Privacy-Preserving Blockchain-Based Data Sharing Platform for Decentralized Storage Systems

Van-Hoan Hoang*†, Elyes Lehtihet*, Yacine Ghamri-Doudane†
*OODRIVE-Trusted Cloud Solutions, 75010 Paris, France.
v.hoang@oodrive.com; e.lehtihet@oodrive.com
†L3i Lab, University of La Rochelle, 17000 La Rochelle, France.
yacine.ghamri@univ-lr.fr

***Abstract*—Cloud-based storage services have been the dominating outsourcing solution for both individuals and organizations to share data digitally. Despite the advantages, users must rely on storage services for data confidentiality, data access control, user privacy, and data availability. Whereas data confidentiality can be protected by advanced encryption algorithms, the rest remain challenging. First, in existing centralized storage services, even though data access controls are mainly defined by data owners, they are maintained and enforced by the services, which can deny data retrieval requests of authorized users or allow requests of illegitimate users. Second, the identity of a user is often known to the services to verify its eligibility to access requested data according to the access control, thus making the user traceable in the system. More importantly, the lack of anonymity may make users reluctant to use such services in sensitive contexts. Third, a huge amount of data is daily generated and stored on a centralized party, simultaneously serving requests from many users, which may cause a collapse of the system during peak periods. To address all these concerns, we propose a privacy-preserving blockchain-based data sharing platform for the InterPlanetary File System (IPFS), a content-addressable peer-to-peer storage system. The platform allows protecting both user anonymity, data confidentiality, and provides high data availability due to being deployed upon the IPFS network.**

***Index Terms*—anonymous access control, decentralized data storage, ring signature, predicate encryption, hidden transaction.**

I. Introduction

Cloud-based storage services have been widely adopted for facilitating data storage and data sharing between remote users. Since data is stored in a centralized third party which is not fully trusted, there are four critical issues to be solved: (1) data confidentiality, (2) data access control, (3) user privacy, and (4) data availability. The advent of advanced encryption schemes, such as attribute-based encryption [1], provides a user-centric model, which can be used to encrypt data before outsourcing to protect data confidentiality. However, the others remain unsolved. First, outsourcing data to a cloud-based storage service partially deprives a data owner of the ownership. Users must rely on the service maintaining the consistency and operations in accordance with user-defined access control policy (2). However, due to legal reasons, for example, data censorship, the service can block access to data of an individual or a group of users. Even worse, the service can remove all data related to some specific users. Second, regarding user privacy (3), a user has to provide its identity to the service for verification before being granted access to the requested data. This, however, leads to the user being traceable in the system. The lack of user privacy enables the service provider to trace all activities of any targeted user or discover the relationship between data stakeholders, i.e., who shares data with whom. This is not desired in many scenarios, for example, in healthcare systems, in which patients prefer concealing their identities while sharing their sensitive Electronic Medical Records (EMR) to healthcare institutions. Observe that simultaneously preserving data confidentiality (1) and user anonymity (3) is also an efficient way of resisting data censorship. Indeed, from a philosophical point of view, we state that if storage nodes have knowledge about neither the content of data nor the real identities of data stakeholders, they will have no incentives to deny authorized parties access or accept unauthorized ones access to the data. Hence, the only way for storage service provider to remove data pertaining to a certain user is to destroy all data of the whole system that is, in most cases, not worth doing. The user anonymity is, however, not a trivial work in the context of data sharing where the data owner has to specify users whom it wants to share data with, and authorized users need to prove their right before accessing the data. Third, with the proliferation of applications, such as Internet of Things applications, where a huge daily volume of data is transferred, stored, and accessed by different actors, such a centralized approach seems insufficient to handle all data-related requests at peak times, thus decreasing data availability (4) or potentially leading to a single point of failure (SPOF).

With that being said, we need to rethink the way of efficiently storing, sharing, and processing data to put back data ownership in the hands of users as well as protect user and data security. This implies a powerful data censorship resistance mechanism. The emergence of the blockchain technology, which has offered many disruptive solutions in various industries, makes it a prominent solution in this context. In this paper, we propose a privacy-preserving blockchain-based data sharing platform for the InterPlanetary File System (IPFS). The deployment of IPFS [2], which inherits the advantages of many peer-to-peer systems, allows simultaneously retrieving data from multiple storage nodes, thus removing the risk of

ISBN 978-3-903176-28-7

SPOF and improving data availability.

Contributions: The platform is built on top of the public blockchain and includes three main contributions:

1) We propose a revocable predicate encryption scheme for a data owner to ensure data confidentiality while sharing data with other users. While the private key of each user is separately computed and given by the data owner to decrypt the data, the data owner is able to revoke the private keys at will. The scheme incorporates proxy re-encryption technique allowing delegating re-encryption task to the storage nodes in case that the data owner wants to revoke some users. This makes the revoked users unable to decrypt the re-encrypted data.
2) We propose a novel mechanism of hiding auditable data access control lists that are stored on the blockchain and fully managed by the data owners. A user can anonymously prove its access right over the shared data, without revealing its identity, to request a data retrieval.
3) We present the notation of hidden transaction that prevents adversaries from analyzing the sender and the recipient involved.

The remainder of the paper is organized as follows: We first present related work in Section II. In Section III, we describe the system and security models. Section IV outlines the overall architecture of the platform. Next, Section V details the main building blocks of the platform. We detail the platform in Section VI. We analyze the security and privacy of the platform in Section VII. In Section VIII, we evaluate its performance. Section IX concludes our work.

II. Related Work

There have been several works targeting user anonymity in the field of peer-to-peer content distribution network, such as Freenet [3] and Free Haven [4]. While Freenet allows data owners to encrypt data with their own names, Free Haven does not provide such an encryption mechanism to protect data confidentiality against storage hosts. Users in Freenet and Free Haven can query the networks to retrieve desired data. The routing protocols in these systems, which are used to pass requests to data hosts and return the data to requesters, allow protecting the identities of the data owners and the data retriever at the network layer. However, these systems do not support data access control mechanism, making data owners unable to restrict access to specific users. Therefore, user revocation, which happens when the data owners want to update access control lists, cannot be featured in such systems.

In the context of Cloud-based storage, Shen et al. [5] attempt to solve user anonymity by using group signature technique. In group signature, the group manager computes a master key (MK) and a system public key (PK). Each user joining the group receives a private key generated based on MK by the group manager. To access the shared data, a member creates a signature δ on behalf of the group using its private key. The storage provider verifies the validity of δ by using PK before granting the user access to the requested data. Verifying δ only allows the storage provider to determine whether it has been created by a group member, without revealing the real identity of the signer. This approach, however, has two main drawbacks. First, group members need to directly interact with the group manager to receive their private keys before being able to prove their data retrieval right. This constraint is eliminated in our system design and will be discussed in details later. Second, in this case, the data access policy is simplified by PK that is used to verify group signatures. Storing access control policy requires to fully rely on the storage service for enforcement and further updates.

The works in [6], [7], aim at blockchain-based access control solutions, but none of them rigorously takes user privacy into account. These simply solve data access auditability, meaning that the blockchain engages in recording access control lists and data access activities in a chronological order so that the data owner can audit these information later. However, trivially storing access control lists on blockchain, which is accessible to everyone, obviously violates privacy of data stakeholders. In [8], [9], the authors make use of Ciphertext-Policy Attribute-Based Encryption (CP-ABE), which allows directly enforcing access control lists into ciphertexts, to protect data confidentiality. The utilization of CP-ABE is promising but still falls short in preserving user privacy. Indeed, in CP-ABE schemes, a private key is associated with a set of attributes while a ciphertext is associated with an access policy. If a user's attributes match the access policy, the user is able to decrypt the data by using its private key. However, the ciphertext must be stored along with the associated access policy which indicates to the data stakeholders the right way of decrypting the data. In storage systems, access policies, however, allow deducing sensitive information about the data stakeholders, thus putting user privacy at risk.

III. System and Threat Models

A. System Model

The proposed platform includes 4 entities illustrated in Figure 1.

Blockchain: is a growing chain of blocks which are validated and chained together. Since being introduced in 2008 [10] as the underlying technology of Bitcoin, the blockchain technology has attracted great interest from both industry and academia. Later, Ethereum [11] appeared as the next blockchain generation, which allows for deployment of smart contracts. A smart contract is a self-executing computer program stored on the blockchain, and is run by the miners in a trustless way. Once being deployed on the blockchain, a

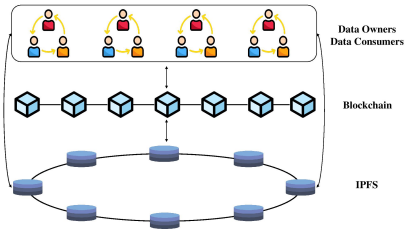


Fig. 1. The system model of the proposed platform

smart contract cannot be modified. This provides the users with integrity, transparency and autonomy guarantees. The proposed platform leverages the Ethereum blockchain to allow users to create smart contracts for data sharing management.
InterPlanetary File System (IPFS): is a peer-to-peer protocol [2] using content-based addressing technique to enable people worldwide to upload, download, and share data together in a fast and safe way. IPFS combines the advantages from the successful peer-to-peer systems such as Kademlia Distributed Hash Table, Bittorent, Git, and Self-Certified File system, thus making it a robust decentralized storage platform, providing quick data block retrieval. Recently, Filecoin [12] is developed to expand IPFS to a global decentralized storage. Specifically, Filecoin integrates an economic incentive mechanism into IPFS, allowing users to set out a price and rent out unused disk for storing data of others.
Data Owner: is any user that stores data on the IPFS. It has the ability of encrypting, anonymously delegating access rights, and revoking access over the data. Once some users are revoked, the data owner computes a re-encryption key and sends it to the storage nodes for re-encrypting the data. Thus, the revoked users are no longer able to decrypt the data.
Data Consumer: is anyone using data shared by data owners.

B. Threat Model

We formalize the security and privacy requirements that the proposed data sharing platform is supposed to achieve:

1) Data Confidentiality: Our model assumes that data is encrypted before outsourcing, making adversaries unable to decrypt it without the decryption key that is only known to authorized users. In case of revocation, the shared data is re-encrypted by the proposed proxy re-encryption technique, preventing the revoked users from decrypting the data. This is also known as forward-secrecy requirement.

2) User Privacy: Data owners and consumers anonymously share data in the system without anyone learning any personal information about them. This also disallows detecting whether two specific users have ever shared data.

3) User Linkability: This implies the impossibility of linking activities made by the same user. For example, one cannot say whether a user has accessed shared data several times or how many data sharings that a data owner has made.

4) User and Data Linkability: This requirement avoids linking any user with data shared in the platform.

IV. The Architecture

This Section highlights the main architectural building blocks constituting the platform as depicted in Figure 2.

On the user side, a user interacts with 4 following internal components to anonymously share data with others:
Data Sharing Engine: allows a data owner to specify the data consumers with whom it wants to share the data to construct as well as update the corresponding auditable hidden access control list $\mathcal{L}$, which is then stored on the blockchain. The platform is designed in an asynchronous model, meaning that the data consumers are not required to be active (online) when

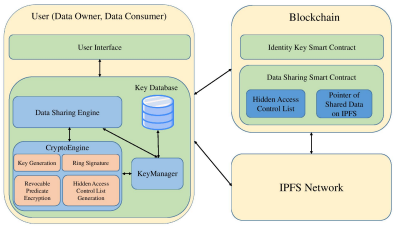


Fig. 2. The architecture of the proposed data sharing platform

the data owner shares the data. This component helps the data consumers to detect new data sharing from others using $\mathcal{L}$.
CryptoEngine: is responsible for all cryptographic operations in the platform. It computes hidden access control lists on requests of Data Sharing Engine and allows encrypting, decrypting data to protect data confidentiality. For data consumers, the component is used to issue proofs of eligibility to access shared data. Moreover, it also allows constructing hidden transactions among users to protect their identities.
KeyManager: is responsible for storing and retrieving keys from the database in an authenticated way.
Key Database: is a secure store of keys on the user side.

On the blockchain side, the platform has two main types of smart contract:
Identity Key Smart Contract: The platform has a common contract for users to register their identity public keys that are anonymously bind to their real identities. This reassures data owners about the existence of data consumers in the platform before sharing data with them. In practice, users exchange their public identity keys over a secure out-of-band channel before sharing data. In case that a user wants to change its own identity keys, it is required to sign the new public identity key with the old private identity key and send both the new public key and the resulting signature to the contract. The mining nodes verify the validity of the signature to update the user's new public key.
Data Sharing Smart Contract: A data sharing smart contract is created by a data owner to manage the location of the shared data on the IPFS and the hidden access control list that anonymously indicates who have the right to access the data. The data owner can audit and make changes to the hidden access control list over time. Data retrieval requests of data consumers are registered into the contract. These ensure the transparency and auditability of activities over the shared data.

V. The CryptoEngine and KeyManager Components

For ease of understanding the functioning of the platform, we describe in details the CryptoEngine and KeyManager which contain our main cryptographic contributions.

A. KeyManager

KeyManager involves in storing and retrieving keys from the Key Database. In the platform, we classify 4 types of key, each of which has a public key (PK) and a private key (SK):

- **Identity Keys** (iPK, iSK): Each user has one iPK and one iSK which are bind to its identity in the system.

Users exchange their public identity keys in a secure out-of-band channel to avoid violating their privacy.

- **Smart Contract Keys** (sPK, sSK): A user can have many smart contract keys, but one private smart contract key sSK is allowed to deploy only one smart contract.
- **Ephemeral Keys** (ePK, eSK): An ephemeral key is a one-time key used to sign all kinds of transaction, except transactions needing to be signed by sSK.
- **Hidden Keys** (hPK_{s-r}, hSK_{s-r}): The aim of hidden keys is to obscure the identities of participants in a transaction with respect to third parties.

B. CryptoEngine

The CryptoEngine plays a vital role in the platform due to its responsibility for all cryptographic computations. We now describe the functionalities that it supports:

1) Key Generation, Hidden Transaction and Hidden Auditable Access Control List:

a) Identity, Smart Contract, Ephemeral Keys Generation:

These keys are generated in the same way in Algorithm 1, which depicts the case of identity keys. The main difference among them lies in their usage purposes as outlined above.

Algorithm 1 Identity Key Generating Algorithm

Input: Given a cyclic group $\mathbb{G}$ of order p, generated by G.
Output: iPK, iSK

1: Chooses at random a number $x \in \mathbb{Z}_p^*$.
2: Sets $iSK = x$ and $iPK = xG$.
3: **return** iPK, iSK

b) Hidden Key and Hidden Transaction:

In any transactions, the platform aims to protect the identities of both the Sender ($\mathcal{S}$) and Recipient ($\mathcal{R}$). Observe that it is straightforward to protect the identity of $\mathcal{S}$ by allowing it to sign the transaction with an ephemeral key. However, preserving the identity of $\mathcal{R}$ is a non-trivial work. Intuitively, the transaction needs to be sent to a *random* address so that only $\mathcal{R}$ can determine that the transaction is intended for it. In our context, $\mathcal{R}$ also needs to know the identity of the sender, thus making the problem more challenging. For example, upon receipt of data sharing, a data consumer needs to know exactly the identity of the data owner to determine the corresponding PRE private key for decryption. Therefore, we make use of hidden keys and hidden transactions to effectively overcome the problem. We assume that a Sender ($\mathcal{S}$) wants to make a hidden transaction with a Receiver ($\mathcal{R}$). To this end, $\mathcal{S}$ first generates an ephemeral key pair ($eSK_s = e$ and $ePK_s = eG$). Next, $\mathcal{S}$ takes as input the public identity key iPK_r of $\mathcal{R}$ and eSK_s, then follows Algorithm 2 to obtain a public hidden key hPK_{s-r}. Next, $\mathcal{S}$ creates a transaction, signs it with the private ephemeral key eSK_s and sends it to the address of the resulting public hidden key hPK_{s-r}.

$\mathcal{R}$ needs to scan the blockchain to determine whether a new hidden transaction inserted into the blockchain is intended for it. That is, $\mathcal{R}$ must be able to correctly compute the corresponding private hidden key hSK_{s-r}. Given the public ephemeral key ePK_s of the hidden transaction, $\mathcal{R}$ uses its private identity key iSK_r to compute $hPK'_{s-r} = H(ePK_s.iSK_r)G + iPK_r = H(e.x_r.G)G + x_rG$. If ($hPK'_{s-r}$ = hPK_{s-r}), the hidden transaction is intended for $\mathcal{R}$, who can then compute the corresponding private hidden key, $hSK_{s-r} = H(e.x_r.G) + x_r$. In contrast, third parties cannot recompute hSK_{s-r} without iSK_r, thus they cannot identify the receiver of the transaction.

Algorithm 2 Public Hidden Key Generating Algorithm

Input: $eSK_s = e$ and $iPK_r = x_rG$
Let H be a collision-resistant hash function: $\{0,1\}^* \rightarrow \mathbb{Z}_p^*$
Output: hPK_{s-r}

1: Computes $z = H(e.x_r.G)$
2: Computes $hPK_{s-r} = zG + x_rG$
3: **return** hPK_{s-r}

In order for $\mathcal{R}$ to know the sender of the hidden transaction, $\mathcal{S}$ needs to encrypt its public identity key iPK_s with the public hidden key and embed it into the transaction. Hence, $\mathcal{R}$ computes the private hidden key to decrypt iPK_s.

c) Auditable Hidden Access Control List:

An auditable hidden access control list $\mathcal{L}$ is composed of a set of hidden public keys of all the data consumers. The data owner adds or revokes data consumers by adding or removing their public hidden keys from $\mathcal{L}$. A data consumer wanting to access the shared data has to issue a proof stating the existence of its hidden key in the list so that the proof reveals nothing about the consumer's identity. We achieve this by using a ring signature scheme which is presented in the following.

2) Predicate Encryption:

Ciphertext-Policy Attribute Based Encryption (CP-ABE) is an advanced cryptographic primitive deployed in many data sharing platforms [8], [9]. Despite its natural suitability for data sharing systems, CP-ABE incurs privacy issue. Specifically, in CP-ABE, user private key is associated to its attributes set (S), i.e., age, name, while data is encrypted with an access policy (P) that is composed of attributes and logical operators. Any user with an attribute set matching with P is able to decrypt the ciphertext. This mechanism thus allows for fine-grained access control over encrypted data. However, such an encryption scheme requires to store P along with the ciphertext to help the user to decrypt it correctly. However, storing P that contains user attributes on a dishonest storage node obviously violates the privacy of data stakeholders. To overcome the privacy issue of CP-ABE while still benefiting from its advantages, we build a CP-ABE scheme supporting hidden access policies from a revocable predicate encryption.

a) Introduction of Predicate Encryption:

Predicate Encryption (PE) is generalizing many advanced cryptographic primitives, such as Identity-Based Encryption and CP-ABE. We refer to [13] for its formal definition. Generally, in a PE scheme, a user private key SK_f is associated with a predicate f, whereas a ciphertext is associated with a set of attributes S. A user can decrypt the ciphertext by using its private key if S matches the predicate f. In [14], Katz et al. introduce a PE scheme supporting conjunctions, disjunctions, and inner product. Also, the authors prove that

the scheme ensures both payload-hiding and attribute-hiding requirements to protect both data confidentiality and privacy of attributes associated with ciphertexts. Thus, the scheme solves the privacy issue of CP-ABE. The construction in [14], however, lacks an efficient revocation mechanism to revoke users' private keys while necessary. This feature is indispensable in data sharing.

In this paper, we develop a new PE scheme, called Revocable Predicate Encryption (RPE), as an extension to [14]. The integrated revocation mechanism allows revoking private keys of users. We integrate an efficient proxy re-encryption technique into the scheme to fulfill the forward secrecy. Specifically, when a data owner revokes some users, it computes a re-encryption key and securely sends it to the storage nodes for re-encrypting the shared data. Thus, the revoked users are no longer able to decrypt the re-encrypted data even if the storage nodes are physically corrupted afterward. The computation of a re-encryption key is computationally lightweight. Therefore, this approach relieves the data owner from the workload of downloading and re-encrypting the shared data. We now detail our RPE scheme and then introduce a way of building a CP-ABE scheme supporting hidden access policies from RPE.

b) Revocable Predicate Encryption (RPE):

Definition 1. *The Lagrange interpolation allows rebuilding a polynomial P(x) of degree t from a set of* $(t+1)$ *points* $S = \{x_i, P(x_i)\}_{i=0..t}$ *through the following formula:*

$$P(x) = \sum_{i=0}^{t} P(x_i)\Delta_{i,S(x)}; \quad \Delta_{i,S(x)} = \prod_{i \neq j; j \in [0,t]} \frac{x - x_j}{x_i - x_j}$$

The RPE scheme consists of the following functions: **Setup**(1^k): Given a security parameter k, the system generates a cyclic group $\mathbb{G}$ of composite order $N = p.q.r$ where p, q, r are large prime numbers. Let $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ be groups generated by g_p, g_q, g_r, respectively. Let $\hat{e}$ be an efficient pairing function over $\mathbb{G}$ such that $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}$. Let H be a collison-resistant hash function such that $H : \{0,1\}^* \to \mathbb{G}$.

To generate the master key (MSK) and pubic key (PK), the system randomly chooses $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ and $h_{1,i}, h_{2,i} \in \mathbb{G}_p$. Next, it chooses $\gamma \in \mathbb{Z}_p^*$, $h \in \mathbb{G}_p$ and computes:

$$\begin{aligned} MSK =& (p, q, r, g_q, h^{-\gamma}, \{h_{1,i}, h_{2,i}\}_{i=1}^{n}) \\ PK =& (g_p, g_r, Q = g_q R_0, P = \hat{e}(g_p, h)^{\gamma}, \\ & \{H_{1,i} = h_{1,i} R_{1,i}, H_{2,i} = h_{2,i} R_{2,i}\}_{i=1}^{n}) \end{aligned} \tag{1}$$

To support revocation, the system first chooses a random number $z \in \mathbb{Z}_p^*$, then forms a random polynomial P of degree t, which allows simultaneously revoking t users:

$$P(x) = \sum_{i=0}^{t} p_i x^i \text{ where: } P(0) = z. \tag{2}$$

The system chooses a random number $a \in \mathbb{Z}_p^*$ then computes a master revocation key MRK, an initial unrevocation proof URP, a public revocation key PRK:

$$\text{MRK} = z;\ \text{PRK} = (g_p^z, g_p^a);\ \text{URP} = g_p^{az} \tag{3}$$

Encrypt(x, M, PRK): To encrypt a message m with a vector $x = (x_1, x_2, .., x_n)$, where $x_i \in \mathbb{Z}_N^*$, the algorithm randomly selects $s, \alpha, \beta \in \mathbb{Z}_N^*$ and $R_{3,i}, R_{4,i} \in \mathbb{G}_r$, computes:

$$\begin{aligned} C =& (C' = mP^s \hat{e}(g_p^z, g_p^a)^s, C_1 = g_p^s, \\ & \{C_{1,i} = H_{1,i}^s Q^{\alpha x_i} R_{3,i}, C_{2,i} = H_{2,i}^s Q^{\beta x_i} R_{4,i}\}_{i=1}^{n}) \end{aligned} \tag{4}$$

KeyGen(v, MSK): To generate a private key for a user associated with a vector $v = (v_1, v_2, .., v_n)$, the algorithm chooses random numbers $r_{1,i}, r_{2,i} \in \mathbb{Z}_p^*$, where $i = 1..n$. Then, it continues picking randomly $f_1, f_2 \in \mathbb{Z}_q^*$, $R_5 \in \mathbb{G}_r$ and $Q_6 \in \mathbb{G}_q$. It generates the private key for the user:

$$\begin{aligned} SK_v =& (K = R_5 Q_6 h^{-\gamma} \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \\ & \{K_{1,i} = g_p^{r_{1,i}} g_q^{f_1 v_i}, K_{2,i} = g_p^{r_{2,i}} g_q^{f_2 v_i}\}_{i=1}^{n}) \end{aligned} \tag{5}$$

The algorithm associates each user with a unique number num. It also sends to the user the initial unrevocation proof URP_{init} and an unrevocation key urk. The unrevocation proof URP is a complementary element used to decrypt data, and will be securely updated by the user using urk as soon as a new revocation happens. The urk of the user is computed:

$$urk = P(num) \tag{6}$$

SKeyUpdate(MRK, L): Suppose that the system wants to revoke n users associated to a list of n unrevocation keys L = $\{urk_i\}_{i=1..n}$. If $(n < t)$, the system picks $(t-n)$ random numbers j, such these numbers were and will be never used again, and computes $urk_j = P(j)$. Next, the system chooses $\tilde{a} \in \mathbb{Z}_p^*$ and computes an unrevocation update PU:

$$\text{PU} = (h^{\tilde{a}}, \{i, h^{\tilde{a} urk_i}\}_{i=1..t}) \tag{7}$$

UKeyUpdate(PU, urk_x): Given PU, an unrevoked user x uses its unrevocation key urk_x to update the unrevocation proof according to the Lagrange Interpolation. This function does not work for revoked users.

$$\text{URP}_{\text{new}} = g_p^{\tilde{a} z} = g_p^{\Delta_{x,\text{PU}(0)} . \tilde{a} . urk_x} \prod_{i=1}^{t} g_p^{\Delta_{i,\text{PU}(0)} . \tilde{a} . urk_i} \tag{8}$$

RKeyGen($\text{URP}_{\text{old}} = g_p^{az}$, $\text{URP}_{\text{new}} = g_p^{\tilde{a} z}$): To compute a re-encryption key, an unrevoked user takes URP_{old} and URP_{new} as inputs, then selects random numbers $X \in \mathbb{G}_{\mathrm{T}}, r \in \mathbb{Z}_N^*$ and computes a re-encryption key:

$$k_{re} = (g_p^{-az} H(X), g_p^r, X \hat{e}(g_p^{\tilde{a}}, g_p^z)^r) \tag{9}$$

Re-encryption(C, k_{re}): Using a re-encryption key, the storage node re-encrypts the ciphertext by converting the element C' in the original ciphetext C into C'_{re}:

$$C'_{re} = C' \hat{e}(g_p^{-az} H(X), C_1) = mP^s \hat{e}(H(X), g_p^s) \tag{10}$$

The re-encrypted ciphertext would be:

$$\begin{aligned} C_{re} =& (C'_{re} = mP^s \hat{e}(H(X), g_p^s), \\ & C_1 = g_p^s, C_r = g_p^r, C_x = X \hat{e}(g_p^{\tilde{a}}, g_p^z)^r, \\ & \{C_{1,i} = H_{1,i}^s Q^{\alpha x_i} R_{3,i}, C_{2,i} = H_{2,i}^s Q^{\beta x_i} R_{4,i}\}_{i=1}^{n}) \end{aligned} \tag{11}$$

Decryption($C/C_{re}, SK_v$, URP): Depending on the type of ciphertext, a user decrypts it by using its private key.
- To decrypt an original ciphertext C, the user follows the below formula:

$$\begin{aligned}\tilde{m} &= \frac{C'\hat{e}(C_1, K)\prod_{i=1}^{n}\hat{e}(C_{1,i}, K_{1,i})\hat{e}(C_{2,i}, K_{2,i})}{\hat{e}(\text{URP}, C_1)} \\ &= m\hat{e}(g_q, g_q)^{(\alpha f_1+\beta f_2)<x,v>}\end{aligned} \tag{12}$$

- To decrypt a re-encrypted ciphertext C_{re}, an unrevoked user uses the updated unrevocation proof URP_{new} to compute:

$$X = \frac{C_x}{\hat{e}(\text{URP}_{\text{new}}, C_r)} = \frac{X\hat{e}(g_p^{\tilde{a}}, g_p^z)^r}{\hat{e}(g_p^{\tilde{a}z}, g_p^r)} \tag{13}$$

Then, the user derives the message from the formula:

$$\begin{aligned}\tilde{m} &= \frac{C'_{re}\hat{e}(C_1, K)\prod_{i=1}^{n}\hat{e}(C_{1,i}, K_{1,i})\hat{e}(C_{2,i}, K_{2,i})}{\hat{e}(H(X), C_1)} \\ &= m\hat{e}(g_q, g_q)^{(\alpha f_1+\beta f_2)<x,v>}\end{aligned} \tag{14}$$

We observe that if $< x, v >= 0$, the user obtains $\tilde{m} = m$, the exact plaintext.

c) Constructing CP-ABE scheme supporting hidden policy: In a CP-ABE scheme, the ciphertext is associated with an access policy while the private key of a user is associated with its identifying attributes. Therefore, to construct a CP-ABE scheme supporting hidden policies from RPE, we only need a preprocessing stage to convert an access policy into an encryption vector x, and convert a set of attributes into a key vector v. If the set of attributes matches the access policy, implying that $< x, v >= 0$, the user with the private key related to v can decrypt the data encrypted with x. Due to the page limit, we refer to [15] for a simple conversion of attributes and access policies. In our data sharing platform, we employ CP-ABE, which is based on RPE and supports hidden policies, to allow users to share data.

C. Ring Signature

Ring signature is a type of digital signature that allows a user to sign a message on behalf of a group of users, which is arbitrarily formed by the signer at the signing time [16]. A verifier can only verify whether the signature comes from the group, without knowing exactly the identity of the signer. Therefore, ring signature provides users with anonymity guarantee whose extent completely depends on the ring size. More specifically, a signer takes its private key SK_s and all public keys $S = (PK_1, ..PK_s, ..PK_N)$ of all members in a group of size N to sign a message. A verifier checks the validity of the signature to determine whether the message was anonymously signed by a member of S. In this paper, we purposely use the ring signature scheme of Herranz et al. [17]. This scheme has been proven computationally efficient and secure in the random oracle model. Second, this scheme is based on Discrete Logarithm (DL) assumption, making it compatible with the current cryptosystem in the Ethereum blockchain and removing the need of deploying another cryptosystem on user side.

In the platform, the ring signature scheme is used by data consumers to issue proofs of eligibility to anonymously access the shared data. The intuition behind the proof is that given an auditable hidden access control list consisting of hidden public keys ($R = \{hPK_i\}$), an authorized data consumer issues a ring signature on a random nonce on behalf of R. While the resulting signature allows verifying the eligibility of the consumer, it allows neither learning the consumer's identity nor linking its sessions. The latter is achieved due to the randomness of ring signature, meaning that the signatures of the same consumer are uniformly random and unlinkable.

VI. Privacy-Preserving Blockchain-based Data Sharing Platform

In this Section, we detail the operations during data sharing in the platform.

A. User Registration

To use the platform, a user first needs to register with the public identity key smart contract deployed on the blockchain. To this end, it creates a wallet containing a public and private identity key (iPK, iSK) according to Algorithm 1. Then, the user sends iPK to the contract for registration. We emphasize that the identity keys represent the user identity and as such they should be carefully used not to violate user privacy.

B. Sharing Key Distribution

A data owner uses RPE to encrypt the data shared with data consumers. Therefore, it needs to share decryption keys to the data consumers for decryption. To this end, the data owner specifies a set of attributes representing each of them. Then, for each attribute set, it runs a **KeyGen** operation, as described in our RPE scheme, to obtain a RPE private key for the consumer. Simply sending a transaction, which includes the RPE private key, to the public identity key iPK_c of the consumer will obviously violate its privacy. Moreover, there might be many other data owners sending RPE private keys to the same consumer, making it an appealing target to attackers. To deal with the concern, the data owner creates a hidden transaction as described in Algorithm 3 and sends it to the blockchain. By scanning the blockchain, the intended consumer knows the arrival of a hidden transaction. It then computes the private hidden key hSK_{o-c} which is used to decrypt σ_1 and σ_2 to obtain the RPE private key and the public identity key iPK_o of the data owner. Then, the consumer inserts a pair of iPK_o and the RPE private key into its local

Algorithm 3 Constructing a key distribution transaction

Input: iPK_c and a RPE private key.
Output: A hidden transaction.
1: Computes a public hidden key hPK_{o-c} with consumer.
2: Creates an encryption σ_1 of iPK_o under hPK_{o-c}.
3: Creates an encryption σ_2 of the RPE private key under hPK_{o-c}.
4: Creates a transaction including (σ_1, σ_2) and signs it with a newly generated private ephemeral key eSK.
5: **return** The resulting hidden transaction.

storage for future usage. On the other hand, by analyzing the hidden transaction, third parties cannot determine the users involved in the transaction.

C. Data Uploading

Before uploading data to the storage network, a data owner first generates a smart contract key pair (sPK and sSK). It then creates and deploys a new smart contract by using sSK on the blockchain. In this contract, sPK is declared as the public key of the contract owner who has all rights over it. In principle, one smart contract only manages one upload. An upload is referred to as a set of data that is simultaneously encrypted under the same access policy using RPE. Then, the data owner can start uploading the encrypted data onto the IPFS and, in turn, receives the address where the data is stored. The address is a hash of the encrypted data and can also be used to verify the integrity of the data. Finally, the data owner sends a transaction signed by sSK to the smart contract to declare the address for future data retrieval.

D. Data Sharing

To share data with a set of data consumers, a data owner specifies a hidden access control list $\mathcal{L}_{hPK}$ corresponding to these consumers. Each element in the list contains two objects. The first is a public hidden key hPK_i corresponding to a consumer i, computed from Algorithm 2. The second is an encryption of the public identity key iPK_o of the data owner with hPK_i. This object is important for the data consumers to determine which RPE private key should be used to decrypt the data if they are in collaboration with many data owners. Then, the data owner creates a transaction to declare $\mathcal{L}_{hPK}$ in the contract. The data consumers scan the blockchain and analyze the public hidden keys to be aware of new data shared with them and who the owner is.

E. Data Retrieval

If a data consumer wants to access data associated with a smart contract, it needs to fulfill two following requirements:

- Proving that it is part of the associated hidden access control list $\mathcal{L}_{hPK}$ of the smart contract, without revealing its identity.
- Possessing a valid RPE private key to decrypt the data.

For the first requirement, an authorized consumer uses its private hidden key hSK_i, which the corresponding hidden public key is included in $\mathcal{L}_{hPK}$, to create a ring signature with all the public hidden keys in $\mathcal{L}_{hPK}$, then sends the signature to the storage nodes. In parallel, a transaction containing a data retrieval request and the ring signature is also made and sent to the blockchain miners that verify and insert the transaction into the blockchain. The storage nodes verify the validity of the ring signature, and send the encrypted data to the data consumer. Finally, these storage nodes create a transaction, and then send it to the blockchain miners for confirming that the data retrieval has been served.

F. User Revocation and Data Re-encryption

If a data owner wants to revoke a set of data consumers, it will first update all hidden address lists in its smart contracts by excluding the addresses related to the revoked consumers. Afterward, the consumers are no longer able to provide valid ring signatures corresponding to the updated hidden address lists for requesting data from storage nodes.

The platform also ensures that the revoked consumers are incapable of decrypting the encrypted data. To this end, the data owner runs a **SKeyUpdate** operation to obtain an unrevocation update PU that is signed with the private smart contract key sSK, and is sent to the contract. Upon receipt of PU, the legitimate consumers update their RPE private keys. The **UKeyUpdate** operation only works for legitimate consumers. The **UKeyUpdate** operation performed by the revoked consumers results in a failure. This procedure is to make the RPE private keys of the revoked users useless in decrypting the data shared after their revocation. Next, a re-encryption key k_{re} is generated by one of the legitimate consumers by carrying out a **RKeyGen** operation. This key is encrypted with the public keys of the storage nodes and is then sent to them for the re-encryption phase. The revoked consumers are then unable to decrypt the re-encrypted data.

VII. Security and Privacy Analysis

This Section proves that the proposed platform fulfills all the security and privacy requirements defined in Section III.

A. Data Confidentiality

1) Security Proof of the Revocable Predicate Encryption (PRE): To ease the security proof of the RPE scheme, we first define a Revocation Scheme, denoted by RS, as follows:
Setup: Let $\mathbb{G}$ be a cyclic group of order p, generated by a generator g_p. The system chooses two random numbers $a, z \in \mathbb{Z}_p^*$, and computes a master revocation key MRK = z, a public revocation key PRK = (g_p^z, g_p^a), and an unrevocation proof URP = g_p^{az}. Users knowing URP will be considered legitimate whereas the others are considered revoked.
Encrypt: To encrypt a message m, one first picks at random $s \in \mathbb{Z}_p^*$ and computes the ciphertext $C = (g^s, m\hat{e}(g_p^a, g_p^z)^s)$.
Decrypt: To decrypt the ciphertext, an authorized user uses its URP and computes:

$$m = \frac{m\hat{e}(g_p^a, g_p^z)^s}{\hat{e}(g_p^s, g_p^{az})}$$

For revoked users, who do not know URP, need to compute $\hat{e}(g_p^a, g_p^z)^s$ from a tuple of $(\hat{e}, g_p^z, g_p^a, g_p^s)$. However, the possibility of achieving that is negligible under the Decisional Bilinear Diffie-Hellman (DBDH) assumption [18]. Thus, the PS scheme is provably secure under the DBDH assumption.

The RPE scheme is developed on the PE scheme in [14], and can be thought of as a combination between the RS scheme and the PE scheme. The latter has also been proven secure, our proposed RPE is thus provably secure.

TABLE I
DEPLOYMENT COST MEASUREMENT OF SMART CONTRACTS

Contracts	Transaction Cost (gas)	Execution Cost (gas)	Total Deployment Cost (gas)
Identity Key Contract	641176	446080	1087256
Data Sharing Contract	1318303	951831	2270134
Ring Signature Contract	2442294	1802498	4244792

2) Forward Secrecy: This security requirement guarantees that revoked users are unable to decrypt the data shared with them in the past even in case that they can compromise the storage nodes and retrieve the encrypted data. In the platform, once some users are revoked access to an encrypted data, the data owner generates a re-encryption key and sends it to storage nodes to re-encrypt the data that only un-revoked users can decrypt by using their private keys. For revoked users to decrypt the re-encrypted data, they need to compute X from the element $C_x = X\hat{e}(g_p^{\tilde{a}}, g_p^z)^r$ by using related available information, a tuple of $(g_p, g_p^{\tilde{a}}, g_p^z, g_p^r)$. This is, however, unfeasible under the DBDH assumption.

B. User Privacy

To share data with a set of data consumers, the data owner $\mathcal{S}$ deploys a new data sharing smart contract with a random smart contract key pair (sPK and sSK). The privacy of the data owner is thus protected. To anonymously include a data consumer ($\mathcal{R}$) into the hidden access control list ($\mathcal{L}$), $\mathcal{S}$ generates a public hidden key (hPK_{s-r}) to $\mathcal{R}$. Hereby, $\mathcal{L}$ is composed of the public hidden keys of all the authorized data consumers, and is stored on the blockchain. We now prove that hPK_{s-r} does reveal any information about neither $\mathcal{S}$ nor $\mathcal{R}$. As described in Algorithm 2, hPK_{s-r} is computed based on an ephemeral key pair ($eSK_r = e; ePK_r = eG$) and the public identity key ($iPK_r = x_rG$) of $\mathcal{R}$ so that $hPK_{s-r} = H(e.x_r.G) + x_rG$. Therefore, to identify the data consumer who has been included into $\mathcal{L}$ by hPK_{s-r}, an adversary has to try the public identity keys $\{iPK_k\}_{k=1..N}$ of all N candidates to recompute the corresponding public hidden key hPK_{s-k} and compare it with hPK_{s-r}. However, given the available information ($ePK_r = eG, hPK_{s-r}$) and $\{iPK_k = x_kG\}_{k=1..N}$, it is computationally unfeasible to compute hPK_{s-k}. Indeed, given ($ePK = eG$ and $iPK_k = x_kG$), the adversary cannot obtain $e.x_k.G$, which is required in Step 1 of Algorithm 2, due to Discrete Logarithm Complexity assumption.

C. User Linkability

We prove that an adversary cannot link activities done by a specific user. For data sharing, a data owner uses a different smart contract key pair (sPK, sSK), which are one-time keys, to deploy a smart contract. This is, thus, impossible to link two contracts made by the same data owner. Similarly, in each data sharing, a data consumer i is represented by a hidden key pair (hPK_i and hSK_i), in which the public hidden key is stored on the blockchain and used to prove the right to access the shared data. The hidden key pair is generated randomly for every data consumer in every data sharing. In other words, if a data consumer is involved in N data sharing contracts, it will have N hidden key pairs, which are computationally indistinguishable, thus making data consumers anonymous across all the smart contracts. Within one smart contract, the adversary cannot distinguish the access sessions of the same user due to the randomness of the ring signature scheme. Hence, we conclude that it is impossible to link activities of the same user over the whole system.

D. User and Data Linkability

This is the result of the combination of the above security and privacy guarantees. Indeed, both the confidentiality of data and user privacy is protected, making it impossible to determine which user has accessed to which data.

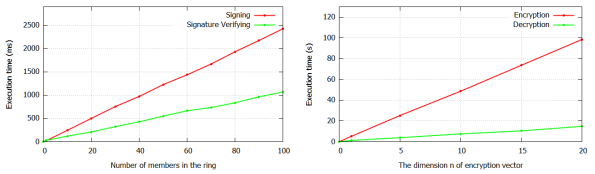
VIII. PERFORMANCE EVALUATION

In this Section, we evaluate the performance of the platform by measuring the computational costs of the computationally heavyweight operations performed by users as well as the costs of deploying the smart contracts on the blockchain.

A. Evaluation of the Smart Contract Deployment Cost

The Ethereum blockchain is maintained by miners that are responsible for verifying and adding transactions into the blockchain. Therefore, all operations must be paid to the miners to be executed on the blockchain. Specifically, *gas* is the unit of measurement that defines the amount of computational effort needed to execute an operation. To initiate a transaction, a sender must specify two variables: *gas limit* and *gas price*. The former indicates the maximum amount of *gas* that the sender is willing to pay for the transaction while the latter indicates the price of *gas*. The *gas* price is set based on ETH, the underlying cryptocurrency of the Ethereum blockchain.

In the platform, there are 3 main types of contract, including Identity Key contract, Data Sharing contract, and Ring Signature contract, which are written in the Solidity language. As explained above, the Identity Key contract allows new users to register to join the sharing system. To outsource an upload, which could be a set of data, the data owner deploys a Data Sharing contract that allows managing anonymous and auditable access control as well as registering the location of the data on the IPFS network. We separately deploy a Ring Signature contract which can be called from Data Sharing contracts to verify the eligibility of data consumers. The ring signature scheme is implemented on the curve ECSecp256k1, which is standardized by SECG [19]. We also open the source code of the smart contracts in [20]. The deployment costs of the smart contracts are illustrated in Table I.



(a) Execution time of ring signature

(b) Execution time of RPE

Fig. 3. Time execution measurement of the ring signature and RPE

B. Evaluation of Computationally Heavyweight Operations

On the user side, the main computational burden is caused by the cryptographic operations which are performed on their devices such as PC or mobile. We observe that the generation of keys or hidden access control lists requires a small number of lightweight operations, including additions and multiplications. Therefore, we concentrate on the computation costs of the heavyweight operations which can also vary and depend on the context: the ring signature and revocable predicate encryption schemes. With regard to the ring signature scheme, it consists of signing and verifying operations. While the former is performed by data consumers, the latter is performed by the blockchain miners and is implemented in the Ring Signature contract in [20]. Due to the huge computation capacity of the blockchain miners, the verifying operation can be done nearly instantly. However, for the completeness of the demonstration, we show the costs of both operations when being carried out on the experimental computer.

We implement the RPE and the ring signature schemes on a computer with Core i7 2.81GHz processor and 8Gb RAM. The ring signature scheme is implemented on the curve ECSecp256k1 as mentioned above whereas the revocation predicate encryption scheme is implemented on the curve $y^2 = x^3 + x$ over the field $\mathbb{F}_q$ for some prime $q = 3 \mod 4$. The latter allows performing pairing of type A1, the pairing over a group of composite order, which is the core building block of RPE. The performance of RPE is measured according to the dimension of encryption vectors whereas the performance of ring signature scheme is evaluated based on the number of members in a ring. As illustrated in Figure 3, the ring signature scheme is practical as it takes less than 2.5s to create a signature on behalf of a large group of 100 members. Moreover, the signature verifying algorithm respectively only requires 1s for such a signature. However, the revocable predicate encryption is slightly inefficient as encrypting a message with a 10-dimensions vector requires 50s and the decryption accordingly requires 7.5s. The inefficiency of RPE is due to the expensive cost of crytographic operations performed over a pairing group of composite order, such as pairing and modular exponentiation. In [21], Freeman develops a method to convert cryptosystems built on composite-order groups, including the original predicate encryption scheme [14], to prime-order groups to improve performance. Applying this method in RPE helps to greatly reduce the computational overhead. Indeed, on our given computer, performing a pairing over a prime-order group is around 33 times faster than the same operation over a comparable composite-order group (15ms versus 500ms). Similarly, 35 times is the improvement factor with regard to modular exponentiation.

IX. Conclusion

In this paper, we propose a privacy-preserving blockchain-based data sharing platform for decentralized storage systems. The platform allows protecting data confidentiality as well as user privacy in any activities in the system. Yet, the auditability of data access is guaranteed through hidden access control lists, which are stored on blockchain for public verifiability. These features make the platform a powerful solution to data censorship in the data outsourcing context. We also conduct an implementation of RPE and the ring signature scheme on the given computer to evaluate their performance.

References

[1] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy*. IEEE, 2007, pp. 321–334.
[2] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.
[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Designing privacy enhancing technologies*. Springer, 2001, pp. 46–66.
[4] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 67–95.
[5] J. Shen, T. Zhou, X. Chen, J. Li, and W. Susilo, "Anonymous and traceable group data sharing in cloud computing," *IEEE Transactions on Information Forensics and Security*, pp. 912–925, 2017.
[6] A. Ouaddah, A. Abou Elkalam, and A. Ait Ouahman, "Fairaccess: a new blockchain-based access control framework for the internet of things," *Security and Communication Networks*, pp. 5943–5964, 2016.
[7] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.
[8] J. Liu, X. Li, L. Ye, H. Zhang, X. Du, and M. Guizani, "Bpds: A blockchain based privacy-preserving data sharing for electronic medical records," in *GLOBECOM*. IEEE, 2018, pp. 1–6.
[9] S. Wang, Y. Zhang, and Y. Zhang, "A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems," *IEEE Access*, vol. 6, pp. 38 437–38 450, 2018.
[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
[11] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[12] "Filecoin: A decentralized storage network."
[13] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," in *TCC*. Springer, 2011, pp. 253–273.
[14] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *EUROCRYPT*. Springer, 2008, pp. 146–162.
[15] J. Lai, R. H. Deng, and Y. Li, "Fully secure cipertext-policy hiding cp-abe," in *ISPEC*. Springer, 2011, pp. 24–39.
[16] A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles," in *TCC*. Springer, 2006, pp. 60–79.
[17] J. Herranz and G. Sáez, "Forking lemmas for ring signature schemes," in *INDOCRYPT*. Springer, 2003, pp. 266–279.
[18] F. Laguillaumie and D. Vergnaud, "Designated verifier signatures: anonymity and efficient construction from any bilinear map," in *SecureComm*. Springer, 2004, pp. 105–119.
[19] M. Qu, "Sec 2: Recommended elliptic curve domain parameters," *Certicom Res., Mississauga, Canada, Tech. Rep. SEC2-Ver-0.6*, 1999.
[20] V. H. Hoang, https://github.com/vanhoanHoang/Privacy-Preserving-Data-Sharing-Platform.
[21] D. M. Freeman, "Converting pairing-based cryptosystems from composite-order groups to prime-order groups," in *EUROCRYPT*. Springer, 2010, pp. 44–61.