

Contents lists available at [ScienceDirect](https://www.sciencedirect.com)

# Blockchain: Research and Applications

journal homepage: [www.journals.elsevier.com/blockchain-research-and-applications](http://www.journals.elsevier.com/blockchain-research-and-applications)

## AudiWFlow: Confidential, collusion-resistant auditing of distributed workflows



Xiaohu Zhou<sup>a,\*</sup>, Antonio Nehme<sup>a</sup>, Vitor Jesus<sup>b</sup>, Yonghao Wang<sup>a</sup>, Mark Josephs<sup>a</sup>, Khaled Mahbub<sup>a</sup>, Ali Abdallah<sup>a</sup>

<sup>a</sup> School of Computing and Digital Technology, Birmingham City University, Birmingham, B4 7XG, UK

<sup>b</sup> Aston Business School, Aston University, Birmingham, B4 7ER, UK

### ARTICLE INFO

#### Keywords:

Auditing  
Distributed workflows  
Confidentiality  
Blockchains  
Smart contracts

### ABSTRACT

We discuss the problem of accountability when multiple parties cooperate towards an end result, such as multiple companies in a supply chain or departments of a government service under different authorities. In cases where a fully trusted central point does not exist, it is difficult to obtain a trusted audit trail of a workflow when each individual participant is unaccountable to all others. We propose AudiWFlow, an auditing architecture that makes participants accountable for their contributions in a distributed workflow. Our scheme provides confidentiality in most cases, collusion detection, and availability of evidence after the workflow terminates. AudiWFlow is based on verifiable secret sharing and real-time peer-to-peer verification of records; it further supports multiple levels of assurance to meet a desired trade-off between the availability of evidence and the overhead resulting from the auditing approach. We propose and evaluate two implementation approaches for AudiWFlow. The first one is fully distributed except for a central auxiliary point that, nevertheless, needs only a low level of trust. The second one is based on smart contracts running on a public blockchain, which is able to remove the need for any central point but requires integration with a blockchain.

### 1. Introduction

Distributed workflows with multiple organisations involved, cooperating towards a certain outcome (such as a supply chain), are now a common way of work leveraging the potential of the Internet. This is common in many domains, including governments, digital health, education, engineering, supply chains, goods distribution, etc. Collaboration is enabled by interoperable applications through which each organisation contributes to a workflow. A key enabler is trust: organisations need to trust each other that each will deliver their part as contracted. When a problem occurs, the workflow needs to be audited to determine what failed.

We start with a simple example. A shopper orders a book online. After payment using an independent payment processor, the book is collected from the warehouse and sent for post dispatch. An independent courier picks up the book and transports it to the destination hub. The book is then delivered by a person to the door of the shopper. The shopper signs a

form, and the workflow is completed. One can see that all the participating parties are, for the most part, unknown to each other. If a problem occurs, the shopper needs to be refunded, and an audit of the particular workflow started. Such an audit is typically time-consuming and lengthy and, most of the time, the costs do not justify the effort. A full refund, “no questions asked”, is then issued to the shopper. A key obstacle is that, if an audit does happen, a dishonest party is able to easily hide or create evidence to waive its liability. The likelihood is that one runs into an inconclusive “finger-pointing” problem that only a human judge can resolve using the law and principles such as the “balance of probabilities”. The problem AudiWFlow tackles is how to generate evidence, as the workflow happens, that is able to guarantee that all the evidence is stored with integrity, is unforgeable, is available, and cannot be repudiated.

With massive digitisation, nearly every domain has similar needs—supply chains [1], inter-department business processes [2], e-government services [3,4], etc. Note that the problem becomes trivial if

\* Corresponding author.

E-mail addresses: [xiaohu.zhou@mail.bcu.ac.uk](mailto:xiaohu.zhou@mail.bcu.ac.uk) (X. Zhou), [antonio.nehme@bcu.ac.uk](mailto:antonio.nehme@bcu.ac.uk) (A. Nehme), [v.jesus@aston.ac.uk](mailto:v.jesus@aston.ac.uk) (V. Jesus), [yonghao.wang@bcu.ac.uk](mailto:yonghao.wang@bcu.ac.uk) (Y. Wang), [mark.josephs@bcu.ac.uk](mailto:mark.josephs@bcu.ac.uk) (M. Josephs), [khaled.mahbub@bcu.ac.uk](mailto:khaled.mahbub@bcu.ac.uk) (K. Mahbub), [ali.abdallah@bcu.ac.uk](mailto:ali.abdallah@bcu.ac.uk) (A. Abdallah).

a central party is able to coordinate and gather evidence; however, trusting a central party is a difficult problem in itself, especially in a distributed workflow, where parties may not even know each other beyond their adjacency. Furthermore, simple log recording is not enough, as any valid evidence cannot be open to manipulation [5]. A further problem is collusion both with a central entity managing the workflow or between two adjacent parties in the workflow topology to tamper with digital evidence. Even if the orchestration of the workflow is managed in the cloud, a privileged insider can tamper with the logging process. Finally, the confidentiality requirements should be noted [6]. In a purely distributed workflow, organisations may want to only deliver the expected outcome and not disclose any other information.

### 1.1. Scenario

To illustrate the challenge, we present and informally analyse a simple working scenario of health insurance, see Fig. 1.

Alice (A) wants health insurance from insurer Bob (B). Alice has to provide her medical history to Bob and allow Bob to contact her doctor, Dr. Cathy (C). Insurer Bob also needs Alice's family medical history, but this has to go through Dr. Cathy, who, following a confidentiality-friendly approach, will provide an overall report after she contacts the family doctors, Dr. Dippy (D) and Dr. Eva (E).

To save on insurance costs, Alice asks her doctor, Dr. Cathy, to provide an untruthful medical record. Bob thus obtains Alice's medical history from Dr. Cathy (which includes reports from Dr. Dippy and Dr. Eva). Happy with the outcomes, insurer Bob offered a deal with which Alice was happy.

After two years, Alice claimed compensation after a medical incident. Referring to Alice's insurance claims, Bob investigated all documents about Alice and found that Alice's medical records provided by Dr. Cathy contradict records in the hospital. Insurer Bob rejects Alice's claim. Alice counterargued that the insurance company was responsible for collecting her medical history and is thus entitled to compensation. Insurer Bob then tries to obtain the contacts of Dr. Dippy and Dr. Eva, which Dr. Cathy refuses to give on behalf of confidentiality.

As we see, there are a number of parties (A, B, C, D, E), which are independent and, a priori, individually unaccountable to any other party.

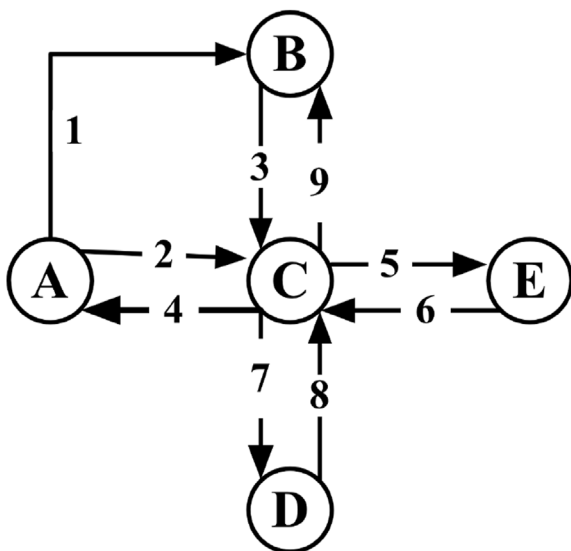


Fig. 1. The example scenario of health insurance.

All have to collaborate towards the end result, which is health insurance for Alice. A further component is that some parties are not known to other parties: B does not know who are D and E and, in fact, only knows that there are other parties beyond C. Overall, we have a workflow topology (as in Fig. 1), where vertices are the collaborating parties and the directed edges are the sequence of actions and deliverables. The audit trail consists of all the interactions represented by the edges of the graph.

In the absence of perfectly shared information, there is no way for any party to verify the accuracy of the information provided. For example, insurer Bob is in a position where he cannot prove Alice is at fault. This is because Alice can claim that insurer Bob colluded with doctors from the hospital and modified her original documents to close a sale. Insurer Bob also cannot find any traces of a collusion attack between Dr. Cathy and Alice. Finally, Dr. Cathy denies ever signing the records that insurer Bob holds and raises the suspicion that her signature was forged.

This scenario illustrates the challenges of obtaining strong evidence in the case of a distributed multi-party workflow. The key requirements are as follows. First, all participants must be accountable for their actions, either by denying a past action or manipulating an audit record. Second, no participant should be able to destroy evidence, as this can be a way to waive liability. Third, one should be able to detect if dishonest parties collude even if it is not impossible to prevent false records between themselves. Fourth, and finally, there are important use-cases where confidentiality is required. This comes in two forms: (1) confidentiality of information exchanged and (2) identity of the participants themselves beyond a certain point in the workflow graph. We discuss these requirements in more detail when formalising our problem in Section 4.

### 1.2. Organisation of this paper

Section 2 reviews the literature on existing schemes of audit trails built. Section 3 formally defines the problem, presents a threat model, and introduces notation and terminology. Section 4 presents two architectural approaches to solve the problem. The first is based on a peer-to-peer distributed architecture but relies on an accessory central node. The second removes the central node by replacing it with a public blockchain; we used Ethereum, a public blockchain and one that now has proven the ability to deliver complex requirements while maintaining the security and integrity expected from blockchains. In Section 5, we describe how we implemented both approaches and discuss the experimental results. In Section 6, we conclude our paper and discuss future work.

## 2. Related work

Despite its crucial importance, workflow auditing is a topic that has not been abundantly discussed in the literature. We split this section into two subsections: conventional systems auditing and using the properties of blockchain to secure evidence and records.

### 2.1. Secure auditing and logging system

The literature about conventional auditing systems commonly reviews the cloud and database storage of logs in order to store proof of evidence. This raises the problem of protecting audit records at storage, for which encryption is a commonly used technique, as followed in the literature [7–9]. However, encryption requires the management of keys, which raises problems of confidentiality or destruction of evidence. In simple terms, whoever owns the keys can modify or breach the confidentiality of evidence. To mitigate these problems, some approaches using secret sharing have been proposed [7,10]; other approaches rely on integrating secret sharing with multi-party workflows, such as adopting a software-defined network controller [5]. The problem of trusted auditing is also pervasive and is a requirement across many different fields, such

as in e-government services [8].

Zawoad et al. built a secure logging service model, named “SecLaaS”, to achieve the confidentiality and integrity of logs in cloud storage [6]. Although this approach minimises the risk of unauthorised modifications of the logs, the cloud service provider is entrusted with the generation of the audit trail without any means to verify the accuracy of the audit records. Flores [11] presented a salting-based authentication module to target credential protection in user registration and authentication. A database intrusion detection module is also mentioned to report and record insider login attempts to the database within the network. Although it concerns malicious behaviors at the back-end and application levels, the irreversibility of records for login attempts in the database is not discussed.

Rajalakshmi et al. [8] proposed a mechanism addressing the requirements of secure logging and auditing between untrusted parties that uses homomorphic encryption. It aims to prevent the exposure of private information and save logs in the tamper-proof cloud so that the logs can be used as audit forensics. Ma and Tsudik [9] used hash chains to verify the logging entity. They aimed to detect attempts to modify logs generated before a compromise of the logging entity and relied on trusted storage for their approach. However, the approaches of Refs. [8,9] do not consider collusion-related attacks between involved participants.

## 2.2. Distributed ledgers in auditing

Distributed ledger technology (DLT) in auditing is a promising direction at present. Blockchain is a popular technology in the DLT. A number of approaches in the literature based on blockchain technology have been applied to data auditing and provenance for different industry fields currently, such as adoption in business collaborative processes [2] and in health data sharing [12,13]. Different proposals achieve different levels of security and data confidentiality in auditing, and some of them do not target the same trust level for the logging entity. These approaches typically rely on smart contracts to produce audit trails and verify the records on the blockchain network.

Cucurull and Puiggalí [14] proposed to add checkpoints to storage that generate logs that are then published to the Bitcoin blockchain; however, tampering with logs is possible between the checkpoint intervals. Putz et al. [15] targeted this limitation by enabling integrity verification of each log entry through hashes published on a permissioned blockchain. They verified each log collected from different organisations. Tian [1] used blockchain with distributed databases to track a food supply chain process. Each participant in the supply chain generates and maintains audit records of its part of the process and submits proof of authenticity to the blockchain. Lu and Xu [16] presented another application of blockchain to verify whether a product is genuine in a supply chain. Ahmad et al. [17] showed a blockchain-based application ‘BlockAudit’ that saves logs into the relational database management system and converts logs into the JSON package as a transaction reported to the Hyperledger blockchain. These approaches reflect a common assumption that an entity is trusted to produce digital evidence; we argue that there needs to be a (near) real-time verification of the audit data as it is generated.

Tapas et al. [18] proposed an approach to leverage DLT to detect misconduct between customers and providers in cloud storage. They relied on mutual challenges between clients and cloud service providers to verify the authenticity of evidence reporting but did not consider workflows including multiple administrative domains, which is a requirement we take into account in this paper. Weber et al. [2] used smart contracts to check and control interactions in the execution of business processes. Audit trails are generated following smart contracts execution. However, this model requires the data not to be encrypted at the level of smart contracts, which jeopardises the confidentiality of their approach. Pourmajidi and Miransky [19] presented a prototype of a blockchain-based log system to store and verify the collected logs from different cloud storage providers. They saved logs and their hashes into the hierarchical ledger and designed APIs for clients to interact with the

storage. Not requiring encryption at rest is a threat to the confidentiality of the records.

## 2.3. Contributions of this paper

In this paper, we introduce AudiWFlow, which is a confidentiality-friendly and collusion-resistant auditing approach for distributed workflows. We discuss two implementations of AudiWFlow: one uses a central coordination point (CP) to have minimal trust requirements between the involved third party and participants, and the other relies on a public blockchain with smart contracts to replace the third party for audit trail record and sharing. The double lock with key pairs of the workflow and of a participant in the workflow enables the encrypted exchange of messages and encrypted audit records between participants. Both of our implementation approaches offer the same assurances for the confidentiality, integrity at generation and storage, and the availability of audit records reflecting the contribution of each participant in a workflow.

Our analysis of approaches in the literature [7–9] that rely on a centralised mechanism for auditing distributed/general workflows concludes that collusion between participants and a party trusted to record audit trails makes tampering with or destroying digital evidence possible, as well as breaching the confidentiality of workflow transactions. Our approach using a central point builds a complete audit trail through timely following of exchanged messages with encrypted audit records and requires the nodes to push these audit records into the central server. The audit trail and timely verification on both sides of the server and the client prevent malicious behaviors from participants, such as collusion attacks. It also solves the problem of data accuracy and integrity discussed in the literature [6,20]. On the other hand, prior practices [1,14,16,17] that use blockchain for auditing workflow collaborations spreading across multiple organisations fall short of providing assurances for the combination of the availability, confidentiality, and correctness of reported audit records. Our approach using a public blockchain runs smart contracts to exchange a valid digest of an audit record for an attestation of the integrity of the verified record, which minimises the risk in the involvement and affection of third-party on malicious acts. The audit record is shared, alongside the exchanged message, with the next participant. Both the sender and recipient store the audit record to compare it with a digest of the message in the blockchain. All transactions are encrypted with a secret key of a workflow or node key pair, including in smart contracts. It addresses the problem of data disclosure that may happen in the model of [2,19]. For any arbitrary distributed workflow, our approach offers a robust and confidentiality-friendly way to record and verify audit records at any desired granularity while giving auditing capability to key shares of participants in a distributed workflow.

## 3. Problem statement

The key requirements an auditing architecture needs to satisfy are accountability, non-repudiation, confidentiality (records and graph), availability, and collusion detection.

Fig. 2 shows a linear workflow that has an auditing component (CP).

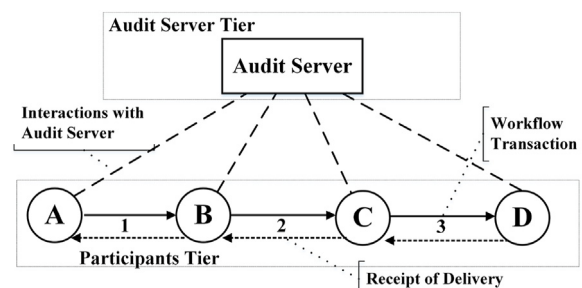


Fig. 2. The representation of the system structure.

The specific role of CP will be discussed later. Whereas in our first approach we use a central CP, in the blockchain approach this is replaced with smart contracts running on a public blockchain. In the simple topology of Fig. 2, participant A starts by requesting work from B; B then requests work from C to complete the request from A. Then, it continues to D. When D performs the expected action based on the request from C, the workflow terminates. The evidence generated while the workflow progresses is composed of the individual audit records. Should a dispute arise at a point in the future, this evidence must hold all participants accountable for their contributions.

Each workflow  $\mathcal{W}$  is modelled as a topology with a directed graph  $G=(V, E)$ . Fig. 3 gives a working example where participants are  $V=\{A, B, C, \dots\}$  and the edge set is  $E=\{1, 2, 3, \dots\}$ . Each participant contributes to the overall work done, in sequence. We assume the topology of a particular workflow is established before it starts and is static for its duration. We further assume, without loss of generalisation, that the graph is acyclic when annotating each participant as a requester or responder. In other words, the workflow uses a path over the graph such that no node is a requester or a responder twice. If a particular workflow uses the same participant twice at different times, the workflow graph is different.

### 3.1. Notation

We use the following notation:

- a workflow  $\mathcal{W}$  executes over a directed graph  $G=(V, E)$  and associated audit evidence  $\mathcal{A}$  produced during the execution of the workflow.
- $V=\{A, B, C, \dots\}$  is the set of members of a workflow.
- $E=\{1, 2, 3, \dots\}$  is a set of the sequence of participants' actions in a workflow.
- CP is an auxiliary Coordination Point.
- $pk_{\mathcal{W}}$  and  $sk_{\mathcal{W}}$  are the public and private keys of a workflow  $\mathcal{W}$ , respectively.
- $pk_i$  and  $sk_i$  are the public and private keys of participant  $i=1, 2, \dots, N$  with  $|V|=N$  participants.
- $k_j$  is the  $j$ -th share of a threshold key, in the sense of secret sharing, and  $j=1, 2, \dots, N$ . It is derived from  $sk^{\mathcal{W}}$ , and any threshold  $K \leq N$  members can recover the key.
- $M_1||M_2$  denotes the concatenation of messages  $M_1$  and  $M_2$ .
- $M_{ij}$  is a message sent from participant  $i$  to  $j$ .
- $sign_i(M)$  is a message  $M$  signed by participant  $i$ .
- $enc_i(M)$  is message  $M$  encrypted with  $pk_i$ .
- $enc_{\mathcal{W}}(M)$  is message  $M$  encrypted with  $pk_{\mathcal{W}}$ .
- $hash(M)$  is a digest of message  $M$  using a one-way collision-resistant function (a "hash").

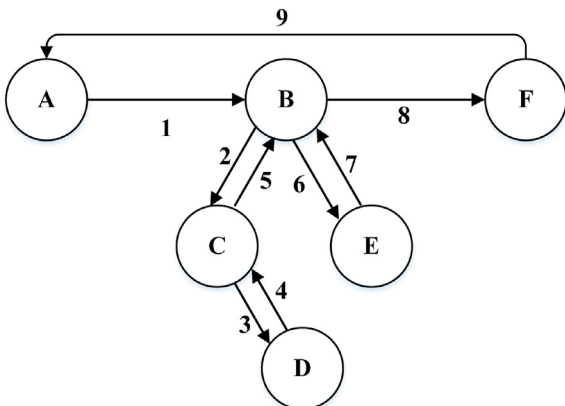


Fig. 3. The representation of an example workflow [20].

We use the following message formats in our signalling diagrams:

- authenticity

$$P_{ij} = enc_j(sign_i(M_{ij}))$$

Node  $i$  is sending a plaintext output  $M_{ij}$  to  $j$  and, to assure authenticity in a future audit, it signs the message. For confidentiality, node  $i$  encrypts the result with  $j$ 's public key.

- receipt

$$R_{ij} = sign_j(sign_i(M_{ij}))$$

After  $i$  sends output to  $j$ ,  $j$  returns a receipt of delivery to  $i$ .

- audit record

$$A_{ij} = sign_i(enc_{\mathcal{W}}(sign_i(M_{ij})))$$

This message generates an audit record that, for confidentiality, is encrypted with the public key of the workflow. The resulting object is then signed again with  $i$ 's private key as we will need a quick verification of the record without the need to inspect its contents. The full audit trail of workflow  $\mathcal{W}$  is  $\mathcal{A}^{\mathcal{W}} = \{A_{1,2}, A_{2,3}, \dots, A_{ij}, \dots\}$  with indexes matching the graph path of the workflow.

- integrity proof

$$I_{ij} = hash(A_{ij})$$

This message simply extracts the digest of an audit record.

### 3.2. Threat model

Our architecture takes into account the following security requirements for our threat model:

- **completeness, integrity, and authenticity** The audit evidence  $\mathcal{A}^{\mathcal{W}}$  generated as workflow  $\mathcal{W}$  executes needs to capture all inputs and outputs, in the sequence as they were generated, with participants' identities embedded in the evidence.
- **non-repudiation** No participant should be able to dispute recorded evidence.
- **confidentiality** The evidence should be confidential to a certain agreed level; furthermore, it should be possible to protect the graph (participants' identities and associations) if desired.
- **availability** No participant should be able to destroy evidence at any time after release.
- **collusion detection** No two or more participants should not be able to collude and not be detected. Note that the effort is in detection.

## 4. Architectures

Our approach to the problem is to store audit records, as they are produced, in all nodes participating in the workflow. This will ensure the availability of the audit trail. In this section, we present our approaches to the problem.

### 4.1. Overview

The degree of availability of the audit trail is bound to the size of the subset of participants that are allowed to inspect the evidence, as all evidence is, primarily, stored in encrypted form. The key technique is to use a workflow-wide secret key  $sk_{\mathcal{W}}$  which is split into  $K \leq N$  shares. The shares are then distributed to all participants, and it requires  $K$  out of  $N$  to open the full audit trail. All audit records are encrypted with  $pk_{\mathcal{W}}$ . A dispute can be resolved if  $K$  nodes decide independently to open the

evidence and verify the claims.  $K$  is dependent on the case, but it should be at  $K > 2$ , as we will not be able to detect collusion in case 2 adjacent nodes are malicious.

As the audit records are generated, they are verified by both nodes involved. If B responds to a request from A, both A and B will verify the records of each other; if a discrepancy exists, the detecting node will alert the whole workflow. If no discrepancy exists, the audit record is sent to the CP. The CP then displays the individual record encrypted with  $pk_{\mathcal{V}}$  to all participants who keep a copy of the encrypted audit records.

Note that the CP only exists to manage the distribution of the individual records. As a malicious actor, its effects are limited to availability, as it could help distribute corrupted evidence. The CP is challenged by every participant to verify that it is not acting maliciously. Details about our approach to challenging the CP are discussed in Section 4.5. In this sense, we propose two alternative architectures:

- **using a CP.** If the availability of the CP can be trusted for the only purpose of displaying encrypted audit records during the workflow, using a CP makes our architecture simple to adopt.
- **using smart contracts with a public blockchain.** If full assurances are required, we propose to replace the CP with a public blockchain which, by being not managed centrally but behaving as if it were, cleanly provides a trust anchor. It brings its own security challenges, however, which we will discuss later.

After the workflow terminates and there is a dispute,  $K$  out of the  $N$  nodes can decide to join their threshold keys and recover the complete audit trail. Note that all  $N$  nodes have a copy of the audit trail, albeit encrypted with  $pk_{\mathcal{V}}$  to protect confidentiality.

#### 4.2. Key Management

Our scheme involves several keys which need to be securely generated, distributed, and locally validated. This preparatory step is common to both of our approaches. There are two sets of keys involved.

- 1) **Node Keys:** We assume that identity credentials  $pk_i$  and  $sk_i$  for every participant are managed before the workflow starts—for example, with conventional certificates that can be verified. It should be noted that storing public keys for a particular workflow can potentially disclose the number and identity of participants. We assume this is an acceptable relaxation and leave this case for future work.
- 2) **Workflow Keys:** We need to generate and distribute, for each workflow  $\mathcal{W}$ , the keys  $pk_{\mathcal{V}}$  and  $sk_{\mathcal{V}}$ . The threshold keys  $k_j$  are derived from  $sk_{\mathcal{V}}$ : we use a verifiable secret sharing mechanism [21–23] to create  $N$  shares where  $K$  is enough to reconstruct the secret. Workflow key distribution can either be done through direct messages to each participant over a secure channel or by encrypting each share of the key with the corresponding participant's private key and posting them to CP.

We adopt a pragmatic approach and task the participant with the least incentive to corrupt the audit trail to generate and distribute the key shares. This is generally the first or last participant depending on the workflow: the first participant in a workflow can be a gift shop salesman required to keep track of orders for its customers; and the last participant in another workflow can be a supermarket manager who needs to keep track of which food, ordered by customers, is from.

Furthermore, using a publicly verifiable secret sharing (PVSS) scheme enables any participant to verify that other participants have received authentic shares of the same secret without revealing this secret. PVSS was first proposed by Stadler [23] and used by Schoenmakers [24] for an electronic voting application. D'Souza et al. [25] later adopted PVSS and explicitly required the secret shared among participants to be a legitimate private key equivalent to a public key in their approach to support key recovery. In our context, this enables any entity knowing the public keys

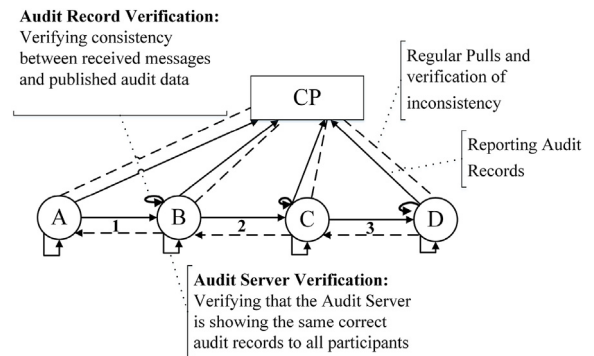


Fig. 4. Architectural approach using a coordination point (CP) [20].

of workflow participants as well as the equivalent public key  $pk_{\mathcal{V}}$  of the shared secret  $k_i$  to verify that the key generator was honest with the key distribution and that the secret key  $sk_{\mathcal{V}}$  can be reconstructed with key shares.

#### 4.3. Approach using a CP

This approach uses a centralised CP as shown in Fig. 4 with the corresponding signalling diagram in Fig. 5. The CP is a special node that can be hosted by any participant or a third party. It is only responsible for distributing the audit records  $A_{ij}$  across all participants. It does not permanently store any data.

We do not strictly need a CP in the topology of the graph because of confidentiality. In that case, nodes can distribute individual audit records (and keys) between themselves. The trade-off incurs added coordination complexity, which may not be acceptable.

Node  $i$  sends message  $P_{ij}$  to node  $j$ . Node  $j$  extracts the original output  $M_{ij}$  and is now able to continue the workflow. Node  $j$  further performs the following verification. First, it verifies that the signature in  $P_{ij}$  is valid. Second, it uses its private key to extract  $sign_i(M_{ij})$ , which it keeps as a receipt. As for  $i$ , it gets receipt  $R_{ij}$  from  $j$ . In parallel,  $i$  sends to CP the audit record  $A_{ij}$ .

Furthermore, node  $j$  needs to confirm that  $i$  sent the expected audit record  $A_{ij}$  to CP in order to prevent the corruption of the audit trail. Node  $j$  extracts  $sign_i(M_{ij})$  and encrypts it with  $pk_{\mathcal{V}}$ . The result is compared to what was stored in CP by  $i$ . Should there be any discrepancy between this record and what  $i$  stored in CP, the workflow is stopped. This is the gist of our approach for audit records verification. The precise way of notifying all participants of a halted workflow is out of scope; solutions could pass by using a messaging protocol based on gossip (peer-to-peer propagation of direct messages).

An important aspect of our proposal is that all participants verify and

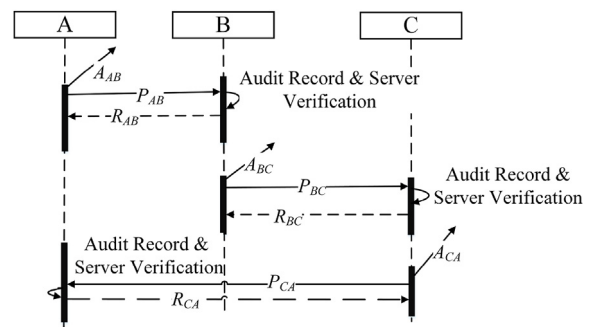


Fig. 5. Signalling diagram using a coordination point (CP) [20].

keep the whole audit trail. Each participant will independently verify that the audit records they periodically pull from CP are validly signed by the node that generated them. This is done by verifying that the records on the CP are correctly signed. This is important to prevent the destruction or corruption of the audit trail by either a node or by CP. Note that a potential attack is that CP will selectively distribute incorrect audit records depending on which node is requesting. To mitigate that, nodes verify that they all received the same copy of the encrypted records by displaying a digest of the records and their signature over the digest on the CP.

Overall, all participants will have the full audit trail when the workflow terminates. No participant will be able to inspect the actual audit trail since it is encrypted with the workflow key  $pk_W$ . However, each participant has a share of the key. Upon open dispute, a subset  $K$  of all participants can decide to gather their key shares and open the whole audit trail and resolve the dispute.

4.4. Approach using a public blockchain

The previous approach uses a central CP. We envision that, in most practical cases, simplicity of use outweighs the weaknesses it introduces. In this section, we show how a public blockchain able to run smart contracts can fully replace any central point while reducing the surface attack area. The trade-off is, essentially, the complexity of integration. Fig. 6 depicts this approach.

We did not consider any private blockchain for two main reasons. Firstly, projects have been abandoning the idea of private blockchains and have been moved to public blockchains such as Ethereum, Cardano, or Algorand. Second, a private blockchain cannot deliver the same security assurances as a public blockchain due to its very high scale. For example, where it is virtually impossible to modify a block of Bitcoin or Ethereum, and even less to modify without detection, a private blockchain, necessarily much smaller, suffers from governance and administration weaknesses and, should access control not be perfect, modification without detection is much easier. We have used Ethereum and adopted the consensus mechanism used by the technology.

The blockchain fully replaces CP. Since we use a public blockchain with auditable and open code, any participant can independently verify that the code meets the intended requirements. The public keys of all nodes need to be stored in the blockchain and are always available. Audit record hashes can be verified at any time, etc. Note that reading a hash of the public records does not disclose the identity of the participant, albeit the number of participants needs attention (which we will presume acceptable). This depends on the smart contract code, which, to reiterate, is publicly available for inspection.

The anonymity of the individuals participating in the workflow is not jeopardised by the use of a public blockchain. The identity of the individual approving or requesting a transaction at the level of each participant (node) is part of the encrypted payload in the audit record. The nodes in Fig. 2 represent the platform (a mobile app or website) that the individuals are using. The signature of the service provider only shows on the public ledger, and this is a reasonable level of privacy given that the

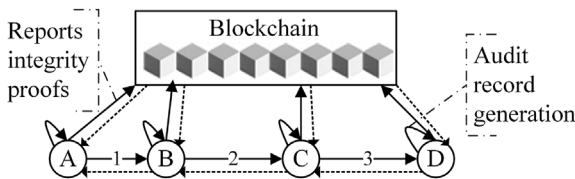


Fig. 6. Architectural approach using public blockchain.

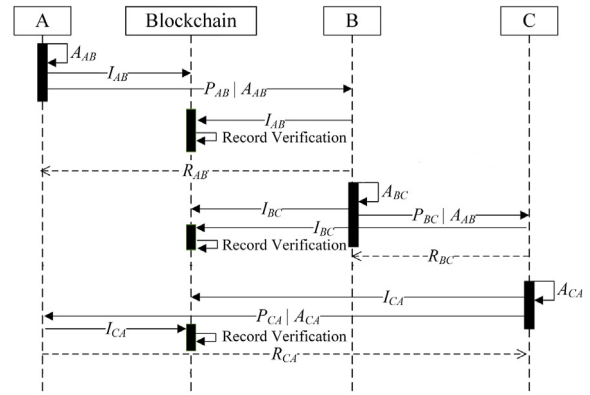


Fig. 7. Signalling diagram using public blockchain.

list service providers are public in most use cases.

The blockchain can also act as a trusted messaging channel, such as halting a workflow and notifying all nodes. Note that one still needs to store audit records somewhere outside of the blockchain. Participants can choose this location for the record storage. The hash of the records (on the blockchain) will assure nodes of authenticity and integrity. The audit records are immediately accessed by each node and locally stored. If a particular node attempts to retrieve an audit record but fails to find the correct record or does not receive a response from the data storage that is chosen by nodes to store the encrypted audit record, it can broadcast an alert on the blockchain to warn other workflow nodes about this failure.

Our protocol uses a public blockchain, which consists of four phases: initialisation, data exchange, record verification, and distribution. See the message diagram in Fig. 7.

The initialisation phase involves key generation and node registration. We need to create a key pair for the wide workflow,  $pk_W$  and  $sk_W$ . Similar to the previous case, we still need a trust anchor (may be the first node in the workflow) to bootstrap the process. We once more delegate the choice of the entity who coordinates the distribution to the specific use case, such as to the participant with the least incentives to be malicious. Whereas  $pk_W$  is stored in the blockchain for public access, the  $k_i$  split shares of  $sk_W$  are distributed to each participant. We further assume that each participant has a cryptographic key pair previously generated and that all have stored and made available their public keys in the blockchain. When a participant wants to get a public key from the blockchain, a smart contract is called to find the respective key. In the first phase, all involved workflow nodes are required to register in the blockchain network before any act starts. Nodes can push and pull records to/from the blockchain.

During the data exchange phase, the actual execution of the workflow happens, and interactions between each participant are similar to those in the CP approach. Nodes will cycle through a record distribution and verification phase. The essential difference is that message  $I_{ij}$  (audit record integrity), along with metadata and a location to the actual audit record, is pushed and stored in the blockchain. The messages are never stored on the public blockchain; only the integrity proofs  $I_{ij}$  of the audit records are stored and verified through smart contracts. Once published, any participant can verify the records and obtain the actual audit record from a public location. Algorithm 1 shows the pseudocode of the smart contract for record verification; the algorithm shows that the smart contract only performs a string comparison operation, without any additional cryptographic operations to the ones dictated by the consensus mechanism of the blockchain, to verify that the integrity proof (the digest value of an audit record) has been published on the blockchain.

**Algorithm 1.** Smart Contract on Record Verification**Algorithm 1** Smart Contract on Record Verification**Input:**  $HashAudit\_Rec[]$ 

▷ hash value of an audit record reported by a participant

 $Integrity\_Rec[]$ 

▷ integrity proofs reporting by the recipient

**Output:** Boolean indicating if a record is stored or verified successfully

```

1: if  $HashAudit\_Rec[] \neq \emptyset$  then
2:   func(saveHash)      ▷ save Hash into blockchain
3:   return True
4: end if
5: Blockchain.push(Integrity_Rec[])
   ▷ upload an integrity proof to blockchain for comparison
6: if Blockchain.push(Integrity_Rec[])  $\neq \emptyset$  then
7:   func(getHash)
8:   func(compareHash)
9:   for each Audit_Rec[]  $\in$  Integrity_Rec[] do
10:    while Audit_Rec[hashId] == Integ_Rec[hashId]
11:      return True
12:    end while
13:  end for
14: else
15:   return False
16: end if

```

#### 4.5. Security analysis

In this section, we discuss our approach by revisiting our key requirements: completeness, integrity and authenticity, non-repudiation, confidentiality, availability, and collusion detection.

1) Completeness, integrity, authenticity and non-repudiation: We consider a dishonest participant who attempts to tamper with data in the existing audit records or disseminate an incorrect audit record. Our approach can detect these situations since each individual audit record ( $A_{ij}$ ) is verified immediately after generation while each node keeps receipts  $R_{ij}$  of locally exchanged (between pairs of nodes) audit records. Unless two adjacent nodes collude (as discussed below) and assuming the key distribution (as below) is secure, this assures non-repudiation. All participants also perform the Audit Record Verification mechanism after the exchanged data are received. In this sense, there are a few points where data corruption or concealment is checked.

In the CP case, the central element can potentially modify audit records. It can also provide invalid records, perhaps even selectively, depending on which participant is requesting. This is prevented since all records need to be signed and verified by all nodes. For the blockchain-based case, the code of the smart contracts is publicly auditable while not disclosing any important information about the workflow itself.

2) Availability and confidentiality: The full audit trail, if desired, can be kept by all participants while not jeopardizing the confidentiality of any participation in the workflow given that the records are encrypted with the workflow key that no participant holds. The protected audit trail can be opened in case of a dispute and a pre-defined (configurable)  $K$  number of participants agree. Availability of the audit trail is, at the limit, guaranteed since (up to) all nodes keep the

whole audit trail. Note that we use a verifiable shared scheme to prevent the distribution of invalid key shares.

3) Collusion-detection: Our scheme, and we informally argue none other, cannot prevent collusion between two adjacent nodes. Consider the following workflow:  $A \rightarrow B \rightarrow C \rightarrow D$ . Nodes B and C are free to decide to forge, modify or entirely destroy their internal audit records since no one else can verify or attest to their integrity. There will, however, be proof of collusion.

It should be noted that this attack can be made much less likely, for small workflows (say, up to 10 participants), if every node is to link the previous audit records in their own record. At least 4 nodes would have to collude. However, the generated audit records need to be open for verification and inspection among the 4. This scheme could be expanded to any number of nodes and, depending on the topology, quickly expand to the whole graph. In this case, no CP would be needed, as the trail of records could be made to propagate across the entire topology. We leave this approach out for simplicity.

Collusion involving the CP could be far more harmful, but as explained before, the CP only serves to coordinate the distribution of the audit trails and has special verification mechanisms associated.

4) Key management: Participants are assumed to protect their credentials, which are assumed to use well-known means such as those in the Public Key Cryptography eXchange. As such, we leave this particular aspect out of the scope of our paper.

Following the Public Verifiable Secret Sharing Scheme for the key distribution of the shares  $k_i$  of  $sk_W$ , participants can verify that the key generation and distribution are correct such that each of them has a correct share of the same secret key. This still raises the problem that  $sk_W$  is exposed by the participating node that generates the key. This is common in both the CP and blockchain cases. To minimise the risk of breaching the confidentiality of audit records, we assign the key generation role to the participants that have the least incentive to cheat, which is, in the case, a participant who does not hold the audit trail yet has an interest in it succeeding, such as the user requesting the workflow. We, nevertheless, acknowledge this is an open problem that is, seemingly, only resolved by either a collaborative generation of the shares that could be resolved or case-based design.

5) Complexity: A CP approach is attractive given its simplicity. The CP is little more than a simple file server with some verification logic. However, it requires a high availability of a CP. The blockchain approach seems to elegantly resolve those problems, but at the cost of integration of complexity, which, in itself, provides the possibility for implementation vulnerabilities both at the code of the smart contracts and also at a system level. For example, audit records need to be sent securely (in some form) to be stored out of the blockchain. Considering this is specific to an actual implementation, we do not discuss it further.

## 5. Evaluation

In this section, we present experimental evaluation results<sup>1</sup> of our implementation of the two described approaches<sup>2</sup>, using a CP and using smart contracts running on a public blockchain.

The workflow topologies were generated using BRITE<sup>3</sup>, a network topology generator. Even though BRITE was designed to generate Internet topologies, we hold the expectation that BRITE is sufficiently representative of real-world workflows for a small number of nodes (say,

<sup>1</sup> Relevant Data: <https://github.com/Jency/AudiWFlow.git>.

<sup>2</sup> <https://github.com/antonionehme/AuditingWorkflows-Blockchain>.

<sup>3</sup> <https://github.com/nsol-nmsu/brite-patch>.

up to 20). We configured BRITE to generate a connectivity graph based on the Barabasi-Albert algorithm with two key parameters:

- the number of nodes  $N$ , where we use  $N=\{10, 15, 20\}$ .
- the connectivity degree  $m$ , which is, on average, the number of nodes each node connects to. For example, for  $m=1$ , the topology is linear and for  $m=N$  the topology is a full mesh with all nodes connected to all nodes. We used  $m=\{2, 5, 7, 10\}$

For every pair of parameters  $(N, m)$ , we generated 5 different random topologies in order to generate enough randomness and extract statistical parameters such as averages. In total, we had about 100 different topologies.

Above the working node topology, we added either a special server (CP) or an interface to the Ethereum blockchain.

In terms of the software implementation, we should stress that this is experimental code, which is not designed for performance but only to demonstrate and validate our approaches. For this demonstration, we used the OpenSSL gensa function to generate the key pairs for the participants and workflow; 2048-bit was selected as the size of each of the keys. We also used a public implementation of Shamir Secret Sharing<sup>4</sup> to generate the shares of the workflow private key. At the level of the workflow participants, we relied on the Nimbus-JOSE library<sup>5</sup> for the cryptographic functions, including the encryption of the audit records and the generation and verification of the integrity proofs. We further used common desktop hardware or low-end, but modern, servers such as the Intel Core i7 at 2.6 GHz with 32 GB RAM. As such, we are only interested in overall behaviour and orders of magnitude. All source code is open and available on request. Specifically,

- for the CP approach, we used Apache Tomcat application servers, which use Java SE 8.
- for the blockchain approach, we combined an Apache Tomcat application server for the workflow nodes and then implemented an interface called an Ethereum smart contract written in Solidity. We run our approach in the local test environment of Ethereum.

To evaluate the performance of each implementation, we record the response time for each transaction, which includes the message propagation time, the generation and reporting time of the audit record, and the time taken by the verification mechanisms, often using cryptography.

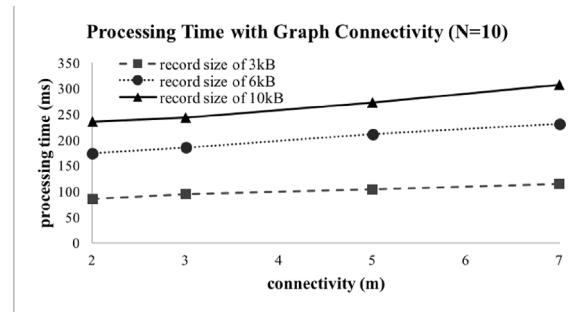
### 5.1. Using a CP

We evaluate our CP implementation with different sizes for the individual record. For a fair and consistent evaluation, we separated the results according to the number of participants  $N$ ; this gives comparable computational resource allocation for each iteration of the experiment after dedicating the resources required to bootstrap participants and the audit server.

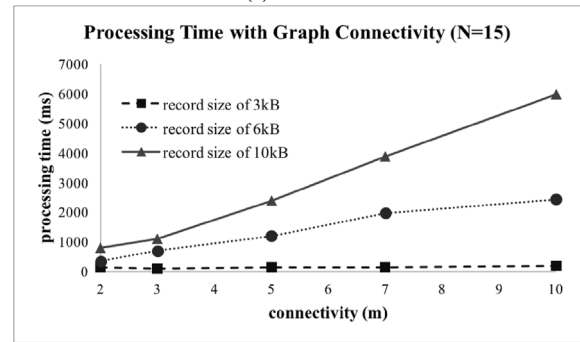
Fig. 8 shows the average response time. The key parameter we evaluate is the response time (vertical axis). We used three sizes for the individual audit record each node generates: 3 KB, 6 KB, and 10 KB.

For small topologies up to  $N=20$ , we obtained an approximate linear relation. This is likely to be shaped like a logarithmic curve if we allowed the size of the topology to scale to different orders of magnitude.

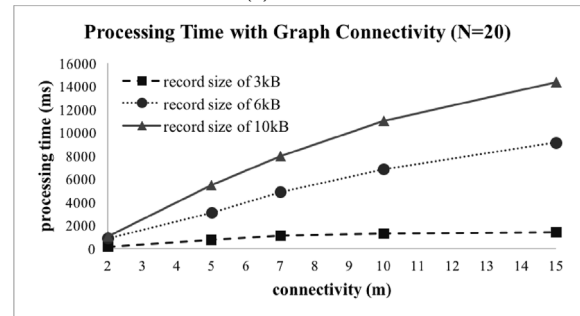
Fig. 9 shows the size of the final audit trail. We evaluated the size of the audit trail (vertical axis) against three sizes of the individual audit record (fixed for the set of runs). It follows the generic pattern and the size of the total audit trail increases sub-linearly with the size of the individual record at  $N=20$ . This was expected since the emulated



(a)  $N = 10$



(b)  $N = 15$



(c)  $N = 20$

Fig. 8. Evaluation of response time using a CP.

workflows run increasingly in parallel with increasing  $m$ , which reduces the size of pair-wise messages. Fig. 10 shows the impact of the connectivity degree  $m$ . We evaluated the size of individual records (horizontal axis) and the size of the final audit trail (vertical axis) when we allowed the increase in the connectivity degree  $m$  of each node connected: 2, 3, 5, 7. If the topology is larger, the size of the final audit trail also increases, as expected, but once again, we see a sub-linear relation.

In absolute terms, we note that the delay of our approach is on the order of seconds. If one considers that (1) each node will have to perform some kind of task that is likely to last more than a few seconds, and (2) our code is likely to not be optimised, these results suggest that our scheme can support fully automated workflows in real-time.

### 5.2. Using a public blockchain

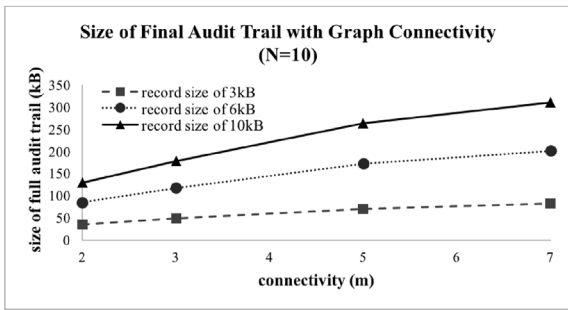
To reduce the time it takes to mine a block, we set up the variable of difficulty to a low value. We evaluated and discussed the processing time and gas cost, and compared the performance of implementations by CP and blockchain.

- 1) Processing Time: Fig. 11 shows the average response time. The key parameter involved is the number of records (vertical axis) that are generated and shared in each iteration. We chose that each node generates 10 KB for the individual audit record. With topology  $N=20$ ,

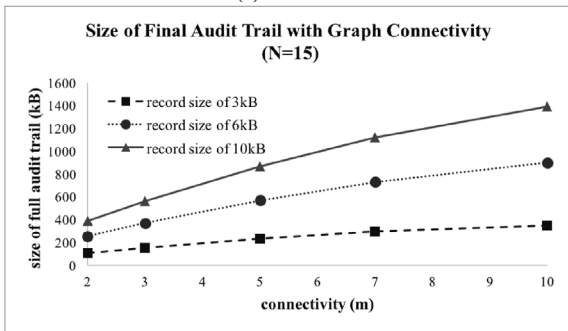
<sup>4</sup> Shamir Secret Sharing Scheme: <https://github.com/iancoleman/shamir/blob/master/src/js/secrets.js>.

<sup>5</sup> Nimbus JOSE: <https://connect2id.com/products/nimbus-jose-jwt>.

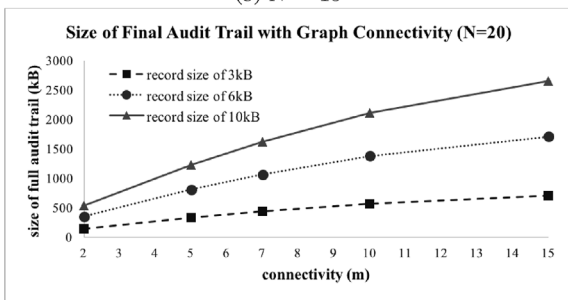




(a)  $N = 10$

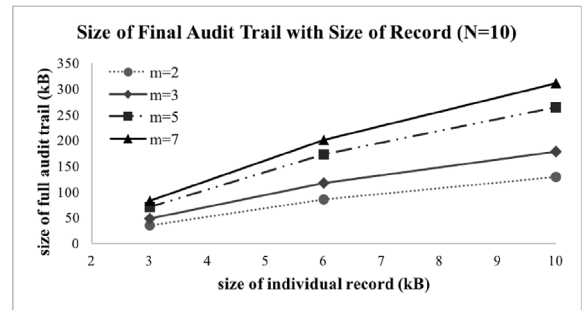


(b)  $N = 15$

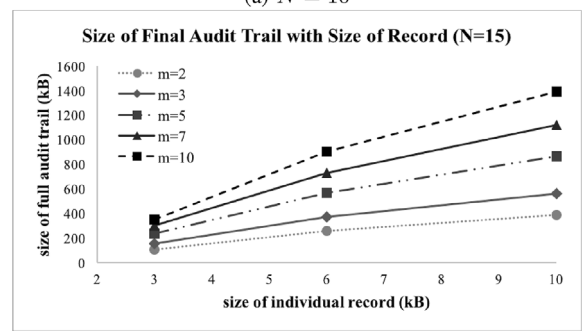


(c)  $N = 20$

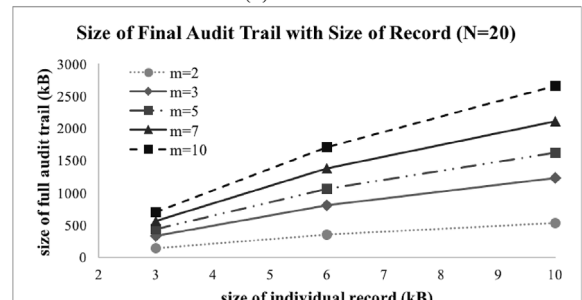
Fig. 9. Evaluation of the size of the audit trail using a CP (coordination point).



(a)  $N = 10$



(b)  $N = 15$



(c)  $N = 20$

Fig. 10. Evaluation of the size of the audit trail and individual record using a coordination point.

we found a stable relationship between the average response time and the number of records at the different graph connectivities. The time to process a set of records is essentially dominated by the mining delay.

2) Cost: Fig. 12 shows the gas cost for different sizes of records. We took the gas spent for each transaction and record size and calculated averages for topologies of  $N=20$ . As expected, we see a linear dependency with the record size. This is because we only store the minimum information, which is digests of the records in the blockchain. Since each node will generate 1 record, the gas used will be  $G \propto N \cdot \text{hash}(M) + g_{alg} + g_{setup}$ , where  $g_{alg}$  is the gas spent to execute the storage procedure (fixed and independent of the topology), and  $g_{setup}$  is any initial, one-time, setup of the smart contract. As such, using the  $O$ -notation, the costs will be  $C=O(\text{hash}(M) \cdot N \cdot C_{gas} \cdot C_{gas})$  is the actual monetary cost per unit of gas. This is dependent on the actual platform (such as Ethereum) and on market conditions.

A remark about Ethereum. We selected this platform purely on the basis of a proof-of-concept. Even though it is the most popular currently, we do not set any expectations or make any recommendations as to which platform to use. In other words, any blockchain that is able to store a value (the hash of the records) is fit for use. We assume that the cost structure will be similar in any smart contract platform, even if current alternatives (such as Algorand or EOS operate differently).

Furthermore, we note that the gas as measured in our implementation

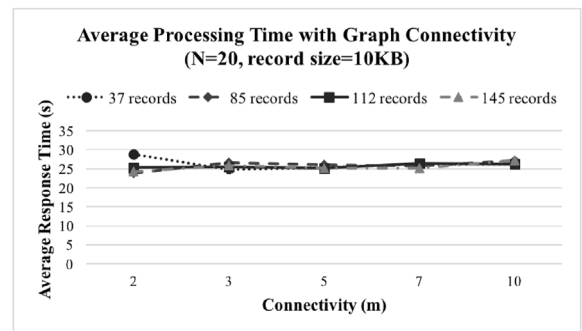


Fig. 11. Evaluation using a public blockchain.  $N=20$  nodes, the size of the audit record is 10 KB.

may not be accurate as we used the source-code of Ethereum as of early 2021. Ethereum changed how gas is calculated and spent recently (in the London upgrade). The structure of costs, however, did not change and still is  $C=O(\text{hash}(M) \cdot N \cdot C_{gas})$ . A similar structure of costs will exist when Ethereum, and perhaps most other platforms, change to Proof-of-Stake instead of Proof-of-Work.

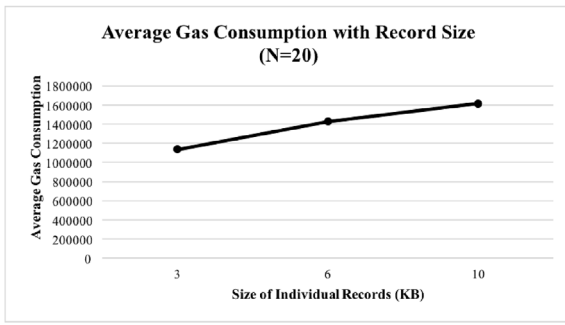
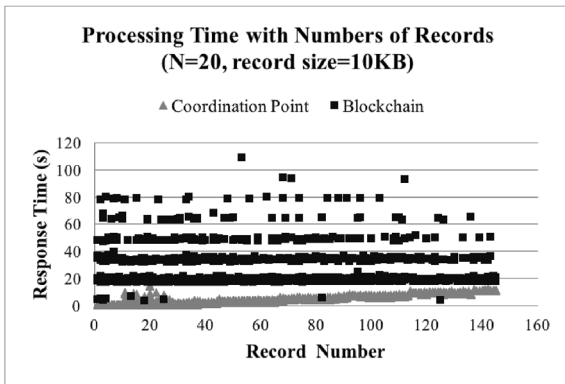
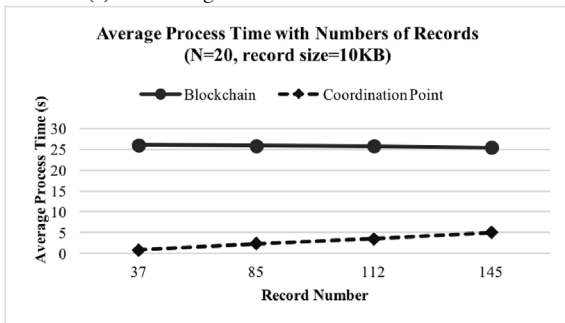


Fig. 12. Average processing gas cost of each iteration with the size of the record.  $N=20$  nodes.



(a) Processing time with numbers of records



(b) Average processing time of each iteration with numbers of records

Fig. 13. Comparison evaluation using CP (coordination point) and blockchain in response time with numbers of records.  $N=20$  nodes, record size=10 KB.

3) Performance Comparison: In Fig. 13 we compare the performance, in time to store the complete audit trail, of our two implementations. We used  $N=20$  and a record size of 10 KB. In Fig. 13a, the relationship between response time and the number of records in the implementation of the CP and blockchain behaves significantly differently. The comparison of the relationship between the average response time and specified numbers of records in Fig. 13b shows a similar result as Fig. 13a. When using a CP, the response time is essentially linear as audit records are generated. For the blockchain implementation, we see jumps of about 15 s, which corresponds to the mining period.

## 6. Conclusion and future work

We presented an architecture that tackles the problem of auditing workflows satisfying accountability, non-repudiation, confidentiality of records and graphs, availability, and collusion detection. We proposed and evaluated two implementations based on either a CP or a blockchain.

Whereas the CP approach is simpler and could meet the real-world requirements of most cases, we showed that using a central point targets the minimal trust requirements and using a public blockchain makes our approach more robust at the expense of processing delay. Although AudiWFlow is designed with confidentiality, integrity, and availability requirements in mind, its tight security measures can be relaxed for applications that do not require, for example, strict confidentiality requirements between participants. Different implementations of our approach can be adopted in different cases; offering both implementations enables the adoption of the approach for organisations that have legal, political, or corporate culture restrictions on the technology that can be used. While the blockchain-based implementation of our approach offers a stable processing time and permanently stored integrity proofs of the audit records on the public blockchain, the implementation with the central CP requires less processing time (limited nodes involved) subject to the availability of a central server. Our discussion and evaluation of each implementation enable adopters to make an informed decision on which implementation is a better fit for their business context.

There remain a number of questions for future work. Two of them are the following. First, we will strengthen our approach to satisfy identity confidentiality: at present, our scheme relaxes this requirement, and the identity of nodes needs to be known beforehand—for example, to distribute cryptographic material such as public keys. Second, our approach requires a starting point that needs to be trusted for some tasks, such as generating the threshold keys. Solving this particular problem will necessarily raise the complexity by, for example, making use of multiparty secure computation techniques. Finally, the overall architecture of an approach using blockchain needs elaboration.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

- [1] F. Tian, A supply chain traceability system for food safety based on haccp, blockchain amp; internet of things, 2017 International Conference on Service Systems and Service Management; 16–18 Jun 2017; Dalian, China, IEEE, Piscataway, NJ, USA (2017) 1–6.
- [2] I. Weber, X. Xu, R. Riveret, et al., Untrusted business process monitoring and execution using blockchain, in: M. La Rosa, P. Loos, O. Pastor (Eds.), Business Process Management, Springer, Cham, Switzerland, 2016, pp. 329–347.
- [3] K. Hartmann, C. Steup, On the security of international data exchange services for e-governance systems, Datenschutz und Datensicherheit-DuD 39 (7) (2015) 472–476.
- [4] I. Pappel, I. Pappel, J. Tepandi, et al., Systematic digital signing in Estonian e-government processes, in: A. Hameurlain, J. Küng, R. Wagner (Eds.), Transactions on Large Scale Data-and Knowledge-Centered Systems XXXVI, Springer, Heidelberg, Berlin, Germany, 2017, pp. 31–51.
- [5] Z. Guan, H. Lyu, H. Zheng, et al., Distributed audit system of SDN controller based on blockchain, in: M. Qiu (Ed.), Smart Blockchain, Springer, Cham, Switzerland, 2019, pp. 21–31.
- [6] S. Zawoad, A.K. Dutta, R. Hasan, Towards building forensics enabled cloud through secure logging-as-a-service, IEEE Trans. Dependable Secure Comput. 13 (2) (2016) 148–162.
- [7] I. Ray, K. Belyaev, M. Strizhov, et al., Secure logging as a service—delegating log management to the cloud, IEEE Syst. J. 7 (2) (2013) 323–334.
- [8] J.R. Rajalakshmi, M. Rathinraj, M. Braveen, Anonymizing log management process for secure logging in the cloud, 2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]; 20–21 Mar 2014; Nagercoil, India, IEEE, Piscataway, NJ, USA, (2014) 1559–1564.
- [9] D. Ma, G. Tsudik, A new approach to secure logging, in: V. Atluri (Ed.), Data and Applications Security XXII, Springer, Heidelberg, Berlin, Germany, 2008, pp. 48–63.
- [10] M.A.M. Ahsan, A.W.A. Wahab, M.Y.I. Idris, S. Khan, E. Bachura, K.R. Choo, Class: cloud log assuring soundness and secrecy scheme for cloud forensics, IEEE Trans. Sustain. Comput. 6 (2021) 184–196, <https://doi.org/10.1109/TSUSC.2018.2833502>.
- [11] D.A. Flores, An authentication and auditing architecture for enhancing security on egovernment services, 2014 First International Conference on eDemocracy eGovernment (ICEDEG); 24–25 Apr 2014; Quito, Ecuador IEEE, Piscataway, NJ, USA, (2014) 73–76.
- [12] C. Esposito, A. De Santis, G. Tortora, et al., Blockchain: a panacea for healthcare cloud-based data security and privacy? IEEE Cloud Comput. 5 (1) (2018) 31–37.

- [13] X. Zhou, V. Jesus, Y. Wang, M. Josephs, User-controlled, auditable, cross-jurisdiction sharing of healthcare data mediated by a public blockchain, 2020 IEEE 19th International Conference On Trust, Security And Privacy In Computing And Communications (TrustCom); 29 Dec 2020–1 Jan 2021; Guangzhou, China, IEEE, Piscataway, NJ, USA, (2020) 978–982.
- [14] J. Cucurull, J. Puiggalí, Distributed immutabilization of secure logs, in: G. Barthe, E. Markatos, P. Samarati (Eds.), International Workshop On Security And Trust Management, Springer, Cham, Switzerland, 2016, pp. 122–137.
- [15] B. Putz, F. Menges, G. Pernul, A secure and auditable logging infrastructure based on a permissioned blockchain, *Comput. Secur.* 87 (2019), 101602.
- [16] Q. Lu, X. Xu, Adaptable blockchain-based systems: a case study for product traceability, *IEEE Softw.* 34 (6) (November 2017) 21–27.
- [17] A. Ahmad, M. Saad, M. Bassiouni, et al., Towards blockchain-driven, secure and transparent audit logs, *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*; 5–7 Nov 2018; New York, NY, USA, ACM, New York, NY, USA (2018) 443–448.
- [18] N. Tapas, G. Merlino, F. Longo, et al., Blockchain-based publicly verifiable cloud storage, in: 2019 IEEE International Conference on Smart Computing (SMARTCOMP); 12–15 Jun 2019; Washington, D.C, USA, IEEE, Piscataway, NJ, USA, June 2019, pp. 381–386.
- [19] W. Pourmajidi, A. Miransky, Logchain: blockchain-assisted log storage, 2018 IEEE 11th International Conference on Cloud Computing (CLOUD); 2–7 Jul 2018; San Francisco, CA, USA, IEEE, Piscataway, NJ, USA, (2018) 978–982.
- [20] A. Nehme, V. Jesus, K. Mahbub, et al., Decentralised and collaborative auditing of workflows, in: S. Gritzalis, E. Weippl, S. Katsikas (Eds.), *Trust, Privacy and Security in Digital Business*, Springer, Cham, Switzerland, 2019, pp. 129–144.
- [21] E.F. Brickell, Some ideal secret sharing schemes, in: J.-J. Quisquater, J. Vandewalle (Eds.), *Advances in Cryptology—EUROCRYPT '89*, Springer, Heidelberg, Berlin, Germany, 1990, pp. 468–475.
- [22] A. Shamir, How to share a secret, *Commun. ACM* 22 (11) (1979) 612–613.
- [23] M. Stadler, Publicly verifiable secret sharing, in: U. Maurer (Ed.), *Advances In Cryptology—EUROCRYPT '96*, Springer, Heidelberg, Berlin, Germany, 1996, pp. 190–199.
- [24] B. Schoenmakers, A simple publicly verifiable secret sharing scheme and its application to electronic voting, in: M. Wiener (Ed.), *Advances in Cryptology—CRYPTO '99*, Springer, Heidelberg, Berlin, Germany, 1999, pp. 148–164, plus 0.5em minus 0.4emBerlin.
- [25] R. D'Souza, D. Jao, I. Mironov, et al., Publicly verifiable secret sharing for cloud-based key management, in: D.J. Bernstein, S. Chatterjee (Eds.), *Progress In Cryptology – INDOCRYPT 2011*, Springer, Heidelberg, Berlin, Germany, 2011, pp. 290–309.