactions signed during a round. The top values of the global per-round hashes are linked together into a perpetual hash-tree and published in a hard-to-modify and widely witnessed media such as newspapers.

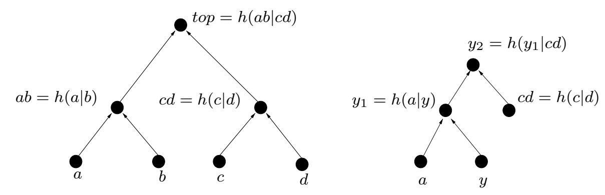

Figure 3: Building a hash-tree and verification of y in the position of b

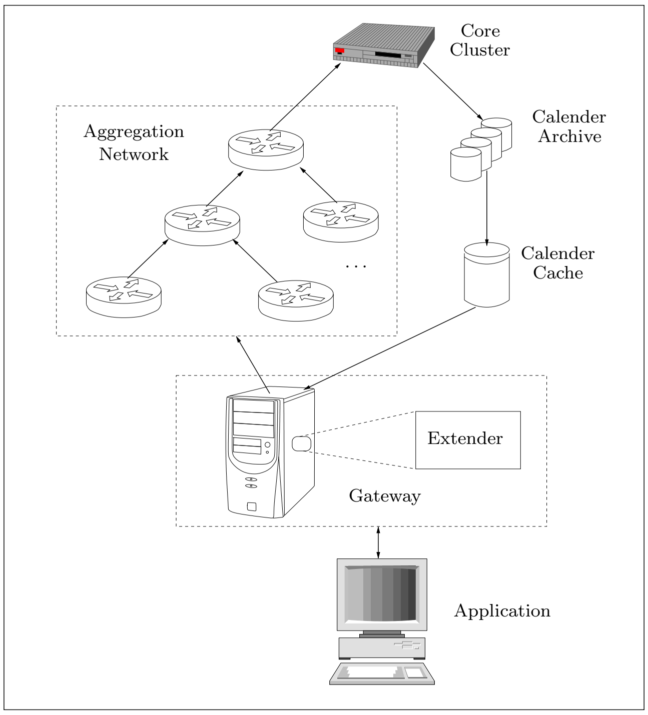

Figure 4: KSI System Architecture

The infrastructure design of KSI(Figure 4) consists of four major components namely application, gateway, aggregators and the core cluster. The application is used to perform the hashing step that forms the signing request. The signing request is sent to the gateway that provides services to the end-user. The gateway accepts the signing requests in application specific formats and forwards them to the assigned aggregators. The aggregators working in rounds of equal duration gather the incoming requests to build a hash tree and pass the top hash values to their upstream aggregators. The requests received during a particular round are aggregated into the same hashtree. An aggregator sends responses to all its child aggregators along with the hash path of its own tree after receiving a response from its upstream aggregator. A core cluster is on the top of the aggregation network and is responsible for achieving consensus about the top hash value in each round, storing it in calendar database and returning it to the aggregation network. There is an extender in the gateway that offers signature token verification services to the users. The values in the calendar database are distributed using calendar cache to the extender services. Refer to [9] for the detailed description of KSI architecture.

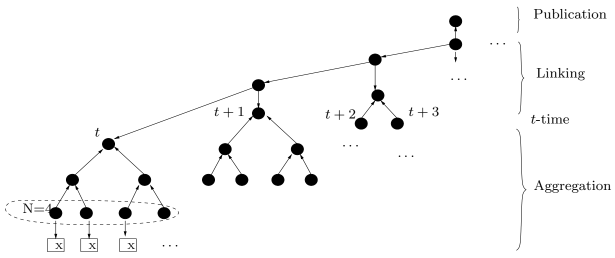

Figure 5: Hash-tree

We conducted the test of KSI for blockchain time-stamping using the open-source library available on Github in Java. The KSI library and test access are available on request. When a block is signed using KSI, we obtain a signature token [Refer Appendix] in return which can be used to verify the block. The size of signature token file in this test example is 1757 bytes.

4.2 Blockchain based on KSI

In the blockchain based on KSI hashchain (Figure 6), the hash of data is signed and sent to the KSI server as input. The KSI server signs this input and returns a signature token corresponding to the particular input, an irrefutable proof that the block existed. Note that the KSI server takes only the hashes of the data as input. Therefore, even if the server is compromised it is impossible to know what the data is. As the hashes are stored on a global hash calendar any modifications are globally visible, especially to other users of the KSI network. This signature token along with the hashes pertaining to that block can be given as a receipt to the user that the block existed which acts as a global proof for his transaction. Using the hashes pertaining to that block and the signature token, user can prove that the transaction was in a particular block on the blockchain at a point of time. It is the user's responsibility to store the signature token at the users' side. The end-user has global proof that the block existed at that point of time in the chain using the signature token of that particular block. The end-user can use this KSI signature token to verify the integrity of the blockchain without the need for monitoring or maintaining the blockchain. The main advantage of using KSI is that the security of the system does not depend on the long-term secrecy of the private keys. As KSI server takes only hash of data as the input, the privacy of the underlying data is preserved. It is impossible to compromise the Gaurdtime's global server infrastructure, without being detected. Therefore, it is infeasible to forge a KSI signature with past or future time-stamps.

5. Conclusion

Leveraging blockchain technology in enterprise applications is an active research area. Organizations are building their own blockchain infrastructure to improve their efficiency in performing operations. We addressed the issue of lack of strong immutability in permissioned environments due to the centralized authority that has the ability to modify the blockchain at some point of time. We proposed use of KSI signature to ensure irrefutable and irreversible proof

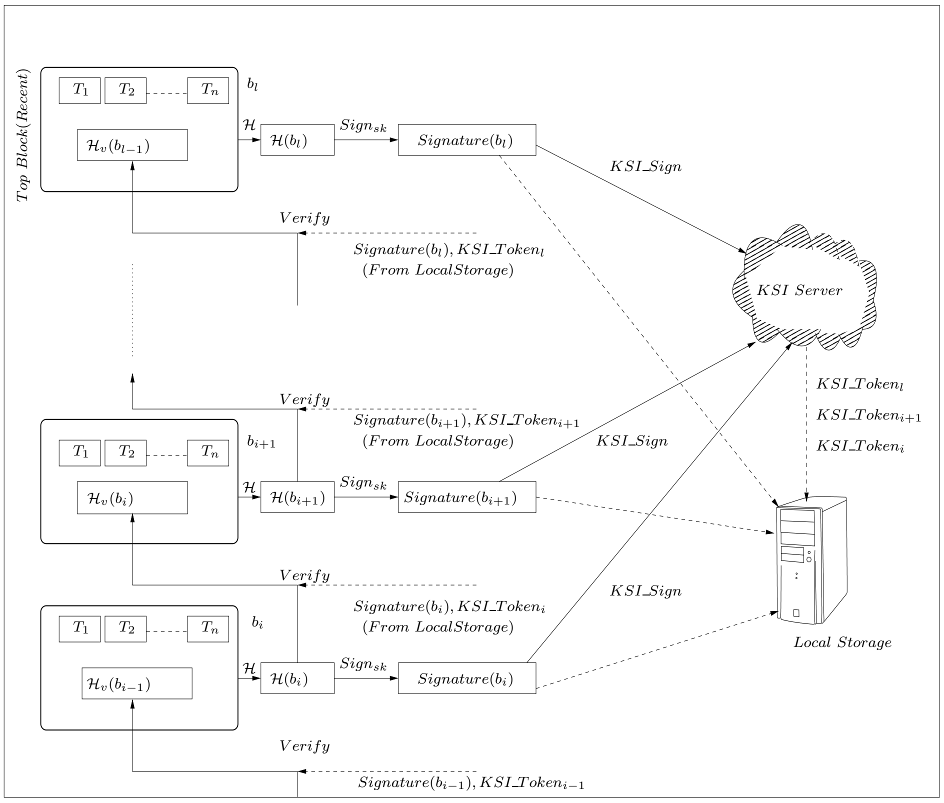

Figure 6: Blockchain based on KSI hashchain

of block confirmations to mitigate the problems in adapting blockchains to permissioned environments. The KSI signature ensures integrity of the blockchain even when the users do not monitor the blockchain.

6. References

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

[2] Adam Back. Hashcash- a denial of service counter-measure. 1997. http://www.cypherspace.org/~adam/hashcash/.

[3] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *CoRR*, abs/1506.03471, 2015.

[4] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *University of Maryland and Cornell University*, 2015.

[5] Ethereum Project. A next-generation smart contract and decentralized application platform, 2014.

[6] Hyperledger. Hyperledger. 2015. https://www.hyperledger.org/ (Accessed: 17-10-2016).

[7] Chain. Chain. 2014. https://chain.com/ (Accessed: 17-10-2016).

[8] Eris Industries. Eris blockchain platform. 2014. https://monax.io/platform/ (Accessed: 21-10-2016).

[9] Ahto Buldas, Andres Kroonmaa, and Risto Laanoja. *Secure IT Systems: 18th Nordic Conference, NordSec 2013*, chapter Keyless Signatures' Infrastructure: How to Build Global Distributed Hash-Trees, pages 313–320. 2013.

[10] Ralph Charles Merkle. Secrecy, authentication, and public key systems. 1979. AAI8001972.

APPENDIX

The structure of KSI signature token from our test results can be seen below:

```
Attributes of the Signature Token:

Input Hash:SHA-256:[E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855]
Aggregation HashChain Length:5

Aggregation Output Hashes:

 Hash of Aggregation Chain[0]SHA-256:[3C80351B010E63EB497C6084767AD3B54E249B822871160D18B9222F0CC61145]
Aggregation Chain Reference:com.guardtime.ksi.unisignature.inmemory.InMemoryAggregationHashChain@b99d62d9
 Number of ChainLinks in Aggregation Chain[0]:1
Chain Link[0] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@e307b26c

 Hash of Aggregation Chain[1]SHA-256:[AD0FBE76FAFEBC063A2CD7B3C0AB88106AF8984C53138CBDFFB3F29AB67873AA]
Aggregation Chain Reference:com.guardtime.ksi.unisignature.inmemory.InMemoryAggregationHashChain@54910693
 Number of ChainLinks in Aggregation Chain[1]:3
Chain Link[0] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@93cb5396
Chain Link[1] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@d15c9d0c
Chain Link[2] Reference:com.guardtime.ksi.unisignature.inmemory.RightAggregationChainLink@292bc4f0

 Hash of Aggregation Chain[2]SHA-256:[B56BF2C17A575D2997B9CFF40D8C7F884C0DD8FA5F676B9A1F793C4D61FAF882]
Aggregation Chain Reference:com.guardtime.ksi.unisignature.inmemory.InMemoryAggregationHashChain@ad10395b
 Number of ChainLinks in Aggregation Chain[2]:3
Chain Link[0] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@42e14227
Chain Link[1] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@a1b71791
Chain Link[2] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@955bbf6c

 Hash of Aggregation Chain[3]SHA-256:[E41C6CE67CD812FD2ED464706BF524559B027D912A331E1C78EC8DBF8949EA3C]
Aggregation Chain Reference:com.guardtime.ksi.unisignature.inmemory.InMemoryAggregationHashChain@66073029
 Number of ChainLinks in Aggregation Chain[3]:4
Chain Link[0] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@5fddf3c0
Chain Link[1] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@1221961f
Chain Link[2] Reference:com.guardtime.ksi.unisignature.inmemory.RightAggregationChainLink@6123e75e
Chain Link[3] Reference:com.guardtime.ksi.unisignature.inmemory.RightAggregationChainLink@bad54cf7

 Hash of Aggregation Chain[4]SHA-256:[EBA64D3B78838A05FFCEA4B2AC4FB163DF7E269F1D886691B5F649020FDB4EDE]
Aggregation Chain Reference:com.guardtime.ksi.unisignature.inmemory.InMemoryAggregationHashChain@f7f8f7c0
 Number of ChainLinks in Aggregation Chain[4]:3
Chain Link[0] Reference:com.guardtime.ksi.unisignature.inmemory.RightAggregationChainLink@aa9d6a5a
Chain Link[1] Reference:com.guardtime.ksi.unisignature.inmemory.LeftAggregationChainLink@60c9a511
Chain Link[2] Reference:com.guardtime.ksi.unisignature.inmemory.RightAggregationChainLink@2423b109

Aggregation Time:Tue Oct 18 10:52:06 IST 2016
Publication Time:Tue Oct 18 10:52:06 IST 2016
Calendar Hash ChainLinks Size:15
Calendar Hash Chain Links:
Calendar Chain[0] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@b23e7944
Calendar Chain[1] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@d845ff9b
Calendar Chain[2] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@64ed9ff2
Calendar Chain[3] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@308daaca
Calendar Chain[4] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@61e6aa86
Calendar Chain[5] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@4df72eef
Calendar Chain[6] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@38c8a9c6
Calendar Chain[7] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@2ec2e947
Calendar Chain[8] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@531f5389
Calendar Chain[9] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@d2d749e1
Calendar Chain[10] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@60eb7304
Calendar Chain[11] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@45ae24e
Calendar Chain[12] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@423f949
Calendar Chain[13] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@9f716fb2
Calendar Chain[14] Reference:com.guardtime.ksi.unisignature.inmemory.RightInMemoryCalendarHashChainLink@67f59dd4
Signature Data Certificate ID:[B@6b2fad11
Signature Data Certificate Repo URI:null
Signature Data Signature Type:1.2.840.113549.1.1.11
Signature Data Signature:[B@79698539
Publication String:AAAAAA-CYAWYX-4AOWUC-BNYPHW-OATJSO-OTHP2X-DS4PYB-SRTL4G-3LSRMM-F55204-IVQ7ZM-IXGNFH
Publication Data Hash:SHA-256:[D6A082DC3CF670269939D33BF571CB8FC06519AF86DAE51630BDEE9DC4561FCB]
Publication Time:Tue Oct 18 10:52:06 IST 2016
```