

Article

A Blockchain-Based Data Sharing System with Enhanced Auditability

Yao Xiao ¹, Lei Xu ^{1,*}, Zikang Chen ¹, Can Zhang ¹ and Liehuang Zhu ^{1,2}¹ School of Cyberspace Science and Technology, Beijing Institute of Technology, Beijing 100081, China² Southeast Academy of Information Technology, Beijing Institute of Technology, Putian 351100, China

* Correspondence: 6120180029@bit.edu.cn

Abstract: Cloud platforms provide a low-cost and convenient way for users to share data. One important issue of cloud-based data sharing systems is how to prevent the sensitive information contained in users' data from being disclosed. Existing studies often utilize cryptographic primitives, such as attribute-based encryption and proxy re-encryption, to protect data privacy. These approaches generally rely on a centralized server which may cause a single point of failure problem. Blockchain is known for its ability to solve such a problem. Some blockchain-based approaches have been proposed to realize privacy-preserving data sharing. However, these approaches did not fully explore the auditability provided by the blockchain. The dishonest cloud server can share data with a requester without notifying the data owner or being logged by the blockchain. In this paper, we propose a blockchain-based privacy-preserving data sharing system with enhanced auditability. The proposed system follows the idea of hybrid encryption to protect data privacy. The data to be shared are encrypted with a symmetric key, and the symmetric key is encrypted with a joint public key which is the sum of multiple blockchain nodes' public keys. Only if a data requester is authorized, the blockchain nodes will be triggered to execute a verifiable key switch protocol. By using the output of the protocol, the data requester can get the plaintext of the symmetric key. The blockchain nodes participate in both the authorization process and the key switch process, which means the behavior of the data requester is witnessed by multi-parties and is auditable. We implement the proposed system on Hyperledger Fabric. The simulation results show that the performance overhead is acceptable.

Keywords: auditability; data sharing; blockchain; privacy-preserving**MSC:** 68P25

Citation: Xiao, Y.; Xu, L.; Chen, Z.; Zhang, C.; Zhu, L. A Blockchain-Based Data Sharing System with Enhanced Auditability. *Mathematics* **2022**, *10*, 4494. <https://doi.org/10.3390/math10234494>

Academic Editor: Jonathan Blackledge

Received: 13 October 2022

Accepted: 23 November 2022

Published: 28 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

With the rapid development of information technology, a large volume of data is generated in consumer and industrial activities. For ordinary users, it is not an easy task to manage the rapidly growing data. Lots of individuals and companies choose to delegate the data storage task to cloud service providers, such as Microsoft and Amazon. Cloud-based data storage also provides a convenient way for users to share data with each other. However, users may lose control of their data if they hand over data to cloud servers. A dishonest cloud server may analyze or share the stored data with others against the data owner's will, which causes the disclosure of the owner's privacy. Therefore, it is important to propose a privacy-preserving scheme for cloud-based data sharing.

Cryptographically tools, such as proxy re-encryption (PRE) [1–3] and attribute-based encryption (ABE) [4–6], are often utilized to deal with the privacy issue in cloud-based data sharing systems. Both PRE and ABE enable the data owner to store his data on an honest-but-curious cloud server in encrypted form. PRE-based approaches treat the cloud server as a proxy. The proxy uses the re-encryption key generated by the owner to translate the ciphertext into a form which can be decrypted by the designated data requester. By this

way, the owner can share his data to the requester without disclosing sensitive information to the cloud server. ABE-based approaches require the data owner to specify an access control policy before encrypting the data. A data requester may be able to get the ciphertext from the cloud server, but he cannot decrypt the ciphertext unless his attributes satisfy the owner's policy.

These privacy-preserving approaches are designed for the traditional centralized cloud service model, thus they inevitably have a single point of failure problem. The centralized server is vulnerable to DDoS attacks. The system will be unavailable once the cloud server fails. Another shortcoming of the above mentioned approaches is that they lack auditability. The cloud server may share the stored data to an unauthorized requester or forge log files for potential benefits. It is difficult for users to detect dishonest behaviors of the cloud server.

To alleviate the aforementioned problems, some researchers have proposed blockchain-based data sharing approaches. Simply speaking, blockchain is a decentralized, traceable, and immutable ledger [7]. Blockchain participants, usually referred to as nodes, maintain the consistency of the ledger through a consensus mechanism. A data sharing system that incorporates blockchain with PRE, ABE, or other encryption schemes generally works as follows. The data owner encrypts his data with an encryption key. The encrypted data are stored on a cloud server, and the decryption key is transferred via the blockchain to data requesters who have the right to access the owner's data. In this way, blockchain can provide an immutable and traceable log, which improves the auditability of the system. In some studies, the smart contracts or blockchain nodes provide functions such as key management, thereby reducing the dependence on a single server.

However, existing blockchain-based data sharing approaches [8–13] still cannot guarantee that the whole data sharing process is auditable. Each node in the blockchain network has a copy of the ledger. The node can parse the ledger to obtain the information stored in it without invoking any smart contract. That is to say, by colluding with a node, a user can “secretly” obtain the information published on the blockchain ledger without leaving a trace on the ledger. For example, in a PRE-based system [8], the data owner generates a re-encryption key for the data requester. The server re-encrypts the ciphertext with the re-encryption key for the requester. The re-encryption key and the new ciphertext are both published on the blockchain. However, once the owner publishes the re-encryption key on the blockchain ledger, the server may acquire it from a compromised blockchain node and re-encrypt more ciphertext for the requester without publishing the re-encryption result. As a result, the requester can receive more data than the owner has agreed. Similarly, in an ABE-based system [12], the encrypted data are stored in the cloud server and the blockchain is used to transfer the key. If the owner encrypts the decryption key with ABE and publishes the result on the blockchain ledger, the requester can get the plaintext of every decryption key from a compromised node without being logged, as long as he satisfies the associated policy. Further, if the cloud server colludes with the requester, then the requester can get all ciphertext corresponding to the decryption keys without requiring the consent of the data owner. From the perspective of privacy protection, the lack of auditability is undesired, since the data owner has no idea who has obtained his data. There are a few studies [14,15] that try to further enhance the auditability of data sharing. However, they only focus on the decryption key issuing or after-the-fact accountability.

In this paper, we propose a new blockchain-based data sharing system that provides stronger auditability. Similar to the aforementioned approaches, the data owner encrypts the document to be shared with a symmetric key and uploads the ciphertext to a cloud server. The symmetric key is encrypted with a public key and the corresponding ciphertext is published on the blockchain ledger. In order to prevent the data requester from obtaining the document secretly, we propose to encrypt the symmetric key with a joint public key which is constructed with multiple nodes' public keys. The data owner publishes the encrypted symmetric key together with an attribute-based access control policy on the blockchain ledger. The data requester invokes a smart contract to request a document.

If he satisfies the policy specified by the data owner, the aforementioned nodes will run a verifiable parallel key switch (VPKS) protocol to convert the encrypted symmetric key into a form that the requester can decrypt. Specifically, each node is required to submit a zero-knowledge proof to the blockchain to prove that it performs the conversion correctly. Only when every node has submitted its result to the blockchain can the requester obtain the plaintext of the symmetric key and then decrypt the document. In this way, only if all the nodes are compromised by the data requester, the data requester is able to decrypt the ciphertext of the symmetric key without being logged. As long as there is one node that does not collude with the requester, the requester has to invoke the smart contract to trigger the nodes to execute the VPKS protocol. As a result, evidence of the invocation and the execution of the VPKS protocol can be found in the blockchain ledger, which proves that the requester has been authorized to access the document and is able to decrypt the ciphertext.

The main contributions of this paper are summarized as follows.

- (1) We propose a blockchain-based auditable data sharing system. The proposed system allows users to share data securely and offers enhanced auditability. As long as all the nodes do not collude, the system can guarantee that the data owner knows who has obtained the data he shares in the first place.
- (2) We propose a verifiable parallel key switch protocol. A cluster of nodes executes this protocol to convert the ciphertext encrypted with a joint public key into a form which can be decrypted with someone's private key. We use zero-knowledge proof techniques to prevent the node from providing an incorrect intermediate result. The nodes perform the calculations and submit the results to the blockchain in parallel.
- (3) We have conducted an experimental evaluation of the proposed system. We have built a simulation environment on the Fabric platform and evaluated the time cost under different settings. The results demonstrate the feasibility of the proposed system.

The remainder of this paper is organized as follows. Section 2 reviews some representative studies on blockchain-based data sharing. Section 3 introduces some preliminary knowledge related to our work. The system model and design goals of the proposed system are presented in Section 4. Section 5 presents the details of the proposed system. An informal security analysis of the proposed system is provided in Section 6. Section 7 presents the performance evaluation of the proposed system. Finally, conclusions are drawn in Section 8.

2. Related Work

Due to its decentralization and transparency, blockchain technology is being used to mitigate the centralization of traditional encrypted data sharing systems and enhance their auditability.

Gao et al. [8,9] used blockchain to manage the PRE keys in data sharing process. As we have discussed in Section 1, once the owner publishes the re-encryption key on the blockchain ledger, the server may receive it from a compromised blockchain node and re-encrypt more ciphertext for the requester without publishing the re-encryption result. As a result, the requester can get more data than the owner has agreed. Zheng et al. [10] proposed a blockchain-based decentralized data trading platform and used PRE for data transmission. Similarly, ref. [11] propose a ciphertext-policy attribute-based proxy re-encryption algorithm. The scheme reduces the number of re-encryption required, thereby reducing nodes' computing overhead and network communication overhead.

Xia et al. [12], provided a cloud-based access control system for IoT environments. The data is stored in the cloud server in the form of ABE encrypted ciphertext. The ciphertext of the key is transmitted on the blockchain. The scheme also provides data owners with the ability to update access policies for encrypted data. Liu et al. [13] proposed a blockchain-aided searchable attribute-based encryption. The blockchain system replaces the traditional centralized server that is responsible for threshold generation, key management, and user revocation to reduce dependence on a single point. Yuan et al. [14] proposed a data sharing approach which combines accountable CP-ABE and blockchain. If a decryption key is illegally shared, any third party can publicly verify whether the key was provided by the

corresponding user or the attribute authority and the key abuser cannot deny it. Before being discovered, a lot of data may have been illicitly obtained through this decryption key. Hei et al. [15] proposed a two-step key publishing mechanism to enhance the auditability of attribute decryption key issuing. Only after multiple attribute authorities publish their keys on the blockchain, can the user obtain the final attribute decryption key.

However, the above studies do not provide sufficient auditability. Once the ABE decryption key or the re-encryption key is published, the malicious user may collude with the cloud server to obtain more data without the owner's awareness. To protect the interest of the data owner against this threat, it is necessary to provide greater auditability for data sharing. In order to achieve stronger auditability, Froelicher et al. [16,17] proposed two systems for privacy-conscious statistical analysis on distributed datasets. These systems utilize technologies such as zero-knowledge proofs and differential privacy to protect privacy of data providers. However, their solution only supports the sharing of statistic values and has limited usage scenarios. In this paper, we focus on data sharing in general scenarios.

3. Preliminaries

In this section, we introduce some basic concepts of the cryptographic tools and the blockchain technique which we employ to build the proposed system.

3.1. ElGamal Cryptosystem

The ElGamal public key encryption system, which was proposed in 1985 [18], has the property of adding homomorphism. In the proposed system, the elliptic curve implementation of ElGamal encryption is adopted. Given a message m , the corresponding ciphertext is:

$$[m]_P = (rB, m + rP), \quad (1)$$

where B is the generate point of the elliptic curve, P is the public key of the receiver, and r is a random element in a finite field. To decrypt the ciphertext, the receiver multiplies its private key p and rB to get rP . Then the receiver subtracts rP from the right part of the ciphertext to get the message m . The additive homomorphism means that the ciphertext satisfies the following equation:

$$[\alpha m_1 + \beta m_2]_P = \alpha [m_1]_P + \beta [m_2]_P, \quad (2)$$

where m_1 and m_2 are two messages, α and β are two scalars.

3.2. Key Switch Protocol

In [16], Froelicher et al., proposed a *key switch* (KS) protocol. By following this protocol, several servers jointly transform a ciphertext produced by the elliptic curve ElGamal cryptosystem into a form that a specified receiver can decrypt with its private key. Formally speaking, for message M , the key switch protocol performs the following transformation:

$$[M]_{CollP} \rightarrow [M]_{P_U}, \quad (3)$$

where P_U is the public key of the receiver U , and the collective public key $CollP$ is an aggregation of the servers' public keys:

$$CollP = \sum_{i=1}^N P_i, \quad (4)$$

where N denotes the number of servers and P_i is the public key of the i th server. Specifically, given the ciphertext $[M]_{CollP} = (C_1, C_2) = (r \cdot B, M + r \cdot CollP)$, each server $S_i (i > 1)$ first receives $(\tilde{C}_{1,i-1}, \tilde{C}_{2,i-1})$ from S_{i-1} , then generates a random nonce v_i and computes $(\tilde{C}_{1,i}, \tilde{C}_{2,i})$ as

$$\tilde{C}_{1,i} = \tilde{C}_{1,i-1} + v_i \cdot B \quad (5)$$

and

$$\begin{aligned} \tilde{C}_{2,i} &= \tilde{C}_{2,i-1} - (r \cdot B)p_i + v_i \cdot P_U \\ &= \tilde{C}_{2,i-1} - r \cdot P_i + v_i \cdot P_U \end{aligned} \quad (6)$$

where p_i is the private key of the i th server. For server S_1 , we define $(\tilde{C}_{1,0}, \tilde{C}_{2,0}) = (0, C_2)$. The last server S_N produces the tuple as:

$$\begin{aligned}\tilde{C}_{1,N} &= \sum_{i=1}^N v_i \cdot B \\ &= v \cdot B\end{aligned}\quad (7)$$

and

$$\begin{aligned}\tilde{C}_{2,N} &= C_2 - r \cdot \sum_{i=1}^N P_i + \sum_{i=1}^N v_i \cdot P_U \\ &= M + r \cdot \text{CollP} - r \cdot \text{CollP} + v \cdot P_U \\ &= M + v \cdot P_U,\end{aligned}\quad (8)$$

where $v = \sum_{i=1}^N v_i$. The ciphertext $(\tilde{C}_{1,N}, \tilde{C}_{2,N})$ is the target ciphertext $[M]_{P_U}$. The receiver U can decrypt it to get M .

The complexity of the protocol increases linearly with the number of servers. In their later work [17], Froelicher et al. reconstructed the protocol. They improved the protocol's algorithm to organize servers in a tree, increasing the efficiency of the protocol.

However, in this original KS protocol, both parts of the Elgamal ciphertext must be points on the elliptic curve. That is to say, the message M has to be a point on the chosen elliptic curve. Upon receiving the ciphertext $(\tilde{C}_{1,i-1}, \tilde{C}_{2,i-1})$, the server first performs an elliptic curve addition for the left part, and then performs an elliptic curve addition for the right part. The final ciphertext $(\tilde{C}_{1,K}, \tilde{C}_{2,K})$ can be decrypted by the requester. If the message m is not a point on the chosen elliptic curve, as shown in Figure 1, one can first map the message to a point mB , and then apply the KS protocol. When the protocol finishes, the requester decrypts the ciphertext and gets the point mB . After that, the requester needs to use a brute-force approach to find the original image m . Considering this, Froelicher et al. restrict m to an integer no greater than 100,000.

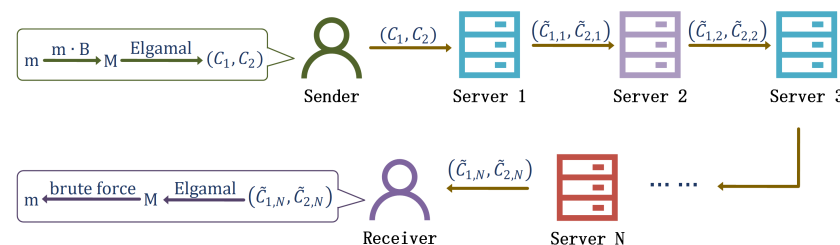


Figure 1. A simple illustration of the KS protocol.

3.3. Zero-Knowledge Proof of Correctness

Goldwasser et al. [19] constructed the first zero-knowledge proof system that can prove the correctness of a statement without leaking any meaningful information. It has been shown that all languages in NP have zero-knowledge proofs, in the premise that the encryption functions exist [20]. In [21], Camenisch and Stadler proposed a notation for describing general statements about knowledge of discrete logarithms, and showed how to construct an efficient proof system.

For example, to prove that the discrete logarithms of $y_1 = g_1^{x_1}$ and $y_2 = g_2^{x_2}$ to the base g_1 and g_2 , respectively, satisfy the representation $a_1^{x_1} a_2^{x_2} = a \pmod{q}$, the prover proceeds as follows:

- $(v_1, v_2) \in_R \mathbb{Z}_q^2$, $t_1 = g_1^{v_1}$, $t_2 = g_2^{v_2}$ and $t_3 = a_1^{v_1} a_2^{v_2}$
- $c = H(g_1, y_1, g_2, y_2, a_1, a_2, a, t_1, t_2, t_3)$
- $r_1 = v_1 - cx_1 \pmod{q}$ and $r_2 = v_2 - cx_2 \pmod{q}$

The expression $\xi \in_R X$ means that ξ is chosen randomly from the (finite) set X according to the uniform distribution. H denotes a collision-resistant hash function. The values (t_1, t_2, t_3) , c , and r_1, r_2 are called *commitment*, *challenge*, and *response*, respectively. The re-

sulting proof is (c, r_1, r_2) and can be verified by first reconstructing the commitments

$$\begin{aligned} t'_1 &= y_1^c g_1^{r_1}, \\ t'_2 &= y_2^c g_2^{r_2}, \\ t'_3 &= a^c a_1^{r_1} a_2^{r_2}, \end{aligned} \quad (9)$$

and then checking the equations

$$c = H(g_1, y_1, g_2, y_2, a_1, a_2, a, t'_1, t'_2, t'_3). \quad (10)$$

The example can be extended to prove that $Y_1 = y_1 B$ and $Y_2 = y_2 B$ to the base point B on elliptic curves $\mathbb{E}$, satisfy the representation $A_1 y_1 + A_2 y_2 = A$. This is applied to our work.

3.4. Blockchain and Smart Contract

In a narrow sense, blockchain refers to an immutable distributed data ledger organized in the form of blocks [7,22]. A blockchain network consists of multiple nodes where each node has a backup of the ledger. In a consortium blockchain (such as Fabric [23]), users need to register their identity certificates with the certificate authority (CA). A user is authenticated by the certificate to use blockchain services, for example, to initiate transactions. A blockchain node packages transactions generated by users into a new block. The block also contains other information, such as the hash value of the previous block, the timestamp, and so on. The node broadcasts the new block to others. Each node validates the contents of the block and performs the chosen consensus protocol, such as Kafka [24] or Raft [25]. The consensus protocol ensures the consistency of the block accepted by each node. Each node appends the block to the local ledger after the consensus protocol.

A smart contract is a protocol combining messages and algorithms that can be executed automatically in public networks, proposed by Nick Szabo [26]. In blockchains, a smart contract refers to a program that includes data and codes running on the blockchain ledger. Users can interact with the ledger through smart contracts. Smart contracts can communicate with users through the event service. Smart contracts can define different events. Users can register a listener for the specified event. The contract can send messages to all users that have registered for the specified event listener.

4. System Model and Design Goals

In this section, we first describe the main entities in the data sharing scenario considered in our work. Then we provide an overview of the proposed data sharing system. After that, we analyze the security risks posed by different entities and introduce the design goals of the proposed system.

4.1. System Model

As shown in Figure 2, there are six main types of entities in the proposed data sharing system, including data owner, data storage platform, data requester, blockchain network, key switch cluster, and attribute authority.

- **Data Owner:** Data owner (DO) refers to the user who wants to share his/her data via the proposed system. DO has the right to decide who can get his/her data. We use *document* to represent the smallest unit of the data shared by DO. A DO can specify different access policies for different documents.
- **Data Storage Platform:** The data storage platform (DSP) provides data storage and acquisition services to data owners and data requesters. In practice, DSP can either be centralized or distributed. For example, the platform can be deployed on a cloud server or an InterPlanetary File System (IPFS). In the proposed system, DSP receives encrypted documents from DOs.
- **Data Requester:** Data requester (DR) refers to the user who wants to obtain data from some data owner. DR sends his/her request to the blockchain and gets the shared document(s) from DSP.

- **Blockchain Network:** The blockchain network consists of multiple nodes from N different organizations. These nodes jointly maintain an immutable ledger. As we will explain in the following section, publicly accessible information, such as the basic description of the shared document, user's access request and the corresponding authorization result, is recorded on the ledger. Data owners and data requesters interact with the ledger through smart contracts.
- **Key Switch Cluster:** Each organization generally selects multiple nodes to join the blockchain network and specifies one of them as the Key Switch Node (KSN). The key switch nodes from all organizations form the Key Switch Cluster (KSC) which is responsible for converting the encrypted data into a form that data requesters can decrypt.
- **Attribute Authority:** Attribute authority (AA) issues attribute certificates to data requesters to certify that they have certain attributes. The proposed system utilizes attributed-based access control. A DR is allowed to get a DO's data only if the attributes defined in the DR's certificate satisfy the access policy specified by the DO.

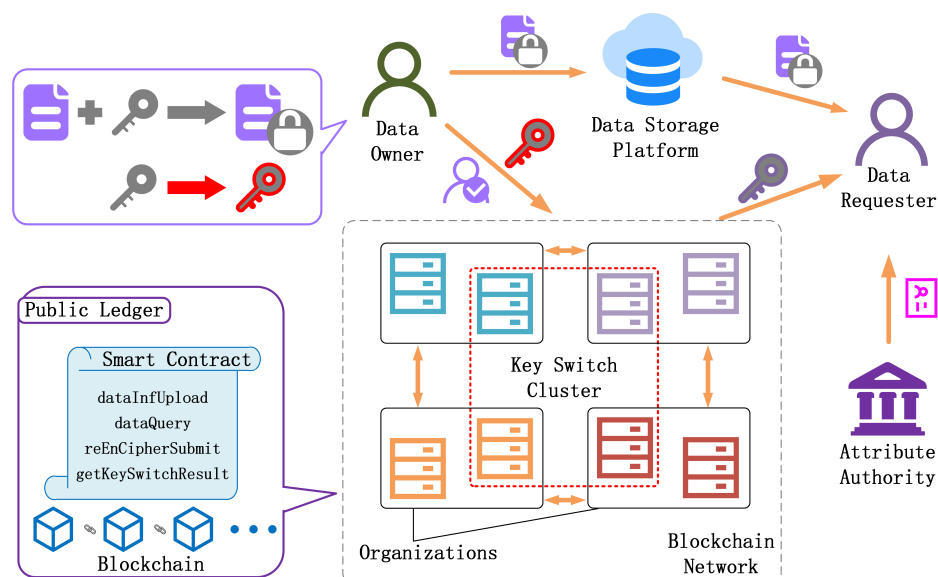


Figure 2. System model.

4.2. Basic Workflow

For simplicity and without loss of generality, we assume that a *DO* shares one document with a *DR* at a time. The proposed data sharing system works as follows. In the initialization phase, system entities generate their private-public key pairs. The *DR* registers at the *AA* and gets an attribute certificate. The public keys of key switch nodes and the attribute certificates of data requesters are published on the blockchain ledger. Then all nodes in the *KSC* aggregate their public keys to get the joint public key which will be used to encrypt the symmetric key of the shared document. We refer to this joint public key as the *KSC's* public key.

To share a document, the *DO* first encrypts the document with a symmetric key and uploads the ciphertext to the *DSP*. Then the *DO* encrypts the symmetric key with *KSC's* public key. After that, the *DO* invokes the smart contract to publish the encrypted key, a basic description of the shared document, and the attribute-based access policy specified for the document on the blockchain ledger.

The *DR* who is interested in the *DO's* document first sends a data access request to the blockchain by invoking the smart contract. Then the attribute-based access control (ABAC) process is executed automatically. If the *DR* satisfies the access policy, an event will be triggered and the *KSC* will start to run the verifiable parallel key switch protocol. After the key switch process finishes, the *DR* is able to decrypt the ciphertext of the symmetric key.

Then the *DR* can download the encrypted document from *DSP* and use the symmetric key to decrypt the document.

4.3. Threat Model

Our work mainly focuses on how to deal with the privacy issues that occur in the data sharing process. In the workflow described above, there are various potential threats to the data owner's privacy.

First, the *DR* is threatening since he/she may try to get *DO*'s data without obtaining authorization from *DO*. We assume that the *DR* is able to get the encrypted document from *DSP* freely, but he/she cannot compromise the *DO* to get the symmetric key directly. Moreover, the *DR* can collude with the *DSP* and some *KSNs*, and it is assumed that the *DR* cannot collude with all the *KSNs* in the *KSC* simultaneously. Furthermore, if the *DR* has the authorization of a document, he/she may also try to get the document without the perception of the *DO*. Secondly, the *DSP* which holds *DO*'s encrypted documents is assumed to be honest-but-curious. The *DSP* will not disclose the documents to others arbitrarily, but the *DPS* may collude with malicious *DRs* or compromised *KSNs*. Thirdly, the *KSN* whose duty is to help the *DR* to decrypt the *DO*'s document may pose a threat to *DO*'s privacy. We assume that some of them may be compromised by the malicious *DR*. A compromised *KSN* will provide all the blockchain ledger data it owns to the malicious *DR* and provide the computing services it needs privately. We also assume that the compromised *KSNs* will execute the verifiable parallel key switch protocol correctly. Otherwise, the data requester and other entities in the system will discover that the node is compromised.

In addition, we make the following assumptions. The blockchain network cannot be controlled by the malicious *DR* and the ledger cannot be tampered with. The *AA* is fully trusted and cannot be compromised by the *DR*. The communication channel between any entity pair is secure. The cryptographic primitives used in the proposed system are secure.

4.4. Design Goals

The proposed system is expected to realize secure and auditable data sharing under the security assumptions described above. Specifically, the system needs to achieve the following goals:

- *Secure Fine-Grained Data sharing*: A *DR* can share multiple documents with others and the confidentiality of these documents should be guaranteed. For each document, the *DR* should be able to specify in detail which users are allowed to access the document. Other than the *DO* and legitimate *DRs*, no one should be able to see the content of the shared documents. The data stored on the *DSP* and the information published on the blockchain ledger should not disclose any sensitive information about the shared documents.
- *Audit-Supporting*: The whole data sharing process should be logged on the blockchain ledger. As long as a *DR* has requested a document, we should be able to find corresponding records from the ledger, such as if the *DR* has been authorized, if the *DR* has got the key to decrypt the document, etc. When privacy leakage happens, these records can help to determine the source. These records can be used for auditing.

5. Proposed Scheme

In the previous section, we have presented a simple overview of the proposed data sharing system. In this section, we first introduce the smart contract designed for the system, then we describe in detail the data sharing process which consists of the initialization phase, data upload phase, data query phase, and data acquisition phase.

5.1. Design of Smart Contract

As mentioned in Section 4, the proposed data sharing system contains a blockchain network, and users outside the blockchain network can only access the distributed ledger

via smart contracts. We design a smart contract to fulfill the main functions of the data sharing system. As shown in Contract 1, we refer to this smart contract as Data Sharing Contract (DSC). DSC contains the following functions:

Contract 1 Data Sharing Contract (DSC)

```

1: // Callable by DO to publish information about the shared document
2: function dataInfUpload( $H(doc)$ ,  $H([doc]_k)$ ,  $Meta$ ,  $Plc$ ,  $[key]_{pk_\Sigma}$ )
3:
4: // Callable by DR to obtain information of specified documents
5: function getDataInfo( $keyWord$ )
6:
7: // Callable by DR to publish request for document
8: function dataQuery( $H(doc)$ ,  $pk_{DR}$ )
9:
10: // Callable by DSP to verify the legitimacy of data request
11: function checkRequst( $H(doc)$ ,  $rqID$ ,  $pk_{DR}$ )
12:
13: // Callable by KSNs to submit re-encrypted data
14: function reEnCipherSubmit( $rqID$ ,  $pk_{KSN_i}$ ,  $share_i$ ,  $\pi_i$ )
15:
16: // Callable by DR to obtain the symmetric key to decrypt the document
17: function getKeySwitchResult( $rqID$ )

```

5.1.1. DSC.dataInfUpload()

As shown in Algorithm 1, the DO calls this function to publish information about the shared document. The published information is supposed to help data requesters to find documents satisfying their requirements.

Algorithm 1 DSC.dataInfUpload

```

1: // Callable by DO to publish information about the shared document
2: function DATAINFUPLOAD( $H(doc)$ ,  $H([doc]_k)$ ,  $Meta$ ,  $Plc$ ,  $[K]_{pk_\Sigma}$ )
3:    $flag \leftarrow dataExist(H(doc))$ 
4:   if  $flag = True$  then
5:     return error
6:   else[this is a new document]
7:      $data \leftarrow [H([doc]_k), Meta, Plc, [K]_{pk_\Sigma}]$ 
8:      $Record(H(doc), data)$ 
9:     return success
10:  end if
11: end function

```

5.1.2. DSC.getDataInfo()

As shown in Algorithm 2, a DR calls this function to search documents whose descriptions fit specified keywords. This function retrieves the published information of all documents, filters out irrelevant documents, and returns information of relevant documents.

Algorithm 2 DSC.getDataInfo

```

1: // Callable by DR to obtain information of specified documents
2: function GETDATAINFO(keyWord)
3:   for each docHash recorded do
4:     meta  $\leftarrow$  GetMeta(docHash)
5:     if keyWord  $\subset$  meta then
6:       cipherHash  $\leftarrow$  GetCipherHash(docHash)
7:       dataInfo  $\leftarrow$  [docHash, cipherHash, meta]
8:       dataList  $\leftarrow$  add(dataList, dataInfo)
9:     end if
10:  end for
11:  return dataList
12: end function

```

5.1.3. DSC.dataQuery()

As shown in Algorithm 3, when a DR wants to obtain a DO's document, the DR calls this function to publish its request. The function extracts DR's attributes from its certificate and executes attribute-based access control. The request and the authorization result will be recorded on the ledger. If the DR is authorized, the function notifies registered entities, such as the KSNs, of this request through the event service.

Algorithm 3 DSC.dataQuery

```

1: // Callable by DR to publish request for document
2: function DATAQUERY( $H(doc)$ ,  $pk_{DR}$ )
3:   flag  $\leftarrow$  queryExist( $H(doc)$ ,  $pk_{DR}$ )
4:   if flag = True then
5:     return error
6:   else[this is a new request]
7:     cert  $\leftarrow$  getCertFromPK( $pk_{DR}$ )
8:     attr  $\leftarrow$  getAttr(cert)
9:     plc  $\leftarrow$  getPlcFromDHash( $H(doc)$ )
10:    // the result valiResult is 0 or 1
11:    valiResult  $\leftarrow$  validation(plc, attr)
12:    rqID  $\leftarrow$  GetTXID()
13:    rqData  $\leftarrow$  [ $H(doc)$ ,  $pk_{DR}$ , valiResult]
14:    Record(rqID, data)
15:    return rqID, valiResult
16:   end if
17: end function

```

5.1.4. DSC.checkRequest()

As shown in Algorithm 4, the DSP calls this function to verify the legitimacy of a DR's request for specified documents. The function retrieves the corresponding request record from the blockchain ledger, checks the authorization decision, and returns the authorization result.

Algorithm 4 DSC.checkRequest

```

1: // Callable by DSP to verify the legitimacy of data request
2: function CHECKREQUEST( $H(doc), rqID, pk_{DR}$ )
3:    $flag_q \leftarrow queryExist(rqID)$ 
4:   if  $flag_q = False$  then
5:     return error
6:   end if
7:    $data \leftarrow GetQuery(rqID)$ 
8:   if  $H(doc) \neq data.H(doc)$  then
9:     return error
10:  end if
11:  if  $pk_{DR} \neq data.pk_{DR}$  then
12:    return error
13:  end if
14:  if  $data.valiResult = False$  then
15:    return error
16:  else [ $data.valiResult = True$ ]
17:    return true
18:  end if
19: end function

```

5.1.5. DSC.reEnCipherSubmit()

As shown in Algorithm 5, during the key switch process, each KSN calls this function to submit its re-encrypted data to the blockchain. The function verifies if the ciphertext is correctly calculated based on the proof submitted by the KSN. The verified ciphertext will be recorded on the ledger, and this submission is notified to the DR through the event service.

Algorithm 5 DSC.reEnCipherSubmit

```

1: // Callable by KSNs to submit re-encrypted data
2: function REENCIPHERSUBMIT( $rqID, pk_{KSN_i}, share_i, \pi_i$ )
3:    $flag_q \leftarrow queryExist(rqID)$ 
4:   if  $flag_q = False$  then
5:     return error
6:   end if
7:   if  $pk_{KSN_i}$  in KSC then
8:      $flag_s \leftarrow shareExist(rqID, pk_{KSN_i})$ 
9:     if  $flag_s = True$  then
10:      return error
11:    end if
12:     $flag_v \leftarrow validProof(\pi_i)$ 
13:    if  $flag_v = False$  then
14:      return error
15:    end if
16:     $shareID \leftarrow GetTXID()$ 
17:     $data \leftarrow [rqID, pk_{KSN_i}, share_i, \pi_i]$ 
18:     $Record(shareID, data)$ 
19:    return shareID
20:  else [illegal KSN]
21:    return error
22:  end if
23: end function

```

5.1.6. DSC.getKeySwitchResult()

As shown in Algorithm 6, the *DR* calls this function to obtain the symmetric key to decrypt the document. By utilizing the re-encrypted data submitted by *KSNs*, this function transforms the ciphertext of the symmetric key into a form that can be decrypted with *DR*'s private key.

Algorithm 6 DSC.getKeySwitchResult

```

1: // Callable by DR to obtain the symmetric key to decrypt the document
2: function GETKEYSWITCHRESULT(rqID)
3:    $flag_q \leftarrow queryExist(rqID)$ 
4:   if  $flag_q = False$  then
5:     return error
6:   end if
7:    $H(doc) \leftarrow getHashbyRq(rqID)$ 
8:    $[K]_{pk_\Sigma} \leftarrow getCiKey(H(doc))$ 
9:    $[K]_t \leftarrow [K]_{pk_\Sigma}$ 
10:  for each  $KSN_i$  in KSC do
11:     $flag_s \leftarrow shareExist(rqID, pk_{KSN_i})$ 
12:    if  $flag_s = False$  then
13:      return error
14:    end if
15:     $share \leftarrow getShare(rqID, pk_{KSN_i})$ 
16:     $[K]_t \leftarrow cipherSum([K]_t, share)$ 
17:  end for
18:  return  $[K]_{pk_{DR}} \leftarrow [K]_t$ 
19: end function

```

5.2. Initialization

Before any document is shared in the proposed system, cryptographic materials such as keys and certifications must be prepared. Two types of encryption schemes are adopted in our work to protect data owners' privacy. The symmetric key encryption scheme, or more specifically, AES-256-GCM, is applied to the shared document, and the public-key encryption scheme, or more specifically, the ElGamal encryption scheme, is applied to the symmetric key. A big prime p and an elliptic curve G on $\mathbb{Z}_p$ with B as the generate point is chosen for encryption. A function $F_{keyGen} : \mathbb{Z}_p \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ is defined to maps a two-dimensional vector to a scalar in the finite field $\mathbb{Z}_p$. That is, for a point $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ on curve G , $z \leftarrow F_{keyGen}(x, y)$ is a scalar in $\mathbb{Z}_p$.

In the initialization phase, the data requester first generates its public-private key pair (pk_{DR}, sk_{DR}) , and then applies an attribute certificate *Cert* from *AA*. The certificate contains *DR*'s public key pk_{DR} and a field that describes the *DR* with a set of attributes. We denote the attribute set as $Attr = \{attr_1, \dots, attr_M\}$.

Suppose there are N organizations. Each organization $org_i (i = 1, \dots, N)$ chooses several nodes to join the blockchain network, and one of the nodes is specified as *KSN*. Each KSN_i generates its public-private key pair (pk_{KSN_i}, sk_{KSN_i}) and publishes the public key on the blockchain. The node KSN_N from the organization org_N calculates the joint public key, namely the *KSC*'s public key, as follows

$$pk_\Sigma = \sum_{i=1}^N pk_{KSN_i}. \quad (11)$$

The node then publishes this result on the blockchain.

Moreover, *AA* publishes the smart contract *DSC* described in V-A on the blockchain. The contract allows entities to register different event services to obtain function results. Once the corresponding function is called, the contract notifies the registrant of its result.

Each *KSN* registers an event listener for function *DSC.dataQuery()*, so that it can convert the encrypted data for *DR* immediately.

5.3. Data Upload

5.3.1. Data Encryption

In order to encrypt the shared document *doc* and avoid brute force as in [16,17], *DO* first generates a random number *ran* and maps it onto *G*:

$$K = \text{ran} \cdot B, \quad (12)$$

Then *DO* uses F_{keyGen} to map *K* to a value $k \triangleq F_{keyGen}(K)$. This value is used as the symmetric key to encrypt the document *doc*. We refer to *K* as pre-key. We denote the encrypted document as $[doc]_k$. *DO* calculates the hash value of *doc*, denoted as $H(doc)$, and the hash value of the encrypted document, denoted as $H([doc]_k)$, respectively. After that, *DO* encrypts *K* with the *KSC*'s public key pk_Σ and gets the ciphertext

$$[K]_{pk_\Sigma} = (r \cdot B, K + r \cdot pk_\Sigma), \quad (13)$$

where *r* is a random element in $\mathbb{Z}_p$.

5.3.2. Data Submit

To share the document in a privacy-preserving way, *DO* submits different data to the blockchain and *DSP*. *DO* first prepares a publicly accessible description *Meta* of the shared document. *Meta* includes the document's name, creation time, type, etc. *DO* also specifies the attribute-based access policy *Plc* for the document. Then *DO* calls the smart contract function *DSC.dataInfUpload()* to publish $H(doc)$, $H([doc]_k)$, *Meta*, *Plc* and $[K]_{pk_\Sigma}$ on the blockchain. After that, *DO* sends $H(doc)$ and $[doc]_k$ to *DSP*. Then *DSP* computes the hash value of $[doc]_k$ and compares the result with $H([doc]_k)$ stored on the blockchain. Only if the two hash values are equal, *DSP* will store the data sent from *DO*.

5.4. Data Query

5.4.1. Data Query

In order to find desired documents, *DR* first specifies the query keyword *keyWord* and calls the function *DSC.getDataInfo()*. The smart contract will check the field *Meta* stored in the ledger and use *keyWord* to pick out relevant documents. *Meta* and hash values of relevant documents will be returned to the *DR*. If *DR* is interested in some document *doc*, *DR* will extract the hash value $H(doc)$ from the corresponding *Meta* and calls the function *DSC.dataQuery()*. In addition, *DR* registers an event listener so that it can be notified when *KSNs* finish their work.

5.4.2. Access Control

After receiving *DR*'s request, the function *DSC.dataQuery()* starts an access control process. The function first extracts *DR*'s attribute set from its certificate. By using the hash value $H(doc)$ as the query condition, the function retrieves the access policy *Plc* specified for the document from the ledger. Then the function evaluates whether *DR* satisfies the access policy. The evaluation result is recorded on the blockchain ledger in the form of a transaction. The ID of the corresponding transaction, denoted as *rqID*, and the evaluation result are returned to *DR*.

If *DR* satisfies the access policy, the smart contract will notify all the *KSNs* through the event service. Each *KSN* can get the requester's public key pk_{DR} , the hash of the target document $H(doc)$ and the encrypted pre-key $[K]_{pk_\Sigma}$.

5.5. Data Acquisition

5.5.1. Verifiable Parallel Key Switch (VPKS)

In order to decrypt $[doc]_k$, *DR* needs to obtain the symmetric key *k*. As mentioned earlier, the pre-key *K* is encrypted under *KSC*'s public key pk_Σ . As shown in Figure 3, to

enable the DR to decrypt the ciphertext $[K]_{pk_{\Sigma}}$, all the KSN s run a verifiable parallel key switch (VPKS) protocol after being notified by the contract.

Each key switch node KSN_i gets the encrypted pre-key $[K]_{pk_{\Sigma}}$, transaction ID $rqID$, DR 's public key pk_{DR} and the hash value of the requested document $H(doc)$ from the smart contract. As shown in Protocol 1, we rewrite the ciphertext $[K]_{pk_{\Sigma}}$ as (C_1, C_2) , where $C_1 = r \cdot B$ and $C_2 = K + r \cdot pk_{\Sigma}$.

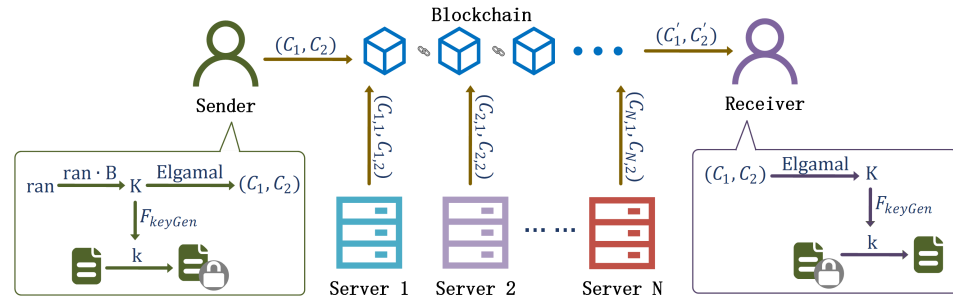


Figure 3. A simple illustration of the proposed VPKS protocol.

Protocol 1 Verifiable Parallel Key Switch (VPKS)

- 1: *Input.* $pk_{DR}, [K]_{pk_{\Sigma}} = (C_1, C_2) = (r \cdot B, K + r \cdot pk_{\Sigma})$
- 2: *Output.* $[K]_{pk_{DR}} = (C'_1, C'_2) = (r' \cdot B, K + r' \cdot pk_{DR})$
- 3: *Protocol.*
- 4: **Share Submit**
- 5: *Each Key Service Node KSN_i :*
 - 6: 1. generates a secret random nonce r_i .
 - 7: 2. computes its re-encrypted cipher as its share:
 - 8:
$$C_{i,1} = r_i \cdot B,$$
 - 9:
$$C_{i,2} = -r \cdot pk_{KSN_i} + r_i \cdot pk_{DR},$$
 - 10:
$$share_i \leftarrow (C_{i,1}, C_{i,2}),$$
 - 11: in which
 - 12:
$$r \cdot pk_{KSN_i} = r \cdot B \cdot sk_{KSN_i} = C_1 \cdot sk_{KSN_i}.$$
 - 13: 3. creates the proof for the calculation:
 - 14:
$$\pi_i \leftarrow PK\{(r_i, sk_{KSN_i}) : C_{i,1} = r_i \cdot B,$$
 - 15:
$$pk_{KSN_i} = sk_{KSN_i} \cdot B,$$
 - 16:
$$pk_{DR} \cdot r_i - C_1 \cdot sk_{KSN_i} = C_{i,2}\}.$$
 - 17: 4. calls $DSC.sahreSubmit()$ to submit $share_i = (C_{i,1}, C_{i,2})$ and π_i on the blockchain.
- 18: **Collection Replacement**
- 19: *DSC.getKeySwitchResult:*
 - 20: 1. gets all N shares $(C_{i,1}, C_{i,2})$ submitted by KSN s.
 - 21: 2. computes $(C'_1, C'_2) = (\sum C_{i,1}, \sum C_{i,2}) = (r' \cdot B, K + r' \cdot pk_{DR})$ where $r' = \sum r_i$.
 - 22: 3. returns $[K]_{pk_{DR}} = (C'_1, C'_2)$.

Each node KSN_i first generates a secret random nonce r_i and computes $(C_{i,1}, C_{i,2})$ as follows:

$$C_{i,1} = r_i \cdot B, \quad (14)$$

$$\begin{aligned} C_{i,2} &= -C_1 \cdot sk_{KSN_i} + r_i \cdot pk_{DR} \\ &= -r \cdot B \cdot sk_{KSN_i} + r_i \cdot pk_{DR} \\ &= -r \cdot pk_{KSN_i} + r_i \cdot pk_{DR}. \end{aligned} \quad (15)$$

We refer to the tuple $(C_{i,1}, C_{i,2})$ as KSN_i 's share.

In addition to calculating the share, KSN_i needs to generate a zero-knowledge proof to prove that the share is correctly calculated. KSN_i generates two random values v_1, v_2 from $\mathbb{Z}_p$ and calculates the commitment $T = (T_1, T_2, T_3)$ as

$$\begin{aligned}
T_1 &= v_1 \cdot B, \\
T_2 &= v_2 \cdot B, \\
T_3 &= v_1 \cdot pk_{DR} + v_2 \cdot C_1.
\end{aligned} \tag{16}$$

KSN_i then calculates the challenge as

$$c = H(B, C_{i,1}, pk_{KSN_i}, pk_{DR}, C_1, C_{i,2}, T_1, T_2, T_3) \tag{17}$$

and the response as

$$\begin{aligned}
r'_1 &= v_1 - c \cdot r_i \pmod{p}, \\
r'_2 &= v_2 - c \cdot sk_{KSN_i} \pmod{p}.
\end{aligned} \tag{18}$$

The proof π_i is (c, r'_1, r'_2) .

Then KSN_i calls the function $DSC.reEnCipherSubmit()$ to submit the share $(C_{i,1}, C_{i,2})$, the proof π_i and the transaction ID $rqID$ to the smart contract. To verify the share, the contract parses π_i as (c, r'_1, r'_2) and reconstructs the commitment $T' = (T'_1, T'_2, T'_3)$ as

$$\begin{aligned}
T'_1 &= r'_1 \cdot B + c \cdot C_{i,1}, \\
T'_2 &= r'_2 \cdot B + c \cdot pk_{KSN_i}, \\
T'_3 &= r'_1 \cdot pk_{DR} + r'_2 \cdot C_1 + c \cdot C_{i,2}
\end{aligned} \tag{19}$$

and checks the equation

$$c = H(B, C_{i,1}, pk_{KSN_i}, pk_{DR}, C_1, C_{i,2}, T'_1, T'_2, T'_3). \tag{20}$$

If the equation is satisfied, the contract writes the verified share to the blockchain ledger. Let TX_{share_i} to denote the corresponding transaction. Then the smart contract will notify the DR through the event service.

After receiving N event notifications, DR calls the function $DSC.getKeySwitchResult()$ to aggregate the shares. The function utilizes all the shares submitted by $KSNs$ and the original ciphertext $[K]_{pk_\Sigma} = (C_1, C_2)$ to compute

$$\begin{aligned}
C'_1 &= \sum_{i=1}^N C_{i,1} \\
&= \sum_{i=1}^N r_i \cdot B
\end{aligned} \tag{21}$$

and

$$\begin{aligned}
C'_2 &= C_2 + \sum_{i=1}^N C_{i,2} \\
&= K + r \cdot pk_\Sigma + \sum_{i=1}^N (-r \cdot pk_{KSN_i} + r_i \cdot pk_{DR}) \\
&= K + r \cdot pk_\Sigma - r \cdot pk_\Sigma + \sum_{i=1}^N r_i \cdot pk_{DR} \\
&= K + \sum_{i=1}^N r_i \cdot pk_{DR}.
\end{aligned} \tag{22}$$

The tuple (C'_1, C'_2) is returned to DR .

5.5.2. Ciphertext Acquisition

If DR is authorized, it can send a request for the encrypted document $[doc]_k$ to DSP . DR needs to provide the hash value $H(doc)$ and the transaction ID $rqID$ to DSP . DSP calls the function $DSC.checkRequest()$ to verify the legitimacy of DR 's request. If the request is valid, DSP will return $[doc]_k$ to DR . DR can then verify the integrity of the received document by computing the hash of $[doc]_k$ and comparing the result with that returned by the $DSC.getDataInfo()$ function.

5.5.3. Decryption

DR uses its private key sk_{DR} to decrypt (C'_1, C'_2) as follows

$$\begin{aligned}
C'_2 - sk_{DR} \cdot C'_1 &= K + \sum_{i=1}^N r_i \cdot pk_{DR} - \sum_{i=1}^N r_i \cdot B \cdot sk_{DR} \\
&= K + \sum_{i=1}^N r_i \cdot pk_{DR} - \sum_{i=1}^N r_i \cdot pk_{DR} \\
&= K.
\end{aligned} \tag{23}$$

Then *DR* uses F_{keyGen} to compute the symmetric key k from K . After decrypting $[doc]_k$ with k and obtaining the original document doc , *DR* can verify the integrity of doc by computing its hash and comparing the result with that returned by $DSC.getDataInfo()$.

For intuition, we depict the communication process between entities and the smart contract functions called during data sharing in the proposed system in Figure 4. It is worth noting that we only demonstrate the process that a document is uploaded by the data owner and shared with a data requester. In fact, after a document is uploaded, it may be shared multiple times. That is, independent sharing processes will be initiated by different data requesters, including the data query phase and the data acquisition phase, respectively.

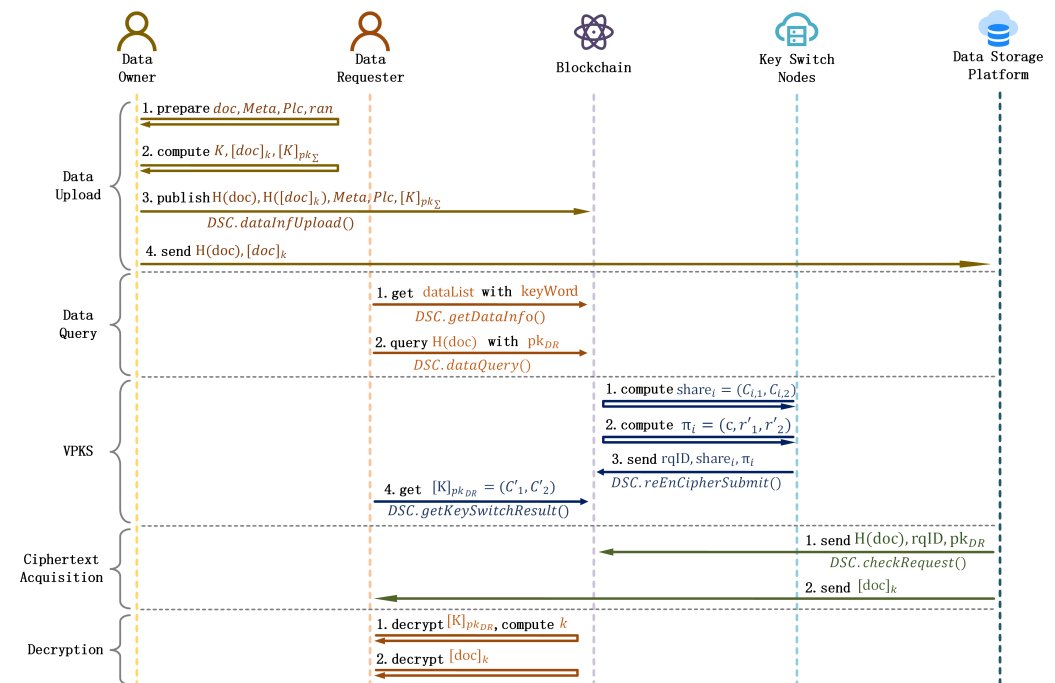


Figure 4. An illustration of the proposed data sharing process.

6. Security Analysis

In the above section, we have presented a detailed description of the proposed data sharing system. In this section, we analyze whether the proposed system, under the security assumption we've made in Section 4, can guarantee the confidentiality of the shared document and the auditability of the data sharing process.

6.1. Data Confidentiality

As mentioned in Section 4.3, a *DR* may try to obtain the document doc without getting proper authorization. It is assumed that *DR* can obtain the encrypted document $[doc]_k$ from the compromised *DSP* through the hash value $H(doc)$. Next the *DR* needs to obtain the symmetric key k . Since *DO* cannot be compromised, *DR* can not get k directly. The *DR* can try to get the pre-key K and then deduce k from K . The pre-key K is encrypted with the joint public key pk_{Σ} , and the ciphertext $[K]_{pk_{\Sigma}}$ is stored on the blockchain. Even without authorization from the smart contract, the *DR* can get $[K]_{pk_{\Sigma}}$ from the compromised *KSNs*. However, *DR* still requires the help of all *KSNs* to decrypt the ciphertext. Normally, *KSNs* will actively execute the *VPKS* protocol only if they receive notification of a successful authorization event. Without

proper authorization, the *DR* has to request the compromised *KSNs* to help with the decryption. However, different *KSNs* belong to different organizations, and it is assumed that the *DR* is not able to corrupt all *KSNs*. That is to say, the *DR* cannot get enough shares to convert $[K]_{pk_\Sigma}$ to a form that it can decrypt with its own private key. As a result, the *DR* cannot get the symmetric key k to decrypt $[doc]_k$.

6.2. Auditability

A *DR* may try to obtain a *DO*'s document without leaving a trace. We assume that the *DO* does not fully trust the *DR* and will not directly share the document with the *DR*. The *DR* can corrupt the *DSP* and obtain the encrypted document $[doc]_k$. Moreover, the *DR* can obtain information stored in the blockchain ledger by parsing the copy maintained by a compromised *KSN*. It means that the *DR* can obtain the encrypted pre-key $[K]_{pk_\Sigma}$ without invoking the smart contract function *DSC.getDataInfo()*. From the compromised *KSNs*, the *DR* can get some shares that are necessary for it to decrypt the ciphertext, without invoking the smart contract. But as described in the previous section, the *DR* cannot corrupt all *KSNs*. Hence, in order to get the symmetric key to decrypt the document, the *DR* must invoke the smart contract function *DSC.dataQuery()* to get the shares submitted by *KSNs*. The *DR*'s identity information and signature are recorded on the blockchain and cannot be deleted or tampered with. Hence the *DR* cannot deny that it has requested the document. After executing the *VPKS* protocol, each *KSN* invokes the function *DSC.reEnCipherSubmit()* to submit the shares to the blockchain. Since the *KSNs* are required to provide zero-knowledge proofs to prove they have correctly executed the *VPKS* protocol, it is guaranteed that these shares are sufficient to convert the ciphertext of the pre-key K into a form that the *DR* can decrypt with its private key. After the shares are recorded on the blockchain, *DR* can call the smart contract function *DSC.getKeySwitchResult()* to obtain the converted ciphertext, or get all the shares from the blockchain ledger held by some compromised *KSNs* and convert the ciphertext privately. In any case, the *DR* is assumed to be able to obtain the pre-key K , and hence the symmetric key k . The transactions TX_{share_i} generated by the function *DSC.reEnCipherSubmit()* not only imply that the *DR* obtains the symmetric key, but also indicate that the *DR* has been authorized and is able to get the encrypted document $[doc]_k$. Therefore, as long as all the N transactions $TX_{share_1}, \dots, TX_{share_N}$, each of which relates to one *KSN*, exist in the blockchain ledger, the *DR* cannot deny it obtain the original document.

In addition to *DR*, *DO* and *KSNs* can also be audited. The *DO* invokes the smart contract function *DSC.dataInfUpload()* to publish descriptions of the document and specify the access policy. With its signature being recorded on the blockchain, the *DO* cannot deny what it has published. Each *KSN* calls the function *DSC.reEnCipherSubmit()* to upload the share it has computed. By checking the signature contained in the transaction, one can learn the source of a given share.

7. Performance Evaluation

To evaluate the performance of the proposed system, we conduct a series of simulations. In this section, we first briefly describe how the proposed system is implemented. Then we provide the evaluation results and necessary explanations.

7.1. Experiment Setting

The blockchain network is implemented on Hyperledger Fabric (<https://www.hyperledger.org/use/fabric>, accessed on 12 October 2022). The network includes $M + 1$ organizations. One organization is responsible for providing the ordering service. There are three orderer nodes which run the Raft consensus algorithm. The remaining M organizations maintain the copies of the blockchain ledger. Each of these M organizations includes two peer nodes, one of which acts as the *KSN*. The smart contract is designed in Go language and instantiated in every peer node. We use IPFS (<https://ipfs.io/>, accessed on 12 October 2022) to fulfill the function of *DSP*. IPFS is a distributed storage system which enables

a user to retrieve a file with the hash of the file. We have built a local IPFS with four nodes. The cryptographic algorithms are implemented based on the SM2 standard. Two libraries, namely Go's native elliptic curve library (<https://github.com/golang/go/tree/master/src/crypto/elliptic>, accessed on 12 October 2022) and a public SM2 library (<https://github.com/tjfoc/gmsm>, accessed on 12 October 2022), are utilized. The system runs on a PC with Intel(R) Core(TM) i7-9700 of 3.00 GHz and 8 GB RAM (Lenovo, China).

As described in Section 5, the data sharing process in the proposed system consists of multiple phases, including the initialization phase, data upload phase, data query phase, and data acquisition phase. Each phase is simulated separately. Specifically, the simulations are conducted under different settings of parameters so that we can observe how the parameters influence the performance. Given a setting of parameters, the simulation is repeated 10 times, and the average evaluation result is reported.

7.2. Results and Discussion

In the initialization phase, we evaluate the generation time of each KSN's key pair and the corresponding KSC public key. We vary the number of organizations, and observe how the generation time changes. Given the number of organizations, KSNs generate key pairs in parallel, and then the KSN from a specified organization calculates the public key of the KSC. As we can see in Figure 5a, as the number of organizations increases, the generation time of the key pairs basically stays the same. The generation time of the KSC's public key increases approximately linearly with the number of organizations.

To simulate the data upload phase, we generate text documents of different sizes. As described in Section 5.3, each document is first encrypted and then published on the DSP. The encryption key is encrypted again and published on the blockchain. We evaluate the time spent on generating and encrypting the key (referred to as key encryption), the time spent on encrypting the document, the time spent on calling the smart contract function *DSC.dataInUpload()* to publish the encrypted key, and the time spent on publishing the encrypted document on the DSP. As shown in Figure 5b, as the size of the document increases, both the time spent on the key encryption and the time spent on publishing on the blockchain increase remarkably. The time spent on the key encryption barely changes and it is far less than that of other processes, while it takes about 2 s to publish the encrypted key on the blockchain, no matter what the size of the document. This publishing process starts from the smart contract function invoked, ends with the corresponding transaction being appended to the ledger. The time spent on this process is mainly affected by the consensus algorithm chosen by the ordering service.

To simulate the data query phase, we fix the number of organizations to 3 and the size of the document to 8 MB, and vary the number of attributes in the access policy. As described in Section 5.3, the proposed system utilizes attribute-based access control. We define the access policy as the conjunction of different attributes, namely $attr_1 \wedge attr_2 \wedge \dots \wedge attr_i$. We vary the number of attributes, and measure the time spent on the data query operation (off-chain) and time spent on the access control operation (on-chain), respectively. From the results shown in Figure 5c we can see that, compared with that of the access control phase, the time cost of the data query phase is negligible. For example, when there are 10 attributes, the time cost of the access control phase is 2208 ms, while that of the data query phase is only 0.11 ms. As described in Section 5.4.2, during the access control phase of the proposed system, the smart contract not only evaluates if the data requester's attributes satisfy the access policy, but also records the evaluation result on the blockchain ledger. In other words, the transaction corresponding to the access control phase involves both read and write operations on the ledger. The simulations are conducted on Hyperledger Fabric platform. According to the transaction flow specified in Hyperledger Fabric, processing a transaction involving write operations is more time-consuming than processing a transaction involving only read operations, and the transaction throughput heavily depends on the consensus algorithm. The Raft consensus algorithm is chosen in our simulations. As shown in Figure 5c it takes about 2 s to process the transaction

corresponding to access control. Moreover, from the simulation results we can observe that the time spent on the access control phase is barely affected by the number of attributes. The reason is that evaluating if the data requester's attributes satisfy the access policy occupies only a small part of the access control phase.

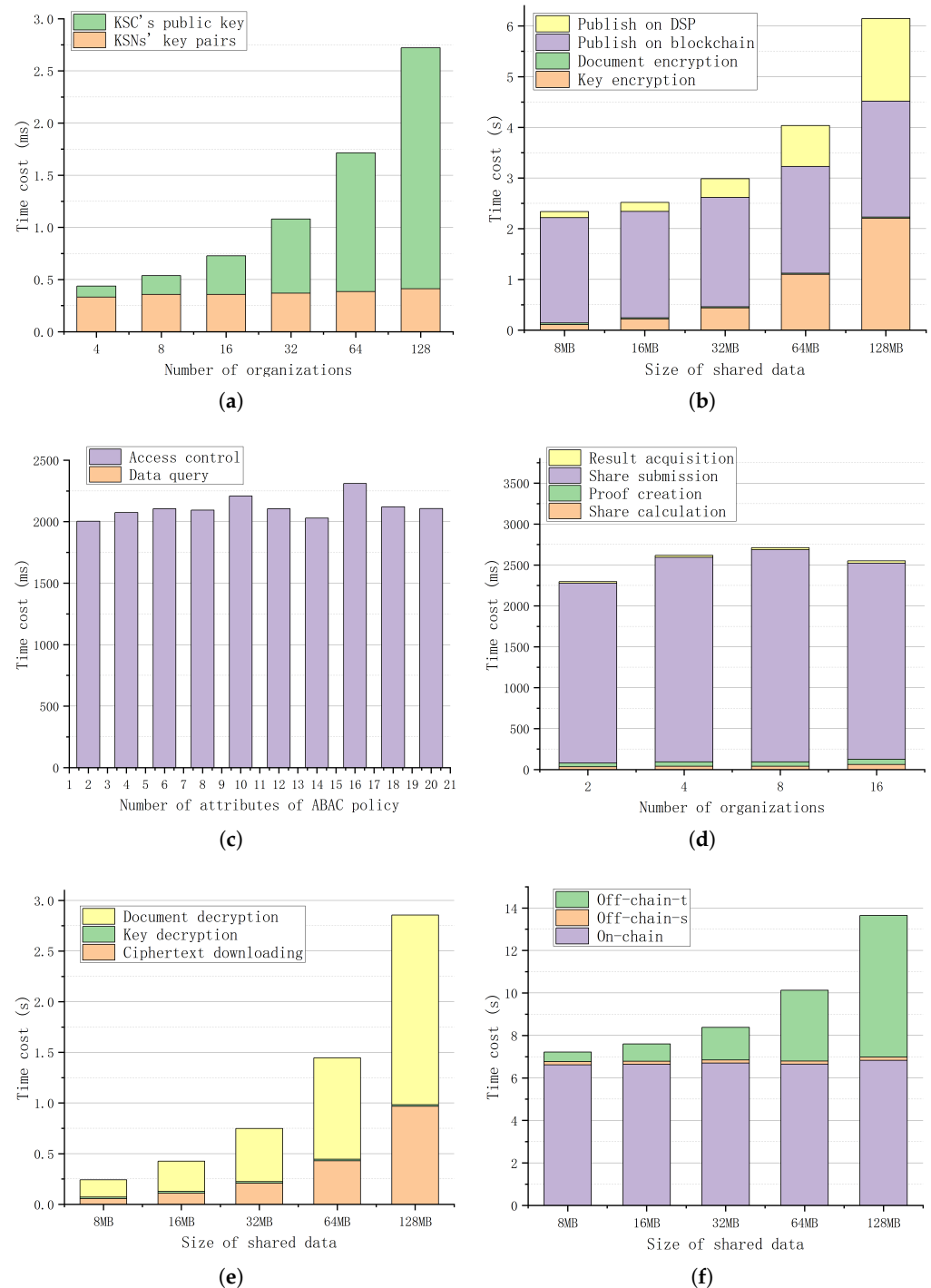


Figure 5. Performance evaluation results of the proposed system. (a) The key generation time under different settings of organization numbers; (b) The time cost of the data upload phase under different settings of data sizes; (c) The time cost of the data query phase under different settings of attribute numbers; (d) The time cost of the VPKS protocol under different settings of organization numbers; (e) The time cost of the ciphertext acquisition and decryption under different settings of data sizes; (f) The time cost of different operation categories under different settings of data sizes.

To evaluate the time cost of the proposed VPKS protocol, we vary the number of organizations, and we measure the time spent on the following four operations separately: share calculation, proof generation, share submission, and result acquisition. The first three operations are performed in parallel by KSNs, and from the results shown in Figure 5d we can see that, the time cost of these operations barely changes with the number of the organizations. The time cost of the result acquisition operation varies with the number of organizations, while it is far less than that of the share submission operation. Although both operations need to interact with the blockchain network, no transaction is recorded on the ledger during the result acquisition phase. For this reason, we consider result acquisition an off-chain operation.

To evaluate the time cost of the ciphertext acquisition phase and the decryption phase, we fix the organization number M to 4 and vary the size of the shared document. Given a document, we evaluate the time spent on downloading the ciphertext of the document from the DSP (referred to as ciphertext downloading), the time spent on decrypting the result of VPKS protocol and generating the symmetric key (referred to as key decryption), and the time spent on decrypting the ciphertext of the document (referred to as document decryption). As shown in Figure 5e, both the time spent on the ciphertext downloading phase and the time spent on the document decryption phase increase with the size of the document, which coincides with our intuition. The time spent on the key decryption phase barely changes, as the size of the pre-key, as well as the symmetric key, is fixed.

To better understand which operation has the largest influence on the performance of the proposed system, we divide all the operations mentioned above into three categories, namely on-chain operations, off-chain-t operations, and off-chain-s operations. The on-chain operations include invoking the smart contract function *DSC.dataInfUpload()*, access control and submitting the VPKS shares. The off-chain-t operations include encrypting the shared document, submitting the encrypted document to the DSP, downloading the ciphertext, and decrypting the ciphertext, which are necessary operations in any privacy-preserving cloud data sharing system. Other operations specific to the proposed system, including share calculation, proof generation, encrypting the pre-key, and result acquisition in VPKS protocol, are referred to as off-chain-s operations. As shown in Figure 5c,d, the number of attributes in the ABAC policy and the number of organizations have less impact on the time spent in the data query phase and the time spent in the VPKS protocol. Therefore, we fix the number of organizations to 8 and the number of attributes to 10, and vary the size of the shared document. As shown in Figure 5f, as the size of the shared document increases, the time cost of off-chain-s operations and on-chain operations barely change, while the time cost of off-chain-t operations increases remarkably. Specifically, when the size of the shared document is small, the time cost of the off-chain-t operations and off-chain-s operations occupy a small portion of the total time cost.

To further demonstrate that the efficiency of the proposed system is acceptable, we conduct a comparative experiment. Specifically, we simulate a PRE-based data sharing system which has a similar workflow with the proposed system. In the PRE-based system, the symmetric key of the shared data is generated by the DO's public key and two random numbers. The DO encrypts his public key and these random numbers as a ciphertext about the symmetric key. The DO publishes the access policy and the ciphertext about the symmetric key on the blockchain. If the DR satisfies the access policy, the smart contract will notice the DO to calculate the corresponding re-encryption key for the DR and publish it on the blockchain. Then a proxy server, which is not a KSN, re-encrypts the ciphertext about the symmetric key for the DR and publishes the result. The DR can decrypt this result to generate the symmetric key. As shown in Table 1, compared with the proposed system, the total time overhead of the PRE-based system is about 2 s longer. It is mainly because the PRE-based system requires the DO to perform an additional calculation and transaction publishing. That is, calculating the re-encryption key for the DR and publishing it on the blockchain.

The above evaluation results show that the performance of the proposed system is reasonable. As shown in Figure 5f, the on-chain operations are the most time-consuming in the proposed system. However, the time cost of on-chain operations mainly depends on the performance of the underlying blockchain platform. In future work, we will explore other blockchain platforms and investigate how to further improve the efficiency of off-chain operations.

Table 1. A time cost comparison between a PRE-based system and the proposed system. The number of attributes of the ABAC policy is set to 10. The number of organizations in the proposed system is set to 8. The evaluation is repeated 10 times and the average result is presented.

Data Size (MB)	Encryption (s)		Publication (s)						Data Request (s)	
			Blockchain		DSP					
	PRE	pro.	PRE	pro.	PRE	pro.	PRE	pro.		
8	0.108	0.139	2.113	2.082	0.118	0.118	2.029	2.094		
16	0.158	0.24	2.191	2.105	0.179	0.179	2.21	2.208		
32	0.321	0.458	2.186	2.164	0.378	0.368	2.12	2.105		
	ReKey Publication (s)		Re-Encryption/vpks (s)		Ciphertext Download (s)		Decryption (s)		Total (s)	
	PRE	pro.	PRE	pro.	PRE	pro.	PRE	pro.	PRE	pro
8	2.019	-	2.551	2.55	0.056	0.055	0.095	0.185	9.089	7.223
16	2.127	-	2.617	2.616	0.111	0.11	0.145	0.315	9.738	7.773
32	2.029	-	2.298	2.297	0.208	0.207	0.297	0.539	9.837	8.138

8. Conclusions

With the rapid development of information technology, cloud-based data sharing has become a common application. A number of cryptography-based solutions have been proposed to deal with the privacy issues in cloud-based data sharing. Some of them have utilized the blockchain technique to enhance the auditability. However, they cannot prevent the data requester from secretly obtaining the shared data by colluding with the cloud server or the blockchain nodes. In this paper, we propose a blockchain-based data sharing system with enhanced auditability. A cluster of blockchain nodes was introduced to execute a verifiable parallel key switch protocol. As long as at least one node does not collude with the requester, the requester has to trigger the VPKS protocol by publishing a query transaction on the blockchain ledger to get the decryption key of the shared document. Each node in the cluster also has to publish its calculation result on the ledger. That is, the requester cannot obtain the plaintext of the key secretly. The security analysis and the performance evaluation demonstrate that the proposed system is secure and practically feasible. In future work, we would like to enhance the protection of privacy during the access control phase, such as hiding the attributes of the requester, to adopt the proposed system to various application scenarios.

Author Contributions: Conceptualization, Y.X.; methodology, Y.X.; software, Z.C. and Y.X.; validation, L.Z. and L.X.; formal analysis, L.X. and Y.X.; investigation, Y.X.; resources, L.Z.; data curation, Z.C. and Y.X.; writing—original draft preparation, Y.X. and C.Z.; writing—review and editing, L.X. and L.Z.; visualization, Y.X.; supervision, L.X. and L.Z.; project administration, L.X.; funding acquisition, L.X. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Defense Basic Scientific Research program of China under grant number JCKY2018602B015 and number JCKY2019602B013, Beijing Municipal Natural Science Foundation under grant number M21035, National Natural Science Foundation of China grant number 61871037, and Shandong Provincial Key Research and Development Program under grant number 2021CXGC010106.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

G, B	Elliptic curve, base point on G
$[*]_k$	encryption using key k
$H(*)$	Hash function
DO, DSP	Data Owner, Data Storage Platform
KSC	The Key Service Cluster
KSN_i	Nodes in KSC
(pk_{KSN_i}, sk_{KSN_i})	Key Service Node i 's public-private key
$pk_{\Sigma} = pk_{KSN_1} + \dots + pk_{KSN_n}$	KSC's public key
(pk_{DR}, sk_{DR})	Requester's public-private key pair

References

- Ge, C.; Liu, Z.; Xia, J.; Fang, L. Revocable identity-based broadcast proxy re-encryption for data sharing in clouds. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 1214–1226. [CrossRef]
- Lu, Y.; Li, J. A pairing-free certificate-based proxy re-encryption scheme for secure data sharing in public clouds. *Future Gener. Comput. Syst.* **2016**, *62*, 140–147. [CrossRef]
- Han, J.; Susilo, W.; Mu, Y. Identity-based data storage in cloud computing. *Future Gener. Comput. Syst.* **2013**, *29*, 673–681. [CrossRef]
- Hur, J. Improving security and efficiency in attribute-based data sharing. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 2271–2282. [CrossRef]
- Shiraishi, Y.; Nomura, K.; Mohri, M.; Naruse, T.; Morii, M. Attribute revocable attribute-based encryption with forward secrecy for fine-grained access control of shared data. *IEICE Trans. Inf. Syst.* **2017**, *100*, 2432–2439. [CrossRef]
- Wang, S.; Liang, K.; Liu, K.; Chen, J.; Yu, J.; Xie, W. Attribute-based data sharing scheme revisited in cloud computing. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1661–1673. [CrossRef]
- Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2009. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=34408022009 (accessed on 12 October 2022).
- Gao, Y.; Chen, Y.; Lin, H.; Rodrigues, J.J.P.C. Blockchain based secure iot data sharing framework for sdn-enabled smart communities. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, 6–9 July 2020; pp. 514–519.
- Gao, Y.; Chen, Y.; Hu, X.; Lin, H.; Liu, Y.; Nie, L. Blockchain based iiot data sharing framework for sdn-enabled pervasive edge computing. *IEEE Trans. Ind. Inform.* **2020**, *17*, 5041–5049. [CrossRef]
- Zheng, S.; Pan, L.; Hu, D.; Li, M.; Fan, Y. A blockchain-based trading platform for big data. In Proceedings of the IEEE INFOCOM 2020—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Toronto, ON, Canada, 6–9 July 2020; pp. 991–996.
- Wang, D.; Zhang, X. Secure ride-sharing services based on a consortium blockchain. *IEEE Internet Things J.* **2020**, *8*, 2976–2991. [CrossRef]
- Xia, Q.; Sifah, E.B.; Agyekum, K.O.B.O.; Xia, H.; Acheampong, K.N.; Smahi, A.; Guizani, M. Secured fine-grained selective access to outsourced cloud data in iot environments. *IEEE Internet Things J.* **2019**, *6*, 10749–10762. [CrossRef]
- Liu, S.; Yu, J.; Xiao, Y.; Wan, Z.; Wang, S.; Yan, B. Bc-sabe: Blockchain-aided searchable attribute-based encryption for cloud-iot. *IEEE Internet Things J.* **2020**, *7*, 7851–7867. [CrossRef]
- Yuan, C.; Xu, M.; Si, X.; Li, B. Blockchain with accountable cp-abe: How to effectively protect the electronic documents. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017; pp. 800–803.
- Hei, Y.; Liu, J.; Feng, H.; Li, D.; Liu, Y.; Wu, Q. Making ma-abe fully accountable: A blockchain-based approach for secure digital right management. *Comput. Netw.* **2021**, *191*, 108029. [CrossRef]
- Froelicher, D.; Egger, P.; Sousa, J.; Raisaro, J.L.; Huang, Z.; Mouchet, C.; Ford, B.; Hubaux, J.-P. Unlynx: A decentralized system for privacy-conscious data sharing. *Proc. Priv. Enhancing Technol.* **2017**, *2017*, 232–250. [CrossRef]
- Froelicher, D.; Troncoso-Pastoriza, J.R.; Sousa, J.S.; Hubaux, J. Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3035–3050. [CrossRef]
- Elgamal, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory* **1985**, *31*, 469–472. [CrossRef]

19. Goldwasser, S.; Micali, S.; Rackoff, C. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing—STOC '85*; Association for Computing Machinery: New York, NY, USA, 1985; pp. 291–304. [CrossRef]
20. Goldreich, O.; Micali, S.; Wigderson, A. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS 1986)*, Toronto, ON, Canada, 27–29 October 1986; pp. 174–187.
21. Camenisch, J.; Stadler, M. Proof Systems for General Statements about Discrete Logarithms; Technical Report. 1997. Available online: <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/69316/eth-3353-01.pdf> (accessed on 12 October 2022).
22. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In *Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress)*, Boston, MA, USA, 11–14 December 2017; pp. 557–564.
23. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Yellick, J. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, Porto, Portugal, 23–26 April 2018.
24. Sax, M. Apache kafka. In *Encyclopedia of Big Data Technologies*; Springer: Cham, Switzerland, 2019.
25. Ongaro, D.; Ousterhout, J.K. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference*, Philadelphia PA, USA, 19–20 June 2014.
26. Szabo, N. Formalizing and securing relationships on public networks. *First Monday* **1997**, *2*. [CrossRef]