Authenticated and Auditable Data Sharing via Smart Contract

Vincent Reniers
imec-DistriNet, KU Leuven
vincent.reniers@cs.kuleuven.be

Yuan Gao
imec-COSIC, KU Leuven
yuan.gao@esat.kuleuven.be

Ren Zhang
Nervos and imec-COSIC, KU Leuven
ren@nervos.org

Paolo Viviani
Noesis Solutions NV
University of Torino
paolo.viviani@noesissolutions.com

Akash Madhusudan
imec-COSIC, KU Leuven
akash.madhusudan@esat.kuleuven.be

Bert Lagaisse
imec-DistriNet, KU Leuven
bert.lagaisse@cs.kuleuven.be

Svetla Nikova
imec-COSIC, KU Leuven
svetla.nikova@esat.kuleuven.be

Dimitri Van Landuyt
imec-DistriNet, KU Leuven
dimitri.vanlanduyt@cs.kuleuven.be

Riccardo Lombardi
Noesis Solutions NV
riccardo.lombardi@noesissolutions.com

Bart Preneel
imec-COSIC, KU Leuven
bart.preneel@esat.kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
wouter.joosen@cs.kuleuven.be

ABSTRACT

Our main use case features multiple companies that iteratively optimize on the architectural properties of aircraft components in a decentralized manner. In each optimization step of the so-called multi-disciplinary optimization (MDO) process, sensitive data is exchanged between organizations, and we require auditability and traceability of actions taken to assure compliance with signed legal agreements.

In this paper, we present a distributed protocol that coordinates authenticated and auditable exchanges of files, leveraging a smart contract. The entire life cycle of a file exchange, including file registration, access request and key distribution, is recorded and traceable via the smart contract. Moreover, when one party raises a dispute, the smart contract can be used to identify the dishonest party without compromising the file's confidentiality.

The proposed protocol provides a simple, novel, yet efficient approach to exchange files with support for data access auditability between companies involved in a private consortium with no incentive to share files outside of the protocol. We implemented the protocol in Solidity, deployed it on a private Ethereum blockchain, and validated it within the use case of a decentralized workflow.

CCS CONCEPTS

• **Software and its engineering → Peer-to-peer architectures**;
• **Applied computing →** *Enterprise information systems*;

SAC '20, March 30-April 3, 2020, Brno, Czech Republic

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373957

KEYWORDS

Blockchain storage, Distributed shared ledger, Data sharing smart contract, Auditable data sharing

ACM Reference Format:
Vincent Reniers, Yuan Gao, Ren Zhang, Paolo Viviani, Akash Madhusudan, Bert Lagaisse, Svetla Nikova, Dimitri Van Landuyt, Riccardo Lombardi, Bart Preneel, and Wouter Joosen. 2020. Authenticated and Auditable Data Sharing via Smart Contract. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3373957

1 INTRODUCTION

Authenticated and auditable data sharing is crucial for business processes involving multiple parties. A typical example of such business processes is distributed multi-disciplinary optimization (MDO), where a number of companies collaborate on optimizing a complicated industrial design, such as that of an aircraft. Due to the design's complexity and the involved companies' different expertise, such as structural stability, aerodynamics, and propulsion, data exchanges happen frequently throughout the decentralized workflow. The exchanged data must only be visible to intended receivers as they are subject to stringent intellectual property constraints. Furthermore, data integrity also needs to be verifiable because of the legal implications in case of an accident. Moreover, the data registration and access operations need to be logged and timestamped as they are backed by signed legal agreements.

This use case can clearly be generalized to many other cases that require tracking of data exchanges between organizations for legal compliance. In such processes, it is highly important to keep a persistent and tamper-proof record of which data, when, and with whom has been shared as to back the signed legal agreements. Therefore, the manner in which such data access operations are logged must be trustworthy and non-repudiable by either party.

To meet with such requirements of non-repudiation, typically a trusted third party is involved as an intermediary in the data

sharing process (e.g. a cloud storage provider) [19, 22]. This trusted third party can be used to log each operation taken by any party, and resolve any potential disputes. This traditional approach is sub-optimal for three key reasons [16]: (i) it requires a minimum of trust by all involved organizations in the third party, (ii) in case of disputes manual intervention is required which is time-consuming and inefficient, and (iii) it may in such cases be required to reveal the contents of the confidential data.

To this end, decentralized ledger technologies (DLTs) present opportunities to remove trusted third parties by placing trust in the decentralized network and its established consensus algorithms. Indeed, many related works have already focused on the topic of blockchain and data sharing. A subset of these works have focused on (i) simply storing file hashes to verify data integrity [9, 13], to (ii) managing file access permissions on-chain which are then applied locally [3], or even (iii) executing file access policies on-chain [14]. In some of these works auditability is provided on operations such as changing the access permissions, or data contents. More importantly, there are works that (iv) enable the effective operation of data sharing itself via the blockchain, and even feature blockchain-managed (P2P) storage [17, 23]. These works support auditability on the stored data's integrity, however crucially not who accessed or modified the files (i.e. the data access trail).

To our knowledge, the only work which provides auditability on file access operations is CALYPSO [11]. File access is logged on-chain when a user requests access to a file's decryption key. A proof of authorization is first requested from a collection of special nodes, referred to as the secret-management cothority, which log the request. Access to the file key itself is granted in a second step, by providing the proof of authorization to an access-control cothority.

We propose a protocol which presents a simpler, alternative request-response scheme via the blockchain to grant file decryption keys. We provide an implementation of the proposed protocol in an executable Ethereum [21] contract, implemented in Solidity [6].

Comparing with CALYPSO, our solution provides three key benefits: (i) we avoid the need for complex distributed key management to preserve confidentiality of data, (ii) our proposed solution is easy to deploy as a smart contract, and (iii) the smart contract is portable across many blockchain technologies that support Solidity. A potential downside of our protocol is that the data owner has to be online to transmit the symmetric file key via the smart contract upon request. Such active involvement is typically already present in many cases, such as our use case of a decentralized workflow, and sharing a key can be automated at each client.

The remainder of the paper is structured as follows. Section 2 presents the motivating use case and Section 3 its requirements. Section 4 details the proposed protocol, and several scenarios. Section 5 discusses a practical implementation of the protocol in a smart contract. Section 6 validates this architecture deployed within our initial use case, that of a decentralized workflow. Section 7 discusses related work. Section 8 concludes the paper.

2 USE CASE AND PROBLEM STATEMENT

We motivate our proposed solution for auditable data sharing in the context of an industrial case by Noesis Solutions [18] regarding multi-disciplinary optimization (MDO). The MDO process is an iterative, distributed process that involves the sharing of optimization results on industrial designs [5, 12]. In the initial phases, the design may be rather simple and communication can remain internal to the company.

As the design grows increasingly complex, more specialists are involved that are experts in their respective domains (e.g. structural stability, aerodynamics). Consequentially, communication becomes increasingly more difficult to coordinate and oversee between these organizations. In our case, Noesis Solutions' [18] has developed the Optimus tool which effectively coordinates these types of workflows.

In a workflow, shown in Figure 1, there are two types of actors: a singular workflow coordinator, and multiple disciplinary expert companies. Every time a parameter is changed, each participant has to re-run their optimization in an iterative fashion, which ultimately converges to the optimal design.

In this paper, we primarily focus on the manner in which data is exchanged, since the data involves optimization results on sensitive designs that have to be delivered within certain time frames.

2.1 Data sharing with traceability

In the MDO process, sensitive data is shared involving aircraft components and designs that are subject to intellectual property rights, and are covered by signed legal agreements between the parties. As a result, it is important to log which parties shared which data, at which time, and with whom. These logs can furthermore help to trace any mistake in the optimization process, or to verify if parties meet with contractual obligations such as delivering results within a given time frame. Consequentially, any operation must be logged in a non-repudiable manner.

2.1.1 Current situation. In order to log any event or operation with non-repudiation, a trusted third party serves as an intermediary in the data sharing process. The shared data is typically encrypted as to preserve its confidentiality from the trusted third party. This intermediary has several responsibilities: (i) storing and managing retrieval of the data, (ii) creating log events on every operation that occurs, (iii) allowing auditability of these events, and (iv) facilitating conflict resolution when disputes arise.

Dispute and conflict resolution. Several conflicts can arise between parties, for example a file recipient can claim that the owner sent rubbish instead of the actual required data. In such a case, the encrypted data can be revealed to the trusted third party, or an external validator, as to resolve the conflict. Conflicts can also arise on for example the timeliness of operations, or the correctness of a decryption key. In the former case, the validity can simply be checked by a third party without revealing too much private information. However, in the case of verifying file contents, this may result in revealing the file's confidential nature.

2.1.2 Problem statement. The involvement of a trusted third party, while highly common in many business cases, is sub-optimal for three reasons [16]: (i) it is costly, and requires a level of trust by all involved entities, (ii) conflict resolution may require exposing the contents of the sensitive files to this party, and (iii) such intervention

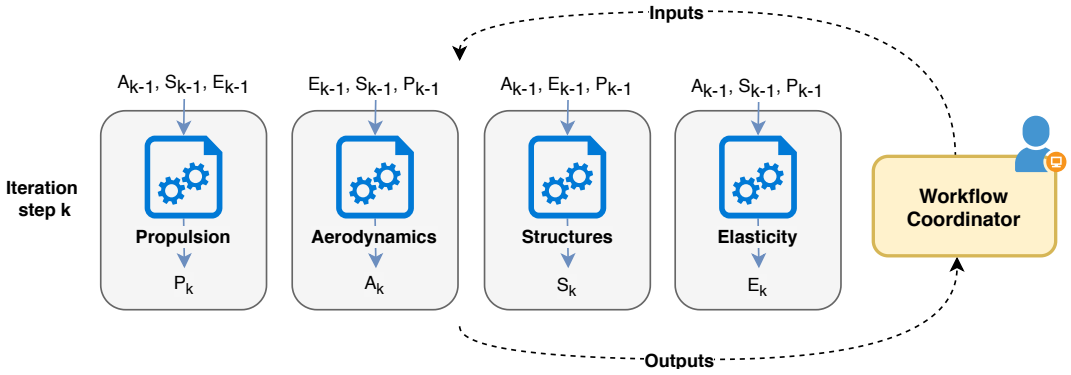

Figure 1: MDO process and data sharing between participants and the coordinator [16].

can involve manual interference which is time-consuming and hinders overall efficiency of the application.

2.2 Decentralized ledger technologies

Decentralized ledger technologies (DLTs) are highly promising in certain cases to avoid the use of trusted parties. Instead, trust can be placed in the entire network and its established consensus algorithms. Numerous blockchain technologies have emerged over the years, the most prominent being Bitcoin [15] and Ethereum [21], each with different trade-offs between performance, security, and functionality. Ledger technologies can be used within our use case to log any data operations taking place in a persistent, and tamper-proof manner, or to actually coordinate the operation itself.

In the next section, we outline the functional and non-functional requirements given the presented use case.

3 REQUIREMENTS

The proposed solution and the case of distributed multi-disciplinary optimization (MDO) has to meet with specific functional, and non-functional requirements [16]. These requirements are typically also in line with other cases that involve communication between organizatios, and which require auditability and traceability of data sharing operations.

Functional requirements

We first outline the functional requirements:

- F1 **Data sharing with access control.** Files can be uploaded, and shared to a specific set of recipients.
- F2 **Auditability of any data operation.** Any data access operation (read, or write) can be checked by anyone, including external auditors.
- F3 **Data validation.** Any party involved can verify independently whether a downloaded file, or a given symmetric file key, is correct.
- F4 **Conflict resolution.** Appropriate mechanisms are in place in the network to resolve either conflicts on (i) the integrity of a file, or (ii) the integrity of a key.

Non-functional requirements

- NF1 **Decentralization.** Trust is placed in the decentralized network and its established consensus algorithms, over the typical traditional approach of involving centralized 3rd parties, which require a level of trust by all involved parties.
- NF2 **Non-repudiation of data operations.** Log creation on any data operation (e.g. write, read) occurs in a trustworthy, non-repudiable, and tamper-proof manner. Neither party can refute the correctness of any logged operation.
- NF3 **Availability and scalability.** We require high availability and scalability of the system as the data set grows, or the number of participants increases.
- NF4 **Data confidentiality.** Any exchanges of for example symmetric file keys, while they may occur on a public blockchain network, must remain confidential.

Blockchain platforms are in essence a tamper-proof decentralized shared ledger, and as a result the shared data is highly available. As such, many of these requirements are either partially or completely satisfied by simply employing the use of a blockchain platform (NF1, NF2, NF3).

Further emphasis can be placed on confidentiality, such as limiting the visibility of data transactions only to organizations involved in the specific process.

In the next section we propose a protocol to satisfy the aforementioned requirements and attain authenticated and auditable data sharing.

4 AUDITABLE DATA SHARING PROTOCOL

When simply using the blockchain as a logging mechanism for sharing data, there are two possibilities regarding malicious data transfer, as also explained in [8, 16]:

- The receiving party can receive the file without acknowledging receipt, or falsely claiming that the data is incorrect.
- The file owner can send rubbish data.

In order to guarantee data access accountability and identification of dishonest parties in case of such malicious transfer, the blockchain has to be an essential part of the data sharing process and cannot be omitted in order to gain access to the files.

Table 1: Terminology and definitions.

Symbol		Symbol	
D_o	Data owner	F	File
D_p	Data participant	C_f	Encrypted file F
K_f	File key	$K_{f,1}$	Part of file key
R_x	Random number	$K_{f,2}$	Part of file key
PK_p	Public key participant		
Symbol			
$Commit_1$	Hash of first file key part and random number R_1		
$Commit_2$	Hash of second file key part and random number R_2		
C_1	Encrypts file key part 1 and R_1 using a public key.		
C_2	Encrypts file key part 2 and R_2 using a public key.		

We present a protocol that is a request-response based scheme to exchange symmetric file keys between parties. The protocol involves two actors: the data owner (D_o) and data participant (D_p). The scheme leverages a smart contract, and the files are exchanged via a shared storage platform.

We outline the protocol specification on the basis of several scenarios that involve one or both of the two actors. The key scenarios that comprise the protocol are explained in order of: creating a file, uploading and announcing a file, to other parties requesting file access, and finally responding with the decryption key.

Table 1 summarizes the terminology and abbreviations used within the protocol's specification.

4.1 Scenario: uploading a file

Assume the case where the data owner D_o wishes to share a file with a set of recipients. Before anything occurs on the blockchain network itself, the file is encrypted and uploaded to a shared P2P or centralized storage system. Figure 2 essentially shows the following steps in the file upload process that take place:

(1) D_o encrypts file F, obtaining C_f using a symmetric key K_f.
(2) D_o uploads the encrypted file C_f to a shared storage system.

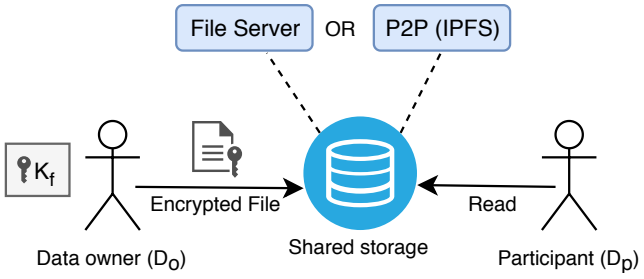

Figure 2: Upload file to shared storage.

The shared storage can be a centralized cloud storage service, or a P2P system such as IPFS [4] The protocol only requires that everyone, or at least each recipient, can access the file, verify it, and that it remains available. With regards to availability, it may be wise to assure that when using a distributed file network the file remains properly replicated (NF3). For such purposes, incentivized P2P file storage solutions such as Sia [2] and Storj [1] are potentially interesting. Alternatively, files can also be hosted on the involved parties' owned servers, or a combination of cloud storage providers.

4.2 Scenario: announcing a file

When a new file has been added, any party that has access to the shared storage can already download the encrypted file, but not yet decrypt it except for D_o. The data owner D_o can now announce to everyone, or a specific list of recipients, that a new file F is ready to be shared. The process is shown in Figure 3 and takes place as follows:

(1) Two random numbers R_1 and R_2 are generated.
(2) Owner D_o splits the symmetric file key K_f into $K_{f,1}$ and $K_{f,2}$, so that $K_f = K_{f,1} \oplus K_{f,2}$.
(3) Owner D_o makes the following commit messages.
    - $Commit_1$ = Hash($K_{f,1}||R_1$)
    - $Commit_2$ = Hash($K_{f,2}||R_2$)
(4) D_o publishes the commit messages and file meta-data, announcing to all involved recipients that a specific file at a given location is available for request.

This announcement is conducted ideally via a smart contract on the blockchain, which can limit a file's access requests to a set list of recipients (i.e. provide a degree of access control). It is only after file announcement that the smart contract's functionality to request a decryption key becomes available. The operations used such as a key split is a simple XOR operation, whereas the hash function can be SHA2, since the latter is typically supported by smart contract languages such as Solidity.

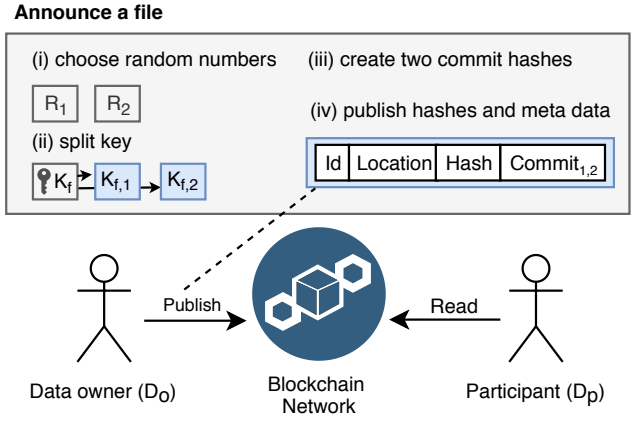

Figure 3: Announce a file to the shared ledger.

The commit messages will later be used when anyone disputes the validity of the key. Such dispute functionality can be implemented in a smart contract to verify one key share without revealing a single bit of the actual key. The random numbers are used to hide the key share, and specifically to prevent anyone from verifying a leaked share's correctness by hashing it, or attempting to find a key share via a brute-force attack.

4.3 Scenario: requesting a file

The next scenario is that of a party who wishes to become a data participant D_p. The prospective data participant D_p publishes a request for the file C_f on the blockchain, or via a smart contract (which can enforce access control on file requests).

(1) D_p downloads the encrypted file C_f to which he wants to gain access.
(2) D_p checks C_f against the announced file hash by D_o.
(3) D_p publishes a request for the file C_f's key, and includes his public key PK_p.

The explicit request ensures that no party can demand the file decryption key without being logged. Moreover, D_p must only send the request after verifying the file's integrity, eliminating potential disputes. The data owner D_o can then check the blockchain for any pending decryption requests.

4.4 Scenario: responding to a file request

The data owner D_o observes on the blockchain that a request has been made for one of his/her files. The owner can decide to either respond, or to ignore the request when the party isn't permitted access. In case the owner responds to the decryption request, the file key is sent to the data participant D_p via the blockchain as follows:

(1) Owner D_o sends the following messages, and encrypts them using the requester's public key PK_p.
   - $C_1 = \text{Encrypt}_{PKp}(K_{f,1}||R_1)$
   - $C_2 = \text{Encrypt}_{PKp}(K_{f,2}||R_2)$
(2) The format of the decrypted text of C_1 and C_2 clearly defines the random number and key part. D_p can now decrypt both messages and discover respectively R_1, $K_{f,1}$, and R_2, $K_{f,2}$.

If the transmitted keys are correct and K_f decrypts the file, then this public exchange serves as a proof of file access. The key can be invalid when either (i) the key doesn't match with the commit hashes published by the owner, or (ii) the key doesn't decrypt the file properly. In case of key incorrectness, it can be challenged later by any involved party which relies on the fact that every event in the request-response based scheme is logged in a tamper-proof manner.

4.5 Scenario: disputing a key

Suppose the data owner D_o sends an incorrect key to the party D_p, as in not being the key that was originally promised. In that case, one or both parts of the following calculation must not match up with the original commit message.

- $Commit_1^{'} = \text{hash}(K_{f',1}||R_1) \neq$ the original $Commit_1$
- $Commit_2^{'} = \text{hash}(K_{f',2}||R_2) \neq$ the original $Commit_1$

If one of these conditions is valid, then this can be checked autonomously by the network when the entire exchange was coordinated via a smart contract. In such a case, the party D_p reveals one of the following conditions to the smart contract:

- The receiving party only reveals R_1 and $K_{f',1}$.
- The receiving party only reveals R_2 and $K_{f',2}$.

The smart contract can then indeed check whether one of the above conditions is correct or not, and thus whether or not the data owner has transmitted the promised key. When both key parts are wrong, only one part is revealed to prove that the key is isn't the promised key by the owner. In this manner, the entire key is never leaked. In case the transmitted key was actually an invalid key, and it does not allow for a proper decryption, then other measures have to be taken, for example via off-chain resolution (e.g. consulting a trusted 3rd party).

5 SMART CONTRACT IMPLEMENTATION

In this section we detail a practical implementation of the aforementioned protocol for auditable data sharing.

There are two options to implement this protocol, either recording each request-response inside meta-data of a blockchain transction, or coordinating the protocol via a smart contract.

A smart contract deployed on a blockchain is a program which runs on all the miner nodes concurrently. The contract lives in its own sandboxed environment, with its own memory stack. Function calls to the contract are executed simultaneously on all its instances in the network. In this manner, the state is kept synchronous and the trust in its output is based on the consensus of multiple simultaneous executions.

Within the context of our protocol, a smart contract provides several benefits versus simply storing data in blockchain transactions, namely: (i) enabling access control and limiting requests, and (ii) enabling key correctness validation autonomously. Thirdly, (iii) a smart contract is portable across multiple blockchain technologies.

5.1 Smart contract functionality

In this section, we briefly outline the contract's functionality. We opted to implement the smart contract using the smart contract programming language Solidity [6], for reasons of its widespread adoption and compatibility with numerous platforms, of which most notably Ethereum [21]. The smart contract's functionality mostly runs on the basis of internally-kept data structures that maintain the state of which files are shared, and with which parties.

5.1.1 Data structures. The internal state of the smart contract is comprised of: the owner's address, lists of recipients, and lists of files. A file is comprised of its hash, location, owner's public key, version number, and the set of parties that are considered either viewers or requesters. These parties are stored as a list of Ethereum addresses and their respective public keys. The file structure also keeps track of the decryption keys that have been shared, which are encrypted using the public keys initially provided by the requesters.

5.1.2 Functionality. Our contract provides functionality that is publicly accessible, as well as functions that are exclusively available to the contract owner (its deployer). We first outline the four core publicly-available functions:

- **createFile**: Anyone can create a file, listing its location, hash, the owner's public key, commit hashes and intended list of recipients.
- **askDecrypt**: Once a file has been created, only parties that are in the list of recipients can request decryption, and they do so by providing their public key.

- **respondDecrypt**: A request can be responded to, only by the owner of that file. In doing so, the owner provides an encrypted copy of the file key.
- **Inspectors**: Methods which reveal the state of the contract. These states include: pending requests for a file, all files owned by a party, file information, and encrypted keys.

5.1.3 Access control. The owner of the contract has special functionality which pertains to the management of the list of recipients. He can create a new list of recipients, to which a name (identifier) is associated, which is used in the creation of files. Importantly, the owner cannot alter existing files, or events that have occurred.

5.1.4 Current implementation. In this version of the contract there is no functionality yet to resolve key conflicts via the smart contract. Such issues will have to be resolved off-chain for now, or can be added at a later stage. The main reason as to why this is the case is that the smart contract is currently not aware of commit messages, and their significance. Instead, we currently resort to off-chain resolution in case of any conflicts that arise, with the blockchain and the smart contract serving as a tamper-proof history of events that took place, and which can not be repudiated by any involved, or external party.

Such automated conflict resolution mechanisms on-chain are out of the scope of the use case, that is instead focused on achieving the properties of non-repudiation and auditability on the data access trail. In fact, conflict resolution in an MDO process would require complex activities that should be taken by actors in the process (i.e. uploading a different file), and it is not desirable by the involved companies to automate at this stage. On the other hand, it is critical to provide a tamper-proof log of reads and writes to a human operator that is in charge of resolving the conflict.

6 DEPLOYMENT AND VALIDATION

Figure 4 illustrates the setup of our deployment. In our use case, that of a decentralized workflow, we setup an Ethereum node within each company's premises. This Ethereum node has an RPC interface, and an Ethereum wallet with a pre-defined set of currency that originates from the genesis block. A sufficient amount of funds has been allocated to each company in a range that it will never become an issue. In our use case, currency isn't the incentive which drives the network, rather it's the organizations' desire to keep the workflow operational.

Each workflow participant or coordinator calls their respective node's RPC interface to execute contract functions, or to inspect the overall state of the blockchain. The Ethereum nodes themselves are running on the go-ethereum (geth) implementation of Ethereum.

6.1 Contract deployment

In our case, we have a single contract pre-deployed, and we simply refer to a static address where it resides. In practice, all participants would have to agree on the location of the smart contract which is going to be used for the exchange of files. If multiple deployments of the smart contract exist, then the file exchange coordination may become fragmented.

However, there may also be benefits for creating multiple deployments of the same file exchange smart contracts when for example

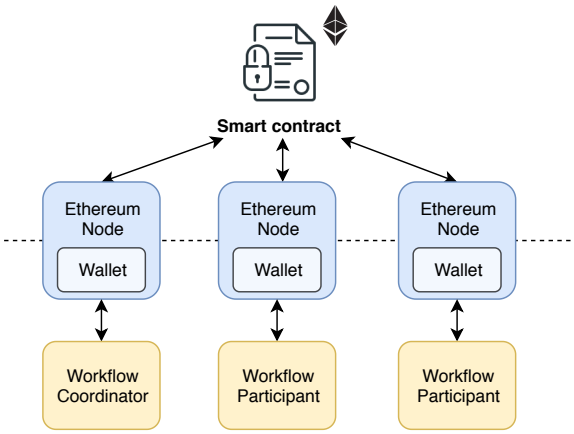

Figure 4: Deployment setup.

the internal data structure grows rather large due to an increasing number of files shared. If a certain workflow is finished, and no conflicts are signalled after a certain time window, then it may be of interest to create a new deployment of the contract.

6.2 Application architecture

In order to connect the blockchain and smart contract functionality to our existing workflow application (written in python), we can make use of several libraries such as the web3py framework. The web3py framework acts as wrapper around the low-level blockchain API and provides functions to load a contract's binary code, deploy it, and call its functions.

6.2.1 Python client. Since the web3py framework isn't tailored to the specific API of our smart contract and its functions, we have written a class `FileTransferContract.py` which neatly packs all the required functionality for a specific company. Each company configures their own Ethereum node IP in the class, and can subsequently inspect all files present, create a file, or request decryption. Since these inspectors reveal a large set of data, we have also provided a visual web interface on top of this wrapper to allow each company to inspect the on-going status.

6.2.2 Web interface. The web interface (shown in Figure 5) is the portal for each company to the smart contract, where it can interact with existing files, check pending file requests, or received files. The web server is a complete standalone service that can be launched on any node, as long as this node can interface with the company's specific Ethereum node and interface with its RPC interface to inspect the blockchain and execute contract functions. Each web service request generally makes use of the previously-discussed Python client functionality.

6.3 Validation

We used this web portal to validate the case of a decentralized workflow, and assure that the design of our setup meets our requirements of non-repudiation, and auditability of data access.

Use case scenarios. We tested several scenarios starting with the announcement of a file, which mentions the owner's public key and

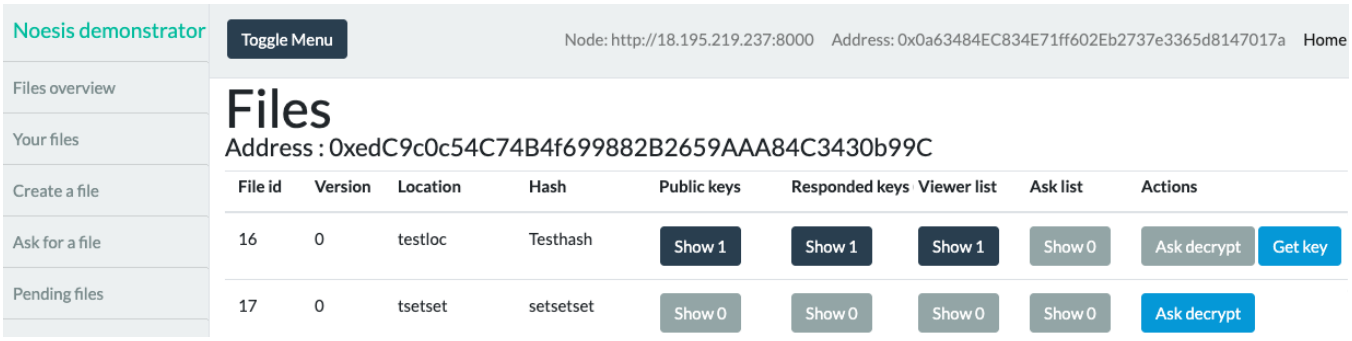

Figure 5: Web interface to manage files and key requests.

a list of recipients (Ethereum addresses). Subsequently, we tested the case where both a permitted, and unpermitted organization request access to the file. The unpermitted request was cancelled by the smart contract, whereas a permitted organization can submit its public key as a request. In our web interface, we then subsequently tested the case where the data owner D_o responds to a key request, which sends the file key encrypted using the provided public key. The other organization was able to properly request its decryption key from the contract, and subsequently decrypt the symmetric key locally. Any other involved organization was able to audit which organizations had shared files and with whom for auditability purposes.

Performance. In terms of performance, we observed that the Ethereum network takes time to process certain transactions which create actual data. For example, creating a file announcement via the smart contract took on average 15 seconds. Whereas a simple call to inspect the existing files returns relatively fast, as it mainly involves network latency.

7 RELATED WORK

Cloud storage services, such as Google Drive or Dropbox, allow users to store, and share their data in a convenient manner. These platforms, while very common in practice, lack the clear functional requirement for data access accountability for our organizations. Moreover, such centralized solutions pose a single point of potential failure or compromise, with for example the user even unaware of any potential or on-going breaches [10]. In light of our requirements, these platforms are therefore not suitable regarding data integrity assurance, data access accountability, and preserving data confidentiality.

In light of avoiding a single point of trust, decentralization of such services presents many opportunities. A main advantage of decentralization is that it can annul the need for a trusted third party. Still, proper caution has to be taken to ensure data integrity, confidentiality and accountability properties.

7.1 Blockchain for file integrity and management of access policies

Several works facilitate data sharing, or integrity verification in a decentralized manner, without access accountability or auditability as a key requirement. We briefly categorize these related work as follows:

- Frameworks that use blockchain to verify data integrity by simply storing file hashes on blockchain-based databases [9, 13]. Any user can check if the file they receive is valid after computing its hash and comparing the hash value with the value stored in the tamper-proof shared ledger.
- The management of file access permissions on the blockchain in combination with local clients that alter these permissions [3]. The blockchain provides a tamper-proof record, and history of file permissions. In [14] the evaluation of access policies is even conducted on-chain.

The frameworks listed above provide auditability in terms of changing file access permissions, or the integrity of file data (via stored hashes) but crucially aren't involved in the data sharing process itself. The latter is a key requirement towards achieving data access auditability.

7.2 Blockchain file sharing platforms

The following academic and industrial implementations enable data sharing via the blockchain as an essential part of the process.

- Zyskind et al. [23] and Shafagh et al. [17] enable data sharing and integrity auditability via blockchain-governed P2P storage nodes.
- Blockchain file storage solutions, such as Sia [2] and Storj [1], manage a shared P2P file network on which users can store and retrieve files in an incentivized manner.

These works unfortunately do not provide a mechanism to track a file's access trail.

Auditable data sharing. NuCypher KMS [7] is a decentralized Key Management System (KMS) that can compliment decentralized file storage solutions such as Sia [2] and Storj [1]. It uses proxy re-encryption (PRE) to grant access to a list of receivers. Selected nodes in the blockchain act as re-encryption nodes that hold re-encryption keys for several files. Whenever a designated receiver requests access to a file stored in distributed storage such as Sia [2], the re-encryption nodes carry out PRE on the file key such that the receiver can decrypt it while no information is leaked to any other party. While access trail logging isn't implemented yet, the authors do argue that it can be integrated [7].

To our knowledge, CALYPSO [11] is the only work that enables extensive auditability of file access operations. The CALYPSO protocol uses threshold cryptography, blockchain technology and a group of witnesses or trustees referred to as a collective authority [20]. It is sufficient and enables the monitoring of the file's access trail, as everything is logged on the blockchain.

Despite the fact that this platform meets with a large portion of our requirements, protocols such as CALYPSO [11] are complex in regards of their underlying architecture. A large communication overhead is involved which stems from two separate consensus mechanisms to authenticate and process a data request. The first mechanism verifies the write and read transaction, which responds with a proof of permitted access. This proof can be send to a second cothority of nodes which govern access to the file decryption key. These consensus mechanisms deliver the proof of file access in an auditable manner, however involve a large communication and upkeep overhead. The main advantage of the CALYPSO protocol is that the owner of a secret does not need to be online after storing his secret on-chain.

Our proposed protocol avoids the need for a complex architecture involving dedicated nodes that manage the file keys on-chain. Instead, we rely on the data owner to respond to file decryption requests with the required key. A single smart contract is used to carry out this request-response scheme, and delivers auditability on file access operations, while storing the files off-chain. This reliance on a single smart contract keeps the architecture of our protocol fairly simple and portable.

8 CONCLUSIONS

Many application cases and especially in an industrial, inter-organizational context (e.g. B2B) today require the involvement of 3rd party intermediaries to enable data access accountability. Unfortunately, the involvement of such a third party requires a minimum degree of trust by all involved participants, and secondly potentially exposes highly confidential files to these intermediaries. Many ongoing research has already focused on the combination of blockchain technologies to remove the necessity of trusted third parties, by placing trust in the network and its consensus algorithms.

In this paper, we have presented the specification for a protocol to auditable data sharing via a blockchain platform. The solution in comparison to existing related work is much more simpler in terms of architecture and algorithms, but does require that the data owner is online and available to respond to document requests (to obtain decryption keys).

We have provided a practical proof-of-concept implementation of the protocol in the smart contract language Solidity, making our solution portable across many blockchain platforms. The smart contract has been tested and validated within the industrial use case of multi-disciplinary optimization (MDO).

In its current state, this smart contract can be deployed on a public blockchain and used reliably to attain data access auditability. Further research is however required to analyze the potential privacy threats, such as the ability of an outsider to infer which parties are involved in a collaboration, or to observe which data is exchanged and how frequently.

ACKNOWLEDGMENTS

This research is funded by the Research Fund KU Leuven and the imec-ICON BoSS project.

REFERENCES

[1] 2019. Decentralized Cloud Storage – Storj. https://storj.io/. Accessed: 2019-09-28.
[2] 2019. Sia - Cloud storage. https://sia.tech/. Accessed: 2019-09-28.
[3] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. 2016. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *2016 2nd International Conference on Open and Big Data (OBD)*. 25–30. https://doi.org/10.1109/OBD.2016.11
[4] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* abs/1407.3561 (2014). arXiv:1407.3561 http://arxiv.org/abs/1407.3561
[5] Evin J Cramer, John E Dennis, Jr, Paul D Frank, Robert Michael Lewis, and Gregory R Shubin. 1994. Problem formulation for multidisciplinary optimization. *SIAM Journal on Optimization* 4, 4 (1994), 754–776.
[6] Chris Dannen. 2017. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners* (1st ed.). Apress, Berkely, CA, USA.
[7] Michael Egorov and MacLane Wilkison. 2017. NuCypher KMS: Decentralized key management system. *ArXiv* abs/1707.06140 (2017).
[8] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. 2006. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive* 2006 (2006). http://eprint.iacr.org/2006/150
[9] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. 2017. Blockchain-based database to ensure data integrity in cloud computing environments. In *Italian Conference on Cybersecurity (20/01/17)*. https://eprints.soton.ac.uk/411996/
[10] The Guardian. 2016. Dropbox hack leads to leaking of 68m user passwords on the internet. https://www.theguardian.com/technology/2016/aug/31/dropbox-hack-passwords-68m-data-breach
[11] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gailly, Linus Gasser, Philipp Jovanovic, Ewa Syta, and Bryan Ford. 2018. Calypso: Auditable sharing of private data over blockchains. *Cryptology ePrint Archive* 209 (2018). https://eprint.iacr.org/2018/209.pdf
[12] Ilan Kroo, Steve Altus, Robert Braun, Peter Gage, and Ian Sobieski. 1994. Multidisciplinary optimization methods for aircraft preliminary design. In *5th symposium on multidisciplinary analysis and optimization*.
[13] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu. 2017. Blockchain Based Data Integrity Service Framework for IoT Data. In *2017 IEEE International Conference on Web Services (ICWS)*. 468–475. https://doi.org/10.1109/ICWS.2017.54
[14] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2019. A blockchain based approach for the definition of auditable Access Control systems. *Computers & Security* 84 (2019), 93 – 119. https://doi.org/10.1016/j.cose.2019.03.016
[15] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved 2019-09-28 from https://bitcoin.org/bitcoin.pdf
[16] Vincent Reniers, Dimitri Van Landuyt, Paolo Viviani, Bert Lagaisse, Riccardo Lombardi, and Wouter Joosen. 2019. Analysis of Architectural Variants for Auditable Blockchain-based Private Data Sharing. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*. ACM, New York, NY, USA, 346–354. https://doi.org/10.1145/3297280.3297316
[17] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. 2017. Towards Blockchain-based Auditable Storage and Sharing of IoT Data. In *Proceedings of the 2017 on Cloud Computing Security Workshop (CCSW '17)*. ACM, New York, NY, USA, 45–50. https://doi.org/10.1145/3140649.3140656
[18] Noesis Solutions. 2018. Software Solutions that enable Objectives Based Engineering. https://www.noesissolutions.com/
[19] S. Sundareswaran, A. Squicciarini, and D. Lin. 2012. Ensuring Distributed Accountability for Data Sharing in the Cloud. *IEEE Transactions on Dependable and Secure Computing* 9, 4 (July 2012), 556–568. https://doi.org/10.1109/TDSC.2012.26
[20] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. 2016. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*. 526–545. https://doi.org/10.1109/SP.2016.38
[21] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014), 1–32.
[22] S. Yu, C. Wang, K. Ren, and W. Lou. 2010. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing. In *2010 Proceedings IEEE INFOCOM*. 1–9. https://doi.org/10.1109/INFCOM.2010.5462174
[23] G. Zyskind, O. Nathan, and A. Pentland. 2015. Decentralizing Privacy: Using Blockchain to Protect Personal Data. In *2015 IEEE Security and Privacy Workshops*. 180–184. https://doi.org/10.1109/SPW.2015.27