Validating data integrity with blockchain

Rosco Kalis
University of Amsterdam
Amsterdam, The Netherlands
rosco@kalis.me

Adam Belloum
University of Amsterdam
Amsterdam, The Netherlands
A.S.Z.Belloum@uva.nl

***Abstract*—Data manipulation is often named as a serious threat to data integrity. Data can be tampered with, and malicious actors could use this to their advantage. Data users in various application domains want to be ensured that the data they are consuming are accurate and have not been tampered with. To validate the integrity of these data, we describe a blockchain-based hash validation method. The method assumes that the actual data is stored separately from the blockchain, and then allows a data identifier and a hash of these data to be submitted to the blockchain. The actual data can be validated against the hash on the blockchain at any time. Several use cases are described for blockchain-based hash validation, and to validate the method it is implemented inside an application audit trail to validate the audit trail data. This implementation shows that blockchain-based hash validation is able to detect malicious and accidental changes that were made to the data.**

***Index Terms*—Blockchain, Ethereum, Data integrity, Data validation, Audit trail**

I. Introduction

US Director of National Intelligence James Clapper stated that the next big cyber threat is data manipulation [1] and technology magazine Wired listed it as one of the biggest security threats in 2016 [2].

Data integrity is paramount in many scientific and societal applications. Many data can be tampered with, and malicious actors could use this to their advantage by making others act on these compromised data, by distributing these data to spread misinformation, or by taking credit of work that isn't theirs.

Consumers of scientific, business, and other data want to be ensured that they can use data without having to worry about the integrity of these data. Likewise, producers of these data want to guarantee data integrity for their consumers, and they want to be ensured themselves that there is no one who can tamper with their data. In this paper, we are interested in the use case where companies want to guarantee data integrity within their organisation, and they want to be sure that all data that are handled through applications can not be changed outside the functionality of these applications.

We use the employment of audit trails to illustrate this use case. Companies employ audit trails to log all state changes made within their applications. This is already a way to enjoy more certainty about the data inside applications, and it offers a way to validate the current state of the application against all past state changes. However, if this audit trail is stored in the same way as the application data, it is equally vulnerable to tampering.

To make hard assertions about data integrity, a method needs to be developed to validate the integrity of any arbitrary data. For this, we look at blockchain [3], which is a promising technology that can improve data integrity. The last few years have really accelerated the progress in blockchain development, and especially smart contracts [4] have opened up new potential use cases for blockchain technology.

A. Blockchain

Blockchain is a data structure that distributes all its data over a network of nodes, so that there is no single point of failure, and no central control that might be compromised [3]. It uses a consensus algorithm that allows these independent nodes to approve correct transactions and reject malicious ones [4].

On the blockchain, data are stored in a chain of so-called *blocks* [4]. Every block header includes the root of a *Merkle tree*, which contains the actual data in the block [4]. Besides this, every block also includes a timestamp and a hash of the previous block in order to make it further resistant to manipulation [4]. This structure can be seen in Fig. 1.

If an attacker would change the data inside a past block, the hash of this block would change with it, and since the changed hash is never referenced by another block, it would not be accepted by the rest of the network [4], and it would effectively create a fork of the blockchain. The rule with forks is that the longest chain is always the leading one, so in order to have the modified block accepted by the network, the attacker would need to grow their chain faster than the rest of the network combined to pass the longest chain [4]. Because the resources of the entire network are extensive in major blockchains, this sufficiently guarantees the integrity of the data in the blockchain [4].

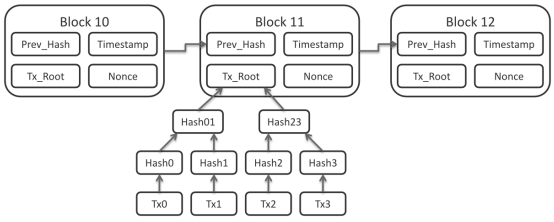

Fig. 1. This image shows the contents of blocks and how they are linked together. Image by Matthäus Wander (Wikimedia), downloaded from https://en.wikipedia.org/wiki/Blockchain in August 2018.

B. Smart contracts

Starting with Ethereum, blockchain technology has been extended with smart contract functionality. Smart contracts are a way to digitally enforce a contract or an agreement between parties through code. This concept predates blockchain more than a decade [5], but only when blockchain was implemented did it finally become possible to implement these smart contracts without the need for a trusted third party.

Ethereum offers the ability to publish smart contracts on its blockchain, which can be executed by the Ethereum Virtual Machine (EVM) [4]. By publishing these contracts to the Ethereum blockchain, all involved parties can easily inspect the contract and they will be assured that the contract will execute exactly as specified.

II. Related work

The need to ensure the data integrity of electronic audit trails and other data already existed decades ago. The different ways an audit trail could be corrupted were described by Weber in 1982, along with methods to overcome some of these corruptions [6]. Then, at the end of the century, Schneier & Kelsey described a method of securing audit log data on untrusted machines [7], which has since been cited in many other works on the subject of data integrity and auditing. US Patent 6968456 uses a similar approach to make the storage of audit trail data inside regular databases more secure [8].

Since the introduction of blockchain, multiple works have been published on the use of blockchain in the validation of data integrity and in other auditing processes. Deloitte has published their research into the use of blockchain in the field of accounting and auditing, in which they envision a hashing based approach similar to our own that allows third party auditors to easily verify the integrity of all data records [9].

Sutton & Samavi also describe a blockchain-based way of audit trail validation. They specifically focus on the non-repudiation of privacy audit logs in the light of privacy policy compliance [10]. US Patent application 20180025181 [11] and Zikratov et al. [12] also present hashing-based methods for data integrity validation. In contrast to our approach, these works specifically use their methods to validate the integrity of data files, while we focus on validating the integrity of any data.

III. Method

Smart Contracts on the Ethereum blockchain allow data to be stored directly inside these contracts as variables. Because of the nature of the blockchain, it is guaranteed that these data can only be changed using the smart contract's functionality, and every interaction with this contract gets recorded as a transaction on the blockchain.

Looking at these properties of blockchain and smart contracts, the easiest way to achieve data integrity for any data is to store all data directly on the blockchain inside these smart contracts. That way the data are easily verifiable, and both producers and consumers of the data are ensured that the data can be trusted. However, there are several issues with directly storing arbitrary data on the Ethereum blockchain. Most importantly these issues present themselves in transaction size limits, high transaction costs, and the potential need for (partial) data confidentiality. Taking these issues into account, a method is described for validating any arbitrary data using the Ethereum blockchain.

A. Transaction limits

Every operation that is executed on the Ethereum blockchain costs an amount of so-called *gas*, and the amount of gas that can be used within a single transaction is limited by the block gas limit. This gas limit is currently around 8 million gas for the Ethereum Main Net, but it can scale depending on demand [13].

The Ethereum Yellow paper specifies that every byte of passed data in a transaction costs 68 units of gas, and every transaction costs 21 000 gas to start with [13]. From this follows that the maximum amount of data passed in a single transaction is approximately 115 kB, as can be seen in (1).

$$(8000000 - 21000)/68 \approx 117300B \approx 115kB \qquad (1)$$

However, this only includes transaction data that are not stored. Data storage is one of the most expensive operations in terms of gas cost, so this can easily become a bottleneck when storing larger amounts of data. The Ethereum Yellow paper specifies that every byte of stored data costs 625 units of gas [13]. From this follows that the maximum amount of stored data in a single transaction is approximately 11 kB, as can be seen in (2).

$$(8000000 - 21000)/(625 + 68) \approx 11500B \approx 11kB \qquad (2)$$

For some use cases this limit will not be relevant, like when storing integer values. But when storing serialised application data or arbitrary-size blobs, this limit can be reached rather quickly. Realistically, this means that larger pieces of data would need to be split up over multiple transactions.

B. Transaction costs

Every unit of gas on the Ethereum blockchain needs to be paid for using the Ether cryptocurrency [13], so the monetary costs of storing data on the blockchain could increase rapidly. At the time of writing the price of one unit of gas is around 10 GWei, or 0.000 000 010 Ether, and the price of one Ether is around €300. This means the price per kB of data is around €2.1, as can be seen in (3). These increased costs make storing larger amounts of data impractical for realistic usage.

$$(625 + 68) \cdot 1024 \cdot 0.000000010 \cdot €300 \approx €2.1 \qquad (3)$$

C. Data confidentiality

All data published on the Ethereum blockchain are publicly accessible. For certain data, this is acceptable or even required, but there are many use cases in which data should only be shared with certain parties. Data that are meant for internal

use in corporations need additional confidentiality measures in order to leverage the strength of blockchain for data integrity.

Two different methods can be used to achieve data confidentiality on the blockchain: encryption and hashing [14]. It is visible from the earlier two points about transaction limits and costs that encrypted data would still realistically reach transaction limits quite often, and the costs of storage would skyrocket [14]. Therefore, only a method using data hashing is viable, as it easily stays under the transaction limits, has constant and relatively low transaction costs, and is able to shield confidential data from the public. In this paper we propose a method using data hashing to validate data integrity.

D. Data validation using data hashing

US Patent application 20180025181 [11] presents a method to reliably store data files and verify their integrity with blockchain technology. The method describes storing data files on a data storage module, and transmitting hashes of these data files to a public blockchain [11]. With this hash they store metadata, such as a timestamp and file size. The method suggests to monitors these files, and whenever a change occurs, this process is re-initiated, storing the new hash on the blockchain, with a link to the previous one [11].

While this method is specifically created for the validation of data files, a modified version of this method can be used for any type of data. The methods for hashing are the same for different types of data, while the metadata and the identifying data can be changed to fit the specific use case.

The method we propose hashes any arbitrary data and stores their hash on the Ethereum blockchain. This hash is identified by a unique identifier representing the specific data. The method for creating this unique identifier is left to the specific implementation of the method. The data can then be verified at any moment by validating that the hash stored for the data identifier matches the actual hash of the data. If the hashes don't match, we know that the data have been changed since they were stored on the Ethereum blockchain. We call the method blockchain-based hash validation.

In contrast to the method described in US Patent application 20180025181 [11], blockchain-based hash validation does not use monitoring to automatically update the hash of specific data, since this method is focused on validating the integrity of a specific version of the data. New versions of data can be added to the smart contract under a new unique identifier. This allows multiple versions of data to be validated simultaneously.

Since the data themselves are not stored on the blockchain, blockchain-based hash validation needs the data to be available in some way to validate them. This means that blockchain-based hash validation can not retrieve the original data when they have been lost or their integrity has been compromised. Use of this method should therefore always be coupled with a way to retrieve lost data, such as a sufficiently strong backup protocol and regular validations of the data. This allows changes in this data to be detected early, and the correct data to be restored.

IV. Proof of concept

Blockchain-based hash validation can be used in several different use cases, including the validation of published scientific results, the validation of audit trail data, and the validation of other shared data between different parties.

To show the possibilities of blockchain-based hash validation we implemented a proof-of-concept audit trail [1] that is validated against an Ethereum smart contract. This audit trail automatically logs all interactions that take place inside the application, and write identifier-hash mappings to the Ethereum blockchain for each of these log entries.

A. Smart contract functionality

The smart contract we created contains a mapping of the unique data identifiers to their corresponding data hash, as well as an array of all identifiers, so it can be iterated over.

Next, the smart contract includes an audit function that allows new data to be added to this mapping. This works by passing an identifier and a hash to this function; this identifier-hash pair is then stored in the mapping and the identifier is stored in the identifier array.

Finally, the smart contract contains a validation function that allows stored data to be validated by passing an identifier and a hash to this function and comparing them with the stored values.

```
contract AuditTrail {
  bytes32[] public identifiers;
  mapping(bytes32 => bytes32) public hashes;

  function audit(bytes32 identifier, bytes32 hash)
  external ownerOnly {
    require(hashes[identifier] == 0,
            "Identifier can only be audited once");
    hashes[identifier] = hash;
    identifiers.push(identifier);
  }

  function validate(bytes32 identifier, bytes32 hash)
  external view returns(uint8) {
    return hashes[identifier] == hash ? 0 : 1;
  }
}
```

B. Apache Isis

For the automatic audit trail we used Apache Isis [2], which is a Java software development framework based on Domain Driven Development and the Naked Objects pattern. It can be used to rapidly develop complex business applications because a UI and REST API are dynamically generated from the domain model of the application at runtime. This leads to a faster development cycle and higher agility, since the focus can remain on the domain model, and no extra time needs to be spent on the presentation layer of the application. More information on Apache Isis' design can be found in [14].

Apache Isis offers several services in the form of Application Programmer Interfaces (APIs) and Service Provider Interfaces (SPIs). The APIs are implemented by the framework

[1] Code can be found at https://github.com/rkalis/blockchain-audit-trail

[2] https://isis.apache.org/ (accessed 2018-04-10)

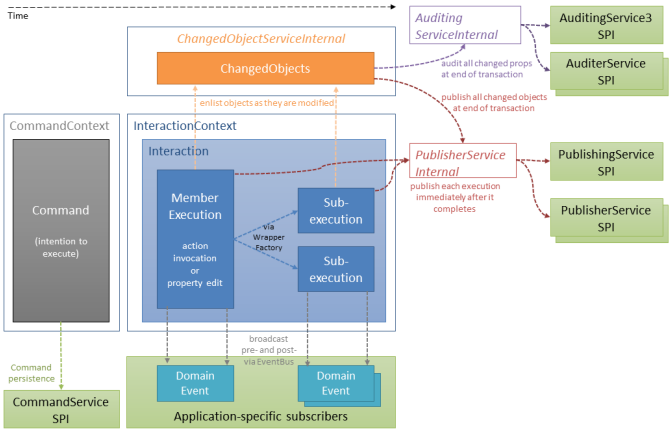

Fig. 2. This image shows the different domain services used for auditing within Apache Isis. Image downloaded from https://isis.apache.org/guides/rgsvc/rgsvc.html in April 2018.

and can be called by the application, while the SPIs are implemented by the developer, and will automatically be picked up and called by the framework [15]. Among these services are several SPIs for auditing or similar purposes, as illustrated in Fig. 2.

Most important of these services are the AuditerService and the PublisherService. The AuditerService captures the actual changes caused by an interaction [15], and the PublisherService captures a memento of the interaction, as well as a summary of the changed properties [15]. Important to note is that the AuditerService gets called every time a change in the database occurs, while the PublisherService gets called at the end of every interaction, which can contain multiple changes.

C. Audit trail implementation

For the automatic audit trail we wanted to log all changes, but at the same time keep the number of blockchain transactions to a minimum. This is why we implemented a combination of the AuditerService and PublisherService.

Every time the AuditerService' audit method gets called, its parameters get stored inside a single ThreadLocal *audit entry* object. At the end of the interaction, the PublisherService' publish method gets called, which is used to submit the audit entry's identifier and hash to the Ethereum smart contract, after which the ThreadLocal audit entry object is reset. The blockchain transaction is sent asynchronously, so that the rest of the application can continue running while the transaction is executed on the blockchain.

D. Audit trail validation

The audit entries inside the audit trail can be validated against the smart contract on the blockchain. Individual audit entries can be validated with a specific validation action, which calls the validation function on the smart contract to verify that the identifier-hash pair of the audit entry still matches the pair that is stored on the blockchain.

There is also the option to validate the entire audit trail. This is done by calling the smart contract's validation function for every single audit entry in the audit trail. In order to fully validate the audit trail, it is also checked for missing entries by traversing the array of identifiers inside the smart contract, and verifying that the audit entry with the corresponding identifier can be found inside the audit trail. The results of this validation are presented in three lists – of validated, invalidated, and missing audit entries.

V. Evaluation

To evaluate the implementation, three different scenarios have been created, in which the data inside the audit trail would be invalidated [14]. After executing these scenarios the audit trail is validated with the audit trail validation method.

For these scenarios, the blockchain audit trail implementation is added to two demo applications, which are based on actual applications that are being used. The first is based on Incode's Contact App [3], and is used for internal contact management within companies. The second is a larger scale application based on Estatio [4], which is a full-fledged estate management system.

Due to space limitation, we briefly describe the three scenarios used for the evaluations, and show the results of the first validation. More information on the scenarios, as well as more result images can be found in [14].

A. Scenario 1 - Inexperienced admin

A new system administrator in training gets a request from their coworker to restore some files from a backup. Lacking enough knowledge, they perform a full server backup, accidentally overwriting the application database. At the end of the day, the audit trail is validated, and the administrator's mistakes are discovered. Luckily, the company can reconstruct most of the correct data with the correct backups, but the changes that had been made the same day could not be recovered. This highlights a part of the weakness of the implementation, but it also displays the way these kinds of mistakes can be detected quite early on.

To simulate this scenario we take a backup of the application database, and restore it after making some changes in the database, effectively overwriting these changes. After following these steps, we run the audit trail validation. Figure 3 shows that the last two audit entries are reported as missing, as these are the ones that were deleted while restoring the database backup. A demo video of this scenario is available on YouTube [5].

B. Scenario 2 - Cover your tracks

An employee of Acme Corporation has decided they want to quit their job and retire. To fill the gap in their finances, they changed the recipient on one of the company's larger invoices to a private bank account, and changed it back to the original after the invoice had been paid. Because the company employs an audit trail, they wish to cover their tracks. They

[3]https://github.com/incodehq/contactapp (accessed 2018-05-29)
[4]https://github.com/estatio/estatio (accessed 2018-05-30
[5]https://youtu.be/nfekoK6pUqU

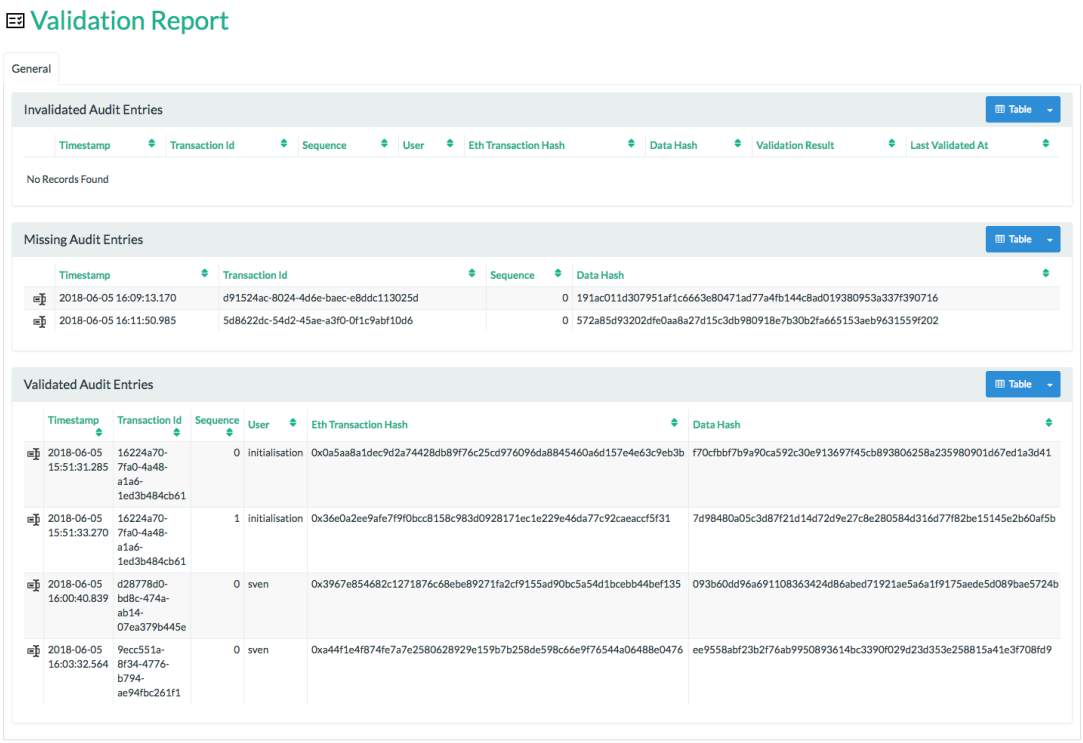

Fig. 3. The final two audit entries are missing as they have been removed in the restoring of the backup.

still have their old company credentials, so they log into the database and remove all audit entries that log their changes.

Because Acme has the policy to routinely validate their audit trail against the Ethereum blockchain they notice the missing audit entries at the end of the day. Since these audit entries had already been included in the company's backups, the correct data can easily be restored, and the employee's malicious actions come to light. Our implementation shows this by reporting the missing audit entries in the validation result.

C. Scenario 3 - Shift the blame

An employee of Acme Corporation maliciously changes the email address of a contact in the application. They then edit the corresponding audit entry directly in the database. In this audit entry he changes the logged user to his coworker to shift the blame. Afterwards, he reports his coworker to their superior. Because Acme corporation uses an audit trail that is validated against the Ethereum blockchain, it is quickly visible that the corresponding audit entry has been tampered with. The audit entry had not yet been included in the company's backups, so there is no hard proof of the employee's intentions. This is because our implementation shows the changed audit entries as invalidated in the validation report, but it does not show what the actual data inside the audit entry should have been.

VI. Discussion

In this paper we proposed a blockchain-based hash validation method. This method is used to validate the integrity of any data by storing a data identifier and a data hash in a smart contract on the Ethereum blockchain. The data can then be validated against the data stored inside the smart contract. We described several potential use cases for this method and we described one of these use cases in further detail in order to test the method. For this use case, we created an automatic audit trail for Apache Isis applications that uses blockchain-based hash validation to validate its data. This audit trail was evaluated using several scenarios.

A. Conclusions

The evaluation of our proof of concept implementation showcased the strengths and weaknesses of blockchain-based hash validation. This evaluation showed that blockchain-based hash validation is able to detect tampering or loss of data. However, this evaluation also shows that it is not able to prevent this tampering or loss of data from happening, as it relies on the data being stored externally.

Therefore, in order to optimally use blockchain-based hash validation, it should be coupled with a way of restoring lost or corrupted data, such as a regular backup protocol or distributed data. Without these measures, the method will only function as validation, which is still valuable on itself.

B. Limits of the proof of concept implementation

In the proof of concept audit trail, multiple changes are aggregated before sending the corresponding blockchain transaction. This means that the application is potentially vulnerable to crashes or outages during the auditing process. This could

lead to an incomplete audit entry being stored to the database, while nothing gets sent to the blockchain, invalidating the audit entry.

The blockchain transactions are sent asynchronously because it can take some time before a transaction is executed and accepted on the Ethereum blockchain. We observed that new transactions fail when five or more transactions are already being processed at the same time. This is usually not a problem for smaller applications, but can definitely impact larger applications. When using this implementation in a production environment, a method needs to be developed to handle these failures.

C. Potential improvements to the method

Currently, the method stores an identifier-hash mapping in a smart contract to validate the integrity of the corresponding data. It is difficult to infer anything about the data by just looking at these data inside the smart contract. In the future, we might want to store additional metadata along with the hash inside the smart contract. This could be achieved by adding a MetaData struct to the smart contract, which could contain additional information. This MetaData struct could then be added to the mapping instead of just a data hash. An example MetaData struct could look like this:

```
struct MetaData {
    string user;
    uint256 timestamp;
    bytes32 hash;
}
```

D. Recommendations for further research

Because blockchain-based hash validation relies on the data being stored separately from the blockchain, it is only able to detect tampering, but it can not prevent tampering. It could be interesting to research we can fully prevent data tampering by storing the actual data on the blockchain.

The Ethereum gas limits put a limit of around 11 kB of data stored on the blockchain per transaction. The price of gas makes storing data on the blockchain very expensive. These issues greatly discourage the storage of larger amounts of data on the blockchain, but they do not make it impossible. While it is probably not practical for real-world usage, it is interesting to see if a proof of concept could be developed for storing any data on the blockchain as a way to guarantee their integrity.

There could also be potential in InterPlanetary File System (IPFS) for data storage. IPFS is a distributed file system that offers deduplication and version history for all stored data. IPFS could provide data storage that is resistant to tampering or corruption because of its distributed nature and checksum verification [16].

The IPFS website [6] also showcases the possibility to integrate IPFS with blockchain technology by storing large amounts of data with IPFS, and placing links to these IPFS data on the blockchain linking to certain specific versions of data in IPFS.

[6]https://ipfs.io/ (accessed 2018-05-31)

Acknowledgement

We would like to thank the EU PROCESS project (grant no 777533) for supporting this work. We would also like to thank Dan Haywood for his input throughout the implementation of our proof of concept.

References

[1] S. Ackerman, "Newest cyber threat will be data manipulation, us intelligence chief says," *The Guardian*, 2015. [Online]. Available: https://www.theguardian.com/technology/2015/sep/10/cyber-threat-data-manipulation-us-intelligence-chief

[2] K. Zetter, "The biggest security threats we'll face in 2016," *Wired*, 2016. [Online]. Available: https://www.wired.com/2016/01/the-biggest-security-threats-well-face-in-2016/

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[4] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[5] N. Szabo. (1996) Smart contracts: Building blocks for digital markets. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[6] R. Weber, "Audit trail system support in advanced computer-based accounting systems," *The Accounting Review*, vol. 57, no. 2, pp. 311–325, Apr. 1982. [Online]. Available: https://search.proquest.com/docview/1301314968

[7] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transactions on Information and System Security*, vol. 2, no. 2, pp. 159–176, May 1999. [Online]. Available: https://www.schneier.com/academic/paperfiles/paper-auditlogs.pdf

[8] A. Tripathi and M. Murthy, "Method and system for providing a tamper-proof storage of an audit trail in a database," Patent 6 968 456, 2005. [Online]. Available: http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=6968456

[9] N. Andersen, "Blockchain technology: A game-changer in accounting?" Mar. 2016. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/de/Documents/Innovation/Blockchain_A%20game-changer%20in%20accounting.pdf

[10] A. Sutton and R. Samavi, "Blockchain enabled privacy audit logs," in *The Semantic Web – ISWC 2017*. Cham: Springer International Publishing, 2017, pp. 645–660. [Online]. Available: https://doi.org/10.1007/978-3-319-68288-4_38

[11] I. Barinov, V. Lysenko, S. Beloussov, M. Shmulevich, and S. Protasov, "System and method for verifying data integrity using a blockchain network," Patent 2 018 025 181, 2018. [Online]. Available: http://www.freepatentsonline.com/y2018/0025181.html

[12] I. Zikratov, A. Kuzmin, V. Akimenko, V. Niculichev, and L. Yalansky, "Ensuring data integrity using blockchain technology," in *2017 20th Conference of Open Innovations Association (FRUCT)*, Apr. 2017, pp. 534–539. [Online]. Available: https://ieeexplore.ieee.org/document/8071359/

[13] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger — byzantium version f72032b – 2018-05-04," 2018. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[14] R. Kalis, "Using blockchain to validate audit trail data in private business applications," 6 2018. [Online]. Available: https://esc.fnwi.uva.nl/thesis/centraal/files/f1051832702.pdf

[15] D. Haywood. (2018) Apache isis domain services. [Online]. Available: https://isis.apache.org/guides/rgsvc/rgsvc.html

[16] J. Benet, "Ipfs - content addressed, versioned, p2p file system (draft 3)," Unknown. [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf