2022 International Conference on Advanced Creative Networks and Intelligent Systems (ICACNIS) | 979-8-3503-3444-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICACNIS57039.2022.10055538

Implementation of Blockchain and Peer-to-peer Network for Digital Document Management

Bagas Fadillah Islamay
School of Electrical Engineering
Telkom University
Bandung, Indonesia
bagasislamay@student.telkomuniversity.ac.id

Yudha Purwanto
School of Electrical Engineering
Telkom University
Bandung, Indonesia
omyudha@telkomuniversity.ac.id

Muhammad Faris Ruriawan
School of Electrical Engineering
Telkom University
Bandung, Indonesia
muhammadfaris@telkomuniversity.ac.id

***Abstract*—Digital document storage usually implemented using conventional methods, namely by using a centralized database system that allows trust issues to arise due to the possibility of manipulation or interference from irresponsible parties. Therefore, we need a secure system that can prevent things like this from happening. For that, blockchain can be a solution to this problem. By using a blockchain-based system, the digital document management process can be carried out more transparently by only using the read and write method so that stored documents can still maintain their legal value. The end of this research results in the form of a decentralized application for digital document processing. Based on the results of white box testing, it is found that the web interface and smart contracts can run 100% however, the continuity of transactions is strongly influenced by several variables such as the computational ability of Rinkeby test network and the number of transactions that are executed at the same time. Based on the results of the black box test, the average file upload speed was 40.61 KBps with the component specifications as written in this journal. Furthermore, it was found that modifying 1 bit of data in a file would make the file identified as a unique file despite the content presented has not changed.**

***Keywords*—Digital Documents, Management, Blockchain, Ethereum**

I. Introduction

Several problems were found related to the mechanism of how digital documents are stored and managed. More to be precise, it concerns how to maintain the legal value of a document while it is stored somewhere.A number of existing applications implement storage with a centralized conventional database system that has a weakness, which is the vulnerability of manipulation or alteration of documents stored by certain parties either intentionally or unintentionally. For this reason, a new method is needed to overcome the weaknesses of this conventional system. Blockchain with a read and write method is one solution for address this transparency issue. Furthermore, the additional implementation of InterPlanetary File System (IPFS) can be used as a container for storing and sharing documents. This research is aimed at designing and creating web-based document management applications using the blockchain system and peer-to-peer network protocol.

The purpose of this research is to find out how to create and implement blockchain on blockchain-based digital document management application and know how to connect applications and store documents on a peer-to-peer network. The form of blockchain which used is Ethereum on the Rinkeby test network, the peer-to-peer network used is InterPlanetary File System (IPFS), document management operations include create, read, and update operations and documents validated based on the hash value provided by IPFS. The research method begins by reviewing several literature studies, followed by designing the system, testing and ends with the writing of this report. In this paper, we implement a document management system using Ethereum blockchain in Rinkeby test network and InterPlanetary File System or IPFS. Ethereum blockchain used to record all the operations that happens in the system as well as maintaining user management through Metamask digital wallet while IPFS used as a storage or a medium for users to store their documents. The storing process involves bl or BufferList package to convert the file into its buffer before it gets uploaded into IPFS with the help of ipfs-api. Finally, the system utilise React.js framework to serve the more user-friendly interface to be able to provide a better user experience.

II. Related Works

The first is the research of N. Nizamuddin, K. Salah, M. Ajmal Azad, J. Arshad and M.H. Rehman [1]. This research was conducted to present solutions related to the problem of centralized version control by creating applications collaborative version control with decentralized networks. Development done using Ethereum and IPFS. This research describes the concept and implementation of IPFS in their design which is relates to our research that also take advantages of IPFS. Further research from Adrian-Tudor Pãnescu and Vasile Manta [2]. This research is intended to design a decentralized database as a medium for sharing research data with protected rights using Ethereum. Application development does not explain the feasibility testing process thus allowing for system vulnerabilities. On the other hand there is also a research from Xingya Wang, Haoran Wu, Weisong Sun and Yuan Zhao [3]. The research conducted is not a decentralized application development, rather a test-suite development for application testing purposes which uses cost. This research examine on how to use gas efficiently during the design of a smart contract.

Furthermore, research by Dan Chirtoaca, Joshua Ellul and George Azzopardi [4]. Researching how to create a framework for deploy smart contracts for applications using non-fungible tokens. In this study, it is explained how to create a deployable framework but has the disadvantage that the OOP model is too complicated. A very interesting research that helps us to find out more about the deployment phase of a smart contract. Next is the research by Xuan Luo, Wei Cai, Zehua Wang, Xiuhua Li, C. M. Victor Leung [5]. Discussing how to exchange tokens on the Ethereum network. In this research, it is explained how the schema token exchange with data transport on the Ethereum network. Exchange flow This helps us in understanding the concept of data transportation on the network Ethereum. Then research by Prathmesh Deshmukh, Shreyas Kalwaghe, Ajinkya Appa and Aprupa Pawar [6]. Explaining an application design freelance without intermediaries, it is used to eliminates third party intervention who can play prices and payouts. This application is run on testnet is in development state. The use of testnet itself can be used to test a decentralized application before running it on the real environment on the mainnet. To further hone the basic understanding of the blockchain concept, the researcher reviews a paper by Dejan Vujičić, Dijana Jagody and Sinia Ranđić [7]. The paper written contains a brief explanation of blockchain, bitcoin and ethereum.

Turning to application securities, the author reviews a paper by Maximilian Wohrer and Uwe Zdun [8] who discussed the security of Ethereum and Solidity. On From the results of this study, several basic principles or pillars of security design are described on smart contracts. This principle can be implemented when designing our applications. Next is the research of Zihan Zou, Yan Xiong, Wenchao Huang and Lu Ma [9]. The results of the paper describe a method for pruning the solidity code by using artificial intelligence which is then packaged into an application called "SPrune". Although it cuts the code, but this app still follow the security principles as described in several papers before. Finally, research from Christopher G. Harris [10]. In this study, the author explains several bugs that usually appear when developing applications. Several tests are also described in this paper. Furthermore, the results of this paper can be used as a guide during the development phase of any decentralized application.

III. Implementation

This research will examine the implementation of blockchain and peer-to-peer network for web-based digital document management application. So it needs a front-end web interface which then connected to a back-end in the form of a smart contract written in a contract-based programming language which is Solidity. The web interface will also connect the application to the Rinkeby test network and InterPlanetary File System (IPFS). Users have access to be able to store, view, retrieve and transfer ownership of their digital documents.

The following is an illustration of the internal system workflow:

A. Software Design

Software design includes the creation of smart contracts and the creation of web interfaces. The software has several main features, namely the store feature, the ownership feature and

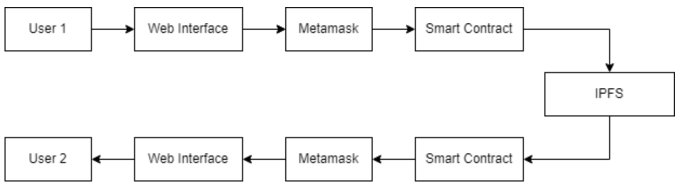

Fig. 1. Block Diagram

the transfer feature. The store feature is a feature for uploading files to IPFS which is both a create operation on the application and write operation on the blockchain. The ownership feature can check the ownership of the uploaded file while allowing users to be able to retrieve files that have been saved, this is a implementation of read operations on the application and on the blockchain. The transfer feature allows users to be able to make transfers ownership of files that have been previously saved, this is the implementation of the update operation on the application and write operation on the blockchain.

In the Store feature, system flow starts from the Metamask check for account selection. Furthermore, the user can perform store operation by affixing the file to be the form from web interface, the application then converts the file into a buffer, user confirms the transaction and proceeding with storing the file to IPFS. As output, the user will get information in the form of file hash and transaction hash. The flow of the store feature is illustrated by Fig. 2.

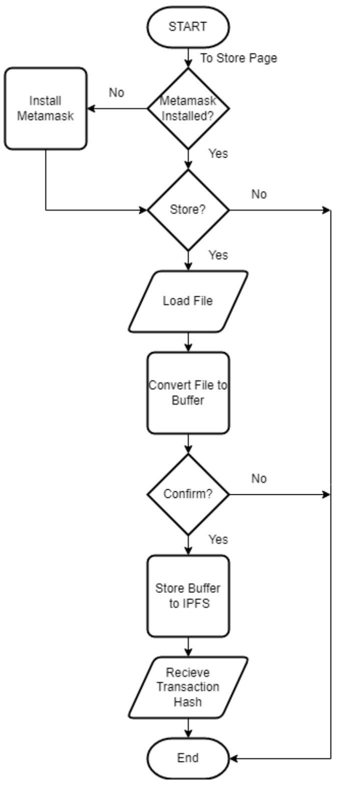

Fig. 2. Store Feature Flowchart

Furthermore, the functionality of the Ownership feature is also started by checking the Metamask account. Followed by the choice of whether the user wants to check the ownership of a file or not. If so, the user can add the file that he wants to find out the owner of, but if the user does not want to check the ownership then the user can continue with the file retrieval feature. In the file retrieval feature, users only need to click the “Retrieve” button to retrieve the files they own. The flow of the ownership feature is illustrated by Fig. 3.

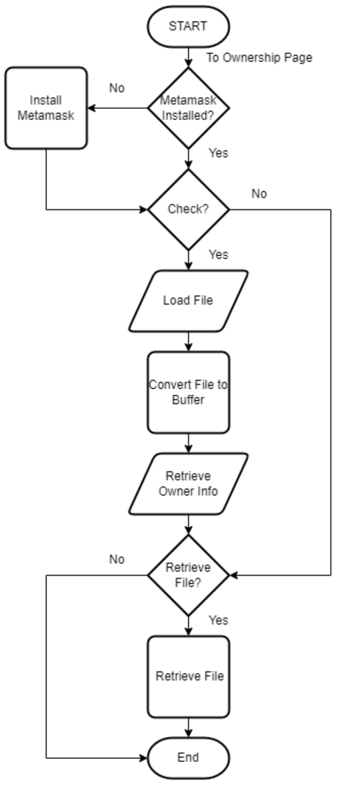

Fig. 3. Ownership Feature Flowchart

As in the previous feature flow, the Transfer feature also starts with a Metamask account check. Then users who wish to transfer ownership of their files can provide information on the destination account to be transferred, followed by confirming the transaction. The end result of this feature is the loss of the file entry that has been transferred and the receipt of the transfer transaction hash as proof that ownership has been transferred. The flow of the ownership feature is illustrated by Fig. 4.

B. Smart Contract Design

Smart contracts consist of several Solidity code files with the concept of linear contract inheritance where each contract is broken down into a single code file and passed down from parent to child to the main contract. There are 5 constructor contracts and 1 main contract. The main contract has all the properties of the 5 other constructor contracts. The six contracts are:

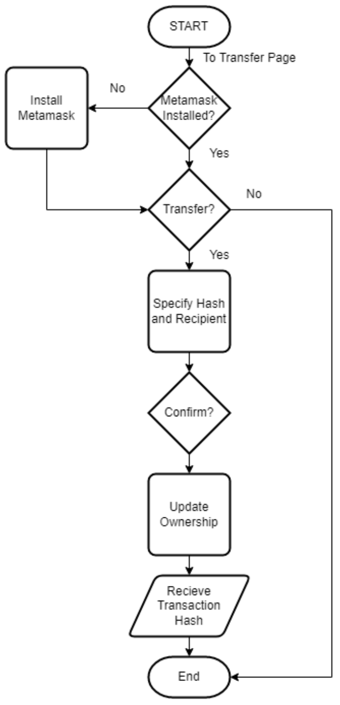

Fig. 4. Transfer Feature Flowchart

- Ownable, is a contract that guarantees the ownership of the contract. This contract also determines which account is doing the migration or deployment.
- Storage, is a constructor contract that describes what information will be written to the blockchain. This contract defines a struct of the file to be uploaded, the data contained in this struct includes the file name, hash, owner and the date and time the file was saved. This contract also includes several mappings that allow users to interact with files owned by them.
- StoreFile, the contract that performs the logging of file storage on IPFS. This contract has a single function, namely the Store() function. The Store() function also has several security feature implementations which are feature to check for replica files and feature to check whether the user has saved the same file before.
- GetFile, is a contract that checks the relationship between a file and its owner. It has 4 functions that perform read operations on the blockchain namely getFilesByOwner(), getOwnerFileDetails(), checkFileExistByHash() and getFileIdByHash() which are self-explanatory. Calling these 4 read functions does not cost money because they do not perform changes to the blockchain.
- FileOwnership, this contract defines the standards and procedures for transferring ownership of files which owned by the user. In short, the transfer of ownership is done by changing the owner variable on file struct that is defined in the Storage contract.
- PlanetOfDocs, is the last contract that inherits all the properties of the previous contract. This contract is empty

and only created for deployment purposes.

C. Web Interface Design

The web interface is designed using the React.js front-end framework using class component style. There are 6 main pages, namely the Home, Store, Ownership, Transfer, Me and NotFound page. Meanwhile, variable management will be done through the setState() feature in React.js. To authenticate users, defining contracts and defining contract addresses can be done by creating a hook interval that check the user address through Metamask using web3.js which will asynchronously make adjustments over a period of time.

D. Contract Migration and Serving Web Interface

Migration is done by configuring contract migration on the Rinkeby test network. After contracts migrated successfully, Truffle Suite will automatically pollute the PlanetOfDocs.json file. The results is the addition of json structured data in the form of contract addresses that have been migrated. Furthermore, the web interface will import PlanetOfDocs.json into the system's front-end configuration. Then contract defined by passing PlanetOfDocs.json to web3.js. Last but not least, the whole front-end code will be compiled to a folder called /build which is used for production purposes. Finally, the /build . folder can be served with the help of the Serve package to be able to run on localhost.

E. Storing and Retrieving Files

File storing on IPFS is carried out with the following flow:

- User puts a new file on the file upload form on the Store page.
- The web interface will extract filenames, perform hashing and convert files into files buffers.
- The web interface will call the StoreFile() function of the migrated smart contract, this will prompts out the Metamask dialog to confirm the transaction.
- If the user confirms the transaction then the file information will be written on the blockchain and the file buffer will be stored on IPFS at *https://infura.ipfs.io/ipfs/file hash*.
- User will receive proof or confirmation of their storing transaction in the form of transaction hash.

Meanwhile, file retrieval is carried out with the following flow:

- User visits the Ownership page on the web interface, at the bottom of the page there is a table listing all files that have been stored by the user.
- Then the user can select the "Retrieve" button in the rightmost column of the file data collection table to retrieve the file.
- Finally, the user is redirected to a new tab that connects where the file located, simply making a request to *https://infura.ipfs.io/ipfs/file hash*.

F. Component Specifications

Making the system in this study will use a laptop unit with the following specifications:

- CPU: Intel Core i5-8250U
- Memory: 8 GB DDR4 RAM
- Storage: 1 TB HDD
- GPU: NVIDIA GeForce MX 130

IV. Testing and Results

A. White Box Testing

White box testing is a test carried out to test the functionality of the application. This test uses a scenario that the tester knows the entire flow of the system and its features. This test is done to find out the existence of bugs and defects in system design. Expected results in This test is a running application based on input from testers.

1) White Box Smart Contract Testing: White box smart contract testing tests smart contract availability on blockchain, scenario execution of key system features and integration with IPFS. Based on the results of white box testing of contract migration it was found that the smart contract was successfully migrated to the Rinkeby test network. Furthermore, to communicate with the web interface, the contract migration process can also modify the json configuration file which will be used as a benchmark in the web interface. Based on the results of testing the Store features, users can perform storage transactions by confirming the Metamask dialog box. Then, some results related to transactions that have already taken place can also be displayed back to the user indicating that the storage process is running well and transaction data is recorded on the blockchain. On testing contract ownership feature, users can perform several ownership operations on digital documents stored on IPFS. This includes finding the ownership of a file and displaying all files owned by logged in user. Lastly is testing the transfer feature. The parameter that was successfully tested was confirmation file transfer and loss of user-owned file entries on the owner page.

2) White Box Web Interface Testing: White box web interface testing tests display availability and user interaction on web interface. It also tests the build scenarios and integration with Metamask. Test build tests the compilation of web interface created using React.js to then be transpiled to a static compiled web form which is lighter. The results obtained are the creation of the /build folder and the presentation of the /build folder can be served with the help of the Serve package. Testing the Home page includes testing the web interface to display page and test some buttons that can redirect the user to another page. The test results which are obtained by this test is that the page can be displayed as it should and all buttons work as expected. Testing the Store page includes user-friendly file storage features. Results that is obtained is that the success of the Metamask check at the beginning of the page loading phase and the user can affix the file on the file upload form. Testing the Ownership page includes file ownership features that owned by the user. The results obtained are the success of the Metamask check at the beginning of the page loading, the success of the page to load a list of files owned by the user and the user can affix file on the file upload form to check the ownership info of that file. Testing the Transfer page includes features file transfer by one user to another. The result obtained is the success of the examination Metamask at the beginning of the page loading and the success of the page to load and allow filling of the form located on the page itself. The Me page test lacks features. Parameters that tested is the identification of the Metamask account and the selection of a button that will redirect the user to the official page Metamask. The final white box test is the NotFound page test. Not Found page will be served to a user accessing a path that is not defined by the web interface. The test results show that all components can work as expected.

B. *Black Box Testing*

Black box testing is an external testing method. The test scenario is carried out with point of view of a user who does not know the functionality of the system. In this test, the functionality that testable is testing file storage and retrieval to and from IPFS.

1) Random File Storage and Retrieval Test: This test is carried out by storing and retrieving 30 files with different extensions and different sizes. It is intended to serve users who want to store various types of files on the system and IPFS. The parameter measured is the ability of the system to store and retrieve various file types and the time it takes to retrieve the file. The storage time is calculated when the user select the "Store" button after affixing the file until the transaction hash is successfully received. Meanwhile, time retrieval is counted when the user selects the "Retrieve" button until the web interface finishes loading the link IPFS file that has been saved. This information can bee seen in Table I.

The test is intended to test the performance of the bl package or BufferList in converting various file types and store them on IPFS. Through 30 different file types, the results show that all kinds of files can be saved by IPFS. However, it can be seen that it takes a different amount of time for varying file sizes.

2) Various Sized File Storing Test: This test is carried out to examine how flexible the system is in storing files with varying sizes. The input data used is a document with .pdf extension ranging from small to large sized files. The parameters tested are the time it took to store the file, specifically the ability of the BufferList library in converting files to buffers and the time it takes to add converted file buffer to IPFS. This information can be seen in table II.

Based on the above results, it can be concluded that the conversion time is strongly influenced by the file size. To conclude, the total upload speed which can be obtained by the following formula:

$$Average\ Upload\ Speed = \frac{\sum File\ Size}{\sum Storing\ Time} \quad (1)$$

TABLE I
RANDOM FILE STORAGE AND RETRIEVAL TEST RESULTS

No	Extensions	File Size	Storing Time	Retrieval Time	Result
1	.pdf	483 KB	0’ 33"	0’ 07"	Succeed
2	.jpeg	174 KB	0’ 19"	0’ 06"	Succeed
3	.rar	19 KB	0’ 19"	0’ 07"	Succeed
4	.png	14 KB	0’ 14”	0’ 11”	Succeed
5	.log	9549 KB	1’ 15”	0’ 20”	Succeed
6	.txt	1 KB	0’ 18”	0’ 01”	Succeed
7	.gz	5 KB	0’ 23”	0’ 08”	Succeed
8	.sql	4 KB	0’ 17”	0’ 09”	Succeed
9	.jpg	9 KB	0’ 15”	0’ 06”	Succeed
10	.mp4	3811 KB	0’ 36”	0’ 17”	Succeed
11	.mp3	6050 KB	1’ 30”	0’ 22”	Succeed
12	.tar	52 KB	0’ 24”	0’ 09”	Succeed
13	.svg	2 KB	0’ 16”	0’ 06”	Succeed
14	.html	1 KB	0’ 16”	0’ 07”	Succeed
15	.htm	14 KB	0’ 16”	0’ 12”	Succeed
16	.wav	3249 KB	0’ 30”	0’ 11”	Succeed
17	.wim	4 KB	0’ 21”	0’ 05”	Succeed
18	.xml	3 KB	0’ 16”	0’ 04”	Succeed
19	.yaml	31 KB	0’ 18”	0’ 05”	Succeed
20	.zip	14 KB	0’ 17”	0’ 05”	Succeed
21	.docx	12 KB	0’ 16”	0’ 11”	Succeed
22	.xslx	9 KB	0’ 13”	0’ 05”	Succeed
23	.pptx	33 KB	0’ 15”	0’ 04”	Succeed
24	.wasm	45 KB	0’ 16”	0’ 03”	Succeed
25	.gba	36 KB	0’ 14”	0’ 04”	Succeed
26	.exe	2243 KB	0’ 21”	0’ 04”	Succeed
27	.ico	362 KB	0’ 19”	0’ 06”	Succeed
28	.7z	11 KB	0’ 14”	0’ 04”	Succeed
29	.config	1 KB	0’ 11”	0’ 04”	Succeed
30	.csv	1 KB	0’ 12”	0’ 04”	Succeed

$$Average\ Upload\ Speed = \frac{32978\ KB}{812\ s} \quad (2)$$

$$Average\ Upload\ Speed = 40.61\ KBps \quad (3)$$

3) Duplicate File Storing Test: This test is used to see if the application can store two files that are same. The first test is done by first saving a .pdf file to IPFS. Then continue by saving the same file to IPFS. As a result, the app will refuse the second save because it is the exact same file that was saved first. The second test is to do a slight modification to the same .pdf file which is still on the user's local side. Modification done by opening the file with a PDF Reader and then saving as a new file with the same name same. This replaces some file metadata like date modified and so on. Then the files that have been modified is saved into IPFS through the application. The result obtained is that the application receives the modified file and store it on IPFS and get a different hash even though the content is the same. This matter indicates that the application performs a file integrity check by comparing hashes not by comparing the contents of the file, so that there is a potential for duplication of content for file storage.

V. CONCLUSION

Based on the results of the design, testing and analysis that have been carried out during the research, it can conclude that n digital document management applications,

TABLE II
VARIOUS SIZED FILE STORING TEST

No	File Size	Storing Time	Upload Speed	Result
1	86 KB	0' 13"	6.615 KBps	Succeed
2	95 KB	0' 13"	7.307 KBps	Succeed
3	159 KB	0' 15"	10.600 KBps	Succeed
4	180 KB	0' 16"	11.250 KBps	Succeed
5	202 KB	0' 16"	12.625 KBps	Succeed
6	284 KB	0' 16"	17.750 KBps	Succeed
7	310 KB	0' 17"	18.235 KBps	Succeed
8	316 KB	0' 17"	18.588 KBps	Succeed
9	373 KB	0' 18"	20.722 KBps	Succeed
10	395 KB	0' 18"	12.944 KBps	Succeed
11	398 KB	0' 24"	16.458 KBps	Succeed
12	421 KB	0' 27"	15.592 KBps	Succeed
13	456 KB	0' 26"	17.538 KBps	Succeed
14	477 KB	0' 28"	17.035 KBps	Succeed
15	576 KB	0' 25"	23.040 KBps	Succeed
16	691 KB	0' 24"	28.791 KBps	Succeed
17	701 KB	0' 28"	25.035 KBps	Succeed
18	788 KB	0' 27"	29.185 KBps	Succeed
19	892 KB	0' 27"	33.037 KBps	Succeed
20	937 KB	0' 30"	31.233 KBps	Succeed
21	1100 KB	0' 36"	30.555 KBps	Succeed
22	1190 KB	0' 33"	36.060 KBps	Succeed
23	1451 KB	0' 35"	41.457 KBps	Succeed
24	1516 KB	0' 34"	44.588 KBps	Succeed
25	1793 KB	0' 37"	48.459 KBps	Succeed
26	2028 KB	0' 35"	57.942 KBps	Succeed
27	3319 KB	0' 39"	85.102 KBps	Succeed
28	3370 KB	0' 43"	78.372 KBps	Succeed
29	3517 KB	0' 46"	76.456 KBps	Succeed
30	4957 KB	0' 49"	101.163 KBps	Succeed

blockchain can be used as a recording medium for transactions from each digital document management operation. Making a blockchain-based digital document management application can be started by creating a smart contract, design the web interface, then connect between the two followed by contract migration and serving web interface. To connect blockchain-based digital document management applications with peer-to-peer network, it can be done using an API, in this case the system used ipfs-api which is a JavaScript API that can communicates with IPFS. Based on the results of white box testing, it is found that the web interface and smart contracts ran as expected however, the continuity of the transaction is strongly influenced by several variables such as Rinkeby testnet's computing capabilities and the number of transactions executed at the same time. Based on the results of the black box test, the average file upload speed time obtained is 40.61 KBps. File hashing is done very strictly, where changes in the form of a difference of 1 bit of data will make a file identified as a unique file even though the content served is not at all experiencing changes.

REFERENCES

[1] N. Nizamuddin, K. Salah, M. Ajmal Azad, J. Arshad, and M. Rehman, "Decentralized document version control using ethereum blockchain and ipfs," *Computers & Electrical Engineering*, vol. 76, pp. 183–197, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790618333093

[2] A.-T. Pănescu and V. Manta, "Smart contracts for research data rights management over the ethereum blockchain network," *Science & Technology Libraries*, vol. 37, no. 3, pp. 235–245, 2018.

[3] X. Wang, H. Wu, W. Sun, and Y. Zhao, "Towards generating cost-effective test-suite for ethereum smart contract," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 549–553.

[4] D. Chirtoaca, J. Ellul, and G. Azzopardi, "A framework for creating deployable smart contracts for non-fungible tokens on the ethereum blockchain," in *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2020, pp. 100–105.

[5] X. Luo, W. Cai, Z. Wang, X. Li, and C. M. Victor Leung, "A payment channel based hybrid decentralized ethereum token exchange," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 48–49.

[6] P. Deshmukh, S. Kalwaghe, A. Appa, and A. Pawar, "Decentralised freelancing using ethereum blockchain," in *2020 International Conference on Communication and Signal Processing (ICCSP)*, 2020, pp. 881–883.

[7] D. Vujičić, D. Jagodić, and S. Ranđić, "Blockchain technology, bitcoin, and ethereum: A brief overview," in *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*, 2018, pp. 1–6.

[8] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018, pp. 2–8.

[9] Z. Zhou, Y. Xiong, W. Huang, and L. Ma, "Sprune: A code pruning tool for ethereum solidity contract static analysis," in *2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, 2020, pp. 66–70.

[10] C. G. Harris, "The risks and challenges of implementing ethereum smart contracts," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2019, pp. 104–107.