Analysis of Architectural Variants for Auditable Blockchain-based Private Data Sharing

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

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

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

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

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

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

ABSTRACT

Many applications by design depend on costly trusted third-party auditors. One such example is the industrial application case of federated multi-disciplinary optimization (MDO), in which different organizations contribute to a complex engineering design effort. Although blockchain and distributed ledger technology (DLT) has strong potential in reducing the dependence on such intermediaries, the architectural complexity involved in designing a solution is daunting.

In this paper, we analyze the architectural variants for decentralized private data sharing while guaranteeing auditability and non-repudiation of data access operations, as well availability of the shared data. The architectural variants analyzed focus on attaining: (i) confidential data exchange, (ii) governing access to the shared data, (iii) providing data access auditability, and (iv) data validation or conflict resolution. We systematically enumerate architectural decisions at the levels of: storage, policy-based file access control, data encryption methods, and auditability mechanisms for private data.

The main contribution of this work is a comprehensive overview of architectural variants for decentralized control of private encrypted data, and the involved trade-offs in terms of performance, storage overhead, auditable trust and security. These findings are validated in the context of the aforementioned industry case that involves federated multi-disciplinary optimization (MDO).

CCS CONCEPTS

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

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297316

KEYWORDS

Blockchain storage, Distributed shared ledger, Decentralized data access control, Decentralized private data auditing

ACM Reference Format:
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 *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3297280.3297316

1 INTRODUCTION AND MOTIVATING CASE

The motivating application case of Noesis Solutions [29] involves a multi-disciplinary optimization (MDO) process between engineering teams that are distributed geographically, and are even affiliated with different organizations. The federated MDO process involves the optimization of airplane component designs, which are safety-critical and highly confidential. Figure 1 illustrates how an aircraft's physical aspects are co-dependent and have to be optimized in an iterative, multi-disciplinary process between different organizations, each skilled in their respective disciplines, such as structural stability or aerodynamics. For example, the exterior of an engine's physical structure is optimized based on the current design of the propulsion system. Such a process is referred to as a multi-interdisciplinary optimization (MDO) process [9, 20]. As depicted in Figure 2, in an MDO process, the results of one task have to be shared as input data to the optimization process of the subsequent tasks.

Due to the safety-critical nature of these airplane component designs, and the potential real-life consequences when things go wrong, these organizations require guarantees in terms of non-repudiation and logging of each action taken in the exchange, consultation or modification of data. Current solutions rely extensively on third-party auditors that provide services in terms of verification of the validity of transferred data and the established data provenance records (which signify the history and data lineage). These auditors are critical to determine when parties are at fault, or for providing proof when they did not meet contractual obligations, such as managing time windows to provide optimization results.

The data sharing scheme involving trusted third-parties is sub-optimal for three reasons: (i) the dependence on a third party may be costly, (ii) manual validation exposes confidential data to trusted

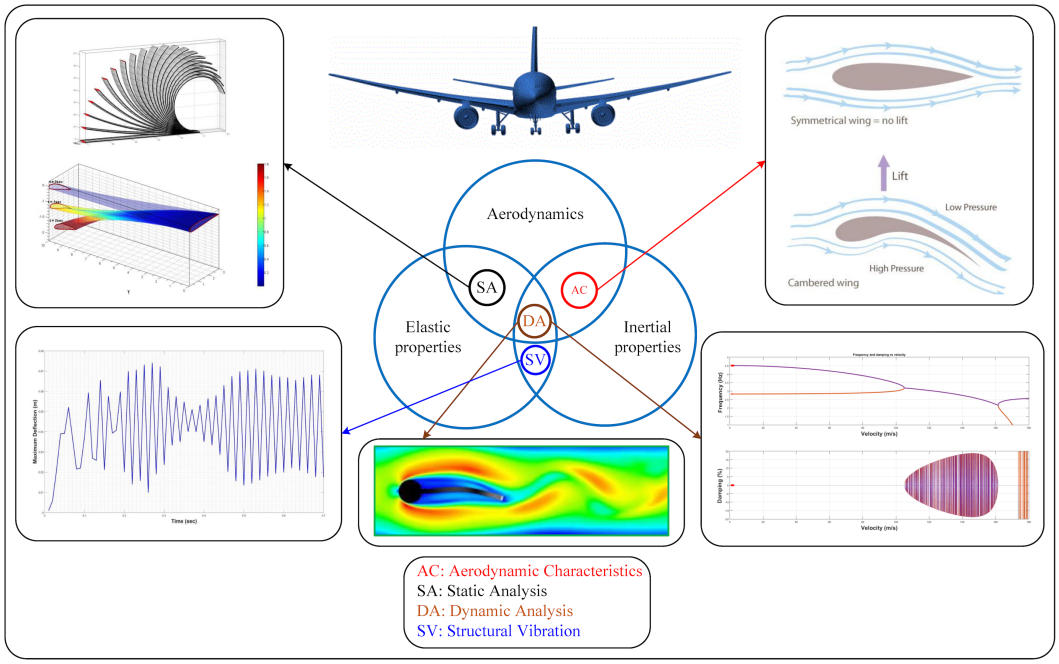

Figure 1: Multidisciplinary design and analysis [5]. Copyright © Ahmed Bayoumy

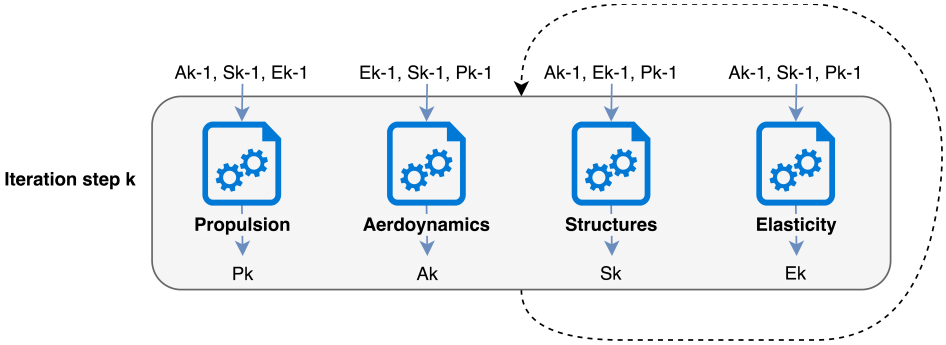

Figure 2: Iterative MDO process, each discipline owned by a specialized engineering team.

third parties, and (iii) the overall efficiency is reduced due to manual interference.

In avoiding trusted third parties, decentralized shared ledgers (DLT) technologies and smart contracts (programs deployed and executed on the blockchain [33]) present major opportunities to solve issues of non-repudiation by maintaining a tamper-proof log of actions, and enable consensus-driven validation followed by audit log creation. Furthermore, the shared data can be governed autonomously by decentralized logic, thereby assuring data availability through automated replication across for example blockchain nodes themselves, or cloud storage platforms. Such decentralized storage systems are considered a viable business application of blockchain technologies. Several existing implementations already accomplish some of these goals, such as Filecoin [21], Storj [1], Siacoin [2] and NuCypher KMS [13].

Additionally, the enterprise nature of this case requires careful consideration of requirements such as security and trust management. The shared data's level of confidentiality complicates matters when data log inconsistencies arise, towards which existing work has focused on verifying data integrity on untrusted storage platforms [34], and external data auditing while preserving confidentiality and privacy [15, 17, 18].

However, the architectural complexity involved in devising a suited ledger-based solution is substantial, and many non-trivial architectural decisions have to be made in terms of storage, policy-based file access control, data encryption methods, and auditability mechanisms for private data.

In this paper, we systematically investigate several architectural approaches to implement auditable blockchain-based private data sharing, for example for adoption in the discussed MDO process.

This effort is based on an extensive analysis of the key requirements and an in-depth review of the current state of the art.

The remainder of the paper is structured as follows; Section 2 introduces the problem of confidential data sharing with access audibility motivated by the industrial case of Noesis Solutions [29], whereas Section 3 establishes the functional and non-functional requirements. Sections 4, 5, and 6 then analyze and discuss the architectural variants and trade-offs for respectively blockchain data transfer and storage, file access control, and data validation. Section 7 concludes the architectural decisions and trade-offs discussed.

2 PROBLEM STATEMENT

The sharing of confidential data between industry partners involved in federated MDO, or a collaboration, is accompanied with a tremendous administrative stream which involves signing non-disclosure agreements (NDAs) and contracts at each step to respectively guarantee intellectual property and non-repudiation. The requirement of non-repudiation applies to designs and results delivered and consulted by all involved parties. In case of failure in the end product, the fault (e.g. in the airplane design) should be traceable to the level of individual parties without allowing any form of repudiation.

2.1 Current state of practice

Many individual solutions exist between the involved participants to support their sharing of data using signed private keys to ascertain the validity on the origin of data. Each company has individual information systems storing copies of the data, which leads to wide-spread fragmentation of data, which in turn hinders overall traceability and access management.

Furthermore, inter-organizational solutions pose the risk of either party claiming inconsistencies at their end, for example not having received or read a file.

In practice, trusted third parties serve as an intermediary to validate the access logs and confirm when the disputed data in fact has been accessed or modified and by whom.

This scheme is sub-optimal for three reasons: (i) the dependence on a third party may be costly and requires a level of trust of all involved parties in the third party, (ii) manual validation exposes confidential data to trusted third parties and leads to a complex administrative overhead, due to the required drawing up of legal contracts and NDAs, and (iii) since these are manual steps, the overall efficiency is reduced.

2.2 Ledger-based architecture

Decentralized shared ledgers (DLTs) have the potential to safely establish a tamper-proof record of events without repudiation through consensus-based integrity [33]. Furthermore, blockchain platforms increasingly feature logic executed in decentralized virtual machines (e.g. Ethereum VM [32]) which can be used to coordinate data exchange and access among parties.

The investigated blockchain architecture serves as a shared ledger to track and share information among engineering teams, eliminating data fragmentation, and enabling fault traceability.

In the context of the motivational case, this means that parties involved in the MDO process can observe and verify when parties comply with their obligations to each other.

In the following section, we outline the required functional requirements (e.g. data sharing), as well as the important non-functional requirements related to data confidentiality, access non-repudiation, availability and overall scalability.

3 REQUIREMENTS

We narrow the application objectives down into functional and non-functional requirements, eliciting the former first:

- **F1 Private data sharing with access control**. Data should be shareable among specific parties and remains only accessible to these designated parties.
- **F2 Logging data access with auditability.** Any operation on the file must be reported, including upload, download, changes and visualizations. Audit logs are only accessible among collaborating parties.
- **F3 Data validation and conflict resolution**. Ensure that the transferred file or the operation is correct, and if not, audit validation can be external, or autonomous, ideally while preserving data confidentiality.

Non-functional requirements:

- **NF1 Non-repudiation of data access.** It is important that no party can refute having seen, shared, accessed or modified data. These non-repudiation requirements when fulfilled can be used to back-up legal and signed commitments among the parties (e.g. to participate in the MDO process, or to maintain intellectual property rights).
- **NF2 Eliminating fragmented central systems and trusted parties** in which the cross-border collaborations store their data, and whom guarantee the integrity and confidentiality of the shared data.
- **NF3 Application scalability and availability** through for example off-chain storage [12], as results can be large and heterogeneous with replication for availability.

Not listed are less stringent requirements on preserving identity privacy during participation in the network, transactions or smart contracts [8, 19] applied in for example zkLedger [25] and Monero [31].

The architectural analysis is structured as follows; Section 4 discusses architectures for blockchain data transfer with, or without, storage (**F1**, **NF2**) while meeting with scalability and availability requirements (**NF3**). Section 5 discusses identity and file access control (**F1**), whereas Section 6 discusses data access auditability (**F2**, **NF1**), and data validation (**F3**).

4 BLOCKCHAIN DATA EXCHANGE ARCHITECTURES

We discuss three approaches for blockchain data exchange (**NF2**) in Section 4.1, followed by Section 4.2 discussing approaches for the decentralized management of stored data (e.g. assuring availability (**NF3**)).

The first data exchange architecture uses the blockchain to simply coordinate and record off-chain file transfers between parties.

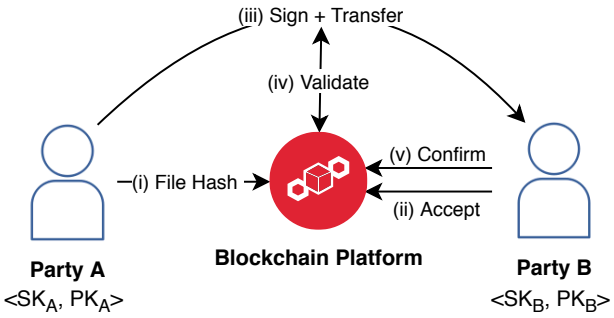

Figure 3: Coordinate an off-chain data transfer recorded on-chain.

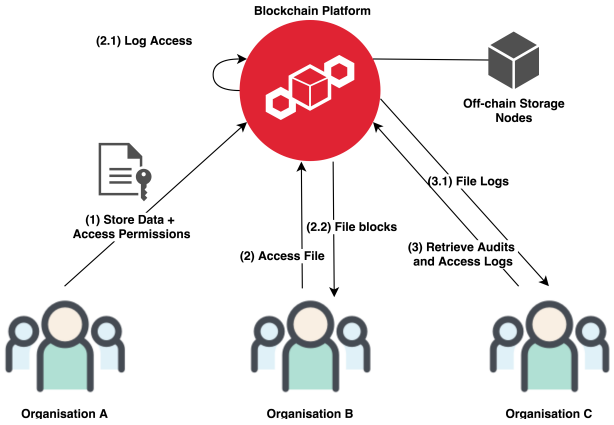

Figure 4: On-chain data transfer with self-governed decentralized storage.

The second and third approach handle the file transfer via the blockchain platform using respectively, its own peer-to-peer (P2P) storage nodes, or centralized cloud storage platforms. Each approach comes with certain challenges in terms of non-repudiation (**NF1**) and trustworthy logging of data access for auditability (**F2**), as well as trade-offs on performance, storage and network traffic overhead (**NF3**).

4.1 Blockchain data sharing architectures

4.1.1 Approach A: Off-chain data transfer and on-chain record. The first approach simply uses the blockchain for tamper-proof bookkeeping of data exchanges, of which the logs are externally created by the parties involved in the off-chain exchange process. It provides a solution with minimal adoption costs by not storing any of the data shared, except for file hashes and parties' digital signatures. The proposed tactic achieves this using the shared ledger as part of a handshake protocol to initiate a data transfer between parties.

Example. Figure 3 (i) depicts party A announcing to the blockchain application that it is ready to transfer data D_A to parties B and C. Party A places the file hash onto the blockchain permanently along with a list of intended parties, thereby announcing that it is ready to transfer a file. Next in Figure 3 (ii), party B accepts the transaction on chain, which is broadcast and stored permanently in a block. What follows is (iii) the transfer of data from A to B off-chain, the file is signed with party A's secret key SK_A, and can be verified using his public key PK_A by party B. Once party B has received the file, he or she can match the file against the hash published by party A on-chain (iv) and create a confirmation of receipt (v).

Cheatable data transfer. This approach still presents two cases where both parties can act maliciously [14]:

- Party B can receive the file without acknowledging receipt, or claiming that the data is incorrect.
- Party A can send rubbish data.

In the former case, the contents of a transferred file can be validated by a trusted third party, since the file is signed using A's public key, and by also establishing a session key on-chain. However, this approach exposes the potential confidential nature of the shared file (**F3**). Gazzoni et al. [14] discuss both aforementioned scenarios of malicious behavior, and a solution is proposed using third-parties without exposing the data by applying homomorphic hashing, the latter of which is exceeding costly in terms of performance.

These non-repudiation challenges cannot be solved when using off-chain transfer without trusted third-parties, therefore we discuss two alternative approaches in which the data transfer is handled by the blockchain itself.

4.1.2 Approach B: On-chain transfer and on-chain record. The blockchain platform can alternatively coordinate the data transfer itself and store the (pending) data on storage managed or hosted by the blockchain. Figure 4 depicts such an architecture. In Figure 4 (i), an engineering team places input data and files to initiate the MDO process, the data features access permissions and application-specific metadata. The file hash is placed onto the blockchain along with a location reference to its full contents stored onto blockchain-governed storage nodes, as is common in for example Siacoin [2] or Storj [1]. The blockchain application ensures data availability by replication and periodically applying a consensus-based proof-of-retrievability (PoR) algorithm [1, 2, 21, 27, 33].

The advantage of the approach shown in Figure 4 is that the blockchain platform handles the private data transfer, thus ensuring that files are in fact transferred between entities followed by trustworthy data log creation. Ultimately, if the transferred file is not correct, a third-party or smart contract can confirm its validity. In addition, the party which submitted a file will not be able to refute that this was the file shared.

Limited data fragmentation and malicious nodes. In this scenario, the file block hashes and their respective locations are public and recorded on-chain. Consequently a party can collect the required file blocks from the storage nodes himself. The storage nodes will have to verify whether the requesting party does indeed have appropriate access permissions. However, when the file is partitioned into a rather small number of blocks and fragmented among few nodes N_f, and the nodes act maliciously they can choose to omit access control and log creation on data access. Only when a file is partitioned across a large enough set of storage nodes (e.g. $N_f > (N/2)$), with N the total amount of storage nodes, can data access control and log creation be implemented with confidence. Consequently,

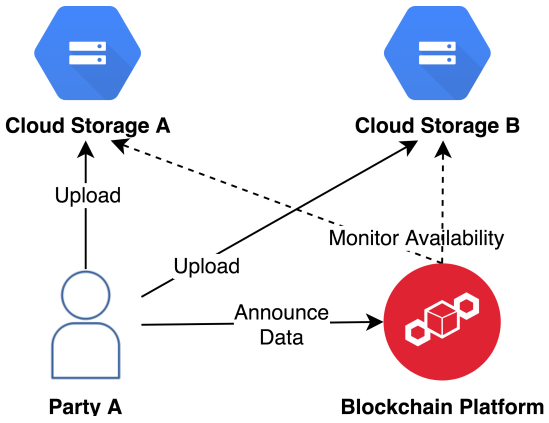

Figure 5: Blockchain application with governance of stored data on cloud storage platforms.

a downside may be that a data transfer will involve significant network traffic to numerous nodes.

Distributed blockchain data decryption. A solution allows minimal data partitioning by enforcing retrieval through consensus among the nodes. For example, the stored file blocks are encrypted client-side by the owner using a symmetric secret. The secret is then encrypted by a public key belonging to the blockchain network. The symmetric file key can only be decrypted after retrieval consensus which gathers private key shares required for decryption, followed by the distribution of these key shares to the requesting party. The requesting party can then decrypt the file's symmetric key using the blockchain's provided private key shares, similar to CALYPSO [18]. Multiple private key shares and respective public keys can be generated per party, or even per file to prevent re-use and compromise.

Either approaches have their trade-offs, and alternatively we present a solution using centralized storage.

4.1.3 Approach C: Blockchain governing cloud storage platforms. An alternative approach to using blockchain nodes for data storage and replication, as previously shown in Figure 4, the data itself can be stored on a cloud storage platform, as shown in Figure 5. In such a scenario, the blockchain application will have to ensure that the files are potentially replicated across multiple cloud platforms to ensure availability (i.e. multi-cloud storage). Cloud storage platforms are inherently more available than P2P storage nodes managed by the blockchain, therefore requiring less file replication and thus minimizing storage overhead.

In dealing with the data access log challenge, potentially the cloud storage platforms are trusted enough to incorporate such data access and audit log creations for the blockchain application. However, the blockchain platform can also impose its own file access control requiring a retrieval consensus for decryption, as previously briefly discussed.

4.1.4 Summary data exchange approaches. We have shown three approaches to implement data exchange using blockchain, Table 1 elicits the access non-repudiation trade-offs for each approach, whereas Table 2 elicits storage and network considerations. The

Storage	**Access non-repudiation.**
Off-chain transfer	Requires 3rd-party for non-repudiation of file transfer and receipt.
On-chain transfer	
- Decentralized nodes	Requires partitioning or decryption consensus among ($n > N/2$) for access non-repudiation.
- Centralized storage	Requires decryption consensus for retrieval ($n > N/2$).

Table 1: Storage architectures and access non-repudiation trade-offs.

Storage	**Trade-offs**
Off-chain	Saves storage space and network traffic. No distributed preservation.
Peer-to-peer	Significant data duplication for availability and cross-node traffic.
BitTorrent/IPFS [6]	Unincentivized storage by P2P community risks availability.
Blockchain storage	Participation reward (e.g. Filecoin [21])
Centralized	
Cloud storage	Minimal duplication due to inherent high availability. Low latency.

Table 2: Storage architectures and considerations.

first approach **A** simply uses the blockchain application to coordinate and log a data transfer. However, as shown there are major challenges on assuring that the data was in fact sent or received off-chain.

Alternatively to solve these issues of non-repudiation, the data can be stored by either approach **B** in a decentralized manner on storage nodes operated by the blockchain, or approach **C** centralized storage platforms managed by the blockchain. Storing the data in a decentralized P2P storage system can require significant network traffic to establish consensus on retrieval among a sufficient set of nodes for trustworthy access logging. Alternatively, less network traffic is involved when consensus is required to decrypt files, allowing for fewer distributed file blocks, albeit still replicated for availability. The latter approach **C** solves this hurdle by trusting a number of cloud storage platforms which are inherently highly available resulting in less storage overhead.

4.2 Decentralized management of storage

Not only files can be shared using the blockchain, but also transaction history can be recorded regarding on-going application-specific (e.g. MDO) processes, the parties involved, and the results exchanged in the respective process. Typically, data stored on-chain are small key-value pairs, for example log records, access permissions, or executable smart contracts.

Large files are best kept off-chain to achieve scalability (**NF3**) with the file hash and its location stored on-chain to verify file integrity and maintain availability [12]. In practice, Siacoin [2], Storj [1] and Filecoin [21] allow users to store files using the blockchain in an encrypted and replicated manner. The file is partitioned into multiple segments, similarly to the BitTorrent [10] protocol, and a hash of the segment is created. File block hashes are stored onto the blockchain, whereas the actual blocks are stored onto multiple nodes in an encrypted format. The nodes can be storage nodes of the blockchain network itself, or for example a P2P file system such as IPFS [6]. However, there is no financial incentive to store files in IPFS as opposed to blockchain storage solutions such as Filecoin [21] which reward data hosts with digital currency. File hosting participants are typically required to lock currency in an escrow account on-chain, which is lost when the participants lose a file block replica. This escrow serves as a deterrent for suddenly leaving the P2P storage network.

Blockchain storage solutions actively apply a proof-of-retrievability (PoR) consensus algorithm to ensure availability of the replicated file blocks across the storage nodes [27]. If a replica disappears, it is duplicated in time to an available storage platform. The PoR-algorithm does not require the entire file to be transferred and extends upon provable data possession (PDP) algorithms which samples portions of the data and establishes a proof that an untrusted storage node holds the data [3, 17].

In the next section we look at implementation challenges and strategies concerning identity management and file access control.

5 FILE ACCESS CONTROL

Next to storage architectures, there are several architectural decisions to implement (decentralized) data access control. Such access control mechanisms allows for sharing private data (**F1**) and attaining auditability for data access operations (**NF1**, **F2**). For example, not every company should have access to the shared designs as these may be confidential and business critical to individual participants. In terms of auditability, we require file access control as otherwise data can be retrieved directly since file hashes and locations are publicly recorded on-chain.

5.1 Confidential data sharing

As it stands, contemporary blockchain storage platforms, such as Siacoin [2], Storj [1], and Filecoin [21] do not allow private file sharing among multiple parties. However, sharing access can naively be done by the exchange of a symmetric key used to encrypt the file. A simple private data sharing approach applied in practice encrypts data or session keys once per entity using its respective public key [34]. These identities are typically the public keys themselves in for example Bitcoin [24] and Ethereum [32].

The work by Zyskind et al. [35] describes a decentralized data management system which shares access to data hosted by blockchain nodes. Whenever data is shared, the access permissions are recorded on-chain for the specific parties. In addition, the session key for a file can also be recorded publicly, however encrypted with permitted parties' public keys. Data retrieval is allowed after a party sends its digital signature to the network, which can then create an audit log and provide the data when permitted. Alternatively to sharing access to a symmetric key, in Shafagh et al. [28] a tactic is implemented which shares private cloud data after proxy re-encryption for the intended party. In essence, a data owner can generate a re-encryption key based on another party's public key, which allows encrypted files intended for the data owner to become decryptable by another party after re-encryption without exposing private keys [4].

5.2 Permission- and identity-based data access control

Considering the enterprise aspect of the application and its confidentiality requirements (**F1**), ideally the blockchain platform is permission-based, such as HyperLedger [7], instead of open platforms such as Bitcoin [24] and Ethereum [32].

Identity management. Only designated identities can join and participate in the blockchain network, which in our case can be coordinated off-chain since the users are involved in an enterprise collaboration. Ideally, data should only be visible on the blockchain to those who have access to it or are involved in the policy.

Permission-based access. Along with the data stored there has to be a permission-based access control mechanism which governs data access. These permissions can be formulated as expressive policies [23], or simply a list of read and write permissions for each identity. These permissions, along with the access control logic, can be handled on- and off-chain in a complementary fashion.

5.2.1 Off-chain encryption. Parties can be granted file access by encrypting an additional file copy using its public key. Parties are removed by removing their respective encrypted file copies, however once the party has had access to the file it may have its own local copy. A potential problem with the off-chain approach is that it requires trust in the person who encrypted the data to be compliant with the metadata of published permissions. Furthermore, creating additional copies creates storage overhead, however a single file can be encrypted for all parties, or ideally only the symmetric session key. In addition, off-chain encryption by itself does not prevent data access without audit log creation.

5.2.2 On-chain file access control. There are two approaches to implement decentralized file access control; either when using decentralized storage and fragmenting the file sufficiently in N_f partitions, or by requiring decryption by the blockchain to read a file. In the first approach, a file can only be retrieved if all of its N_f fragments can be retrieved from $N_f * N_r$ nodes, of which N_r is the replication factor. The consulted nodes can then after consensus generate a data access log recorded on-chain. In the second approach, on-chain decryption can be implemented by additional encryption via a public-private key belonging to the blockchain itself via distributed key generation (DKG) [16, 18]. The private key can be fragmented into key shares and distributed among the nodes, which can validate that the shares are in fact correct by using publicly verifiable secret sharing (PVSS) schemes [18, 26] At the time of file upload, the file or its symmetric key is encrypted using the blockchain's public key. The decrypted data can be retrieved after permission consensus, after which the nodes provide

a sufficient number of n out of m private key shares to the party requesting retrieval if it indeed has permission-based access. Such an approach is applied in CALYPSO [18], a blockchain for sharing encrypted data with auditability. Trade-offs can be made between performance and the level of trust in the network depending on the ratio between $N_f * N_r$ and the number of nodes N, or similarly to the required number of keys N_k and the total nodes N.

5.3 Dynamic file access permissions

Access permissions can also change when parties leave or join the file access group (or policy) over time, requiring an update to the listed permissions on-chain, as well as altering any original encrypted files or session keys if needed to prevent unauthorized access. It is best practice to also encrypt files client-side by the owner or parties' shared keys to prevent unwarranted access by the network in case of malicious network collusion, however this also hinders dynamic file access to a degree as we will show.

5.3.1 Access for new members. Suppose a new party is granted access to a given file, we can update the permissions and naively either re-encrypt the file or its session key with the party's public key, which requires network traffic to update the files. However, we can also leave existing files unaltered via proxy re-encryption [4]; we can share the data owner's public key encrypted files by letting the owner generate a re-encryption key based on its private key and the new member's public key, which allows decryption of files intended originally for e.g. A by the new member. Parties retain data access by using their original re-encryption keys while granting a new member instant access. The proxy re-encryption keys are preferably held by the blockchain network to easily revoke access at a later point in time.

5.3.2 Access revocation. In this case when using proxy re-encryption, the re-encryption key intended for the party who's access is being revoked simply has to be forgotten by the network. Alternatively, if there's a single re-encryption key for all parties in the access policy, this key will have to be renewed excluding the revoked party, a procedure which requires the data owner to be online. Removing the key is enforceable if it is partitioned sufficiently enough across N_f key management nodes to prevent malicious network collusion. Furthermore, it is best practice to use multiple key pairs, ideally one for each file fragment, file, or one for each group. It is important to also re-encrypt stored data appropriately when a user's access is revoked and has previously obtained keys to unread files, or the corresponding encrypted symmetric file keys. Table 3 summarizes the file access control approaches and their trade-offs on file duplication and fragmentation versus consensus to (proxy) decrypt and distributed key management.

5.3.3 Sufficient fragmentation. On-chain file decryption for access non-repudiation does not need to be applied when a file is sufficiently fragmented across nodes $N_f * N_r$ in comparison to N, which verify access permissions on-chain. The file can then be encrypted using a session key, encrypted by the parties' public key, which does however require still altering the key files when a party joins or leaves.

Static access	**Trade-offs**
Enc. PK_{users}	Requires high N_f of single file for access non-repudiation.
Enc PK_{users} + PK_{net}	Requires low N_f and high N_k of file with consensus for proxy decryption and access non-repudiation.
Dynamic access	
Enc. PK_{users}	Requires re-encryption of file blocks with high N_f against collusion.
Enc. PK_{owner} + Proxy decrypt	Requires high N_k and consensus to decrypt, ideally key-pair per file or fragment, however low N_f.

Table 3: Encryption methods for file access control.

6 PRIVATE DATA AUDITABILITY

Auditability is comprised of (a) publicly verifiable data correctness and (b) private data access auditing. Public verification is a requirement as to allow the network to establish consensus on correctness at, for example, the levels of: (i) shared confidential data correctness, (ii) integrity auditing, however also (iii) validating or enforcing a correct sequence of application-specific operations. The verification process ideally does not expose any confidential properties during the evaluation.

In terms of data correctness, we already discussed for the off-chain transfer case that malicious behavior is possible as party B can claim to have received no file or an incorrect file, or party A did not actually send a file at all. In terms of data integrity auditing, file blocks may be compromised due to untrusted nodes, or even by the sender when confidential files in an unencrypted format do not match the hashes recorded on-chain.

6.1 Auditability and log creation

Traditionally, external trusted third parties are assigned as auditors which gain access to the audit logs. However, in this case we can also let the collaborating users gain access to the audit logs, which can be decrypted using private key shares as to govern audit log access control in a decentralized manner.

6.1.1 Data access auditability. File access control enforced by the blockchain enables automatic log creation whenever someone accesses or alters data stored by the blockchain. In the decentralized storage approach, file access can also be checked and logged by individual storage nodes for their respective partitions. However, the latter approach requires sufficient partitioning to attain trustworthy data access log creation. Cloud storage managed by the blockchain for file transfer does however require decentralized file access control on, an in fact, centralized storage for trustworthy audit log creation. Alternatively when the centralized storage platform is fully trusted it can also be allowed to generate audit logs on-chain.

6.1.2 Public data integrity auditing. Typical data auditing solutions focus on assuring that cloud storage platforms preserve the integrity of files hosted [15, 30, 34]. These files hashes can be stored

Validation tactic	Trade-offs
- Trusted parties	Exposes confidential data to trusted intermediaries. Manual intervention is costly and inefficient.
- Trusted environment	Sends data to trusted execution. Prone to side-channel attacks.
- Partial Homomorphic Encryption (PHE)	Can only be applied on integers and is somewhat efficient.
- Fully Homomorphic Encryption (FHE)	FHE signatures with public verification are very inefficient [14].

Table 4: Public validation of private data.

in a tamper-proof manner using the blockchain. However, publicly verifying the correctness of encrypted private data poses several challenges, for example when trying to match it with the unencrypted file's hash recorded on-chain.

6.2 Private data validation

Data exchanged between parties can in certain cases be incorrect when the sender sends a wrong file and disputes arise. In such cases, automated or external resolution is required. In a worst-case scenario the collaboration process is simply stopped. However, typically this is solved by exposing the actual contents of the confidential shared data to a trusted third party, e.g. a group manager or a trusted execution environment.

6.2.1 Verification by sharing data access. ProvChain [22] is a blockchain application which stores the history of operations on data objects, and the lineage of these operations, typically referred to as data provenance records. ProvChain allows auditors to gain access to encrypted meta data on-chain for the verification of data provenance records.

Trusted platform execution environments (e.g. Intel® SGX) can also receive the private key and the encrypted data thereby allowing the trusted execution environment to verify the file's hash recorded on-chain. This file hash can either reflect the hash of the unencrypted or encrypted file. The latter which can always be checked without data leakage. Trusted execution can also serve to create trusted audit logs on data access in for example untrusted cloud storage environments [30]. Such execution environments however are not flawless and can be prone to side-channel attacks.

6.2.2 Verification while preserving data confidentiality. Recent techniques elicited in Table 4 allow public validation of private data without affecting its confidential properties, for example via homomorphic hashing and authenticators [14, 17], which however remains rather impractical due to high computation costs. Other solutions apply partially homomorphic encryption (PHE) to validate encrypted numeric data (e.g. meta-data) from the application workflow without exposing the private contents [11]. Contemporary blockchain solutions, such as zkLedger [25] and Monero [31], apply PHE and zero-knowledge proofs to assure that for example the output of currency spent is less or equal than the sum of received input in previous transactions. Such algorithms work efficiently for encrypted data holding numerical information, whereas for example homomorphic file integrity validation is highly inefficient and currently impractical [14]. Alternatively within our case, private data correctness can also be verified simply by exposing the confidential data to other users involved in the collaboration by revealing party A and B's recorded session key, and granting access to the specific file copy.

6.3 Transaction and private meta-data validation

Data access logs also present the option to validate and control the sequence of operations as they are executed. This requires implementing application-specific rules on the order of transaction and operations depending on the current state of the chain. For example, read operations may only be executed when a file in fact exists, or within the MDO case a party may not upload output data twice without receiving any new input data, as it becomes ambiguous to what data is now intended as input for the next iteration step. Such logic can be implemented using smart contracts on-chain, which can interpret meta-data from previous transactions to determine (i) the MDO process ID, and (ii) previous MDO results and thus the overall process state.

Such application-specific rules can be implemented within smart contracts to for example validate meta-data that links outputs to inputs. Such meta-data validation can be implemented in smart contracts to preemptively detect any run-time errors by the network. However, meta-data can also be encrypted to for example hide the order of application-specific operations. These smart contracts can then apply homomorphic encryption to validate meta-data without exposing the data contents and lineage.

In the case of encrypted meta-data, Raziel [8] and Hawk [19] enable secure multi-party execution for smart contracts, which allows code execution on encrypted inputs while keeping these inputs confidential. However, computation costs increase depending on the number of involved nodes, the complexity of the algorithm, and the required security level, and therefore such techniques are more suitable for permission-based ledgers.

7 CONCLUSION

We have systematically discussed the main architectural decisions and trade-offs involved in the design of a decentralized platform for managing and governing access to private encrypted data. By leveraging distributed ledger technology (DLT) and smart contracts, guarantees can be provided in terms of data availability and consistency, and by employing consensus-based protocols audit and access logs can be created that are non-repudiable. Such an architecture is highly compelling as it allows handing over control of a valuable files (in an encrypted fashion) to a decentralized platform while preserving its confidentiality, managing file access, and in the end eliminating costly trusted third parties in auditing processes.

Regarding file storage, we argue in favor of either a strongly decentralized P2P infrastructure, or a more centralized deployment (e.g. based on multi-cloud storage) that is managed by the blockchain. Without the requirement for access non-repudiation, centralized storage systems are a clear winner in terms of fast latency, and minimal data duplication or fragmentation due to inherent high availability.

We have identified complex trade-offs to avoid network collusion by either fragmenting files in a P2P setup, or increasing the required nodes to decrypt a file block by combining private key shares. Such decentralized decryption requires careful file and key management over time as parties leave and join the access policies. In addition, file deletion after access revocation is not immediate in a P2P environment and nodes are volatile and key refreshment protocols are required.

Auditability also applies to auditing the validity of private data and transactions, and this involves meta-data such as the order of application-specific transactions (e.g. first upload, then download). Our study highlights that private data validation can be achieved using third-party verifiers, trusted execution, but also by leveraging techniques such as homomorphic encryption and zero-knowledge proofs. However, the performance and practical applicability of such approaches are currently limited to the applied algorithm and encrypted data.

The main contribution of our study is the systematic exploration of approaches to accomplish auditability and decentralized private data access control which is especially relevant in the context of federated collaborations. In future work, we will further investigate and quantify the nature of these trade-off through empirical evaluation. Additionally, we aim to work towards solutions to allow dynamically exploiting these trade-offs between data duplication, file and key fragmentation, and the achieved levels of trust and performance.

ACKNOWLEDGMENTS

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

REFERENCES

[1] 2018. Decentralized Cloud Storage – Storj. https://storj.io/. Accessed: 2018-12-10.
[2] 2018. Sia - Cloud storage. https://sia.tech/. Accessed: 2018-12-10.
[3] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. 2007. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 598–609.
[4] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. 2006. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 1–30.
[5] Ahmed H. Bayoumy. 2018. Multi-fidelity model management framework for multidisciplinary design analysis and optimization. Retrieved 2018-12-10 from http://sol.research.mcgill.ca/research.html
[6] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* abs/1407.3561 (2014). arXiv:1407.3561 http://arxiv.org/abs/1407.3561
[7] Christian Cachin. 2016. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.
[8] David Cerezo Sánchez. 2018. Raziel: Private and Verifiable Smart Contracts on Blockchains. *ArXiv e-prints*, Article arXiv:1807.09484 (July 2018), arXiv:1807.09484 pages. arXiv:cs.CR/1807.09484
[9] Shamsheer S. Chauhan, John T. Hwang, and Joaquim R. R. A. Martins. 2018. An automated selection algorithm for nonlinear solvers in MDO. *Structural and Multidisciplinary Optimization* (June 2018). https://doi.org/10.1007/s00158-018-2004-5
[10] Bram Cohen. 2008. The BitTorrent protocol specification.
[11] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. 2012. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*. Springer, 643–662.
[12] Jacob Eberhardt and Stefan Tai. 2017. On or off the blockchain? Insights on off-chaining computation and data. In *European Conference on Service-Oriented and Cloud Computing*. Springer, 3–15.
[13] Michael Egorov, MacLane Wilkison, and David Nuñez. 2017. Nucypher KMS: decentralized key management system. *arXiv preprint arXiv:1707.06140* (2017).
[14] 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
[15] Anmin Fu, Shui Yu, Yuqing Zhang, Huaqun Wang, and Chanying Huang. 2017. NPP: a new privacy-aware public auditing scheme for cloud data sharing with group users. *IEEE Transactions on Big Data* (2017).
[16] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In *Advances in Cryptology — EUROCRYPT '99*, Jacques Stern (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 295–310.
[17] Longxia Huang, Gongxuan Zhang, and Anmin Fu. 2018. Privacy-Preserving Public Auditing for Non-manager Group Shared Data. *Wireless Personal Communications* 100, 4 (01 Jun 2018), 1277–1294. https://doi.org/10.1007/s11277-018-5634-4
[18] 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. (2018).
[19] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. 2016. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 839–858.
[20] 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*.
[21] Protocol Labs. 2017. Filecoin: A Decentralized Storage Network. https://filecoin.io/filecoin.pdf.
[22] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. 2017. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 468–477.
[23] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2017. Blockchain based access control. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 206–220.
[24] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved 2018-12-07 from https://bitcoin.org/bitcoin.pdf
[25] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*.
[26] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*. Springer, 148–164.
[27] Hovav Shacham and Brent Waters. 2008. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 90–107.
[28] Hossein Shafagh, Anwar Hithnawi, Lukas Burkhalter, Pascal Fischli, and Simon Duquennoy. 2017. Secure Sharing of Partially Homomorphic Encrypted IoT Data. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM.
[29] Noesis Solutions. 2018. Software Solutions that enable Objectives Based Engineering. https://www.noesissolutions.com/
[30] D. Song, E. Shi, I. Fischer, and U. Shankar. 2012. Cloud Data Protection for the Masses. *Computer* 45, 1 (Jan 2012), 39–45. https://doi.org/10.1109/MC.2012.1
[31] Nicolas van Saberhagen. 2013. Cryptonote v 2.0. Retrieved 2018-12-07 from https://cryptonote.org/whitepaper.pdf
[32] Gavin Wood. 2014. Ethereum: a secure decentralised generalised transaction ledger. (Yellow Paper).
[33] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba. 2017. A Taxonomy of Blockchain-Based Systems for Architecture Design. In *2017 IEEE International Conference on Software Architecture (ICSA)*. IEEE, 243–252. https://doi.org/10.1109/ICSA.2017.33
[34] Lei Zhou, Anmin Fu, Shui Yu, Mang Su, and Boyu Kuang. 2018. Data integrity verification of the outsourced big data in the cloud environment: A survey. *Journal of Network and Computer Applications* 122 (2018), 1 – 15. https://doi.org/10.1016/j.jnca.2018.08.003
[35] 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