Available online at http://www.advancedscientificjournal.com
http://www.krishmapublication.com
IJMASRI, Vol. 2, issue 12, pp.817-820, Dec. -2022
https://doi.org/10.53633/ijmasri

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY ADVANCED SCIENTIFIC RESEARCH AND INNOVATION (IJMASRI)

ISSN: 2582-9130 **IBI IMPACT FACTOR 1.5** **DOI: 10.53633/IJMASRI**

RESEARCH ARTICLE

A BLOCKCHAIN AND IPFS BASED DECENTRALIZED SECURE STORAGE FOR FILES

Rumeet Singh Kohli[1], M.L. Sharma[2] and K.C. Tripathi[3]

[1,2,3] Department of Information Technology, Maharaja Agrasen Institute of Technology affiliated to Guru Gobind Singh Indraprastha University, Rohini, Delhi
Email- rumeet.00114803119@it.mait.ac.in , mlsharma@mait.ac.in , kctripathi@mait.ac.in

Abstract

We have introduced a secure decentralised application for sharing files and information beginning. It overcomes the integrity and possession problems within the existing solutions for file sharing and information provenance. In planned framework, a decentralised Application on prime of Ethereum is liable for user registration and for beginning functions. Ethereum sensible contract is employed to manipulate, manage, and supply traceability and visibility into the history of the shared content from its origin to the most recent version. It employs IPFS, a distributed classification system, as its information storage layer, avoiding the pitfalls of centralized storage solutions. The planned framework utilizes IPFS. The files are going to be keep in an encrypted kind on IPFS and might solely be accessed within the application. Modify and share operations performed on shared files square measure recorded on an individual basis to the blockchain, ensuring high integrity, resiliency, and transparency.

Keywords: Blockchain, IPFS, Ethereum contract, Sharing file, Ownership

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 an application

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

Functional Overview

Users first register to the 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. 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 application proposed here can be divided into four main phases: User Registration and Authentication, File creation and storage, File retrieval, Provenance data collection and storage.

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. Our smart contract 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, application 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. Smart Contract also contains a mapping of every registered user's 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 the 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 application. For authentication, user will provide registration, public as well as their private key as an input to the application. Initially, registration key will get encrypted using private key, and generated string will be A content identifier, or CID. Using the received public key as an input, user's smart contract deployed address will be fetched from the smart contracts mapping. As the user's smart contract is fetched from the obtained address, to validate the user, the application will send the key to validation function of the user's smart contract. Now the key 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.

File creation and storage

Owner uploads a file in application and requests for sharing this file on the application. Application now creates a random key, to encrypt the file using AES-256 symmetric encryption technique. This random key will be identifier for given file which will only reside in owner's application. The app encrypts the file with the CID. This encrypted file is added to the IPFS network. IPFS network returns hash of the uploaded file. we propose to create a smart contract for every deployed file on IPFS. 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, application will receive deployed file smart contract's address. Now, Owner can specify following types of access control for the specified file. 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 application as an input, the application will encrypt the id of the file with public key of the user with whom the file has to be shared with to create an encryption. This is asymmetric encryption, whereas encryption 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 encryption of the file. This mapping will be added to the shared file's smart contract. Application will access smart contract 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 application. Owner will specify the key in the file's smart contract. Also, owner will send their public key along with deployed file smart contract's address, to the 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 application. uploaded content can only be accessed by using the application editor. The content cannot be downloaded.

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 chain point standard

Validation and Analysis of application

Implementation details:

The Ethereum's test net Kovan 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 is used for developing Smart Contracts. web3.js is used to interact with Ethereum node, using a HTTP connection. We used Meta mask 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 application achieves the following objectives:

1. As any shared file can only be decrypted using the application, it cannot be downloaded in any end users operating system. Thus, no copies of the file exist.
2. Data record is collected and then published to the blockchain network which renders the document tamper proof.
3. Data record is published globally on the blockchain network. Thus, ownership can be cross checked by the blockchain nodes.

Conclusion

The application provides a secure, tamper proof model for sharing files in a distributed file system. The 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 data is stored in the blockchain, it creates an immutable record, and any malicious modifications to the data can be stopped.

Reference

1. Lakshman, A and Malik, P. (2010). Cassandra: a decentralized structured storage system. ACM SIGOPS Oper. Syst. Rev., 44, 35-40.
2. Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp and Yiannis Psaras.(2022). Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web.
3. Rajalakshmi, A., Lakshmy, K.V and 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 and Salah, Khaled. (2018). IPFS-Blockchain-Based Authenticity of Online Publications. 10.1007/978-3-319-94478-4_14.
5. Hasan, R., Sion, R and Winslett, M. (2009). "Sprov 2.0: A highly configurable platform-independent library for secure provenance," ACM, CCS, Chicago, IL, USA, 2009.
6. Ko, R.K and Will, M.A. (2014). "Progger: An efficient, tamper evident kernel-space logger for cloud data provenance tracking," in 2014 IEEE 7th International Conference on Cloud Computing. IEEE, 2014, pp. 881–889.
7. Liang, X., Shetty, S. Tosh, D. Kamhoua, C. Kwiat, K and Njilla, L. (2017). Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (pp. 468-477)
8. "Chainpoint. (2016). A scalable protocol for anchoring data in the blockchain and generating block chain receipts," http://www.chainpoint.org/.

*****