Decentralised and Collaborative Auditing of Workflows

Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah

Birmingham City University, School of Computing and Digital Technology, Birmingham, UK
{antonio.nehme, vitor.jesus, khaled.mahbub, ali.abdallah}@bcu.ac.uk

Abstract. Workflows involve actions and decision making at the level of each participant. Trusted generation, collection and storage of evidence is fundamental for these systems to assert accountability in case of disputes. Ensuring the security of audit systems requires reliable protection of evidence in order to cope with its confidentiality, its integrity at generation and storage phases, as well as its availability. Collusion with an audit authority is a threat that can affect all these security aspects, and there is room for improvement in existent approaches that target this problem.
This work presents an approach for workflow auditing which targets security challenges of collusion-related threats, covers different trust and confidentiality requirements, and offers flexible levels of scrutiny for reported events. It relies on participants verifying each other's reported audit data, and introduces a secure mechanism to share encrypted audit trails with participants while protecting their confidentiality. We discuss the adequacy of our audit approach to produce reliable evidence despite possible collusion to destroy, tamper with, or hide evidence.

Keywords: Audit Trails · Confidentiality · Accountability · Collusion.

1 Introduction

A virtual organisation, defined as a collaboration of independent organisations to fulfill a business requirement [13], requires tasks and decision making to be spread among the different administrative and security domains of participants. These collaborations necessitate a way to keep track of transactions between the involved parties, and evidence of actions needs to be available to assign accountability in case of a dispute. Evidence should be reliable, and its processing should not disclose confidential information to any party.

A number of studies shed light on the importance of audit trails in systems involving multiple participants. Examples of these systems include banking scenarios, logistics, supply chain, and e-government [12,24,26,27]. Werner et al. [25] highlight the importance of reliable audit trails in the financial accounting sector. Kuntze et al. [11] stress on the importance of data authenticity, integrity and privacy for evidence stored in forensic databases; they state that maintaining the confidentiality of parties involved in a chain of evidence is challenging.

To illustrate the challenge, we present a hypothetical scenario: car insurance companies rely on data about the driver's behaviour and habits to calculate their annual fees. When an individual applies for renewal, the insurance company sends a request to assess the applicant to the Ministry of Transport. This ministry requests the car mileage from garages that perform vehicles annual checkups and any arrest warrant for the applicant from the police department, and checks the databases of traffic cameras for the neighbourhood in which the car is most frequently used. Insurance companies do not get all the details about the driver due to the confidentiality of this data; they receive a report with an assessment from the Ministry of Transport and use it to determine the insurance fees. Audit records for every request should be kept: each organisation maintains a log, and an audit service is used to build audit trails. A privileged insider at the Ministry of Transport, colluding with an insurance company, modifies the mileage received from the garages to increase insurance fees. This insider also modifies the local logs, and colludes with the audit service to modify the audit records. As seen in this scenario, a collusion with an entity managing a central audit service renders its data unreliable. Moreover, audit records in workflows include sensitive information for all participants. In this scenario, audit records stored at the Ministry of Transport or with a central audit system expose personal and confidential data.

For a workflow audit architecture to be trusted, one participant should not have the option of colluding with the authority that manages the audit system to breach the confidentiality of, tamper with, or destroy evidence [29]. A central approach in which a single system is trusted to collect, verify, and store logs is a single point of failure in this case. In practice, workflow engines, commonly used to coordinate interactions between workflow participants and to provide auditing services, are a representation of a central approach for audit [12, 17]. Some approaches, [26] for example, propose holding each participating administration accountable to store its own audit logs. Although this practice protects the confidentiality of participants, it does not provide protection against tampering with and destroying evidence.

We, therefore, follow a trustless approach to produce reliable evidence. We define trustless audit by not giving an organisation the ability to generate evidence without a verification of its authenticity. Our architecture retrieves and verifies audit records in a distributed way immediately after generation by participants, while protecting the confidentiality, integrity, and availability of this data from malicious entities working individually or colluding with each other. Following our approach, malicious behaviour of tampering with, deleting, or false reporting of audit records is detected by honest participants. Our contribution is an architecture that offers mitigation from collusion-related threats to tamper with or destroy multi-party workflow audit data. This depends on the collaboration of participants to verify and obtain a copy of encrypted audit records reported to an audit server, as well as to ensure the authenticity of this server. We use Shamir secret sharing mechanism [18] to minimise the risk of confidential data exposure.

In a multi-party workflow, our architecture:

- Offers a means to verify reported audit records.
- Supports distributed storage of audit data at any degree of details.
- Introduces a data structure for audit trails covering arbitrary topology.
- Offers audit capability to any K out of N participants.

This rest of this paper is organised as follows. Section 2 presents related work. Section 3 presents the problem statement and the threat model. A detailed explanation of our approach is presented in Section 4, and an analysis of its adequacy to fulfil our goals in section 5. Implementation and evaluation are covered in sections 6, and we conclude the paper in section 7.

2 Related Work

Many audit frameworks proposed in the literature log events as evidence to verify abiding to security policies and regulatory requirements. Rudolph et al. [17] designed an audit trail with a summary of participation exchanged between participants to show fulfilment of tasks and enforce behavioural policies during workflow execution; they extended their work in [22] to cover the anonymity of participating entities, but did not discuss the security of audit trails at storage and their protection from destruction. Hale et al. [10] present a design and verification framework for services interactions across different clouds to verify abiding to information sharing policies. Other solutions [5–8, 12, 19, 27] cover a variety of audit data including signatures, users' consents for access control, data provenance, service level agreements related logs, and records of database access. However, a trusted central system is used to process logs in [19] and to store audit records in the other frameworks. This raises trust and confidentiality problems since collusion with the central point in any of these cases makes tampering with or destroying evidence possible, as well as breaching the secrecy of audit data.

To protect the confidentiality of audit data, especially when relying on a third party service as proposed in [15] and [16], a common practice is to encrypt audit data prior to its submission to the outsourced storage. [1, 2, 23] are proposals that require a trusted party to protect encryption keys for audits, and Wouters et al. [26] rely on the user to protect the key in their approach. Entities with encryption keys can breach the confidentiality and in some cases the integrity of audit data; also, refusing to share these keys when an audit is required leads to withholding evidence. The framework proposed by Ray et al. [16] uses a secret sharing scheme to split encryption keys to multiple logging hosts; however, their approach covers logging for single applications, and does not consider multi-party interactions.

For forensic investigations, a high degree of assurance for the integrity of digital evidence is required at the collection, generation, and storage phases [29]. A variety of hash based approaches, relying on hash chains and hash trees, are proposed in [4, 21, 28, 29] to assure the integrity of audit records collected by a cloud service provider. Zawoard et al. [28] mitigate possible tampering

with evidence that can result from collusion between users, investigators, and cloud service providers. They encrypt the log files with a public key belonging to a law enforcement agency to protect users' privacy, and adopt a hash-chain scheme referred to as 'Proof of Past Log (PPL)', covering the entire log history for every user on a daily basis. However, these approaches trust the logging entity, being the cloud provider, to report correct audit data and to assure its availability. A proposal by Ahsan et al. [3] is an extension that covers some of the limitations of [28]. In their work, log records generated by cloud service providers are not trusted, and users are required, within a specific timing, to verify their logged activity with their cloud provider and to file a complaint in case of any activity that needs to be denied. Logs attributed to a user are then stored on the cloud service provider servers and encrypted with this user's private key. To mitigate withholding evidence by users, a secret sharing mechanism is proposed to distribute shares of every user's keys to different cloud providers that are assumed not to collude. This approach covers auditing users' activities in a single cloud only, and does not consider multi-party workflows; it also assumes continuous human cooperation for its verification phase, and does not consider destruction of evidence by a cloud provider.

Blockchain is another approach that uses hash chains and the consensus of a large network of nodes to protect data at storage [24]. Some approaches, [20, 24] for example, use blockchains to produce audit trails for processes involving multiple parties; however, this technology does not scale to store large transactions, and therefore only hashes of these transactions are stored on the blockchain [20]. While similar approaches verify the integrity of audit records, they do not protect against destruction of evidence.

To the best of our knowledge, our approach improves the availability, confidentiality, and integrity of audit records in distributed systems. We combine practices in the literature for collective building of audit trails with forensic approaches to protect digital evidence, and introduce verification mechanisms in our architecture to assert accountability of participants in multi-party workflows.

3 Problem Statement and Threat Model

This work argues the need for improvement in audit approaches used for workflows covering multiple administrative domains. Every participant should be accountable for a contribution reflecting an action or a decision in a workflow. Existent audit approaches, discussed in the previous section, depend on trusted parties to record or store audit records; this renders the integrity and availability of these records questionable. Also, trusting a single entity with encryption keys leaves room for withholding evidence. When following approaches discussed in the literature, the problems below can be faced:

- Destroying or withholding evidence
- Tampering with audit data at storage
- Reporting incorrect audit data

3.1 Threat Model

This model excludes compromised elements being under the control of an attacker. Actions are considered to reflect the intentions of the administration managing an element, rather than a fault in the system. Certificate authorities and key issuers are honest; they distribute correct cryptographic keys and do not expose participants' secrets.

The adversary vector includes participants that aim to rig an audit reporting process to avoid non-repudiation. Adversary's goal can be achieved by destroying or tampering with audit records, or by using incorrect cryptographic keys for encryption or signatures to jeopardise the records' usability. The entity managing the audit server can also be malicious, and may attempt to tamper with, destroy, or hide audit records from some participants. A number of malicious entities (including workflow participants and the audit server) can collude to falsify the audit process; this includes skipping the verification of reported audits, hiding audit data from honest participants, cooperating to distribute different versions of records, and delaying the workflow with false alerts. Breaking the confidentiality of workflow transactions using audit records is another malicious goal for an adversary.

Under this model, we aim to verify the authenticity of reported audit records as they are generated as well as their integrity during storage or distribution, and to protect their confidentiality at every stage of the audit process. We also achieve the availability of evidence when a definable number of participants are honest. We consider audit data reported by colluding entities to be unreliable, but prove any malicious behaviour resulting from collusion by using reliable audit records reported by honest participants.

4 Proposed Approach

We propose a confidentiality friendly and decentralised approach for workflow audit. Encrypted audit trails, covering arbitrary workflows, are checked for authenticity as they are built, and are distributed to participants to protect from tampering with or destroying evidence. To safeguard the confidentiality of this data, a threshold of key shares is required to reconstruct the decryption key. This also minimises the chance of witholding evidence by an entity that is holding the key. Audit trails can be constructed with any degree of details ranging from a summary of interactions to full transactions.

4.1 Notation

We use a graph to represent workflows, in which nodes(vertices) represent an organisation in the workflow, and edges with weights represent the order of their interactions. In this regard, edge (A, B) in Figure 1 is the first interaction in the workflow manifested by a message sent from organisation A to B. Workflows covered in this work are pre-established, meaning that the participating entities

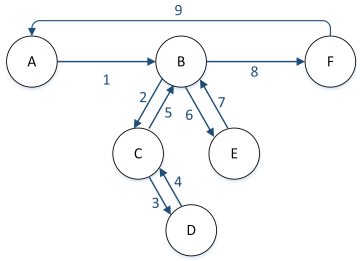

Figure 1. Our representation of workflows.

and their order of interaction are known by participants at the beginning of the workflow execution and do not change until it finishes. For ease of representation, we introduce the notation below used to describe cryptography operations:

- An encrypted message sent from A to B, containing $Payload_{A\to B}$ and the signature of the sender over the payload, is represented by

  $$Payload_{A\to B}(Sign_A, Enc_B) = Enc_B[(Payload_{A\to B}) + Sign_A(Payload_{A\to B})].$$

- An additional signature over the encrypted message, $Sign'_{sender}$, can be sent alongside the cipher text. The message in this case is represented by:

  $$Payload_{A\to B}(Sign_A, Enc_B, Sign'_A) =$$

  $$Payload_{A\to B}(Sign_A, Enc_B) + Sign_A[Payload_{A\to B}(Sign_A, Enc_B)]$$

- $Hash$ is a one-way hash function with a strong collision resistance.
- Shares of a workflow private key wfl_Pri, split with Shamir secret sharing to N shares, are referred to as $K_1 ... K_N$. The equivalent public key is wfl_Pub.
- String X concatenation with Y is represented by: $X||Y$.
  When presenting this architecture, the terms *node* and *participant* are used interchangeably to indicate a participants in a workflow.

4.2 System Overview

Our architecture uses a special node, referred to as the 'Audit Server'. This audit server can be hosted by any node or managed by a separate entity as a central or distributed system, but is not trusted by participants. It is only used to display audit records to participants during the workflow, and not to permanently store any data. For every action, an audit record is published on the audit server by a participant, and verified by another. Every participant in the workflow is given credentials to publish encrypted data on this server. Participants check the authenticity of the information on the server, and update their local storage of audit data continuously. Every participant has a certificate, and every workflow has a key pair which public part is known by all participants, but the private part is shared between them following Shamir secret sharing mechanism [18].

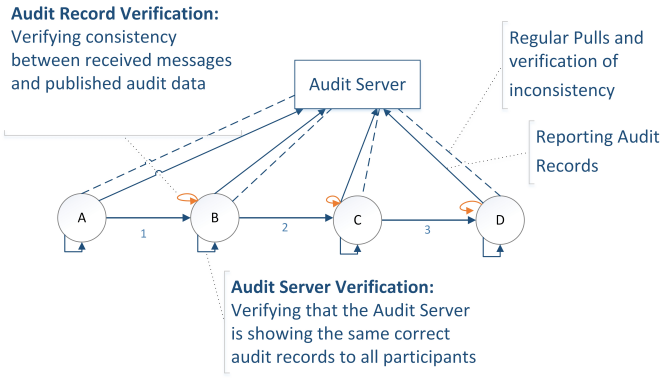

Figure 2. Overview of the proposed architecture.

As shown in Figure 2, the proposed approach includes two types of security verifications.

- The *Audit Record Verification* requires every node that receives a message to check it against its equivalent audit record published on the audit server.
- The *Audit Server Verification* requires every node to check for any inconsistency suggesting malicious behaviour from the audit server.

Before going through the details of these verification mechanisms, we present our key management scheme, and the data structure we use for our audit trails.

4.3 Key Management

We assume that certificates for every participant are managed with public key infrastructure (PKI). While certificates can be used in multiple workflows, workflow keys are only used in a single workflow topology.

Workflow key management is handled by an entity that generates a key pair (wfl_Pub, wfl_Pri), splits (wfl_Pri) following Shamir secret sharing mechanism [18], and securely distributes (wfl_Pub, K_i) to participants; a threshold K of shares (K_i) can reconstruct wfl_Pri. 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 the audit server.

This critical role should be given to the participant that has the least incentive to be dishonest with the key distribution and to expose wfl_Pri. This is generally the first or last participant depending on the workflow: a first participant in one workflow can be a travel agency required to keep track of bookings for its customers, and the last participant in another workflow can be a car manufacturing industry that needs to keep track of where parts of customised vehicles, ordered by customers, are from. Alternatively, this role can be assigned to a dedicated key manager.

4.4 Audit Data Structure

This work covers logging for every interaction between participants in a workflow. A message, sent from a participant to another, contains a payload which includes the data for the intended recipient. Cryptographic operations are performed to assure the authenticity and confidentiality of the transaction. For simplicity, we assume the linear topology shown in Figure 2. A message sent from C to D in the workflow would be of the form

$$Msg_{C\to D} = Payload_{C\to D}(Sign_C, Enc_D)$$

In turn, audit data published by C on the audit server should have the exact same payload, but is encrypted with the workflow public key, and signed after encryption:

$$Audit_{C\to D} = Payload_{C\to D}(Sign_C, Enc_{wfl-Pub}, Sign'_C).$$

The second signature verifies that the audit server has not tampered with the audit record. Each payload includes data to the receiver and audit records of the previous transactions of the workflow when applicable. In this case

$$Payload_{C\to D} = Data_{C\to D} + Audit_{B\to C}.$$

Recursively, a decryption of an audit record with wfl_Pri, recovered from a threshold K out of N parts (K_i) of wfl_Pri, leads to another encrypted one which in its turn gets decrypted with the same key.

4.5 Audit Record Verification

Throughout the workflow, participants are required to verify the correctness of the audit records equivalent to the messages that they receive.

As shown in Figure 3, an audit record is published by every participant after sending a message as well as the last participant after receiving the final message. We only show the content of the *Payload* in the exchanged messages for the clarity of the figure.

A recipient decrypts the Msg with its private key, and verifies the sender's signature. It then performs *Audit Record Verification*: the recipient updates its storage of audit records from the audit server, then checks if the audit records of previous transactions in the message are identical to what is shown on the audit server. It also verifies that the sender of the message did not omit any audit records that should be included, and that these audit records are for the right previous transactions; this is feasible since every participant knows the topology of the pre-established workflow, and can therefore verify the signatures on the audit records. The recipient then re-encrypts the payload and sender's signature with wfl_Pub, and checks if the cipher version of it is on the audit server.

Back to Figure 3, after a successful *Audit Record Verification* the recipient of a message keeps the payload with the sender's signature as a receipt. The

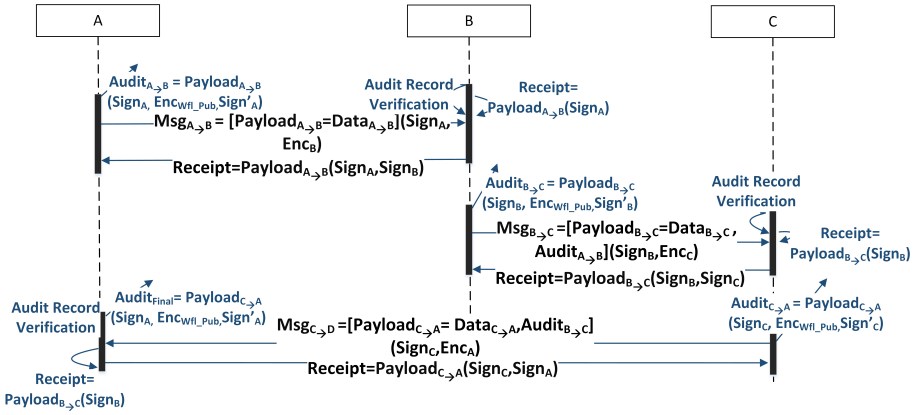

Figure 3. Sequence diagram for our approach on a workflow that starts and ends with participant A. Upwards arrows represent reporting to the Audit Server.

recipient then sends this receipt back after adding its signature to the sender; this is a proof of delivery. In case any of these steps goes wrong, protocol is to raise concern for malicious behaviour.

Another test to verify the authenticity of the audit server is *Audit Server Verification*. Details about this mechanism is covered in the next section.

4.6 Audit Server Verification

Audit Server Verification algorithm is executed by every participant right after sending or receiving a message, and at a recurrent basis during the workflow. The frequency of execution is configured depending on the average execution time of a workflow. The aim is to ensure that the audit server is displaying the same authentic workflow audit records to every participant, and to distribute these records to participants. Tailoring responses to participants is possible if the Audit Server is malicious. We present the following mechanisms to target this challenge:

Digest on the fly: when publishing audit records, each participant is required to add a digest of its local audit records storage, including the one that is being published, signed with its private key. Back to Figure 2, $Audit_{B\rightarrow C}$ in this case would be published alongside

$$Hash(Audit_{A\rightarrow B}||Audit_{B\rightarrow C}) + Sign_C[Hash(Audit_{A\rightarrow B}||Audit_{B\rightarrow C})].$$

Referring to Algorithm 1, participants verify that the audit records that they published on and pulled from the audit server are still displayed; they then update their storage of audit records. After that, they compare the hash value of their stored audit records with the signed digest reported by the last publisher. Different digest values suggest that the audit server is showing different records

to participants. Detecting a malicious activity from the audit server requires an honest participant to publish an audit record after this activity was committed.

Algorithm 1 Audit Server Verification Algorithm.

Requirement: Verify the authenticity of the Audit Server
Update local storage of audit records for participants
Input: *Reported_Recs*[] ▷ Audit records reported by a participant
Stored_Recs[] ▷ Audit records stored Locally with a participant
Output: Boolean indicating if Audit Server is honest

```
1: Server_Recs[] ← Pull_Recs()                    ▷ Pull audit records from the server
2: if Reported_Recs !=Ø then
3:     for each R∈ Reported_Recs[] do
4:         if R does not exist in Server_Recs[] then
5:             return False
6: if Stored_Recs !=Ø then
7:     for each R∈ Reported_Recs[] do
8:         if R does not exist in Server_Recs[] then
9:             Return False
10: Local_Recs[] ←Server_Recs[]                   ▷ Updating Stored Records
11: H← Hash(Local_Recs[])                         ▷ Comparing Digests
12: if H==Last Published Digest then
13:     Return True
14: else
15:     Return False
```

Verify at the End: alternatively, the same steps of the Algorithm are followed, except that the hash comparison is only done at the end of the workflow. The final node publishes a signed digest of the records it has, and other participants follow and do the same. Not having identical digest values suggests a malicious behaviour from the audit server.

Combo: This combines both of the previous two approaches.

4.7 Protocol

This section lists the actions required to reach our security goals:

- Participants keep their receipts, and are required to alert others if a signed receipt of delivery is not received after sending a message.
- *Audit Record Verification* is always performed by recipients on message delivery. Failure of verification alerts all participants.
- *Audit Server Verification* is performed by a sender after publishing a record, a receiver when getting a message, and by every participant on a recurrent basis during the workflow. Failure of verification alerts all participants.

Following this protocol is essential to reveal any malicious activity by entities working on their own or colluding. We discuss potential attack scenarios in the next section.

5 Analysis

In this section, we discuss attack scenarios caused by a malicious node or a malicious audit server working individually, collusion among participants, and collusion between the malicious participants and the audit server. We analyse how each scenario is handled following our approach.

5.1 Malicious Participant

Working individually, a malicious participant can attempt to **truncate audit trails**: such participant attempts to publish incorrect audit data for the message it sent, or to send incorrect audit records of previous transactions in a message. This is caught by the *Audit Record Verification* performed by the honest recipient of the message.

5.2 Malicious Audit Server

A malicious audit server may attempt to hide audit records from all or some participants, or to tamper with these records.
Hiding Audit Records: a malicious audit server can attempt to hide audit records from some participants to limit the distribution of audit data and facilitate destruction of evidence. This is detected by digest verification of any adopted mechanism.
Tampering with Audit Records: this is detected as soon as an honest node reports and audit record with **Digest on the Fly** or **Combo** mechanisms, or at the end of the workflow when **Digest at the End** is followed.

5.3 Collusion Between Nodes

What colluding participants can achieve and the impact of their collusion vary according to their positioning in a workflow. We discuss possible malicious behaviour of consecutive and non-consecutive colluders.
Non-Consecutive Colluders: one malicious node in Figure 4.1 can report an audit record for the other to use as part of their message. This is detected by the honest participant performing *Audit Record Verification*.

Figure 4. Collusion between participants with coloured background.

Consecutive Colluders: one node can cover for the other by skipping the *Audit Record Verification* phase. This leads to one of the cases below:

- Truncating the audit trail if C in Figure 4.2 publishes a corrupted audit record, or a record not including previous audit data
- Having audit trail with a false record, if C reports different data for audit than what it sent to D.

Audit records posted or verified by nodes not following protocol are unreliable. Audit data published by B is reliable, since B has a receipt of delivery signed by C to prove its honesty. Honest node E also has a receipt, and it makes sure D publishes the equivalent audit record of the message it sent on the audit server. If there are more than two consecutive colluding nodes, the nodes that have followed the protocol and that surround the colluding participants will have credible audit data.

5.4 Collusion Between Participants and the Audit Server

In this section, we illustrate cases of collusion between participants and the audit server and discuss the adequacy of each verification mechanism. In Figure 5.1, malicious node C can sign two different versions of an audit record; it relies on the audit server to display the correct record for only D and E, since they require it for the *Audit Record Verification*, and to display the faulty one to A and B. This is detected with any *Audit Server Verification* mechanism that is used:

1. If we publish signed digests with audit records (**Digest On the Fly**), this malicious activity is caught when node D, the first honest node following the malicious activity, publishes its audit data.

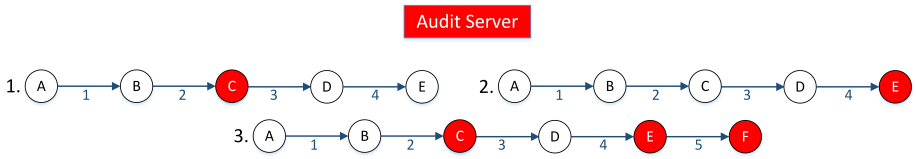

Figure 5. Malicious audit server colluding with participants.

2. With **Verify at the End**, the malicious behaviour gets detected when comparing hash values at the end. When comparing audit records, two versions of a record incriminate both the audit server and the publishing node.
3. The combination of methods (**Combo**) also detects this malicious behaviour.

In 5.2, **Digest on the Fly** does not detect the collusion with the Audit Server since there is not an honest node that follows the malicious one. However, it does not lead to any data loss since all transactions have been covered in the record before it.

The combination of methods (**Combo**) combines the advantages of the two verification mechanisms; Figure 5.3 is a scenario where **Combo** method is a good option:

- Attempts from C and the audit server to show A and B different versions of $Audit_{C \to D}$ than the one sent to D is detected on the fly.
- E and F cannot collude with the audit server to show honest participants different audit data.

6 Preliminary Evaluation

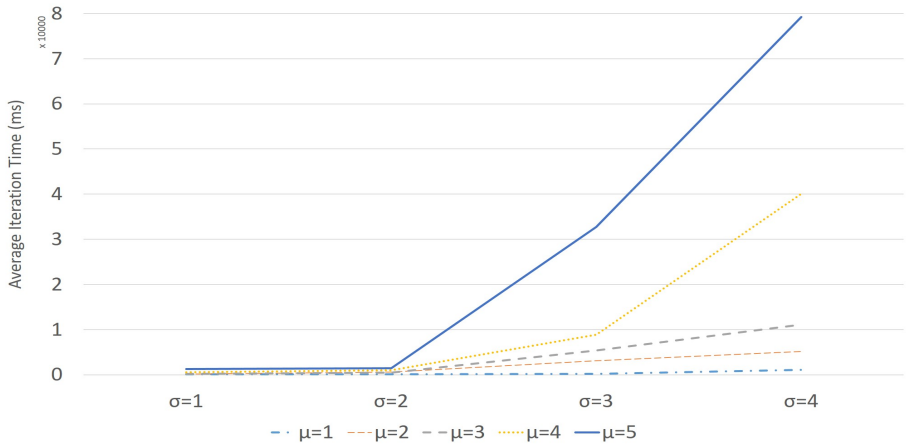

Figure 6. Average processing time for different log-normally distributed delays.

The implementation of the proposal is available on our Github repository[1]. In this section, we simulate a number of workflows relying on the same audit server. For this evaluation, we use a linear topology of nodes following our protocol, and consider the traffic of requests to the Audit Server to be log-normally distributed [9,14]. To simulate server load, we introduce a log-normally distributed delay of the form $e^{\mu+\sigma Z}$ to the Audit Server's response time. The figure below shows the average processing time for each case. While the audit server processing power is inversely proportional to μ, σ reflects the multitude of workflows running simultaneously.

Following this evaluation method, changing the number of nodes and the size of the exchanged *Payload* resulted in graphs with different scales for the Average Iteration Time, but with similar slopes for the lines. Comparing the slope of curves in Figure 6, servers with high computational power show stability and reliability at scale. Less powerful servers can still be used for systems that can afford latency at busy times.

[1] `https://github.com/antonionehme/audit-repository-simulations/tree/master/AuditProject/AuditProject`

7 Conclusion and Future Work

In this work, we presented an approach to enhance the availability, integrity and confidentiality of audit trails of workflows throughout the collection and storage of evidence. Collusion to hide or tamper with audit data is detectable, and the chance of withholding evidence is reduced due to the audit capability of K out of N participants.

This is part of a larger project to develop solutions for digital government workflows in which reliable audit is a key requirement. Decision making involving multiple government departments requires a collusion-proof audit system to assure the accountability while protecting the confidentiality of participants. This approach meets these requirements by providing a high level of assurance on the availability, integrity and confidentiality of digital evidence. We proposed this approach as part of a government digital transformation strategy, and we are working on integrating government-tailored scenarios and on evaluating our audit architecture for this purpose.

References

1. Accorsi, R.: Bbox: A distributed secure log architecture. In: European Public Key Infrastructure Workshop. pp. 109–124. Springer (2010)
2. Accorsi, R.: A secure log architecture to support remote auditing. Mathematical and Computer Modelling **57**(7), 1578 – 1591 (2013)
3. Ahsan, M.M., Wahab, A.W.A., Idris, M.Y.I., Khan, S., Bachura, E., Choo, K.K.R.: Class: Cloud log assuring soundness and secrecy scheme for cloud forensics. IEEE Transactions on Sustainable Computing (2018)
4. Alqahtani, S., Gamble, R.: Embedding a distributed auditing mechanism in the service cloud. In: 2014 IEEE World Congress on Services. pp. 69–76 (June 2014)
5. Aravind, A., Sandeep, A.: Workflow signature for business process domain: A new solution using ibmkd. In: Communication Technologies (GCCT), 2015 Global Conference on. pp. 619–622. IEEE (2015)
6. Bates, A., Hassan, W.U., Butler, K., Dobra, A., Reaves, B., Cable, P., Moyer, T., Schear, N.: Transparent web service auditing via network provenance functions. In: Proceedings of the 26th International Conference on World Wide Web. pp. 887–895. International World Wide Web Conferences Steering Committee (2017)
7. Flores, D.A.: An authentication and auditing architecture for enhancing security on egovernment services. In: 2014 First International Conference on eDemocracy eGovernment (ICEDEG). pp. 73–76 (April 2014)
8. Gajanayake, R., Iannella, R., Sahama, T.: Sharing with care: An information accountability perspective. IEEE Internet Computing **15**(4), 31–38 (July 2011)
9. Goseva-Popstojanova, K., Li, F., Wang, X., Sangle, A.: A contribution towards solving the web workload puzzle. In: International Conference on Dependable Systems and Networks (DSN'06). pp. 505–516. IEEE (2006)
10. Hale, M.L., Gamble, M.T., Gamble, R.F.: A design and verification framework for service composition in the cloud. In: 2013 IEEE Ninth World Congress on Services. pp. 317–324 (June 2013)
11. Kuntze, N., Rudolph, C.: Secure digital chains of evidence. In: Systematic Approaches to Digital Forensic Engineering (SADFE), 2011 IEEE Sixth International Workshop on. pp. 1–8. IEEE (2011)

12. Lim, H.W., Kerschbaum, F., Wang, H.: Workflow signatures for business process compliance. IEEE Transactions on Dependable and Secure Computing **9**(5), 756–769 (2012)
13. Nami, M.R., Malekpour, A.: Application of self-managing properties in virtual organizations. In: Computer Science and its Applications, 2008. CSA'08. International Symposium on. pp. 13–16. IEEE (2008)
14. Paxson, V.: Empirically-derived analytic models of wide-area tcp connections (1993)
15. Rajalakshmi, J.R., Rathinraj, M., Braveen, M.: Anonymizing log management process for secure logging in the cloud. In: 2014 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2014]. pp. 1559–1564 (March 2014)
16. Ray, I., Belyaev, K., Strizhov, M., Mulamba, D., Rajaram, M.: Secure logging as a service-delegating log management to the cloud. IEEE Systems Journal **7**(2), 323–334 (June 2013)
17. Rudolph, C., Kuntze, N., Velikova, Z.: Secure web service workflow execution. Electronic Notes in Theoretical Computer Science **236**, 33–46 (2009)
18. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
19. Sundareswaran, S., Squicciarini, A.C., Lin, D.: Ensuring distributed accountability for data sharing in the cloud. IEEE Transactions on Dependable and Secure Computing **9**(4), 556–568 (2012)
20. Tian, F.: A supply chain traceability system for food safety based on haccp, blockchain & internet of things. In: Service Systems and Service Management (ICSSSM), 2017 International Conference on. pp. 1–6. IEEE (2017)
21. Tian, H., Chen, Z., Chang, C.C., Kuribayashi, M., Huang, Y., Cai, Y., Chen, Y., Wang, T.: Enabling public auditability for operation behaviors in cloud storage. Soft Computing **21**(8), 2175–2187 (2017)
22. Velikova, Z., Schütte, J., Kuntze, N.: Towards security in decentralized workflows. In: Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on. pp. 1–6. IEEE (2009)
23. Waters, B.R., Balfanz, D., Durfee, G., Smetters, D.K.: Building an encrypted and searchable audit log. In: NDSS. vol. 4, pp. 5–6 (2004)
24. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: International Conference on Business Process Management. pp. 329–347. Springer (2016)
25. Werner, M., Gehrke, N.: Multilevel process mining for financial audits. IEEE Transactions on Services Computing **8**(6), 820–832 (Nov 2015)
26. Wouters, K., Simoens, K., Lathouwers, D., Preneel, B.: Secure and privacy-friendly logging for egovernment services. In: 2008 Third International Conference on Availability, Reliability and Security. pp. 1091–1096 (March 2008)
27. Yao, J., Chen, S., Wang, C., Levy, D., Zic, J.: Accountability as a service for the cloud: From concept to implementation with bpel. In: Services (SERVICES-1), 2010 6th World Congress on. pp. 91–98. IEEE (2010)
28. Zawoad, S., Dutta, A., Hasan, R.: Towards building forensics enabled cloud through secure logging-as-a-service. IEEE Transactions on Dependable and Secure Computing (1), 1–1 (2016)
29. Zawoad, S., Dutta, A.K., Hasan, R.: Seclaas: secure logging-as-a-service for cloud forensics. In: Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. pp. 219–230. ACM (2013)