FileShare: A Blockchain and IPFS framework for Secure File Sharing and Data Provenance

Shreya Khatal, Jayant Rane, Dhiren Patel, Pearl Patel, Yann Busnel

FileShare: A Blockchain and IPFS framework for Secure File Sharing and Data Provenance

Shreya Khatal[1], Jayant Rane[1], Dhiren Patel[1], Pearl Patel[2], Yann Busnel[3]

[1] Veermata Jijabai Technological Institute, Mumbai, India
[2] Vidyalankar Institute of Technlogy, Mumbai, India
[3] IMT Atlantique, Rennes, France
khatalshreya@gmail.com, jayantrane811@gmail.com, dhiren29p@gmail.com, pearl207@gmail.com, yann.busnel@imt-atlantique.fr

Abstract. In this paper, we introduce FileShare - a secure decentralized application framework for sharing files and data provenance. It overcomes the integrity and ownership issues in the existing solutions for file sharing and data provenance. In proposed framework, a Decentralized Application (dApp) on top of Ethereum is responsible for user registration and for provenance purposes. Ethereum smart contract is used to govern, manage, and provide traceability and visibility into the history of the shared content from its origin to the latest version. It employs IPFS, a distributed file system, as its data storage layer, avoiding the pitfalls of centralized storage solutions. The proposed framework utilizes an in-built editor to view and modify files. The files will be stored in an encrypted form on IPFS and can only be accessed in the FileShare text editor. Modify and share operations performed on shared files are recorded separately to the blockchain, ensuring high integrity, resiliency, and transparency.

Keywords: Data Provenance, Blockchain, IPFS, Ethereum smart contract, Sharing file, Ownership

1 Introduction

Blockchain is seen as a distributed ledger that can be accessed globally by anyone to verify stored data with high integrity, resilience, credibility, and traceability. Distributed ledger technology can be used to write smart contracts which are self-executing contracts, and can be replicated, shared and supervised by a network of computers that run on blockchain. Smart contracts avoid middleman by automatically defining and enforcing rules and obligations made by the parties in the ledger. Blockchain, however is an expensive medium for data storage. For efficient storage of digital content, we can use InterPlanetary File System (IPFS) which is a peer-to-peer hypermedia protocol and distributed file system. Since IPFS is distributed, it has no single point of failure.

This paper presents a framework where digital content is shared in a secure, tamperproof environment and achieves provenance of read, modify and share operations on data. Our application is based on IPFS and smart contracts of Ethereum blockchain. Blockchain technology is utilized for access control of digital content and storage of provenance data. The proposed FileShare application ensures that the digital content

would only be accessible in the application and will not be available in the end-users' operating system.

Rest of the paper is organized as follows: Section 2 briefly explains underlying concepts and summarizes related work. Section 3 describes the proposed application's workflow. Section 4 discusses validation and analysis of FileShare application. Section 5 concludes the paper.

2 Background and Related Work

Morgan [1] presents the idea of using blockchain technology to prove the existence of a document using the timestamping concept. The legitimacy of the document can be verified, but this system is not focused about the authority of the owner on his/her document. IPFS [2] is the distributed and versioned file system which can connect many computing nodes with the same system of files and manage them by tracking their versions over time. Rajalakshmi et al [3] propose a model of academic research record keeping with access control methods. Nizamuddin et al [4] propose a solution that is based on using IPFS and smart contracts of Ethereum blockchain. Both of the papers mentioned above do not restrict duplication and piracy of the shared content as once the document is downloaded, it can be replicated and any other user can claim the ownership. Further any changes made to the document in users operating system in offline mode are not logged and hence ownership as well as originality is threatened.

Smart contracts provide an easy way to access the Ethereum blockchain written in a high-level coding language called Solidity [5]. To develop Ethereum smart contracts, Remix IDE [6] can be used, which is a browser-based IDE. Another one is the Truffle framework [7], which supports built-in smart contract compilation, linking, deployment and binary management. In order to run Ethereum decentralized apps in the browser itself, without running a full node, MetaMask [8] can be used. The above tools can be combined for an effective Ethereum decentralized application development.

Data provenance is very critical as it provides the history of the origins of all changes to a data object, list of components that have either forwarded or processed the object and users who have viewed and/or modified the object. Hasan et al [9] proposed SPROVE that protects provenance data confidentiality and integrity using encryption and digital signature. However, SPROVE does not possess provenance data querying capability. Ko and Will [10] proposed Progger which is a kernel-level logging tool which can provide log tamper-evidence at the expense of user privacy. Liang et al's [11] proposal ProvChain is a distributed, cloud-based data provenance architecture, which creates tamper-proof record of events by embedding the provenance records into the blockchain as transactions. All of the above methods have no restriction on piracy and hence possess a breach to integrity.

3 Proposed Application's Workflow

Figure 1 shows components of the proposed architecture. Users first register to the FileShare application. The registration details of the user are added to the Ethereum

blockchain by the application. After creation of the file in the application's inbuilt text editor, user decides if the file has to be made shared or public. If the file is supposed to be shared, the file owner provides the public key of recipients with whom the file has to be shared with.

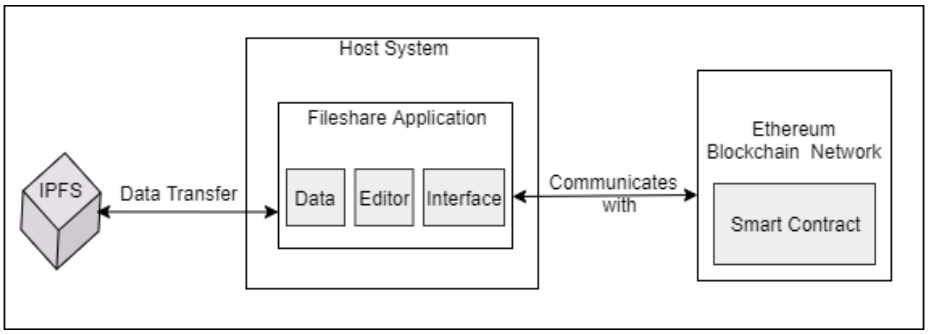

Fig. 1. Components of Proposed Architecture

The application then deploys a smart contract which stores the file metadata. It then encrypts the document and adds to IPFS in an encrypted format. In order to access the files, users are required to use the file sharing application editor as the file would be decrypted only in the application editor. The application uses the file smart contract to access the file metadata, fetches the file from IPFS, decrypts the file and opens in the inbuilt editor. In order to collect provenance data, we log call to functions of smart contracts as file operations performed in the editor. After an operation is performed, the record is generated, which will be uploaded to the blockchain network and stored in the provenance blockchain.

The framework proposed here can be divided into four main phases: User Registration and Authentication, File creation and storage, File retrieval, Provenance data collection and storage.

3.1 User Registration and Authentication:

Users are required to register to the system in order to have a unique identity. We propose to create a smart contract for every user, which will act as a unique identity for them. 'AllUsersMetadata' is a smart contract which acts as a factory to generates a smart contract for every user after their registration. During the registration process, user provides registration key in the form of a string as an input to the application. Using this registration key and current timestamp, application generates public-private key pair using ECDSA algorithm Now, AllUsersMetadata deploys a smart contract of the registered user and obtains address of the deployed smart contract. The deployed user's smart contract contains user's metadata which includes user's public key, registration key, an array of information details regarding the files which have been shared with the user.

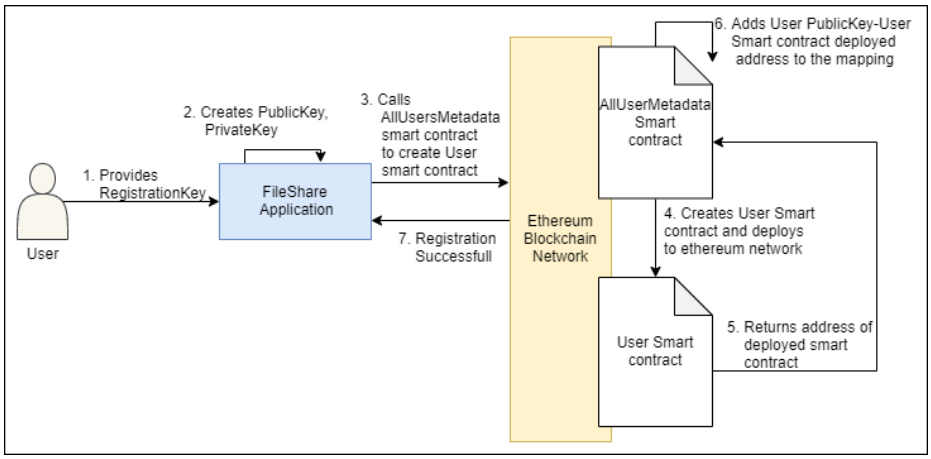

Fig. 2. FileShare system interaction for user registration and authentication

AllUsersMetadata also contains a mapping of every registered users public key to the address of their deployed smart contract. After the deployment of the user's smart contract, the received deployed address of the user's smart contract is added to the mapping in AllUsersMetadata smart contract. Public key generated during the registration process will be used by file owner while specifying recipient to whom the file must be shared with. While registration key and private key will be used to validate the user authenticity during the login process of the FileShare application.

For authentication, user will provide registration, public as well as their private key as an input to the FileShare application. Initially, registration key will get encrypted using private key, and generated string will be the 'EncryptedRegistrationKey'. Using the received public key as an input, user's smart contract deployed address will be fetched from the AllUsersMetadata's mapping. As the user's smart contract is fetched from the obtained address, to validate the user, the application will send the EncryptedRegistrationKey to validation function of the user's smart contract. Now the EncryptedRegistrationKey will be decrypted using public key of the user, and if resulting string is same as registration key of the user's smart contract, then user will be validated otherwise the authentication would fail.

3.2 File creation and storage:

Owner creates a file in FileShare application editor and requests for sharing this file on the FileShare application. FileShare now creates a random key, to encrypt the file using AES-256 symmetric encryption technique. This random key will be 'Secret Key' for given file which will only reside in owner's FileShare application. FileShare encrypts the file with the SecretKey. This encrypted file is added to the IPFS network. IPFS network returns hash of the uploaded file. As shown in figure 3, we propose to create a smart contract for every deployed file on IPFS. 'AllFilesMetadata' smart contract acts as a factory to generate smart contract for every file shared on the application. File's smart contract contains metadata which includes filename, IPFS address of the encrypted file and owner's public key. After deployment of the smart contract, FileShare

application will receive deployed file smart contract's address. Now, Owner can specify following types of access control for the specified file.

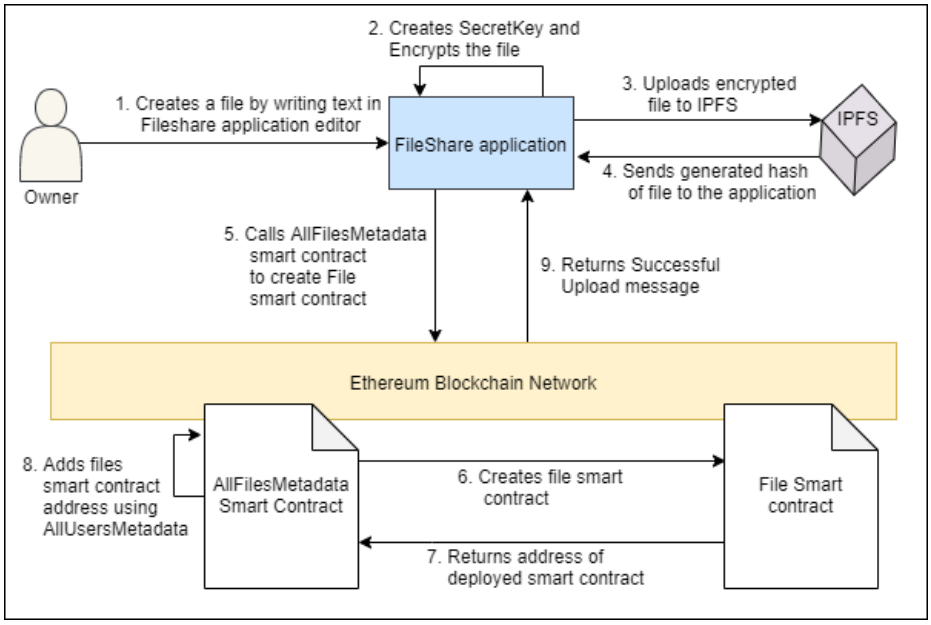

Fig. 3. FileShare system interaction for sharing files

Shared: In this access control, the owner can share the file to other users by using the public key of the user, they want to share the file with. After giving this public key to the FileShare application as an input, the application will encrypt 'SecretKey' of the file with 'Public Key' of the user with whom the file has to be shared with to create an 'EncryptionKey'. This is asymmetric encryption, whereas 'EncryptionKey' can only be decrypted by the user who has corresponding Private Key. Smart contract of the file, for shared mode contains a mapping of the receiver user's public key to the EncryptionKey of the file. This mapping will be added to the shared file's smart contract. AllFilesMetadata will access AllUsersMetadata to obtain deployed address of receiver's user smart contract. The shared files smart contract address will be added to the receiver's user smart contract. Thus, the receiver user smart contract will contain an array of deployed address of all the files which are shared with them.

Public: In this access control, the owner can share the file to every user who is registered on the FileShare application. Owner will specify the SecretKey in the file's smart contract. Also, owner will send their public key along with deployed file smart contract's address, to the AllFilesMetadata smart contract.

After these specifications are set to file's smart contract, other users will be able to access it if they are authorized of the FileShare application.

3.3 File retrieval:

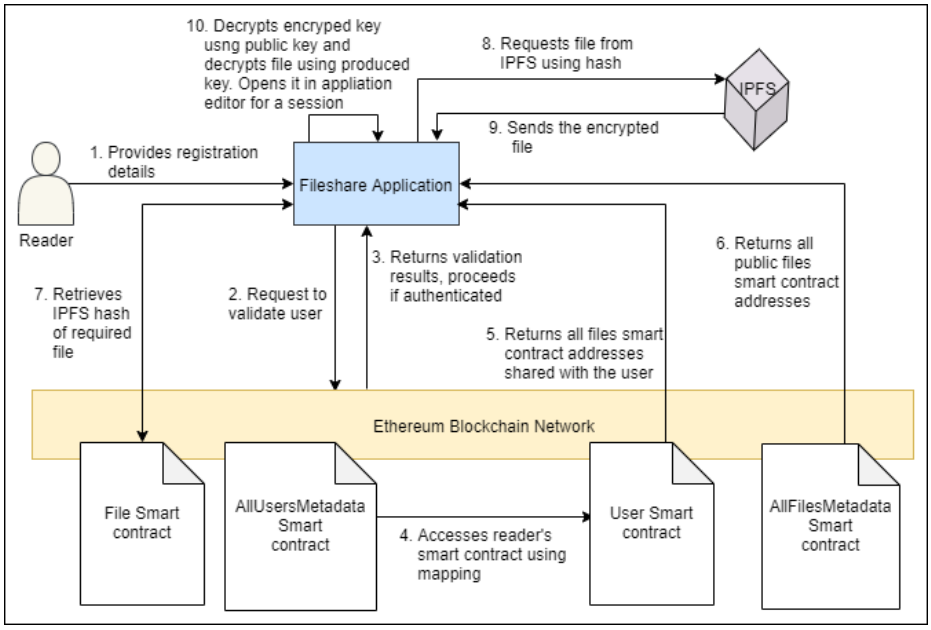

Fig. 4. FileShare system interaction for accessing files

On the FileShare application interface, after giving user's logging details such as registration key, public key and private key, application will retrieve user's deployed smart contract using AllUsersMetadata smart contract as shown in figure 4. If user is validated, then FileShare application will now access the user's smart contract using the AllUsersMetadata. The user smart contract contains the address of deployed smart contracts of all files shared with them. These files will appear on the application interface as 'Shared with me'. Application interface will also retrieve all files which are publicly shared using AllFilesMetadata smart contract. Following mechanism are performed for the proposed access control types:

Shared: Using the file's deployed smart contract address, FileShare application will retrieve key available in the mapping of publicKeyToEncryptedKey using user's own public key. Received key will then be decrypted by user's private key in the application, and generated key will be used to decrypt the accessed files by the FileShare application.

Public: Using file deployed smart contract address, FileShare application will request decryption key of file from corresponding file's smart contract deployed on blockchain. This key will be internally sent to the application and FileShare will decrypt the file and open in the application editor.

File accessed will be available to read for a session where session time would be a defined parameter. Also, file can be modified in FileShare application, which will be redeployed in application along with original owner's public key attached to it. The

uploaded content can only be accessed by using the application editor. The content cannot be downloaded nor be copied to clipboard of operating system, from the editor.

3.4 Provenance data collection and storage

Every time a user performs operations such as read, sharing files, it needs decryption key of the file. This key is available only in corresponding deployed smart contract. Whenever user request this key, smart contract logs these events in blockchain. The provenance data will contain the unique id of the user who has accessed the content, corresponding file's deployed smart contract address, time of access and type of operation accessed by user. For publishing data records to blockchain network, we adopt Chainpoint standard [12].

4 Validation and Analysis of FileShare application

Implementation details:

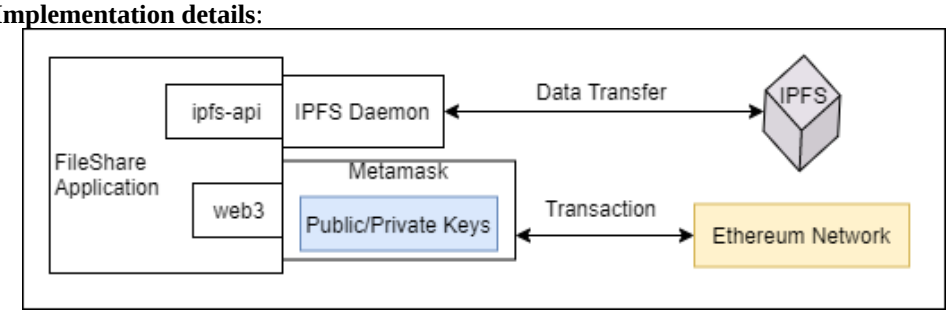

Fig. 5. FileShare Application Architecture

Figure 5 gives an overview of the FileShare application architecture. The Ethereum's Testnet Ropsten blockchain is used to store the user details as well as the audit logs. IPFS to get free hosting forever in a decentralized platform. React.js with webpack is used for the front end. Solidity 0.4.11 is used for developing Smart Contracts. web3.js is used to interact with Ethereum node, using a HTTP connection. We used Metamask to use the final application like the end user would. As our solution doesn't store any unencrypted data on blockchain, it is not prone to Ethereum blockchain hacks. The FileShare application achieves the following five objectives:

1. **No duplication of shared files:** As any shared file may it be in public or private mode can only be decrypted using the FileShare application, it cannot be downloaded in any end users operating system. Thus, no copies of the file exist.
2. **Real time data provenance:** The audit logs obtained include user information and the operations performed on the file may it be view, modify or share and is then added to the blockchain network making it tamper-proof.
3. **Tamper-proof Environment:** Data provenance record is collected and then published to the blockchain network which protects the provenance records.

4. **End to End traceability:** Users can access the provenance data to know the owner of the file, number of people who viewed the file and the edit operations on the file. Thus, no ownership problem occurs.
5. **Provenance Data Validation:** Data provenance record is published globally on the blockchain network. Thus, provenance data is validated by the blockchain nodes.

5 Conclusion

The FileShare application provides a secure, tamper proof model for sharing files in a distributed file system. The provenance data can be used to obtain analytical information. The file owners who made the file public can obtain analytical information about the number of people who viewed the file. The private file owners can keep accessing the modification operations performed by the users with whom the file has been shared. The owner of the files can be traced easily avoiding the ownership problems. Further, as the provenance data is stored in the blockchain, it creates an immutable record, and any malicious modifications to the provenance data can be prevented.

References

1. P. Morgan, “Using Blockchain Technology to Prove Existence of a Document”, last accessed 2018/2/20.
2. Benet, J. (2014). IPFS-content addressed, versioned, P2P filesystem. arXiv preprint arXiv:1407.3561.
3. Rajalakshmi, A. & Lakshmy, K.V. & Amritha, P.P.. (2018). A blockchain and IPFS based framework for secure Research record keeping. International Journal of Pure and Applied Mathematics. 119. 1437-1442.
4. Nizamuddin, Nishara & Hasan, Haya & Salah, Khaled. (2018). IPFS-Blockchain-Based Authenticity of Online Publications. 10.1007/978-3-319-94478-4_14.
5. Solidity — Solidity 0.4.23 documentation. (2018). Solidity.readthedocs.io. [Online]. Available: http://solidity.readthedocs.io/en/v0.4.23/.
6. Remix - Solidity IDE. (2018). Remix.ethereum.org. [Online]. Available: https://remix.ethereum.org/.
7. Truffle Suite - Your Ethereum Swiss Army Knife. (2018). Truffle Suite. [Online]. Available:http://truffleframework.com/.
8. MetaMask. (2018). Metamask.io. [Online]. Available:https://metamask.io/.
9. R. Hasan, R. Sion, and M. Winslett, “Sprov 2.0: A highlyconfigurable platform-independent library for secure provenance,” ACM, CCS, Chicago, IL, USA, 2009.
10. R. K. Ko and M. A. Will, “Progger: An efficient, tamperevident kernel-space logger for cloud data provenance tracking,” in 2014 IEEE 7th International Conference on Cloud Computing. IEEE, 2014, pp. 881–889.
11. Liang, X., Shetty, S., Tosh, D., Kamhoua, C., Kwiat, K., &Njilla, L. (2017, May). Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (pp. 468-477).IEEE Press.
12. “Chainpoint: A scalable protocol for anchoring data in the blockchain and generating blockchain receipts,” http://www.chainpoint.org/. last accessed 2019/11/08