Journal of Systems Architecture 150 (2024) 103133

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

BBS: A secure and autonomous blockchain-based big-data sharing system[☆]

Shan Wang[a,b], Ming Yang[a,*], Shan Jiang[b,*], Fei Chen[c], Yue Zhang[d], Xinwen Fu[e]

[a] *Southeast University, China*
[b] *The Hong Kong Polytechnic University, Hong Kong*
[c] *Shenzhen University, China*
[d] *Drexel University, United States of America*
[e] *University of Massachusetts Lowell, United States of America*

ARTICLE INFO

Keywords:
Blockchain
Big data sharing
Off-state
Hyperledger fabric
Enterprise ethereum

ABSTRACT

Chain of custody is needed to document the sequence of custody of sensitive big data such as various healthcare data. In this work, we propose a secure and autonomous big data sharing system (BBS) based on a permissioned blockchain. In our system, we refer to the data stored at a blockchain node but outside the ledger for sharing as "off-state", such as the large amounts of data (called "big data") that cannot be stored directly in a blockchain ledger in practice. To securely and autonomously share the off-state data, we further design an off-state sharing protocol. In the proposed protocol, a sender registers the big data with BBS for sharing. To acquire the big data, an authenticated and authorized receiver has to propose transactions and interact with BBS in five phases, including data transfer authorization, data transfer, two-time key request, and data decryption. The key advantage of BBS is its ability to defeat dishonest users and autonomously perform the aforementioned five phases. The corresponding transactions are recorded in the ledger, serving as chain of custody evidences that document the data trail. To demonstrate the effectiveness of our proposed system, we have designed and implemented prototypes of BBS for sharing big files over two representative permissioned blockchains, i.e., Hyperledger Fabric and Enterprise Ethereum. Extensive experiments are performed to validate the feasibility and performance of our system.

1. Introduction

A consortium of organizations such as hospitals and research institutions may need to share sensitive data such as biomedical and scientific data with each other free of charge for better cooperation and scientific discovery. The data sharing brings an attractive value to disease diagnosis, therapeutic regimens, and precision medicine [1]. However, the shared data may be sensitive and large in volumes, such as videos, diffusion MRI images [2], and confidential files. For instance, numerous diffusion studies can yield datasets of several gigabytes for a single patient. Due to growing concerns regarding intellectual property theft and industrial espionage [3,4], it is imperative to develop a secure and reliable "big data" sharing system that incorporates a record of the chain of custody [5]. This record will serve to document the complete trail of the data, including information on who requests the data and who owns it. Centralized systems for data sharing often suffer from issues such as single point of failure, data abuse, and privacy violations [1,6,7]. Therefore, a decentralized data sharing system is highly demanded.

Blockchain technology is a promising solution to establish the chain of custody of the shared sensitive data without a central authority, and can preserve non-repudiable evidence [8]. In conventional blockchain systems, data is typically stored in a ledger that consists of a world state and a blockchain. The ledger is synchronized across all blockchain nodes. The world state maintains the current system state, which includes information like the user's cryptocurrency balance in Bitcoin [9], while the blockchain records the entire transaction history. This transaction history contains the operations performed on the world state and the data required to update the world state. A smart contract, i.e., a decentralized application running over a blockchain system, controls operations on the world state and is usually treated as a trustworthy third-party arbiter. The complete transaction history generates the current state values and can serve as evidence for auditing purposes.

☆ An earlier version of this paper entitled "BBS: A Blockchain Big-Data Sharing System" is accepted to IEEE International Conference on Communications (ICC) 2022.
* Corresponding authors.
E-mail addresses: shan-cs.wang@polyu.edu.hk (S. Wang), yangming2002@seu.edu.cn (M. Yang), cs-shan.jiang@polyu.edu.hk (S. Jiang), fchen@szu.edu.cn (F. Chen), yz899@drexel.edu (Y. Zhang), Xinwen_Fu@uml.edu (X. Fu).

https://doi.org/10.1016/j.sysarc.2024.103133
Received 21 June 2023; Received in revised form 21 February 2024; Accepted 25 March 2024
Available online 27 March 2024

However, existing blockchain frameworks cannot be directly applied to big data sharing. This is primarily because existing frameworks impose limitations on ledger data size and data types, driven by factors such as transaction fees, system performance, and other concerns [10–13]. Traditionally, the ledger is designed to store state data, such as cryptocurrency balances. It is impractical to store big files in the world state database or use transactions to carry big files across the blockchain network. Furthermore, all blockchain nodes typically maintain the same ledger data. However, due to privacy and intellectual property concerns, data owners may be reluctant to share large-scale data across all nodes. Sharing data within ledgers presents storage and privacy challenges for big data sharing.

Off-chain schemes have been proposed for blockchain-related big data sharing systems. However, these off-chain based schemes are inappropriate for our purposes, especially for establishing the chain of custody, for the following reasons. First, some off-chain schemes are proposed for big data selling applications over public blockchains [14–17] such as FairSwap [16]. In these systems, users must pay transaction fees to miners for smart contract execution or pay fees to data owners for data exchange. They are not specifically designed for facilitating *free* data sharing applications, which is the objective we aim to achieve. Second, data transfer in those big data selling systems is often performed off-chain, not as an integral part of the blockchain system, and cannot be executed autonomously. Third, in the big data selling scheme, the encryption key is revealed to a public blockchain. The revealed key may incur the temporary censorship attack [18]. If an encrypted file is leaked, such as being intercepted by malicious cyber actors, there is a risk that the encrypted file may be decrypted by adversaries. Fourth, in [19–21], a blockchain module is used to manage data access tokens for data sharing, and users use the token to access data off-chain. The blockchain system has no control over the actual data transfer and cannot know the status of the data transfer. A dishonest sender may not share the data with the receiver, and a dishonest receiver may falsely claim that he/she does not receive the data.

In this paper, we design a novel blockchain-based big data sharing system (BBS) which can autonomously and securely build the chain of custody of the shared data without privacy issues. We introduce a novel "off-state" data sharing system model and protocol, which can concurrently address the storage, privacy, autonomy, and security issues in existing on-chain and off-chain data sharing schemes. Our BBS system is built upon a permissioned blockchain so as to freely share sensitive big data with authenticated and authorized users, and manage varied types of big data. Our major contributions are summarized as follows.

- We introduce the concept of "off-state", which is data, particularly big data, maintained at a separate storage space to distinguish from the ledger at blockchain nodes. Off-states can be shared between parties of interest and do not need to be synchronized across all nodes for privacy considerations. A trustworthy smart contract can directly operate on off-state data, such as sharing, so that the blockchain system can autonomously control the data sharing process and record all necessary evidence in ledgers.
- We propose a novel off-state sharing protocol utilizing the smart contract for autonomous data management and securing the chain of custody. During big data sharing, a receiver only interacts with the system and does not need to involve the sender. Users must interact with the blockchain system through transactions, which document the chain of custody. Dishonest receivers cannot deny that they obtained the original data since the transaction history in the blockchain provides non-repudiable evidence.
- We implement a prototypical BBS running our off-state data sharing protocol for big file sharing over two mainstream permissioned blockchain paradigms, i.e., "execute-order-validate" and "order-execute", used by Hyperledger Fabric [22] (denoted as *Fabric*), and Enterprise Ethereum [23] respectively. This demonstrates that our protocol is general. We present our protocol over Fabric in detail and discuss the Enterprise Ethereum version in Section 8. Extensive experiments are performed to evaluate the feasibility and performance of BBS, such as the latency of big file sharing between pairs of blockchain nodes. BBS over Fabric performs similarly or better compared to secure file transfer applications *SFTP/SCP* in Linux in terms of performance.

Table 1
Comparison to existing work.

Schemes	Storage	Privacy	Security	Autonomy
On-chain [24–29]	✗	✗	✓	✓
Off-chain [14–17,19–21]	✓	✓	✗	✗
Off-state (ours)	✓	✓	✓	✓

Note: The symbol ✓ indicates that the corresponding property is achieved, while the symbol ✗ indicates that the corresponding property is not achieved.

The rest of this paper is organized as follows. The related work is presented in Section 2. Section 3 introduces two representative permissioned blockchain frameworks and their permission mechanisms. Section 4 presents the design goals, system model, and threat model. Section 5 proposes a secure off-state sharing protocol and presents the protocol details. The security analysis of the protocol is presented in Section 6. We evaluate the prototypical BBS based on Fabric in Section 7. Section 8 discusses the BBS system over *Enterprise Ethereum* and shows that the proposed protocol is general. Section 9 concludes this work.

2. Related work

In this section, we review related work on data sharing pertaining to blockchain.

In some work, blockchain systems are employed to store, manage, and share data such as biomedical data within ledgers. In [24–29], smart contracts are utilized to control the data access. Users interact with the system by proposing transactions to set or retrieve shared data. Transactions carry the shared data. However, sharing big data in ledgers is impractical due to ledger storage limitations and privacy concerns.

In [14–17], blockchain systems are designed for selling big data and charging users in cryptocurrency. For example, FairSwap [16,17] is a digital goods selling protocol. It is unsuitable for a free or autonomous big data sharing system for three reasons. First, FairSwap sells big files in Ethereum. Users need to pay fees to miners for smart contract execution and pay for requested data. It is not designed for *free* data sharing applications for the purpose of scientific discovery. Second, in the FairSwap protocol, users are required to transfer files off-chain and perform encryption/decryption, while the blockchain system handles tasks such as cryptocurrency transfer and encryption key exchange. Therefore, FairSwap segregates file transfer from the blockchain system. In contrast, file transfer is an integrated part of our BBS, and can be executed autonomously. Third, FairSwap operates on a public blockchain, where the encryption key is openly disclosed on the ledger. This means that if an encrypted file is compromised, such as being intercepted by adversaries, there is a possibility for the encrypted file to be decrypted using the publicly available encryption key. Our system securely manages the encryption key by a data isolation mechanism, and is not revealed to the public. DataTBC [14] and FaDe [15] have similar drawbacks to FairSwap. Revealing the encryption key by FairSwap to a public blockchain may incur the temporary censorship attack [18], in which a transaction is censored by malicious miners and cannot be committed to the blockchain, but the key has been revealed.

The blockchain modules in [19–21] utilize the blockchain system to manage data access tokens. The data is stored outside of the blockchain.

Users utilize those tokens to access data off-chain. However, they overlook security issues such as dishonest users. For example, in [19,21], a dishonest sender may refuse to share the required data when the receiver requests the data off-chain. In [20], encrypted data is stored in a decentralized file storage system InterPlanetary File System (IPFS), and a blockchain system is utilized to manage the encryption key. A dishonest sender may upload the wrong requested data to the IPFS or share a wrong encryption key, so as to refuse to share the data. In [20], users have to off-chain interact with each other. The system is not autonomous. The aforementioned blockchain systems as well as many others [30–33] lack control over the actual data sharing process and are unable to know the status of data sharing, e.g., whether the data is actually shared. Most of such related work often misses a detailed description of the sharing protocol.

As summarized in Table 1, existing blockchain-based data sharing schemes can be categorized into on-chain schemes and off-chain schemes. On-chain schemes often suffer from storage and privacy issues, while off-chain schemes usually encounter security and autonomy challenges. In comparison, our off-state scheme can address all of these issues, resulting in a secure and autonomous data sharing protocol for big data without compromising privacy.

Furthermore, an earlier version of this paper [34] introduces an off-state scheme. However, the earlier version scheme assumes that one organization only has one user who represents that organization. It only allows data sharing among organizations not among users. This paper extends the system model and generalizes the data sharing protocol, where each organization can set multiple users and users can share data within one organization or across different organizations. This paper also formalizes the security of the proposed protocol in detail, and provides an Enterprise Ethereum version of BBS which demonstrates the generality of the proposed protocol.

3. Background

In this section, we introduce two mainstream permissioned blockchain paradigms, i.e., “execute-order-validate” and “order-execute”, and the related built-in permission mechanisms for access control and data confidentiality. A permissioned blockchain framework is more suitable for our purposes than a public blockchain, because a permissioned blockchain involves multiple permission mechanisms for security and privacy purposes, and does not involve transaction fees.

3.1. Permissioned blockchain paradigms

Permissioned blockchains mainly follow two types of paradigms, i.e., execute-order-validate and order-execute. In the execute-order-validate paradigm, a transaction is executed before ordering the transaction into a block. In the order-execute paradigm, a transaction is executed after transaction ordering.

3.1.1. Execute-order-validate

In the execute-order-validate blockchain architecture, smart contracts execute before transaction ordering. The representative framework is Hyperledger Fabric. The “execute-order-validate” paradigm involves three types of nodes, i.e., peers, orderers, and clients, each responsible for specific tasks in the transaction workflow. Peers are responsible for maintaining ledgers and smart contracts, and serve as the backbone of the blockchain network. Orderers are responsible for block generation. Clients are the users of the blockchain system.

Fig. 1 illustrates the “execute-order-validate” transaction workflow. (Steps 1–2) A client/user signs and initiates a transaction *proposal* which is then sent to a subset of peers called endorsers. The endorsers are specified by an endorsement policy; (Steps 3–6) Each endorser independently executes the chaincode (i.e., smart contract in Fabric), and signs the execution results as an endorsement. The signed results are then returned to the client as the proposal response. It is important to note that the execution results are not yet updated to the world state at this stage; (Steps 7–8) If the execution results obtained from different endorsers are consistent, *the client* has the option to construct and submit a transaction. This transaction includes the transaction proposal, execution results, and a list of signatures on the execution results as endorsements. The client then sends this transaction to the orderer nodes for further processing. (Steps 9–12) The orderer nodes bundle collected transactions into a new block following a consensus protocol such as *Raft*, and distribute the new block to all peers including both endorsers and non-endorsers; (Steps 13–15) All peers validate the transactions in the received new block. For each transaction, if its endorsements meet the endorsement policy check and its execution results pass the version conflict check, each peer updates the world state according to execution results. After validating all transactions, each peer appends the new block to its local blockchain; (Step 16) Finally, a peer will notify the client if the transaction initiated by the client has been successfully committed.

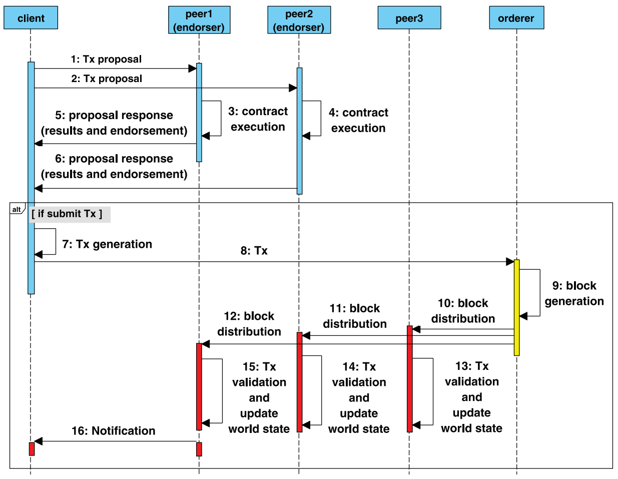


Fig. 1. “Execute-order-validate” transaction workflow. “Tx” is the abbreviation of “Transaction”.

3.1.2. Order-execute

In the order-execute blockchain architecture, smart contracts execute after the transaction ordering. The representative framework is Enterprise Ethereum [23]. The “order-execute” architecture involves two types of nodes, i.e., client and full node. Full nodes maintain ledgers, run the smart contract, generate blocks and form the blockchain system. A client proposes transactions to trigger actions of the blockchain system.

Fig. 2 shows the “order-execute” transaction workflow. (Steps 1–3) A client/user signs, crafts and sends a transaction to a full node. The transaction contains the index of the smart contract function that the client intends to invoke with a parameter list. This full node verifies and broadcasts the transaction to the network of full nodes; (Steps 4–12) A full node is elected as a proposer through the consensus protocol such as Proof-of-Authority (PoA). This proposer generates and signs a new block with the received transactions, and broadcasts the new block to the network of full nodes. Full nodes can work as validators to validate, sign the block and broadcast to notify others of its endorsing of the block; (Steps 13–15) If the new block is properly signed by a group of validators, e.g., $\frac{2}{3}$ of all the validators, each full node accepts and appends the new block to its local blockchain, and processes transactions in the new block one by one. The node executes the specified smart contract function with provided parameters, and updates its world state; (Step 16) The full node that received the client’s transaction sends a notification to the client.

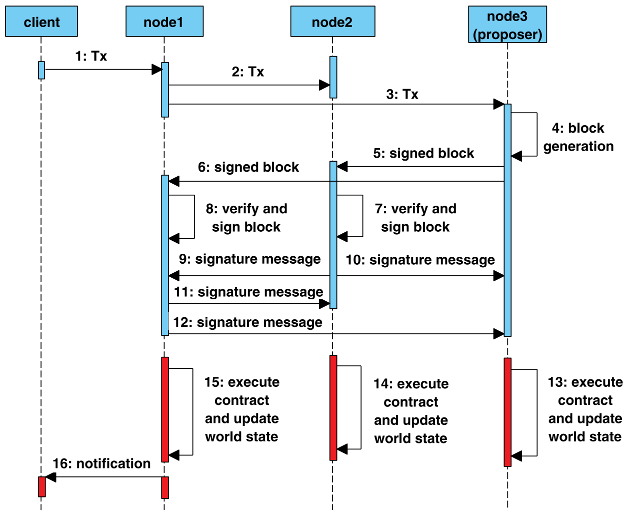


Fig. 2. "Order-execute" transaction workflow.

3.2. Permission mechanisms

A permissioned blockchain usually adopts multiple mechanisms for access control and data confidentiality. We mainly utilize the following three properties of the permissioned blockchain to design our BBS system.

Membership Service: Each node and user has a certificate which indicates its identity. The identity is bonded with one party within the consortium who builds/uses the blockchain system. Only authorized nodes and users can participate in a permissioned blockchain network.

Data Isolation: A data isolation mechanism is required when a subset of participants want to keep sensitive data private from others within the consortium. Fabric incorporates a private data collection (PDC) mechanism [35] to maintain the confidentiality of PDC data within a specific group of PDC members. In this mechanism, the original data is stored only by PDC member peers, while PDC non-member peers only store the hashes of the data.

In Enterprise Ethereum, the private transaction mechanism is adopted to manage private data, which is shared within a *privacy group*. A private transaction operates on the private world state at the privacy group nodes. A private transaction has a *data* field that contains function index and parameters, and is shared and stored only in the privacy group nodes. Only the hash of the *data* field is included in the private transaction, which will be bundled in the blockchain and stored at all nodes.

Endorsement Policy: In the "execute-order-validate" architecture, i.e., Fabric, a special mechanism called *endorsement policy* is adopted. An endorsement policy stipulates which peers are required to perform as endorsers for endorsing a transaction. Endorsing involves executing smart contracts and signing the execution results. Each smart contract has a default chaincode-level endorsement policy, which governs all public data and PDC data in the world state. A key-level endorsement policy can be flexibly customized to manage specific key–value data. A collection-level endorsement policy is employed to manage the PDC data.

4. Design goals and models

In this section, we first present the design goals and then present our system model and threat model. A naive protocol is introduced at the end and its issues are discussed.

4.1. Design goals

Our goal is to design a blockchain based big-data sharing system (BBS) that enables the establishment of a chain of custody for the big data. The ledger serves as evidences that cannot be repudiated. The big data is freely shared between a sender and a receiver within a consortium of organizations.

Given the diverse types and varying sizes of data, along with the challenges of applying existing blockchain frameworks to build BBS, our design goals are as follows.

- *Big data storage*. Given the storage limitations of traditional ledgers, it is not feasible to store large volumes of big data directly in the ledger or include extensive amounts of big data within transactions. BBS should overcome the storage limitation for big data sharing.
- *Privacy preservation*. Big data may be sensitive. The owner may not want to share the data with everyone. The blockchain system cannot be directly used for big data sharing due to all nodes maintain the same data in a conventional blockchain system. The data should be maintained only by an owner node and be shared with desired users. Access control shall be adopted.
- *Autonomy*. BBS shall be autonomous. Users shall not interact with each other off-chain for data transfer as done in related work. A sender shall be able to register his/her data with BBS, which takes over the data sharing process. The whole process of data sharing can be autonomously completed.
- *Security*. During a big data sharing session, either the sender or the receiver has the potential to behave dishonestly. The sender might fail to really share data or may provide the data that does not align with the description that she/he advertises. The receiver may dishonestly deny receiving the data from the sender. In the event of a dispute, the blockchain shall serve as a source of evidence to resolve the case.

4.2. System model

Permissioned Blockchain Settings: A consortium of organizations $\mathcal{O} = \{O_1, \dots, O_n\}$ cooperate together to build a permissioned blockchain system $\mathcal{B}$. One organization has multiple users. For clarity and simplicity but without loss of generality, assume each organization contributes a single node, i.e., peer node in Fabric, and authorizes multiple users to participate in the system. The node set is denoted as $\mathcal{N} = \{N_1, \dots, N_n\}$. The user set of O_i, $i = 1, 2, \dots, n$ is denoted as $U_i = \{U_{i1}, \dots, U_{im}\}$. A user interacts with $\mathcal{B}$ through a client node. N_i and U_i belong to O_i. All organizations collaborate to jointly develop and deploy a smart contract C on nodes $\mathcal{N}$, aiming to achieve a common business objective such as facilitating big data sharing.

We introduce a new concept of "off-state", which is stored at blockchain nodes $\mathcal{N}$, but outside of the ledgers, particularly the world state. For example, the off-state data can be big data. Therefore, we generalize big data sharing as off-state data sharing. Node N_i maintains separate off-state storage spaces for different users in U_i. A user only can access its own off-state space. Off-states can be inconsistent across $\{N_1, \dots, N_n\}$. Off-state data can be shared between two users whose off-state storage spaces may locate at pairs of nodes or locate within one node. A smart contract C can directly operate on the off-state for reading, writing, encryption/decryption and other operations including data sharing.

Fig. 3 illustrates the model of our blockchain off-state sharing system. Each node in the system maintains a world state database and some off-state storage spaces when needed. Users initiate transactions to activate the smart contracts to operate the off-state data, such as data transfer. An off-state sharing session occurs between two users denoted as S and R. The two users belong to organizations, O^S and O^R respectively, within $\mathcal{O}$. We denote their nodes as N^S and N^R, and

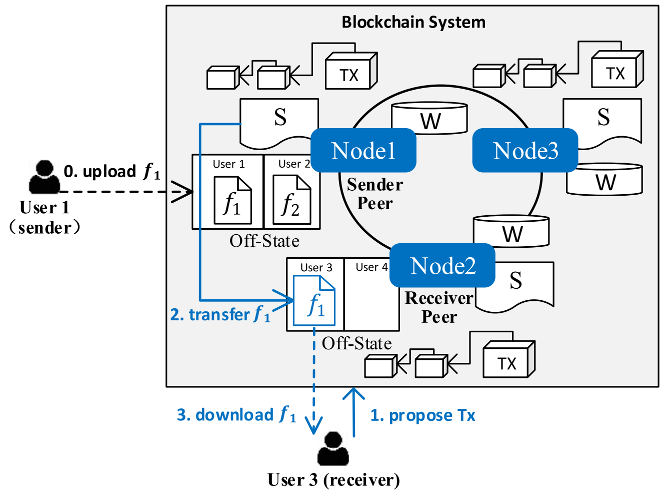


Fig. 3. Blockchain off-state sharing system model. Denote "World state" as "W", "smart contract" as "S", "transaction" as "Tx".

A naive off-state sharing protocol Π running in a permissioned blockchain system $\mathcal{B}$. User S and R interacts with $\mathcal{B}$.

Stage 1: Preparing Data for Sharing

S uploads f to off-state D^S at node N^S, and proposes a transaction with a message $(name, owner, rule, description)$ to $\mathcal{B}$. $\mathcal{B}$ stores $\langle id_f, name, owner, rule, description\rangle$ in the world state across $\{N_1, ..., N_n\}$.

Stage 2: Sharing Data

(**Phase 1: *Data Request***) R proposes a transaction $(request, id_f)$ to $\mathcal{B}$. If the identity of R satisfies the $rule$, $\mathcal{B}$ transfers f from D^S at N^S to D^R at N^R.

Fig. 4. A Naive off-state sharing protocol.

their off-state spaces as D^S and D^R. O^S and O^R can be the same one, where two users who belong to the same organization share big data. The smart contract has complete control over the sharing process and can facilitate the transfer of off-state data f from D^S in N^S to D^R in N^R using an off-state sharing protocol. Prior to data sharing, the data owner, such as User 1, uploads its data such as f_1 to its own off-state space for sharing. During the data sharing process, a data receiver, such as User 3, proposes a transaction to request a specific data such as f_1. Then the smart contract running on blockchain nodes follows the data sharing protocol to manage data access and perform operations on the off-state data. If the receiver is authorized according to the access rules, the smart contract transfers the data from the sender's off-state space to the receiver's off-state space. Subsequently, the data receiver can retrieve the shared data from its own off-state space.

4.3. Threat model

We assume that the underlying blockchain infrastructure $\mathcal{B}$ including all permission mechanisms is secure. All parties securely store their private key. User U_{ij} trusts the node N_i which belongs to the same organization O_i as the user, and does not trust nodes which belong to different organizations $O_k, k \neq i$ from the user. A user U_{ij} can retrieve the world state data in N_i, and retrieve, upload and download data only in its own off-state storage space in N_i. In Fabric, the smart contract is deployed in a decentralized manner, ensuring that no single organization has exclusive control over it. Consequently, we assume the smart contract C is trusted.

Users (e.g., sender/receiver) may be dishonest due to various factors (e.g., economic incentive, internal/external attacks). The sender may withhold the data or may share the data that does not align with the description that she/he advertises. The receiver may dishonestly deny having received the data from the sender.

Security Goals. Denote a malicious adversary (e.g., sender/receiver) as $\mathcal{A}$. We call a tuple $(id_f^*, name^*, f^*)$, which contains the data ID, name and the data plaintext, as a $forgery$ if the sender/receiver could cheat on this data. A sender cheats by providing a $forgery$ which matches with the metadata in the world state but is different from the actually transferred data. A receiver cheats by getting a $forgery$ which matches with the data metadata in the world state before transaction ordering. Intuitively, the proposed system is secure if the polynomial-time adversary $\mathcal{A}$ is computationally infeasible to find a $forgery$ of the proposed system. We formalize the protocol security of the BBS system as follows.

Definition 1. Let $\Pr[\text{Cheat}] = \Pr\left[\mathcal{A}^{\Pi}(\lambda) \rightarrow forgery\right]$ be the probability that the sender/receiver is able to cheat. We say the proposed protocol is secure if $\Pr[\text{Cheat}]$ is negligible.

4.4. Naive protocol and issues

We begin by introducing a naive protocol to illustrate the security issues that need to be carefully addressed during protocol design. This serves to motivate the development of a secure protocol, which we delve into in Section 5.

Fig. 4 shows a straightforward off-state sharing protocol Π. Π has two stages. In the preparing stage, user S registers data f with the blockchain system $\mathcal{B}$. S defines the data access rule and advertises the data description. During the sharing stage, user R proposes a transaction to request f from $\mathcal{B}$. Contract C facilitates the transfer of f to a legitimate R. These two transactions will be recorded in the blockchain as evidences of the data sharing process.

Under our threat model, the naive protocol Π is insecure. (i) A sender may deny that he/she delivers wrong data which does not match with advertised metadata in the world state. For example, in the preparing stage, S may upload wrong data f which does not match with the description but deny sharing f. (ii) A receiver may get f without recording the transaction in the blockchain so as to repudiate receiving the data. For example, under the "execute-order-validate" architecture discussed in Section 3.1.1, the smart contract executes before ordering. In the *sharing* stage, after the smart contract transfers the data f, a dishonest R may change the client application and does not submit the transaction for ordering. Then the blockchain cannot record this data transfer activity but R has received f. A secure off-state sharing protocol is needed to securely establish the chain of custody. In the next section, we propose a secure off-state sharing protocol. This protocol utilizes the trustworthy smart contract to calculate the hash of the shared data and stores it in the world state as undeniable evidence, thereby thwarting dishonest sender. It also leverages the smart contract to encrypt the off-state data and employs a private data mechanism to securely share the encryption key on-chain, effectively defeating dishonest receivers.

5. Off-state sharing protocol

In this section, we propose a secure off-state sharing protocol and present the protocol details based on Fabric. Main notations used in the following are summarized in Table 2.

Table 2
Main notations summary.

Notations	Description
f	Data to be shared
k	Encryption key used to encrypt data f
f_z	Ciphertext of data f encrypted by k
h_z	Hash of the data ciphertext f_z
h_k	Hash of the encryption key k
k_R	Ciphertext of encryption key k encrypted by the public key of the receiver user
k_z	Ciphertext of k_R encrypted by the public key of the receiver peer node

5.1. Protocol overview

In the protocol, a sender user S shares data f with a receiver user R. These two users can belong to different organizations, such as when a hospital shares some biomedical data with a research institute. Alternatively, these two users can belong to the same organization, like when two departments within a hospital internally share documents.

The secure off-state sharing protocol $\mathcal{F}$ is shown in Fig. 5. We use a big file f as an off-state data example. In *Stage 1*, sender S uploads and registers f with system $\mathcal{B}$. f is stored at the sender's off-state space D^S at the sender node N^S. $\mathcal{B}$ derives the commitment information, i.e., hash, and stores such information as well as the metadata of f in the world state. In *Stage 2*, the blockchain system $\mathcal{B}$ can perform the whole off-state sharing process autonomously. Different from related work, receiver R only interacts with $\mathcal{B}$, and does not need to involve sender S. Smart contract C encrypts f using a symmetric encryption key k, and transfers the encrypted data f_z. The received f_z is stored at the receiver's off-state space D^R at the receiver node N^R. k is encrypted two times using the public keys of the receiver node N^R and the receiver R. k is stored in the world state and managed by a data isolation mechanism such as PDC in Fabric so that receiver R has to propose transactions to get k so as to decrypt f_z. Critical information such as hashes of f_z and k will be recorded in the world state and in transactions, which perform as evidences for the chain of custody of the data. With our protocol $\mathcal{F}$, the big data can be securely shared between two users who may belong to different organizations or the same organization.

5.2. Stages of protocol

We now introduce the protocol $\mathcal{F}$ in detail.

5.2.1. Stage 1: Preparing data for sharing

The sender and data owner S prepares the big data/file to be shared. S uploads f to its off-state storage D^S at the sender's blockchain node N^S. S also proposes a transaction to trigger smart contract C to derive the metadata $(name, h, owner, rule, description)$ of f and record it as a $File$ entity in the world state so that people can search for the data information.

- The file hash h can be used to verify the integrity of f. C calculates and records h in the world state. Every node maintains h.
- The access rule $rule$ protects the privacy of f. S defines the access rule based on the identities in the permissioned blockchain. As an example, a rule denoted as $\{O_1, O_2\}$ means that f can only be shared with users belonging to organization O_1 and organization O_2.
- File metadata in the world state database is public to all users for searching.

5.2.2. Stage 2: Sharing data

Fig. 6 shows the workflow of sharing a big file over Fabric. The sharing process involves five transactions, which follow the same

A secure protocol $\mathcal{F}$ interacts with S and R. $\mathcal{F}$ runs in a permissioned blockchain system $\mathcal{B}$. $\mathcal{H}(\cdot)$ is a cryptographic hash function. $\mathcal{E}(\cdot)$ is an encryption function. $\mathcal{D}(\cdot)$ is a decryption function.

Stage 1: Preparing Data for Sharing
Sender S uploads f to off-state space D^S at node N^S, and proposes a transaction that contains parameters $(name, owner, rule, description)$ to the blockchain system $\mathcal{B}$. $\mathcal{B}$ calculates $h = \mathcal{H}(f)$ and stores $\langle id_f, name, h, owner, rule, description\rangle$ in the world state across $\{N_1, ..., N_n\}$.

Stage 2: Sharing Data
(**Phase 1:** ***Data Transfer Authorization***) Receiver R proposes a transaction specifying the file id_f to ask for permission of the particular data transfer. Based on the authorization $rule$ of f, an authorization $flag$ is set as either true or false and is stored in the world state across $\{N_1, ..., N_n\}$.

(**Phase 2:** ***Data Transfer***) R proposes a transaction with id_f to start the actual data transfer. If $flag$ is true, $\mathcal{B}$ encrypts the file $f_z = \mathcal{E}(k, f)$ and transfers encrypted file f_z. $\mathcal{B}$ calculates $h_z = \mathcal{H}(f_z)$, $h_k = \mathcal{H}(k)$, $s = Sign(sk, f_z)$, and stores h_z, h_k and s in the world state across $\{N_1, ..., N_n\}$. $\mathcal{B}$ calculates $k_R = \mathcal{E}(pk_R, k)$ and $k_z = \mathcal{E}(pk_N, k_R)$, where the two layers of encryption of k ensure that only R can derive k and other users including those from the same organization cannot. $\mathcal{B}$ stores k_z as a private data with one member $\{O^S\}$, which is the sender's organization.

(**Phase 3:** k_z ***Request***) After R receives f_z and verifies its hash matches with h_z in the world state, R proposes a transaction with id_f to request k_z. The system checks if R requested the data. If yes, k_z is set as private data with two members $\{O^S, O^R\}$.

(**Phase 4:** k_R ***Request***) Receiver R now proposes a transaction with id_f to request k_R. $\mathcal{B}$ decrypts k_z, derives $k_R = \mathcal{D}(sk_N, k_z)$ and stores k_R as private data with members $\{O^S, O^R\}$.

(**Phase 5:** ***Data Decryption***) R decrypts k_R and derives $k = \mathcal{D}(sk_R, k_R)$. R may request the receiver node to decrypt the encrypted data, or do it locally. The hash h of f is used for integrity check.

Fig. 5. A secure off-state sharing protocol.

workflow introduced in Section 3.1.1. Endorsers for each transaction can be different per the specific endorsement policy, which stipulates who should endorse the transaction. Fig. 6 shows only transaction steps of interest while ignoring other steps such as ordering in Fig. 1.

Phase 1: Data Transfer Authorization. R shall request the permission of transfer of a specific file f and the request shall pass through the access control governed by a specific endorsement policy. The endorsement policy can be set as $AND(O^S, O^R)$, which mandates peers

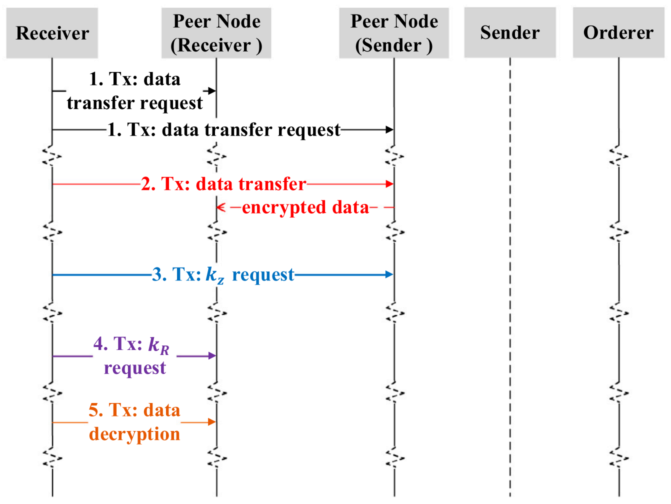


Fig. 6. Five phases in the off-state sharing protocol based on Fabric.

from both organizations O^S and O^R must endorse the transaction. Smart contract C can get the identity of the user who initiates the transaction proposal. C running at endorser peers including the sender peer node and the receiver peer node checks if the user identity meets the access rule in the $File$ entity, and generates and signs the result $flag$ in an $Event$ entity as an endorsement, which will be packed into a transaction by the receiver client. This transaction will record the result $flag$ in an $Event$ entity in the world state. The variable $flag$ is a Boolean value that indicates whether the endorser allows the transfer of the file. A valid transaction occurs when both organizations, O^S and O^R, reach a consensus and sign the same result (i.e., $Event$ entity). This result can either signify the denial of or agreement on the file transfer. We call this transaction as “Tx: data transfer request”.

Phase 2: Data Transfer. After the file transfer is authorized, R proposes another transaction—“Tx: data transfer”—to initiate the file transfer. The endorsement policy for “Tx: data transfer” can be set as $AND(O^S)$, which means the sender peer N^S works as the endorser and signs the transaction following the consensus process as shown in Fig. 1. Smart contract C at sender node N^S first checks $flag$ in the $Event$ entity. The file transfer continues only if $flag$ is $true$.

Smart contract C in sender blockchain node N^S encrypts f using a symmetric key k. C sends the encrypted file f_z to the receiver node N^R via a file transfer protocol such as *SFTP*, and calculates the hash h_z of f_z, hash h_k of k and signature s of f_z, and records h_z, h_k, and s in the corresponding $Event$ entity in the world state. Hashes and the signature can be used to verify the integrity of f_z and k by the receiver. C encrypts k using the public key pk_R of receiver R and gets k_R, and further encrypts k_R using the public key pk_N of receiver peer node N^R and gets k_z. k_z is put in the world state as a Key entity. Specifically in Fabric, the key entity is set as PDC data with sender organization O^S as the only PDC member. PDC can guarantee that only PDC member peers can see k_z and others only get its hash. Transactions involving k_z contain only its hash in Fabric. In this phase, only sender node N^S keeps the original k_z. Others such as N^R and R only have the hash of k_z.

Phase 3: k_z ***Request.*** Receiver R can send a query request to N^R and use C to check if N^R has received f_z and f_z matches with the hash h_z in the world state. In Fabric, the query request will not generate a transaction. If the check is successful, R proposes a transaction—“Tx: k_z request”. The endorsement policy for “Tx: k_z request” is set as $AND(O^S)$ so that N^S performs as an endorser. Smart contract C at N^S checks if R is the legitimate receiver by checking the corresponding $Event$ entity, which is generated in Phase 1 and stored in the world state. If yes, C sets k_z as a new PDC data with members of organizations O^S and O^R. During the consensus process of this transaction, the PDC data k_z will be transferred from the node of O^S to the node of O^R. After this transaction is committed to the ledger, the PDC data k_z at the node of O^R will be updated to the world state [35]. Then the PDC mechanism stores k_z in both N^S and N^R. Therefore, N^R will be able to retrieve k_z. Please note that the corresponding transaction only stores PDC hash, and the PDC value k_z is transmitted using a specific protocol in Fabric, and k_z is updated to the world state of N^R after the transaction is validated. PDC ensures that k_z is shared with the legitimate peer N^R and receiver R and is not publicly disclosed. Please also note when the sender and receiver belong to the same organization, i.e., $O^S = O^R$, Phase 3 can proceed as expected.

Phase 4: k_R ***Request.*** After N^R obtains k_z, R proposes a transaction—“Tx: k_R request”—to $\mathcal{B}$ to request the decryption of k_z to get k_R. The endorsement policy for “Tx: k_z request” is set as $AND(O^R)$ so that N^R performs as an endorser. If R is the legitimate receiver according to the $Event$ entity in the world state, smart contract C at N^R decrypts k_z using the private key of N^R and gets k_R. C further sets k_R as PDC data with members O^S and O^R. Similarly to Phase 3, the PDC mechanism securely stores k_R in both N^S and N^R after the transaction is validated and recorded in the blockchain [35], and ensures that k_R will not appear in the transactions. R is able to retrieve k_R from the world state.

Phase 5: Data Decryption. After receiver R gets k_R, R can choose to use $\mathcal{B}$ to decrypt the shared data (or decrypt f_z itself). R first decrypts k_R using its private key sk_R to get k. Then R proposes the transaction —“Tx: data decryption”—to $\mathcal{B}$. R passes k to C through the transaction proposal to decrypt f_z. In Fabric, k can be set as a special transient parameter, which is pruned during building the transaction so that k will not appear in the world state. k is only shared between peer N^R and user R. The endorsement policy can be set as $AND(O^R)$ so that N^R performs as an endorser. N^R can access f_z in the off-state space D^R, use k to decrypt f_z to get f, and verify if f matches with the recorded hash h in the $File$ entity.

6. Security analysis

In this section, we first prove the security of our protocol. Then, we analyze how the proposed protocol achieves the design goals.

6.1. Proof

Cryptographic tool. Let $\mathrm{Adv}[\mathcal{H}] = \Pr[\mathcal{A}^{\mathcal{H}}(\lambda) \rightarrow \text{collision}]$ be the probability that an adversary $\mathcal{A}$ finds a collision of the cryptographic hash function $\mathcal{H}$. Let $\mathrm{Adv}[\mathcal{E}]$ be the advantage of an adversary that distinguishes two ciphertexts under chosen plaintext attacks, as in standard secure encryption schemes (e.g., [36]). If $\mathcal{H}$ and $\mathcal{E}$ are said to be secure, $\mathrm{Adv}[\mathcal{H}]$ and $\mathrm{Adv}[\mathcal{E}]$ are negligible respectively for any polynomial-time adversary.

Proposition 1. *If the cryptographic hash function $\mathcal{H}(\cdot)$ and the encryption scheme $\mathcal{E}(\cdot)$ are secure, the system BBS is then secure under Definition 1.*

Proof. We first respectively analyze the security in the sender cheating case and the receiver cheating case. Then, we analyze the privacy-preserving property of BBS.

Sender Cheating. In protocol $\mathcal{F}$, the trusted smart contract C generates critical information of shared data f such as hash h and h_z, and puts them in the corresponding transactions and in the world state. Sender S cannot deny sharing data f, unless S finds a forgery $(id_f^*, name^*, f^*)$ where $\mathcal{H}(f^*) = h$ but $f^* \neq f$. That is S finds a collision of the cryptographic hash function $\mathcal{H}(\cdot)$. Then it holds that $\Pr[\text{Cheat}] \leq \mathrm{Adv}[\mathcal{H}]$. A secure $\mathcal{H}(\cdot)$ means that $\mathrm{Adv}[\mathcal{H}]$ is negligible. Then $\Pr[\text{Cheat}]$ is also negligible. S cannot deny the sharing of data f which may not be consistent with the advertisement description.

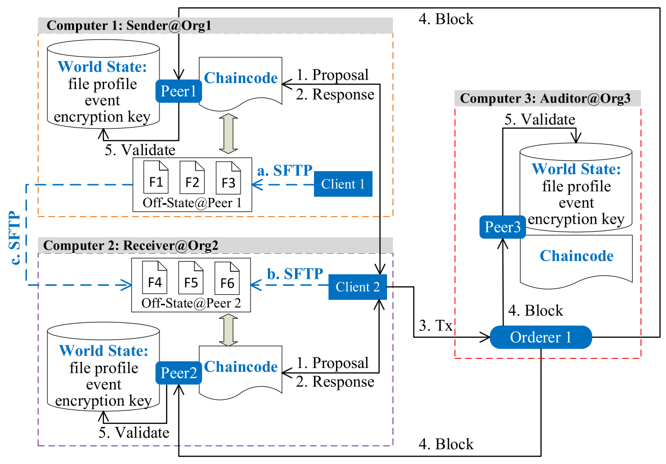


Fig. 7. Prototypical off-state sharing system over Hyperledger Fabric.

Receiver Cheating. In the case of sharing data between two users from two different organizations, after *Phase 2: Data Transfer*, the built-in PDC mechanism guarantees that N^R and R only know the ciphertext f_z of f and the hash of k_z. If R does not submit the transaction—"Tx: k_z request", organization O^R cannot get f even peer N^R and user R collude together, unless O^R finds the preimage of the hash of k_z or the plaintext of f_z. For the former case, the adversary could find a collision of the employed cryptographic hash function; in the latter case, the adversary could distinguish ciphertexts of the employed encryption scheme. Then it holds that $\Pr[\mathsf{Cheat}] \le \mathsf{Adv}[\mathcal{H}] + \mathsf{Adv}[\mathcal{E}]$. Then we have that $\Pr[\mathsf{Cheat}]$ is negligible with a secure hash function and encryption scheme.

In the case of sharing data between two users from the same organization, after *Phase 2: Data Transfer*, the built-in PDC mechanism guarantees that R only knows the ciphertext f_z of f, and ciphertext k_z of k_R. Under the assumption that a user trusts the node which belongs to the same organization as the user, both sender and receiver trust N^R. If R does not submit the transaction—"Tx: k_R request", R cannot get f, unless R finds the plaintext of k_z or the plaintext of f_z. Similarly, $\Pr[\mathsf{Cheat}]$ is negligible.

If $\Pr[\mathsf{Cheat}]$ is negligible, R has to propose and submit "Tx: k_z request" to get k_z (if cross organizations) and "Tx: k_R request" to get k_R so as to get k and decrypt f_z. The trusted C generates k, k_z and k_R and can guarantee their correctness. The blockchain provides non-repudiable evidences. R cannot deny she/he has received f.

Privacy Preservation. Regarding the privacy goal, our protocol incorporates access control mechanisms such as Private Data Collections (PDC) to allow off-states to be shared between pairs of nodes, not across all blockchain nodes. Other nodes are only aware of the hash h of the shared data and the hash h_k of the encryption keys. Similarly, a secure hash function $\mathcal{H}(\cdot)$ can ensure that it is negligible for other nodes to derive the plaintext of the data or of the encryption keys. Therefore, the data is maintained at only an owner side node and is shared with authorized users, and the privacy property can be ensured.

To summarize, our system BBS running protocol $\mathcal{F}$ can securely establish the chain of custody of the off-state data and achieve the privacy goal if the underlying cryptographic function and encryption scheme are secure. □

6.2. Design goals achievement

We now summarize how BBS achieves the four design goals in Section 4.1. (i) **Storage**. We introduce the concept of "off-state" and store the big data off the ledger to address the big data storage challenge in blockchain. (ii) **Privacy**. As analyzed in Section 6.1, BBS can preserve the data privacy by adopting off-state sharing and access control mechanisms. The shared data is stored at only the sender and the authorized receiver nodes. This feature also saves storage space at nodes that do not need the off-state data. (iii) **Autonomy**. BBS is autonomous and does not need users to interact with each other off-chain. A sender can register and delegate his/her data to BBS, which takes over the data sharing process. A receiver has to propose transactions to BBS to request data and does not need to interact with the sender. This design is user-friendly. (iv) **Security**. As analyzed in Section 6, BBS can securely build the chain of custody of the shared data to defeat dishonest users, and the blockchain provides non-repudiable evidences.

7. Evaluation

We have implemented a prototypical off-state sharing system, *BBS*, based on Fabric, and evaluate its feasibility and performance in this section.

7.1. Experiment setup

Fig. 7 shows the prototypical BBS, which has three organizations $\mathcal{O} = \{O_1, O_2, O_3\}$. Each party contributes one physical computer that runs Ubuntu 18.04 with 16 GB memory and 1TB disk. These three computers access the network through WiFi. O_1 and O_2 perform as the sender and the receiver respectively. The sender computer and the receiver computer are located in separate buildings within a university campus. We adopt Hyperledger Fabric v2.3.3 to build the prototypical system, and choose Raft as its consensus protocol. Fabric nodes run in Docker containers. The sender computer and the receiver computer run one peer node and one client node respectively. The computer of O_3 runs one peer node and one orderer node, which performs the equivalent functions as a group of orderer nodes, based on the official Fabric test-network. We incorporate off-state storage space and *SFTP* [37] service into peer docker containers of interest. The smart contract can directly operate on the off-state storage. In Hyperledger Fabric, the smart contract is referred to as chaincode and is developed using *Golang*. The chaincode runs at peer nodes, and can transfer and retrieve data to and from the local off-state storage. For the client application used by users to propose transactions, we employ *Node.js* for its development.

7.2. Performance and feasibility

We utilize *Golang*'s *SFTP* package for file transfers, and it has come to our attention that the sender's buffer size parameter significantly impacts the speed and latency of *SFTP* process. Fig. 8 shows how the buffer size affects the file transfer latency between two docker containers representing N^S and N^R using a 576 MB file and a 1.2 GB file. It can be observed that a buffer size of more than 1 MB will not reduce the latency further. So we set the buffer size as 1 MB in our system.

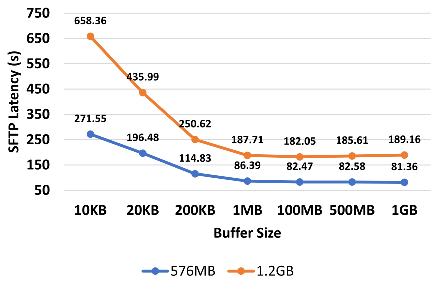


Fig. 8. *SFTP* latency versus buffer size.

Table 3
Test file list.

Type	Size	Description
.pdf	67 MB	An electronic book
.mp4	218 MB	An over 5h audio file
.tif	576 MB	A high-resolution image
.zip	1.2 GB	A collection of medical images
.rar	2.6 GB	The compressed file of one movie
.zip	5.3 GB	The compressed file of two movies

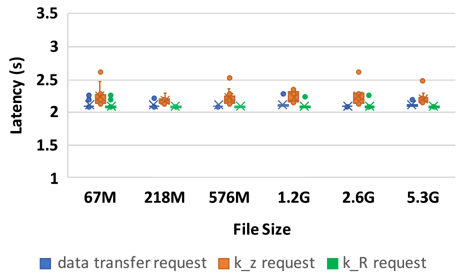


Fig. 9. Transaction latency of "Tx: data transfer request", "Tx: k_z request" and "Tx: k_R request".

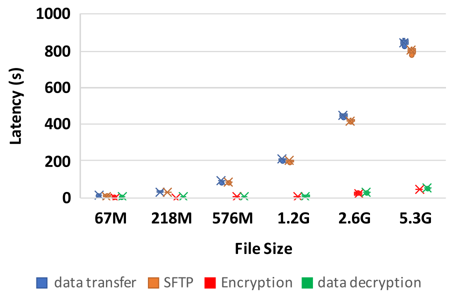


Fig. 10. SFTP latency and transaction Latency of "Tx: data transfer" and "Tx: data decryption".

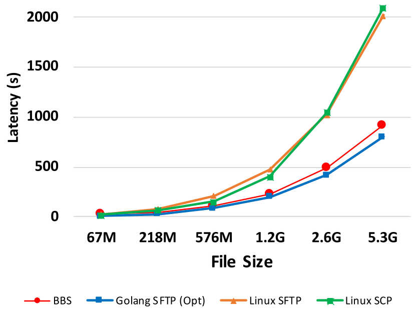


Fig. 11. File sharing session latency.

We evaluate the feasibility and performance of BBS by utilizing files of various types and sizes. Table 3 provides a list of the test files used in our evaluation. Recall that a file sharing session comprises five transactions as illustrated in Fig. 6. Our evaluation involves measuring the latency of individual transactions as well as the overall latency of the entire session. We first present the cases of sharing data between two users who belong to two organizations.

7.2.1. File size

For each file in Table 3, we conduct the file sharing session 16 times. Each time only one file sharing session is performed. The results are shown in Figs. 9–11. As shown in Fig. 9, the file size has minor effect on the latency of "Tx: data transfer request", "Tx: k_z request" and "Tx: k_R request". This is reasonable since these three transactions only operate on the world state data. As shown in Fig. 10, the latency of "Tx: data transfer" and "Tx: data decryption" rapidly increases as the file size increases. The reason is that these two transactions involve multiple cryptographic calculations and the Golang *SFTP* operation. According to Fig. 10 and Table 4, the Golang *SFTP* latency accounts for a large proportion of the whole "Tx: data transfer" latency. As is well known, the *SFTP* latency can be optimized by improving the network bandwidth.

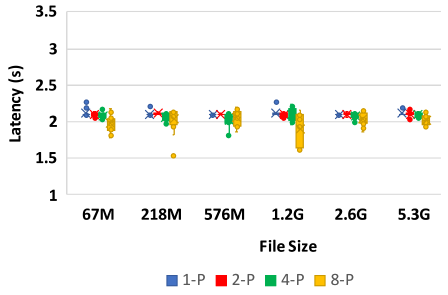


Fig. 12. Transaction latency of "Tx: data transfer request" in different parallel transfer cases (across different organizations).

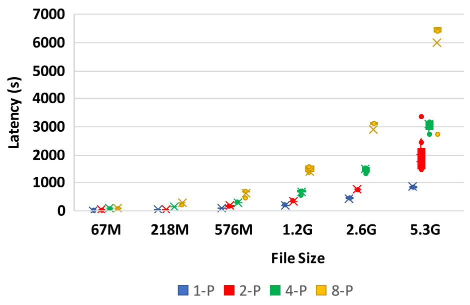


Fig. 13. Transaction latency of "Tx: data transfer" in different parallel transfer cases (across different organizations).

Table 4
Percentage of SFTP latency in "Tx: file transfer" latency.

Size	1 Parallel	2 Parallel	4 Parallel	8 Parallel
67 MB	78.45%	86.74%	96.10%	96.38%
218 MB	90.19%	94.65%	97.78%	91.29%
576 MB	94.79%	96.88%	98.26%	82.64%
1.2 GB	96.30%	95.45%	97.52%	87.17%
2.6 GB	94.00%	91.04%	84.46%	82.65%
5.3 GB	94.66%	90.66%	83.03%	79.26%

Fig. 11 illustrates the latency of a single file sharing session of BBS, and provides a comparison with the latency of Linux *SFTP* and *SCP* applications. In BBS, we utilize the optimized *Golang SFTP* with a buffer size of 1 MB. The latency measurement for BBS includes the latency of file transfer via *Golang SFTP*, as well as the latency incurred by the cryptographic computation and transaction processing and transmission. It can be observed that the latency of BBS increases rapidly as the file size grows. This is because normally more time is required to transfer larger files. We evaluate the default Linux *SCP* command for file transfer and there is no buffer size parameter. For Linux *SFTP* command, there is a buffer size parameter. We set the buffer size of Linux *SFTP* to 99 999 Bytes which is its maximal buffer size value. It can be observed that BBS performs similarly or better compared with Linux SFTP/SCP applications.

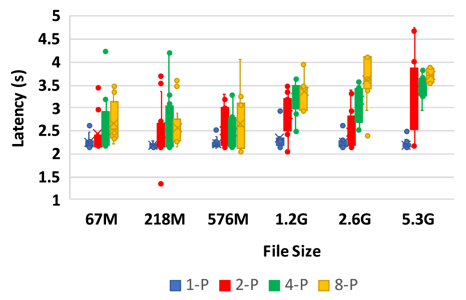


Fig. 14. Transaction latency of "Tx: k_z request" in different parallel transfer cases (across different organizations).

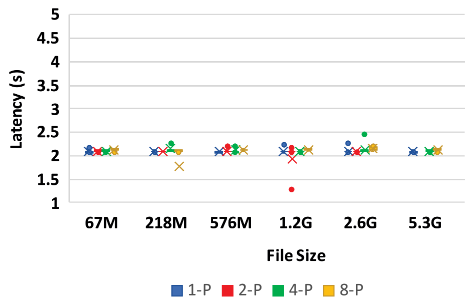


Fig. 15. Transaction latency of "Tx: k_R request" in different parallel transfer cases (across different organizations).

7.2.2. Parallel transfer

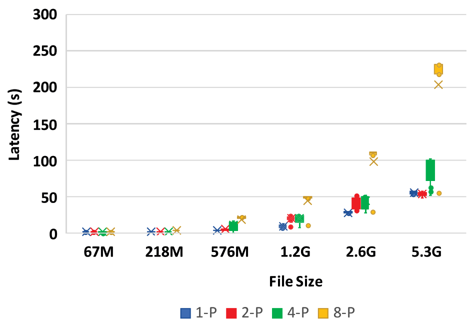
We also demonstrate that BBS has the capability to process multiple concurrent big file transfer sessions and other transactions in parallel. In BBS, a long transaction involving the big file transfer does not stop other transactions from running. For each file, the number of simultaneous parallel file sharing sessions is 1, 2, 4 and 8, denoted as 1-P, 2-P, 4-P and 8-P in Figs. 12–16, which show the experiment results. In each case, the file is transferred 16 times in total. For example, in the case of 2-P (2 parallel file sharing sessions), we perform the experiments 8 times and the file is transferred 16 times (2 × 8).

We make the following observations. As the number of parallel sessions increases, the latency of "Tx: data transfer request", "Tx: k_z request" and "Tx: k_R request" is relatively stable as shown in Figs. 12 and 14–15. However the latency of "Tx: data transfer" and "Tx: data decryption" increases as the number of parallel sessions increases. Table 4 shows the percentage of *SFTP* incurred latency in the "Tx: data transfer" latency. The *SFTP* latency accounts for the most of the transaction latency.

Fig. 16 presents the average latency of a single file sharing session in various parallel transfer cases. It can be observed that the average latency of a file sharing session increases as the number of parallel sharing sessions increases. This observation is expected because when multiple parallel file sharing sessions are active, they share the available network bandwidth. Consequently, with more sessions, the network bandwidth available for each file sharing session decreases, leading to increased latency.

7.3. Users from the same organization

For sharing data between two users who belong to the same organization, we conduct similar experiments. For each file in Table 3, we conduct the file sharing session 16 times in different parallel cases with *SFTP*. Figs. 17–22 shows the experimental results of sharing data. They show the same tendency as sharing data cross different organizations. For example, more time is needed to handle larger data.



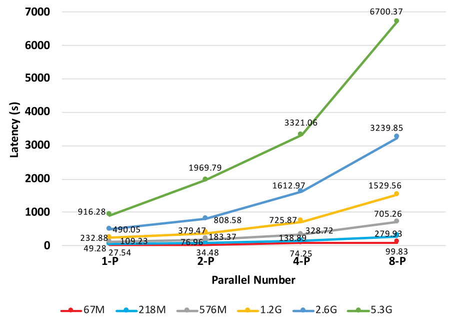


Fig. 16. Average file sharing session latency in different parallel transfer cases (across different organizations).

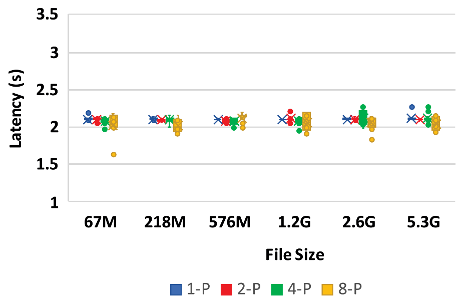


Fig. 17. Transaction latency of "Tx: data transfer request" in different parallel transfer cases (within one organization).

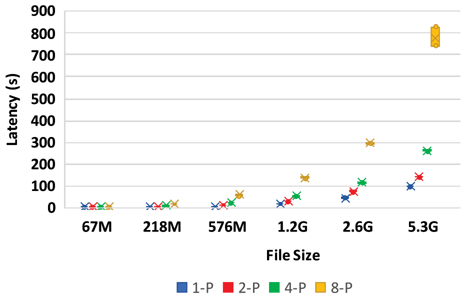


Fig. 18. Transaction latency of "Tx: data transfer" in different parallel transfer cases (within one organization).

8. Discussion

In this section, we first compare different permissioned blockchain frameworks. Then we demonstrate the generality of our off-state sharing protocol by discussing the off-state sharing over *Enterprise Ethereum*.

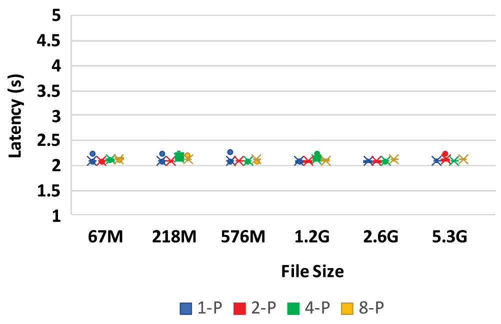


Fig. 19. Transaction latency of "Tx: k_z request" in different parallel transfer cases (within one organization).

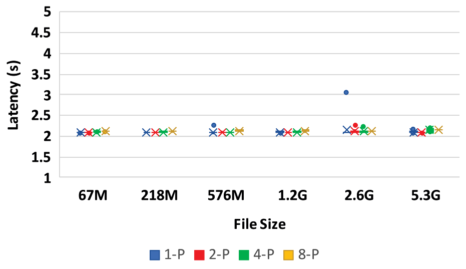


Fig. 20. Transaction latency of "Tx: k_R request" in different parallel transfer cases (within one organization).

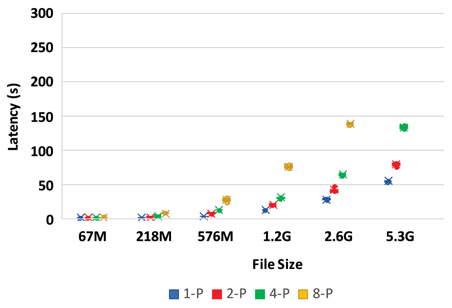


Fig. 21. Transaction latency of "Tx: data decryption" in different parallel transfer cases (within one organization).

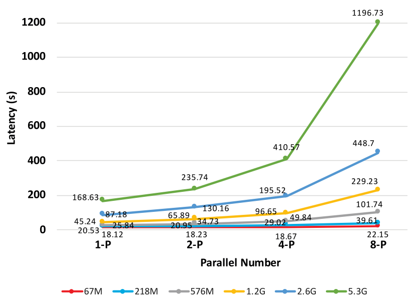


Fig. 22. Average file sharing session latency in different parallel transfer cases (within one organization).

8.1. Comparing permissioned blockchain frameworks

We argue that Fabric is a better choice to implement an off-state sharing system. Enterprise Ethereum's functionalities are limited. (i) Enterprise Ethereum does not support a mechanism like the endorsement policy of Fabric to control which nodes execute the smart contract and sign the execution results. (ii) We need to use precompiled contracts of Enterprise Ethereum to manage the off-state big data due to limited functions of the built-in Solidity contract in Enterprise Ethereum. A precompiled contract needs manual deployment. A node may also deploy a malicious precompiled contract since it is not deployed on-chain through a transaction. In comparison, Fabric's smart contract can be developed in general-purpose language such as *Golang*, and is deployed in a decentralized way. It is easier to use and is more secure. (iii) Enterprise Ethereum does not group users or nodes into organizations. Fabric's membership service provides the extra organization identifier for each node/user. It is easier in Fabric to define access rules and manage nodes/users at the organization level. (iv) Enterprise Ethereum does not have the transient parameter mechanism like Fabric. It cannot securely pass secret information such as key k from a user to nodes in the blockchain system.

8.2. Off-state sharing over enterprise ethereum

Our off-state sharing protocol is general and can be implemented over *Enterprise Ethereum* though *Enterprise Ethereum*'s functionalities are limited. We now introduce the Enterprise Ethereum version of our off-state sharing protocol in Fig. 5. We will present the threat model, protocol phases and evaluation results, and point out the differences between the Enterprise Ethereum version and Fabric version.

8.2.1. Threat model

We assume that the underlying blockchain infrastructure $\mathcal{B}$ including all nodes and all permission mechanisms is secure. A user trusts the node that belongs to the same party as the user. In Enterprise Ethereum, the smart contract is not fully trusted. Enterprise Ethereum adopts *Solidity* [38] to develop the smart contract. A *Solidity* contract is deployed through a transaction and each node runs the same solidity contract. Solidity's functionalities are very limited though Solidity is turing-complete. Enterprise Ethereum also adopts the *precompiled contract* to perform complex functionalities such as cryptographic operations. The precompiled contract can be developed in *Java*, and used to perform the off-state data sharing. The precompiled contract is not deployed through a transaction like a Solidity contract. A precompiled contract is locally configured at each node. A party may deploy a malicious precompiled contract. Therefore, we assume that the Solidity contract is trusted but the precompiled contract may not be trusted.

Users (e.g., sender/receiver) may be dishonest. The naive protocol Π in Section 4.4 over Enterprise Ethereum works as follows. A user proposes a transaction to the blockchain system to request the data and the smart contract directly transfers the data to the receiver. This protocol is also insecure. (i) Under the order-execute architecture of Enterprise Ethereum, the data transfer request transaction is recorded in the blockchain and then the precompiled contract for data sharing executes. However, the precompiled contract can be changed locally, not transferring the data. The data transfer may also fail due to network failure. A dispute will happen when the sender says the data is transferred while the receiver claims the data is not received. (ii) A malicious sender may also transfer wrong off-state data not matching with the advertised metadata. The naive protocol Π under Enterprise Ethereum cannot securely establish the chain of custody.

8.2.2. Secure off-state sharing protocol

Fig. 23 shows the workflow of off-state sharing over Enterprise Ethereum, which is similar to the Fabric version in Fig. 6. We will point out the difference below.

Phase 1: Data Transfer Authorization. A receiver proposes a similar transaction—"Tx: data transfer request"—to one node such as the receiver blockchain node. This transaction is forwarded to all nodes. Every node executes this transaction/smart contract and stores the

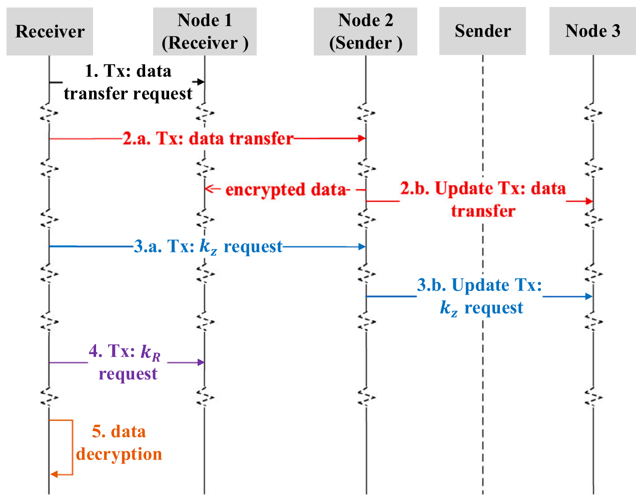


Fig. 23. Five phases in the off-state sharing protocol based on Enterprise Ethereum.

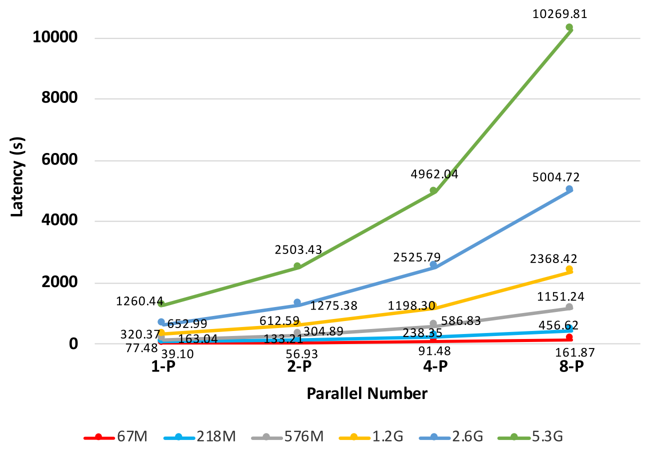


Fig. 24. Average file sharing session latency (across organizations) over Enterprise Ethereum.

$Event$ entity which contains the authorization result $flag$ in the world state.

Phase 2: Data Transfer. The receiver proposes a transaction—"Tx: data transfer" to initiate the file transfer. The smart contract is written in such a way that only if this node is the sender node, it encrypts the data f using key k, signs the encrypted data f_z and generates signature s, transfers f_z, derives the hash h_z of f_z, and performs the two-layer encryption of k and get k_z. At the end of executing the transaction—"Tx: data transfer", the precompiled contract at the sender node proposes a new transaction called "Update Tx: data transfer", which updates h_z and s to the world state across all nodes, and sets k_z as a private data with one member N^S by utilizing the data isolation mechanism—private transaction mechanism—in Enterprise Ethereum. This private transaction mechanism is similar to the PDC in Fabric and securely manages k_z in the world state. Only the sender node keeps k_z and other nodes keep the hash of k_z.

Phase 3 & 4: k_z / k_R Request. After the receiver node receives the data f_z, the receiver proposes the transaction—"Tx: k_z request". The corresponding smart contract at sender node checks the receiver's identity and initializes a new transaction—"Update Tx: k_z request"—to invoke a built-in Solidity contract to add the receiver node N^R to the private data member group of k_z. The receiver node obtains k_z after the transaction is committed to the blockchain. The receiver then proposes the transaction—"Tx: k_R request". The smart contract at the receiver node decrypts k_z and gets k_R, and sets k_R as the private data with two members $\{N^S, N^R\}$. k_R is then updated to the world state at both sender node and receiver node.

Phase 5: Data Decryption. Receiver R gets k_R from the world state at node N^R and decrypts k_R to obtain k. The receiver retrieves the received f_z in the off-state space at N^R. Then R locally does the decryption and derives the desired data f. Unlike in Fabric, the receiver in Enterprise Ethereum cannot propose the transaction—"Tx: data decryption" and ask the blockchain system $\mathcal{B}$ to do decryption. Enterprise Ethereum does not have a mechanism like *transient parameter* in Fabric to avoid exposing sensitive parameters such as k in transactions.

8.2.3. Security analysis

(i) Sender S cannot deny sharing data f unless S finds a collision or preimage of the cryptographic functions. h, h_z, h_k and signature s carried in related transactions can be used to verify the genuineness of the shared data. (ii) Receiver R cannot deny receiving data f. The built-in private transaction mechanism of Enterprise Ethereum manages keys in the world state and guarantees that R can get the key k to derive the desired data f only after proposing required transactions. Our off-state sharing protocol $\mathcal{F}$ can securely establish the chain of custody of shared data over *Enterprise Ethereum*.

8.2.4. Evaluation

We implement a prototypical *BBS* system over Enterprise Ethereum and evaluate its feasibility and performance. We also use the test files in Table 3 and evaluate the parallel cases for each file. The number of simultaneous parallel file sharing sessions is 1, 2, 4 and 8. We use an Enterprise Ethereum solution named Hyperledger Besu [39] to develop our system. The consensus protocol employed in this system is Proof-of-Authority. The smart contract is developed in Solidity. We configure precompiled contracts which are written in Java to manage the off-state data. The client application is developed in Node.js. We set the transaction fee for executing smart contract to zero to implement a free system. Fig. 24 shows the experiment results of the average latency of one file sharing session. The session latency shows the similar tendency as in the Fabric-based BBS although higher. The higher latency is mainly caused by Java-based encryption/decryption calculations which are slower than Golang-based encryption/decryption calculations.

8.3. Scalability discussion

We argue that the performance of data sharing is not influenced by the number of nodes. In our off-state system model, the data is shared between two nodes without the need for synchronization across all nodes. Moreover, as per Table 3, the overall latency primarily arises from the use of *SFTP* for file transfer. The number of nodes may slightly impact the latency incurred during the consensus process. Nonetheless, this effect is not significant enough to seriously impact the performance of BBS. Consequently, the BBS prototype system can readily accommodate a larger number of organizations and nodes.

9. Conclusion

In this paper, we propose a blockchain based big data sharing protocol, which is able to establish the chain of custody of various shared data. Such a decentralized data sharing application is critical for managing data of various types and sizes such as healthcare data. We denote data such as a big file stored at a blockchain node but outside of the ledger as off-state. We design and implement a blockchain-based big data sharing system (BBS), which addresses the challenge of storage by storing big data outside of the ledger, and exchanges data between only users of interest autonomously, thus preserving data privacy. The transactions generated by our protocol serve as auditing evidences for the chain of custody. We implement BBS over Hyperledger Fabric and

Enterprise Ethereum, and conduct extensive experiments to validate the feasibility and performance of the blockchain based big data sharing protocol. It appears that Fabric is a better choice for BBS given Fabric's framework and SDK features. In the future work, we would investigate methods for optimizing the overhead associated with data transfer, aiming to explore the integration of traditional big data sharing techniques into the smart contract framework.

CRediT authorship contribution statement

Shan Wang: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Ming Yang:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Methodology, Funding acquisition. **Shan Jiang:** Writing – review & editing, Writing – original draft, Validation, Supervision. **Fei Chen:** Writing – review & editing, Writing – original draft, Validation, Supervision, Formal analysis. **Yue Zhang:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology. **Xinwen Fu:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

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

Acknowledgments

This research was supported in part by National Key R&D Program of China (No. 2023YFC3605804), National Natural Science Foundation of China (Nos. 62072103, 62272318), Jiangsu Provincial Key R&D Programs (Nos. BE2021729, BE2022680, BE2022065-5), Hong Kong Research Grant Council Theme-based Research Scheme (No. T43-513/23-N), US National Science Foundation (NSF) Awards 2325451, 1931871 and 1915780, Shenzhen Science and Technology Program (No. JCYJ20220531102802005), Key Laboratory of Computer Network and Information Integration of Ministry of Education of China (No. 93K-9), Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201), Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Research Institute for Artificial Intelligence of Things of The Hong Kong Polytechnic University. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] B.E. Dixon, C.M. Cusack, Measuring the value of health information exchange, in: Health Information Exchange, Elsevier, 2016, pp. 231–248.

[2] D. Le Bihan, Looking into the functional architecture of the brain with diffusion MRI, Nature Rev. Neurosci. 4 (6) (2003) 469–480.

[3] K. Budd, Foreign data theft: What academic institutions can do to protect themselves, 2019.

[4] Y. Xu, M.Z.A. Bhuiyan, T. Wang, X. Zhou, A.K. Singh, C-fdrl: Context-aware privacy-preserving offloading through federated deep reinforcement learning in cloud-enabled IoT, IEEE Trans. Ind. Inform. 19 (2) (2022) 1155–1164.

[5] Wikipedia, Chain of custody, 2021, [Online]. (Accessed 27 October 2021). [Online]. Available: https://en.wikipedia.org/wiki/Chain_of_custody.

[6] X. Zhou, X. Ye, I. Kevin, K. Wang, W. Liang, N.K.C. Nair, S. Shimizu, Z. Yan, Q. Jin, Hierarchical federated learning with social context clustering-based participant selection for internet of medical things applications, IEEE Trans. Comput. Soc. Syst. (2023).

[7] X. Zhou, W. Liang, I. Kevin, K. Wang, Z. Yan, L.T. Yang, W. Wei, J. Ma, Q. Jin, Decentralized P2P federated learning for privacy-preserving and resilient mobile robotic systems, IEEE Wirel. Commun. 30 (2) (2023) 82–89.

[8] Y. Xu, S. Xiao, H. Wang, C. Zhang, Z. Ni, W. Zhao, G. Wang, Redactable blockchain-based secure and accountable data management, IEEE Trans. Netw. Serv. Manag. (2023).

[9] A.M. Antonopoulos, Mastering Bitcoin: Unlocking Digital Cryptocurrencies, O'Reilly Media, Inc., 2014.

[10] B. Wiki, Weaknesses, 2021, [Online]. (Accessed 26 March 2021). [Online]. Available: https://en.bitcoin.it/wiki/Weaknesses.

[11] B. Forum, New Bitcoin vulnerability: A transaction that takes at least 3 minutes to be verified by a peer, 2021, [Online]. (Accessed 26 March 2021). [Online]. Available: https://bitcointalk.org/index.php?topic=140078.0.

[12] Etherscan, Ethereum average gas limit chart, 2021, [Online]. (Accessed 26 March 2021). [Online]. Available: https://etherscan.io/chart/gaslimit.

[13] C. Gorenflo, S. Lee, L. Golab, S. Keshav, FastFabric: Scaling hyperledger fabric to 20 000 transactions per second, Int. J. Netw. Manage. 30 (5) (2020) e2099.

[14] D. Hu, Y. Li, L. Pan, M. Li, S. Zheng, A blockchain-based trading system for big data, Comput. Netw. 191 (2021) 107994.

[15] Y. Chen, J. Guo, C. Li, W. Ren, FaDe: A blockchain-based fair data exchange scheme for big data sharing, Future Internet 11 (11) (2019) 225.

[16] S. Dziembowski, L. Eckey, S. Faust, Fairswap: How to fairly exchange digital goods, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 967–984.

[17] L. Eckey, S. Faust, B. Schlosser, Optiswap: Fast optimistic fair exchange, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, 2020, pp. 543–557.

[18] F. Winzer, B. Herd, S. Faust, Temporary censorship attacks in the presence of rational miners, in: 2019 IEEE European Symposium on Security and Privacy Workshops, EuroS&PW, IEEE, 2019, pp. 357–366.

[19] P. Zhang, J. White, D.C. Schmidt, G. Lenz, S.T. Rosenbloom, FHIRChain: applying blockchain to securely and scalably share clinical data, Comput. Struct. Biotechnol. J. 16 (2018) 267–278.

[20] S. Wang, Y. Zhang, Y. Zhang, A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems, IEEE Access 6 (2018) 38437–38450.

[21] U.U. Uchibeke, K.A. Schneider, S.H. Kassani, R. Deters, Blockchain access control ecosystem for big data security, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2018, pp. 1373–1378.

[22] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the 13th EuroSys Conference, 2018, pp. 1–15.

[23] Ethereum, Ethereum mainnet for enterprise, 2021, [Online]. (Accessed 13 June 2021). [Online]. Available: https://ethereum.org/en/enterprise/.

[24] J. Wang, K. Han, A. Alexandridis, Z. Chen, Z. Zilic, Y. Pang, G. Jeon, F. Piccialli, A blockchain-based eHealthcare system interoperating with WBANs, Future Gener. Comput. Syst. 110 (2020) 675–685.

[25] A. Zhang, X. Lin, Towards secure and privacy-preserving data sharing in e-health systems via consortium blockchain, J. Med. Syst. 42 (8) (2018) 1–18.

[26] S. Jiang, J. Cao, H. Wu, Y. Yang, M. Ma, J. He, Blochie: a blockchain-based platform for healthcare information exchange, in: 2018 IEEE International Conference on Smart Computing, smartcomp, IEEE, 2018, pp. 49–56.

[27] Q. Xia, E.B. Sifah, K.O. Asamoah, J. Gao, X. Du, M. Guizani, MeDShare: Trust-less medical data sharing among cloud service providers via blockchain, IEEE Access 5 (2017) 14757–14767.

[28] M. Shen, J. Duan, L. Zhu, J. Zhang, X. Du, M. Guizani, Blockchain-based incentives for secure and collaborative data sharing in multiple clouds, IEEE J. Sel. Areas Commun. 38 (6) (2020) 1229–1241.

[29] J. Chen, Z. Lv, H. Song, Design of personnel big data management system based on blockchain, Future Gener. Comput. Syst. 101 (2019) 1122–1129.

[30] K. Fan, S. Wang, Y. Ren, H. Li, Y. Yang, Medblock: Efficient and secure medical data sharing via blockchain, J. Med. Syst. 42 (8) (2018) 1–11.

[31] Q. Xia, E.B. Sifah, A. Smahi, S. Amofa, X. Zhang, BBDS: Blockchain-based data sharing for electronic medical records in cloud environments, Information 8 (2) (2017) 44.

[32] X. Cheng, F. Chen, D. Xie, H. Sun, C. Huang, Design of a secure medical data sharing scheme based on blockchain, J. Med. Syst. 44 (2) (2020) 1–11.

[33] Z. Wang, Y. Tian, J. Zhu, Data sharing and tracing scheme based on blockchain, in: 2018 8th International Conference on Logistics, Informatics and Service Sciences, LISS, IEEE, 2018, pp. 1–6.

[34] S. Wang, M. Yang, T. Ge, Y. Luo, X. Fu, BBS: A blockchain big-data sharing system, in: ICC 2022-IEEE International Conference on Communications, IEEE, 2022, pp. 4205–4210.

[35] S. Wang, M. Yang, Y. Zhang, Y. Luo, T. Ge, X. Fu, W. Zhao, On private data collection of hyperledger fabric, in: 2021 IEEE 41st International Conference on Distributed Computing Systems, ICDCS, IEEE, 2021, pp. 819–829.

[36] V. Shoup, Sequences of games: a tool for taming complexity in security proofs, IACR Cryptol. ePrint Arch. 2004 (2004) 332.
[37] Wikipedia, SSH file transfer protocol, 2021, [Online]. (Accessed 6 Apirl 2021). [Online]. Available: https://en.wikipedia.org/wiki/SSH_File_Transfer_Protocol.
[38] Wikipedia, Solidity, 2021, [Online]. (Accessed 13 June 2021). [Online]. Available: https://en.wikipedia.org/wiki/Solidity.
[39] Hyperledger, Hyperledger Besu, 2022, [Online]. (Accessed 26 March 2022). [Online]. Available: https://besu.hyperledger.org/en/stable/.

Shan Wang received the B.Sc. degree in computer science and engineering from Southeast University, China, in 2016, and Ph.D. degree in School of Cyber Science and Engineering, Southeast University, China, in 2022. From 2019 to 2023, she studied at University of Massachusetts Lowell as a visiting scholar. Currently, she is a postdoctoral fellow at The Hong Kong Polytechnic University. Her research interests include blockchain security.

Ming Yang received the Ph.D. degree in computer science from Southeast University, China, in 2007. Currently, he is a professor at the School of Computer Science and Engineering in Southeast University, China. His research interests include network security and privacy. He is a member of CCF and ACM, as well as deputy director of Key Laboratory of Computer Network and Information Integration, Ministry of Education.

Shan Jiang received the B.Sc. degree in computer science and technology from Sun Yat-sen University, Guangzhou, China, in 2015 and Ph.D. degree in computer science from The Hong Kong Polytechnic University, Hong Kong SAR, in 2021. He is currently a Research Assistant Professor in Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR. Before that, he visited Imperial College London from November 2018 to March 2019. He won the best paper award from 2021 International Conference on Blockchain and Trustworthy Systems. His research interests include distributed systems and blockchain, blockchain-based big data sharing, and blockchain as a service.

Fei Chen is an Associate Professor with college of computer science and engineering, Shenzhen University, China. He received the B.Eng. and M.Sc. degrees in computer science and engineering from Chongqing University, China, and the Ph.D. degree in computer science and engineering from The Chinese University of Hong Kong, China. His research interests include information and network security, data protection, and privacy.

Yue Zhang is an incoming assistant professor at Drexel University's Computer Science department, and is currently working as a Postdoc in the Department of Computer Science and Engineering (CSE) at The Ohio State University. His research primarily focuses on System Security, specifically in the areas of IoT Security and mobile security. He has published more than 40 papers in prestigious security conferences (e.g. nine of his papers have appeared in USENIX Security, ACM CCS, and NDSS) and journals (e.g., TDSC, TPDS). He received a Best Paper Honorable Mention Award at ACM CCS 2022, and the Best Paper Award at 2019 IEEE International Conference on Industrial Internet. He has also served on the organization committees of the conferences (e.g., track chair for IEEE MSN) and technical program committee of the conferences (e.g., USENIX Security, NDSS, ACM CCS, RAID). His research had led to the discovery of many vendor-acknowledged vulnerabilities, such as by Bluetooth SIG, Apple, Google, and Texas Instruments, and had attracted intense media attention such as Hacker News, and Mirage News.

Xinwen Fu is a Professor with the Department of Computer Science, University of Massachusetts Lowell, Lowell, MA, USA. He received the B.S. degree in 1995 from Xian Jiaotong University, China, the M.S. degree in electrical engineering in 1998 from the University of Science and Technology of China, China, and the Ph.D. degree in computer engineering in 2005 from Texas A&M University, College Station, TX, USA. His current research interests include computer security and privacy, and digital forensics. He has published at prestigious venues including the top four cybersecurity conferences. His research was reported by various media such as Wired and aired on CNN and CCTV 10. He is a senior member of IEEE.