applied sciences

MDPI

Article

Blockchain Implementation to Verify Archives Integrity on Cilegon E-Archive

Intan Permatasari [1,2], Meryam Essaid [1], Hyeonwoo Kim [1] and Hongtaek Ju [1,*]

[1] Department of Computer Engineering, Keimyung University, Daegu 42601, Korea; intanishere16@gmail.com (I.P.); maryama.essaid@gmail.com (M.E.); kimhw84@gmail.com (H.K.)
[2] The Agency for the Assessment and Application of Technology (BPPT), Jakarta 10340, Indonesia
* Correspondence: juht@kmu.ac.kr

Received: 10 February 2020; Accepted: 8 April 2020; Published: 10 April 2020

Abstract: A good archive management system must consider information security aspects, such as availability, confidentiality, and integrity. The Cilegon E-Archive (CEA) system is a centralized system for managing the lifecycle of archives. The existing CEA system has several problems, including a single point of failure, low data availability, and difficulty in proving the originality of files. This paper introduces a prototype for a new CEA system that integrates IPFS and the Ethereum private network. In addition, CEA DApp is developed as an interface for users in interacting with CEA system, and its functionality is managed by a smart contract. The results show that the conducted improvements into the CEA system highly improved the system security in terms of preventing archival forgeries.

Keywords: blockchain; Ethereum; IPFS; decentralized archive system

1. Introduction

According to Article 1, Number 7 [1] of the Indonesian Constitution concerning Basic Archival Requirements, archives are texts which are made and accepted by state institutions and government agencies in any form, whether in a single or group condition, in the context of carrying out government activities. Archives are essential because of the substantial information they offer in the administrative process. Therefore, they should be managed properly; for instance, there should be an orderly, neat storage system and a way to retrieve, recover, and secure the archives. An excellent archive management system leverages the organization in question, such as the government, to work efficiently.

Nowadays, many people use information technology to help manage archives. Cilegon E-Archive (CEA) is a local government archive management system in Cilegon City, West Java, Indonesia. It has features for managing the archives, including their collection, retention, and retrieval. Figure 1 illustrates the mapping of the CEA system. It is located in Serpong, in the south of Tangerang City. However, the archive users are in Cilegon City, which is 99.4 km from Serpong. In Indonesia, archives are important instruments in the auditing process. The Badan Pemeriksa Keuangan (BPK), as Indonesian the government auditor, audits the archives at least once a year. Even though the system server is located near the BPK, the BPK must go to Cilegon because the BPK does not have access to the system. Moreover, the BPK prefers to inspect printed archives that are stored in a physical storage unit in Cilegon City, because there are many archival forgeries in Indonesia.

The CEA system is a centralized system that is located on a single cloud server belonging to Badan Pengkajian dan Penerapan Teknologi (BPPT), which is the Agency for the Assessment and Application of Technology in Indonesia. A centralized system uses a client–server architecture in which nodes connect to one server and are controlled by the network administrator. It makes it easy to track and manage the data. However, a centralized system has several vulnerabilities, such as a single point of failure, low data availability, and lack of data confidentiality.

Figure 1. Map of existing Cilegon E-Archive (CEA) system.

A single point of failure is a condition in which the entire system stops working when the server goes down. The clients cannot connect to the server, so the system cannot process requests to provide data. Moreover, if there is no backup, users can lose data immediately. Therefore, it can cause low data availability.

A good archive management system should consider data confidentiality in order to ensure data can be accessed only by registered users. The existing CEA system has a feature to manage user control. The archivists are allowed to alter the archives in their respective units. Even so, this system cannot guarantee whether the modifications are valid. This condition triggers archival forgeries; because the data and files are only stored in the server without any encryption, they are not secure and unauthorized users can replace legitimate files with fake files or modify the data.

Considering these issues, this paper integrates the Interplanetary File System (IPFS) and blockchain. IPFS is a distributed file storage system that uses content-based addressing and guarantees high-throughput by combining a distributed hash table, BitSwap exchange protocol, and naming system [2]. Blockchain has a consensus mechanism to validate the transactions and guarantee the integrity of each node [3]. In turn, the transactions are unchangeable. Nevertheless, blockchain has limitations in data storage, and there are costs of executing transactions. The cost depends on how big the stored data are.

This paper proposes a new CEA system, using a decentralized architecture. It integrates IPFS and blockchain technology. IPFS shares the files with other nodes in the network instead of only storing the files in one server. Pinning service is a feature of IPFS that ensures high availability and long-term retention. IPFS generates a hash to identify the content of the file. A blockchain-based storing system is expensive to store massive data. Therefore, the hash of a file will be stored in the blockchain, using a smart contract.

It is difficult to modify transactions in a blockchain because blockchain has a consensus mechanism to validate them. The transactions are put in blocks, and each block is associated with each other, secured with a digital signature, and proved with a timestamp. Both IPFS and blockchain will be configured as a private network, to restrict file access from public IPFS and Ethereum networks. This configuration can improve data confidentiality and integrity, to prevent archival forgeries.

2. Background

CEA is a local government archive management system in Cilegon City, West Java, Indonesia. It provides the lifecycle of government archives, including their collection, retention, and retrieval. The users of this system consist of 130 archivists from 32 units in Cilegon City. All the users access this

system via an internet connection. The CEA system uses centralized architecture, which means all users access one server in order to store applications, data, and uploaded files of digital archives.

Figure 2 illustrates the existing concept of the CEA system. The archives come from external systems. In every unit, there are several external systems, such as finance, human resource, and procurement. These systems produce printed documents as output. Based on the Indonesian government's rules, the archivists in all units have to input the data of the printed-out documents. Sometimes the archivists scan and upload the papers into the CEA system. However, because scanners are not available in all units, sometimes archivists only input the data. When they do, they only re-input the data, which are already stored in the external systems. It causes data duplication and human error. For instance, typing mistakes results in mismatches between the data in the external systems and the data in the CEA system. This is a crucial error, and the BPK (Indonesian Government Auditor) must audit the archives periodically. The auditor will inspect the archives in the external systems and the CEA system.

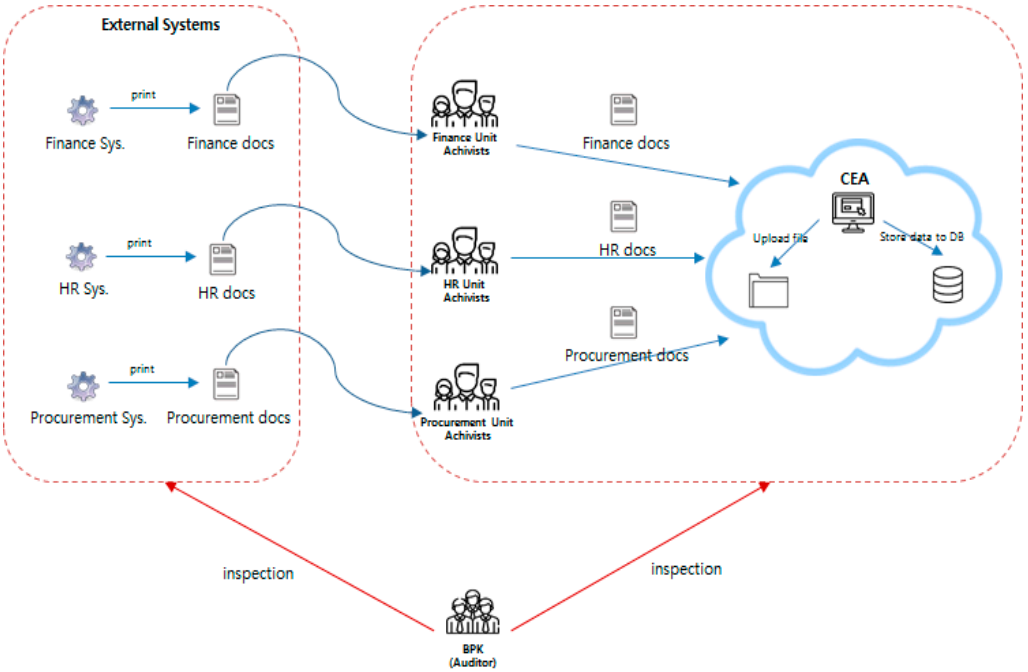


Figure 2. Concept of the existing CEA system.

The centralized web system uses location-based addressing. It uses the internet protocol (IP) addresses in the uniform resource locators (URLs) to show the location where the data are stored. It does not have a direct relationship with the content of the data. The existing CEA system works on hypertext transfer protocol (HTTP), which downloads files from a single host server in response to requests. Centralized data are easier to deliver and manage. However, it allows any party, such as server administrators and archivists, to access, modify, and even remove data. The problem with location-based addressing occurs when the location is not accessible. If the server is offline, users cannot retrieve the data.

Furthermore, the existing CEA system has challenges in three aspects of information security meant to ensure the protection of data [4], as mentioned below:

- Confidentiality
  The data and files of archives are stored in a single server, which does not have secured protection, besides requiring login usernames and passwords. The server administrator can access the data and files without permission. There is no monitoring or logging system to control who accesses what.

- Integrity
  The CEA system allows users to modify the data of the archives. There is no mechanism to validate the data that are modified. Each archivist is responsible for all archives in his or her unit. However, no one can guarantee whether the modifications tamper with archival integrity. This condition creates vulnerability and can lead to archival forgeries.
- Availability
  Data availability is low in a centralized system. When the server is off, all processes stop working. The users cannot send requests to the system, and the system cannot respond to the requests or provide users with the data and information they need.

To overcome these challenges, this paper develops a decentralized CEA system by implementing a distributed file system to manage the archives. The files are distributed in several nodes, to address the issue of having a single point of failure problem. This system uses a smart contract to manage the functionality of the application and deploy it on an Ethereum private network. It will keep the archives confidential and safe from unauthorized users. The consensus mechanism in the blockchain validates each transaction before it is committed in a block to maintain archival integrity.

The blockchain network can assure the integrity of any information system [5]. Running a peer-to-peer (P2P) protocol guarantees a solution to the problem of a single point of failure. Every transaction is stored in a block, which is linked and secured with cryptography. It also has a consensus mechanism to validate the transaction. These features make it almost impossible to modify a transaction. IPFS is a distributed file storage system that involves all nodes to store each file securely and provides high data availability. By combining blockchain and IPFS, the CEA system can improve system security and preserve the integrity of its archives.

The integrity of archives is essential to preventing archival forgeries. Blockchain is able to store data immutably, secure them with a digital signature, and timestamp them. Blockchain can be used to store less data, like transaction metadata information and hash values [6]. This system uses Ethereum private blockchain to store the hash of the file. In addition, a consensus mechanism in the blockchain can validate the transaction securely. When there is any modification, the nodes in the blockchain will check whether it is valid or not.

Related Works

Conceptually, blockchain is a decentralized, distributed digital ledger that verifies and stores transactions immutably and transparently, without relying on a third party or central authority. However, blockchain is not appropriate for storing a large number of data files. For that, IPFS offers a reasonable solution. This review discusses related research on blockchain-based platforms and the IPFS distributed file system. This section also reviews previous research on smart contracts and DApps.

Hasan et al. [7] developed an off-chain communication exchange protocol to stabilize a secure channel for downloading digital content using Ethereum and IPFS.

The integration of blockchain and IPFS technology was also undertaken, to form a framework for a secure, tamper-proof model for academic-research record-keeping with access control methods [6]. The framework utilized smart contracts for storing the metadata of research in the blockchain network and auditing purposes. To enhance file confidentiality, the researchers encrypted the files with the master key before they were uploaded into IPFS.

Yatskiv et al. [8] tried to prove the integrity of videos on cloud services by calculating the hash of each frame and sending the hash to the blockchain, which is trusted by interested parties.

To store large amounts of data on distributed storage systems using blockchain technology, one study [9] created the protocol for notarizing files over blockchain (NFB), to ensure communication between the hyperledger blockchain and OKORO (a secured centralized archiving document management system).

The integration of Ethereum and IPFS has been applied in a social media application [10] for recording data in a distributed data storage service. The researchers developed a web-based, front-end interface and used Infura [11] IPFS API to bridge the communication between the main Ethereum network and the public IPFS.

Research has also shown that DApps may be the solution for the single point of failure issue, as they have several advantages over traditional applications [12]. For instance, DApps are effective for payment processing because users can use cryptocurrencies for transactions. Users can use private and public keys to authenticate themselves and sign transactions. Moreover, transactions stored on a public ledger are transparent and, in turn, more trustworthy.

Previous research [13] introduced the Ethereum-based Smart Home System (SMS) for distributing personal data for emergency calls via IPFS. The personal data were signed via the digital signatures of the users.

These related works leverage the key features of blockchain and IPFS. Likewise, this solution will adopt some of the methods applied in these solutions with several modifications. For instance, instead of using a symmetric key, like a master key, to protect the confidentiality of files, this paper's proposed system relies on creating IPFS and Ethereum private networks. Moreover, in this case, users have to be registered in the private network before they can sign a transaction.

3. System Design

This section describes the design of the proposed system, which consists of system architecture, scenarios, and file distribution.

3.1. System Architecture

Figure 3 illustrates the proposed CEA system architecture. To interact with this system, each user must install the decentralized application named CEA DApp, IPFS private network, and Ethereum private network named CEA Network. The users consist of archivists and BPK. One account is set as an administrator, and it is responsible for registering and monitoring other users. Each user has two identities. First, there is the peer ID for interacting with the IPFS private network. Second, there is the Ethereum address for communicating with CEA Network.

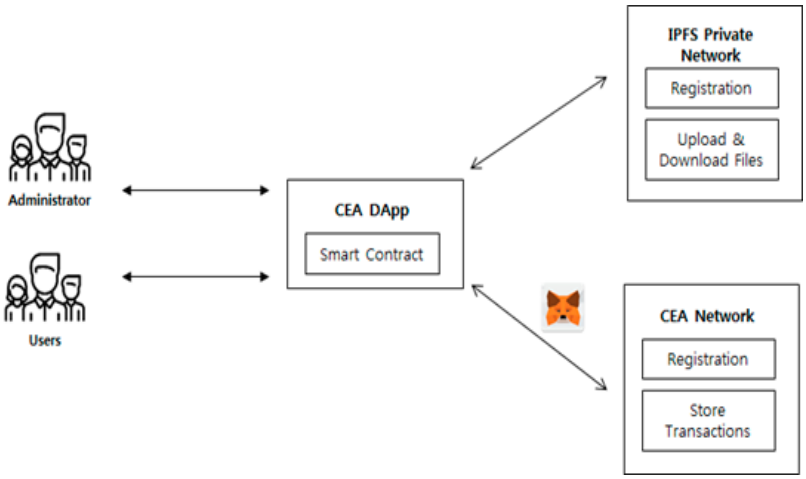


Figure 3. Proposed CEA system's architecture.

To maintain the confidentiality of archives, both the IPFS and CEA Network are set as private networks. The files will only be distributed between archivists and BPK. The hash of files will also be stored and retrieved only by them. The user must install Metamask [14] to run CEA DApp on traditional browsers, such as Google Chrome or Firefox. In addition, Metamask has to import the

Ethereum addresses, which are generated on the CEA Network. The CEA DApp has a smart contract to store the hash in CEA Network and retrieve the hash of files from the CEA Network.

3.2. System Scenarios

3.2.1. Registration on IPFS Private Network

Figure 4 illustrates the scenario of registering on the IPFS private network. An archivist is assigned as an administrator, who has access as Boot Node. All other users are granted access as Client Nodes. All nodes have to set a bootstrap node configuration, using a boot node peer ID and IP address. Hereafter, the boot node generates and copies a single swarm key for all client nodes. A swarm key allows each IPFS node to connect with other nodes. All nodes in the private network must have the exact same swarm key. The IPFS private network is restricted from the public IPFS network.

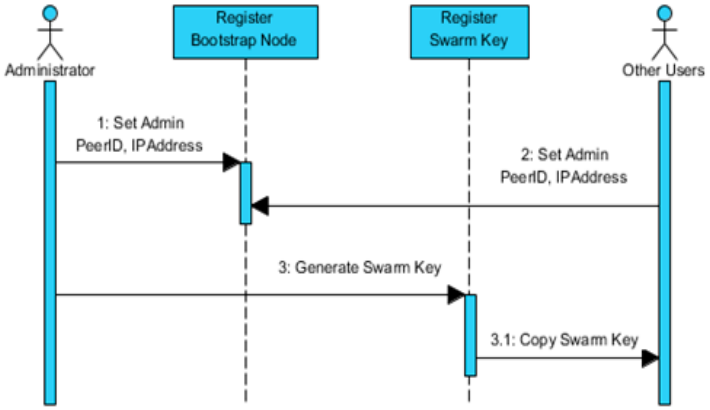


Figure 4. Sequence diagram of registration on IPFS private network.

3.2.2. Registration on CEA Network

The CEA Network is an Ethereum private network. As seen in Figure 5, an administrator should enter a passphrase, to generate an account, which is an externally owned account (EOA). Users need this account to interact with the Ethereum blockchain via transactions [15]. Each EOA has a balance (Ether) to sign each transaction. The accounts are identified by a public and a private key, which are encoded in a key file on the KeyStore directory in each account. The public key is used as the account address.

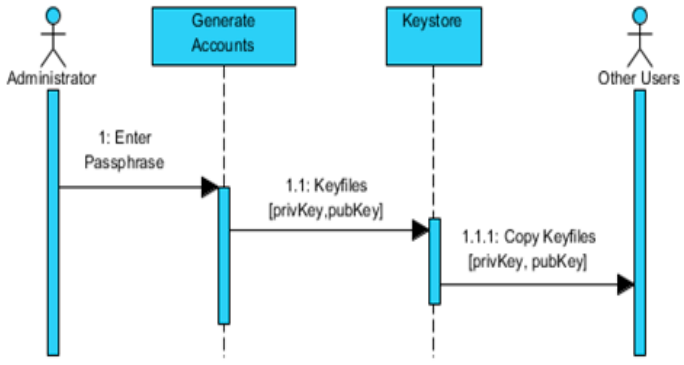


Figure 5. Sequence diagram of registration on the CEA Network.

3.2.3. Uploading a File on CEA System

Figure 6 illustrates the scenario of uploading a file on CEA system. Users employ the CEA DApp interface to upload a file. The file is stored as an object on a local storage IPFS client node. Then, the object is pinned recursively in each client's local storage node to provide file availability. It pins

all links to the object to be stored locally. After the object is successfully pinned in local storage, IPFS leverages DHT, to distribute the object to other nodes in the private network. In this distribution, the closest peer ID among the nodes has the highest priority. IPFS returns a hash to identify the file. The system calls Metamask to sign the transaction containing the hash of a file. However, the user must unlock the account to enable the signing of the transaction on CEA Network. When the transaction is committed, CEA Network returns a transaction ID as its identity.

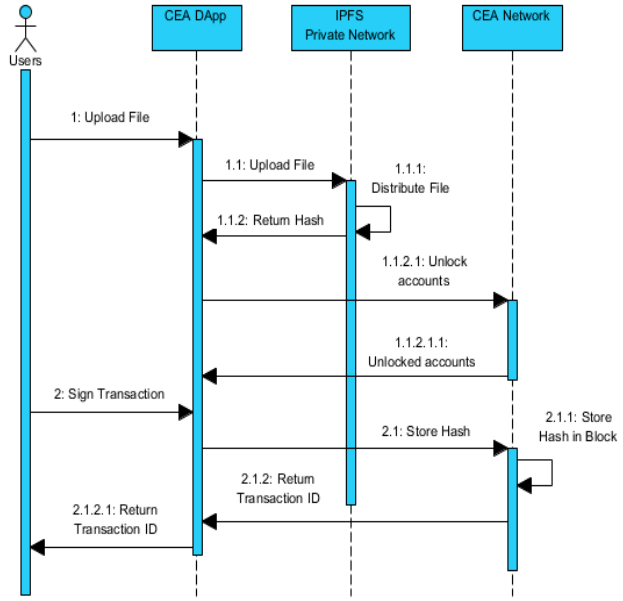


Figure 6. Sequence diagram of uploading a file on the CEA system.

3.2.4. Downloading a File on CEA System

Figure 7 illustrates how users can download a file on the CEA system. Users use a transaction ID to retrieve transaction data from CEA Network. The transaction data contain a hash of the file and other transaction information. The users use that hash to download a file from the IPFS private network.

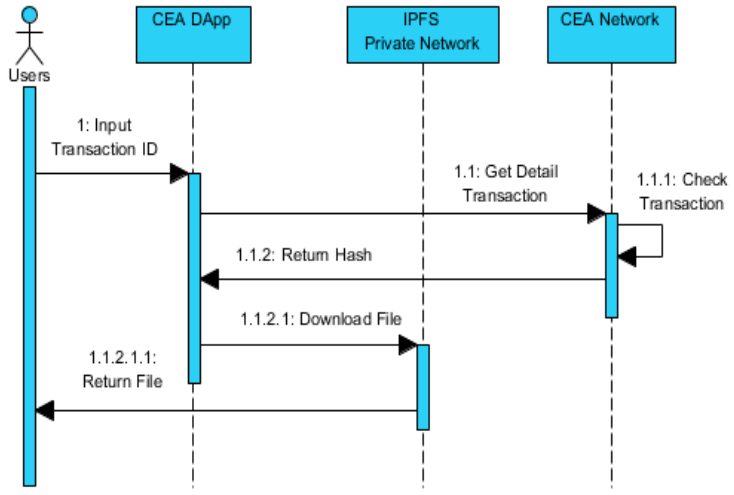


Figure 7. Sequence diagram of downloading a file on the CEA system.

3.3. File Distribution on IPFS

IPFS stores each file as an object that contains data and links. The object storing size is limited to 256 Kb. Objects that are larger than 256 Kb are divided into small parts (chunks) that are equal to or less than 256 Kb. Every chunk is identified by a hash and linked to other chunks, as illustrated in Figure 8.

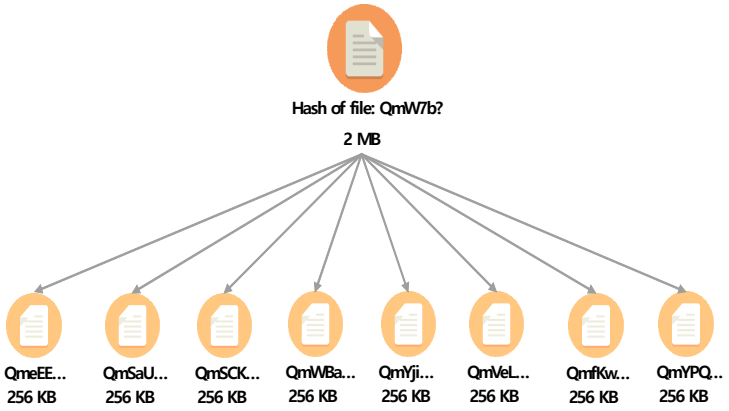


Figure 8. Illustration of distributing a file on IPFS private network.

4. Implementation

This section explains the implementation of CEA Network, the IPFS private network, and CEA DApp.

4.1. CEA Network

To maintain file confidentiality in the file distribution system, this paper implements an Ethereum private blockchain named CEA Network. After installing Geth-Ethereum, we generated three Ethereum accounts on CEA Network. These accounts are protected with a passphrase, which is similar to a password. In addition, these accounts are used for deploying the smart contract and signing the transactions on the CEA Network.

After generating the accounts, the next step is creating a genesis block to provide Geth with basic information, such as chainID, difficulty, gasLimit, and alloc. The chainID is an identification number that distinguishes the CEA Network from other blockchains. The difficulty determines the difficulty of the blocks to be mined. A lower number expresses that the mining process will be quicker. In turn, the transactions will be faster. The gasLimit defines the limit of gas cost per block. This should be set high, in order to prevent limitations when testing a transaction. The alloc allocates the ETH balance to each account. The genesis block is a JSON file and will be copied to other nodes on CEA Network.

Before starting the CEA Network, the data dictionary has to be instantiated with the genesis file, to store data related to the CEA Network blockchain. In this case, the name is myDataDir. In order to use the DApp, HTTP JSON-RPC and flag --rpc are enabled when starting the CEA Network. This network runs at http://localhost:8543. Metamask is used as a communication bridge between the DApp and the CEA Network by importing the existing accounts on the CEA Network into Metamask, using a private key that is stored on the UTC file in the Keystore directory.

4.2. IPFS Private Network

This research uses an IPFS private network to restrict file distribution. This implementation creates a boot node and client nodes. A boot node acts as a network administrator, which is responsible for monitoring the client nodes. It is also used by the client nodes to connect to the IPFS private network. Client nodes have access to upload or download files on the IPFS private network. The Rivest–Shamir–Adleman (RSA) algorithm is used to generate key public/private key pairs to authenticate peers in IPFS. Every node has a peer ID as its identity, which is generated during IPFS initialization.

There are two requirements to enable communication between peers on the IPFS private network. First, the default bootstrap node must be replaced with the IP address and peer ID of the boot node. This should be done for the boot node and the client nodes. Second, the boot node must install

ipfs-swarm-key-gen to generate a swarm key. This swarm key must be available in the ~/.ipfs directory in the boot node and the client nodes.

Table 1 shows how the private network successfully starts the boot node and the client nodes. It shows that all nodes have a swarm key with the fingerprint afcc0321ceefb1bc0a8c13b187bc369b. This network is limited to peers with a fingerprint of the swarm key, as mentioned above. It ensures that public peers without the same swarm key cannot access the network. To enable the communication between IPFS private network and DApp, the requests from DApp at http://localhost:3001 have to be allowed in the IPFS configuration. By default, IPFS is designed to reject request from unknown domains. Therefore, firstly the domains are configured in 'Access-Control-Allow-Origin', to enable the interaction with the IPFS private network.

Table 1. Starting the local nodes in the IPFS private network.

Nodes	Screenshot
Boot Node	com3@com3-OptiPlex-3050:~/.ipfs$ IPFS_LOGGING=critical ipfs daemon Initializing daemon... go-ipfs version: 0.4.18- Repo version: 7 System version: amd64/linux Golang version: go1.11.1 Successfully raised file descriptor limit to 2048. Swarm is limited to private network of peers with the swarm key Swarm key fingerprint: afcc0321ceefb1bc0a8c13b187bc369b
Client_1 Node	intan@intan-VirtualBox: ~ intan@intan-VirtualBox:~$ ipfs daemon Initializing daemon... go-ipfs version: 0.4.22- Repo version: 7 System version: amd64/linux Golang version: go1.12.7 Swarm is limited to private network of peers with the swarm key Swarm key fingerprint: afcc0321ceefb1bc0a8c13b187bc369b
Client_2 Node	Command Prompt - ipfs daemon Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved. C:\Users\INTANISHERE>ipfs daemon Initializing daemon... go-ipfs version: 0.4.22- Repo version: 7 System version: amd64/windows Golang version: go1.12.7 Swarm is limited to private network of peers with the swarm key Swarm key fingerprint: afcc0321ceefb1bc0a8c13b187bc369b

4.3. CEA-DApp

This study developed a CEA-DApp as a front-end system, to make it more convenient for the user to interact with the IPFS private network and CEA Network. It was built on the Truffle Suite framework, using ReactJS programming. This DApp has a smart contract named CEAContract.sol, to store the hash of a file. The CEAContract.sol contains a state variable storedData typed string to model the hash of a file from IPFS. It also contains set and get functions. The set function receives an input value and stores it in storedData, while the get function displays the storedData value.

Before deploying a smart contract in CEA Network, generated accounts from CEA Network are import into Metamask and unlock them. Figure 9 shows CEAContract.sol deployment on the CEA Network. It is stored as a transaction on the distributed ledger and identified by its transaction hash. After it is deployed, the smart contract will get a contract address as its identity and an application binary interface (ABI) as a description of the deployed contract and its function [16]. As seen in Table 2, the contract address and ABI are used in a web3.js library, to allow the interaction between the ReactJS program and the smart contract.

```
2_deploy_contracts.js
=====================

   Replacing 'CEAContract'
   -----------------------
   > transaction hash:    0x43b35feb7a63307cd3e3dc4e08efd6f618032c6aa3d02b6bf940e963b91f4e65
   > Blocks: 1            Seconds: 4
   > contract address:    0x73201F41447321DeE8AD045838973D7Cc20Cad75
   > account:             0x15928827fF1eE61d81182D914a8b7f5416B246B1
   > balance:             99999.97554388
   > gas used:            271008
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00542016 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
```

Figure 9. Smart contract deployment on the CEA Network.

Table 2. Algorithm for Integrating Smart Contracts with ReactJS.

Algorithm 1 Integrating Smart Contracts with ReactJS
1: import *web3*
2: initiate smart contract address = *address*;
3: initiate smart contract abi code = *[abi]*;
4: export *address* and *[abi]* into *web3* provider

The CEA DApp interacts with CEA Network, which is run on http://localhost:8543, and accesses the IPFS private network API using an ipfs-http-client library, which runs on IPFS daemon at port 5001. Therefore, CEA DApp can upload and download files on IPFS. The front-end consists of the upload and download page. A class-named app is used on the upload page. It contains a state as an object that holds information, like ipfsHash, buffer, and transactionHash.

As seen in Table 3, the system calls function onSubmit to store the file on the IPFS private network when uploading the file. The file is converted into a buffer (an array with data) before it is uploaded into the IPFS private network. The IPFS private network returns a hash of the file. Then it triggers the Metamask to confirm the transaction. After the transaction is successfully confirmed, the CEA Network start the mining process to put the transaction in a block. Then, the CEA Network will return a Transaction ID. Before doing this transaction, the account in the CEA Network must be unlocked.

Table 3. Algorithm for Uploading a File.

Algorithm 2 Uploading a File
1: Function onSubmit
2: get ethereum account from *web3*
3: get contract address from *statevalue*
4: convert content of file into *buffer*
5: call function to upload file into ipfs
6: return *hash*
7: put hash into state
8: call method *set()* in *CEAContract.sol*
9: send hash into set()
10: if *set()* is success
11: return *transactionid*
12: else
13: return *error*
14: end if
15: end function

5. Experiment

This section discusses the experiment that was conducted on the proposed CEA system, to ensure its proper functionality and evaluate the transaction fee and system performance.

5.1. Testing of File Availability

This experiment involved three nodes in the IPFS private network that are run on a similar WIFI network. One node is set as a boot node to monitor the client node's activity. Two nodes are set as client nodes. Table 4 shows the hardware and software specifications for testing the nodes.

Table 4. Hardware and software specifications for testing the nodes.

Boot Node	
Processor	Intel(R) CoreTM i5-7500T CPU @2.70 GHz
RAM	8 GB
HDD	250 GB ATA
OS	Ubuntu 18.04
Client_1 Node	
Processor	Intel(R) CoreTM i7-4702 MQ CPU @2.20 GHz
RAM	2 GB
HDD	10 GB ATA
OS	Ubuntu 16.04
Client_2 Node	
Processor	Intel(R) CoreTM i7-4702 MQ CPU @2.20 GHz
RAM	8 GB
HDD	500 GB ATA
OS	Windows 7 Professional

All nodes identify each other by using their peer IDs, which are unique identities for each node with which to communicate in the IPFS private network. Table 5 shows the peer ID of each node in this experiment.

Table 5. Hardware and software specifications for testing the nodes.

Node Name	**Peer ID**
Boot Node	QmfSUyHddqjJpBju6NQehsCeWoGLbnSaVBWir1qSyYg9Wm
Client_1 Node	QmUu2ezNF3zritD9ZK9iaR7JCXm1zTZDYDHqHqp8aQBzsH
Client_2 Node	QmervD64ABohpJuVEHetFoRebw4rhBpfJd1WP6tzAc5uRk

Next, the upload and download operations were tested on the IPFS private network. For the upload operation, IPFS creates an immutable object, depending on file size. If the file is larger than 256 kB, it is divided into several chunks that are identified by their CID. First, the data are stored on the local repository of Client Node 1. Then, Client Node 1 updates the DHT to distribute the data. In this experiment, Client Node 1 uploaded files with various sizes. Figure 10a shows that the file distribution with size 100 kB is stored only in one object, which is identified by one CID. Figure 10b shows that the file distribution with size 1 MB is stored in four objects. It has a root CID and CID for each part.

(**a**) File distribution of 100 kB. (**b**) File distribution of 1 MB.

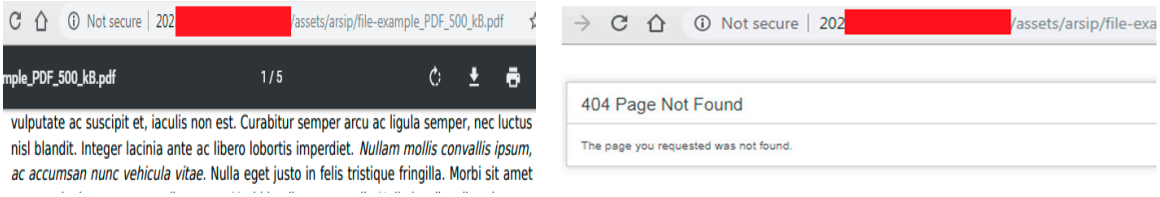
Figure 10. Testing of file distribution on different sizes.

To check file availability, we conducted an experiment where we tried to upload a file into Client_1 node and download it from Client_2 node while the node of Client_1 was down. The result shows that the Client_2 node can properly download the file even when the node contains the file (Client_1 node) is not available as other nodes in the network provide it, Table 6 presents the experiment details.

Table 6. Testing of availability on the proposed CEA system.

Description 1	Client_2 Node checks its Peer ID.
Screenshot 1	intan@intan-VirtualBox:~$ ipfs config show | grep "PeerID" "PeerID": "QmervD64ABohpJuVEHetFoRebw4rhBpfJd1WP6tzAc5uRk"
Description 2	Client_2 Node checks available nodes. It shows only the boot node is available.
Screenshot 2	an-VirtualBox:~$ ipfs swarm peers [redacted]/tcp/4001/ipfs/QmfSUyHddqjJpBju6NQehsCeWoGLbnSaVBWir1qSyYg9Wm
Description 3	Client_2 Node can download the file even though Client Node 1 is not available.
Screenshot 3	intan@intan-VirtualBox:~/testing$ ipfs get QmbBp5huHazG582ean3eGG wWXkkVKnRc7V5te16ByWNs2N Saving file(s) to QmbBp5huHazG582ean3eGGwWXkkVKnRc7V5te16ByWNs2N 99.72 KiB / 99.72 KiB [==========] 100.00%

To compare the proposed CEA system with the existing one, we attempted to download a file when it was not available on the existing system. As seen in Figure 11, the existing CEA system stores the file on the central server and uses location-based addressing; therefore, the file can be downloaded only if it is available on that server or location.



(**a**) The file is available. (**b**) The file is not available.

Figure 11. Testing of file availability on the existing CEA system.

These experimental results show that the functionality of the proposed CEA system offers improved file availability. A distributed file system can provide files, because, instead of storing files on a centralized server, it stores them on all nodes in the IPFS private network. When one node is turned off, other nodes can provide the files.

5.2. Testing of File Integrity

Figure 12 presents an example of how to upload a file by using the CEA DApp. When the user uploads a file, CEA DApp automatically detects its Ethereum Address. Then, the CEA DApp interacts with the IPFS private network, to upload the file. First, the file is stored in a local repository. Then it is distributed to other nodes in the network. IPFS returns a hash of the file, and CEA DApp triggers Metamask to confirm the transaction to be stored on the CEA Network.

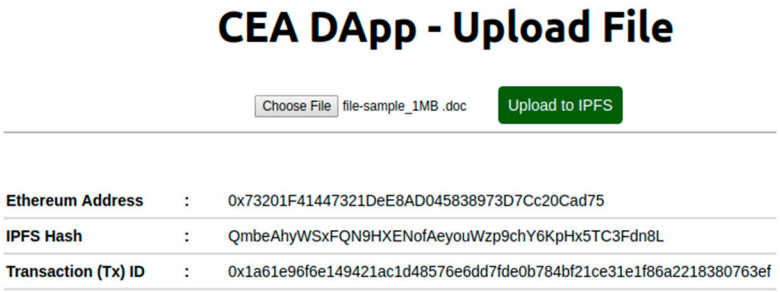


Figure 12. Testing of uploading a file on the CEA DApp.

A transaction ID is used to retrieve data from the CEA Network. Figure 13 shows the results on the download page. It returns the block number, gas used, timestamp, input, and decoded input. The input contains an encoded hash of a file in Hex data format, with a length of 132 bytes. It should be decoded into a human-readable format. The hash will redirect to the IPFS gateway, on http://127.0.0.1:8080/ipfs/, so that the user can download the file without interacting with IPFS command-line interface (CLI).

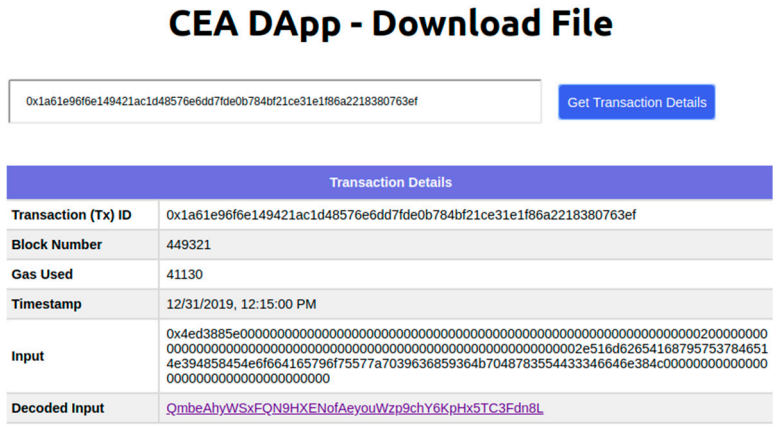


Figure 13. Testing of downloading a file on the CEA DApp.

To ensure collaboration between CEA DApp and CEA Network or the IPFS private network, this experiment manually checks the IPFS and Ethereum console. Figure 14 shows the transaction that is stored on CEA Network, using the transaction ID. All transactions on CEA Network are immutable. It helps users to prove the hash of files that are stored on the IPFS private network. Moreover, the transactions will not spread into another Ethereum network, because the CEA Network is private.

Figure 15 shows that a file is stored on the IPFS private network, using a hash. Because the file size is 1 MB, it is divided into 4 small parts, each of which is less than or equal to 256 kB.

```
> eth.getTransaction('0x1a61e96f6e149421ac1d48576e6dd7fde0b784bf21ce31e1f86a2218380763ef');
{
  blockHash: "0x2dc86a845d3bd06ccfe1f9b87d4e594514c084e7b97c76f98cbb872e9821e342",
  blockNumber: 449321,
  from: "0x4fd4970ecd53c44a94452a790dc45df6cb744384",
  gas: 61695,
  gasPrice: 18000000000,
  hash: "0x1a61e96f6e149421ac1d48576e6dd7fde0b784bf21ce31e1f86a2218380763ef",
  input: "0x4ed3885e0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000002e516d6265416879575378
46514e394858454e6f664165796f75577a7039636859364b704878355443334646e384c000000000000
0000000000000000000000000",
  nonce: 50,
  r: "0x2b6503b27ecc0880857246cb9258a9bee85f89317b551d5a729a034a275aeafd",
  s: "0x35c2aca0248d1a47eadb652cde93253b622891e73c13ec3bb46c201686e79ceb",
  to: "0x73201f41447321dee8ad045838973d7cc20cad75",
```


Figure 14. Testing of storing a transaction on the CEA Network.

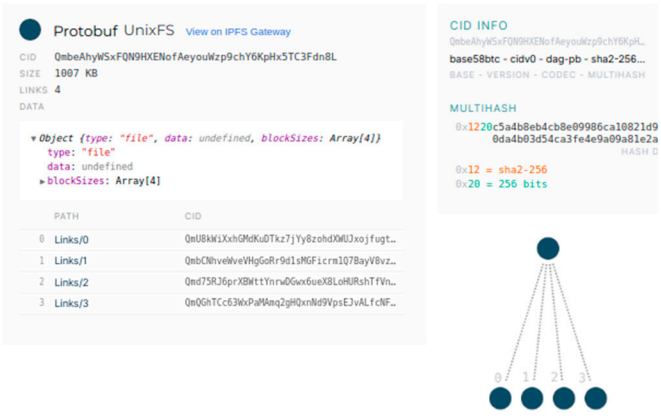


Figure 15. Testing of file distribution on IPFS private network.

IPFS generates a hash to identify the content of a file. Even though files have similar names and sizes, if their content differs slightly, the files will have totally different hashes. This point is shown in Figure 16.

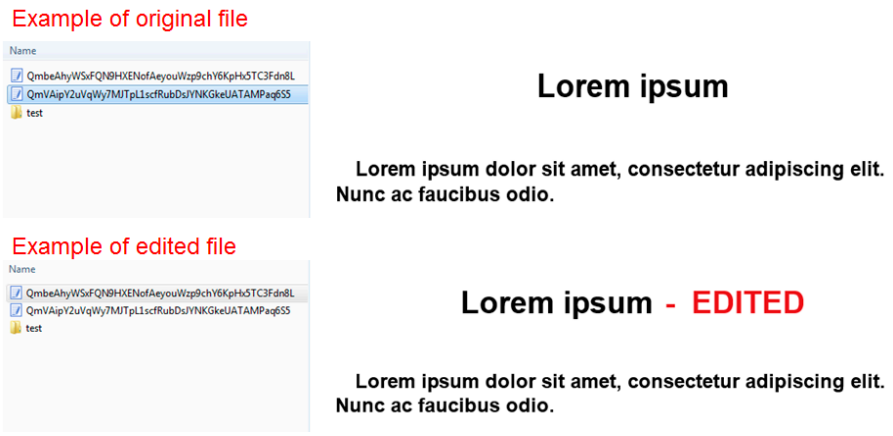


Figure 16. Testing of file integrity on the proposed CEA system.

This experiment uploads two files with similar names and sizes but with very small modifications in their content. The IPFS identifies them with different hashes. This feature of IPFS can prove the integrity of each file. It can help users to check whether files are being tampered with.

5.3. Testing of File Confidentiality

In this paper, the IPFS private network restricts file distribution to nodes with the same swarm key. Checking confidentiality can be achieved by trying to download a file from a public IPFS gateway and

from another IPFS client node outside the private network. In this case, a swarm key in Client Node 1 is removed, so Client Node 1 is no longer registered in the IPFS private network. After Client Node 2 uploads a file, Client Node 1 and the public IPFS gateway try to download and read the content of the file. As seen in Figure 17, neither Client Node 1 nor the public IPFS gateway can download and read the content of the file. The IPFS network sets the local gateway inside the private network.

```
C:\ipfs\test>ipfs cat QmTFBrUYGGDkfPvtyjgpRxnYLADFHzz6m34AsB9hoAz3Yb

Error: Post http://127.0.0.1:5001/api/v0/cat?arg=QmTFBrUYGGDkfPvtyjgpRxnYLADFHzz
6m34AsB9hoAz3Yb&encoding=json&stream-channels=true: context canceled

C:\ipfs\test>ipfs get QmTFBrUYGGDkfPvtyjgpRxnYLADFHzz6m34AsB9hoAz3Yb

Error: Post http://127.0.0.1:5001/api/v0/get?arg=QmTFBrUYGGDkfPvtyjgpRxnYLADFHzz
6m34AsB9hoAz3Yb&encoding=json&stream-channels=true: context canceled
```

(**a**) Testing of confidentiality, using IPFS Client Node outside private network.

(**b**) Testing of confidentiality, using IPFS Public Gateway.

Figure 17. Testing of file confidentiality of the proposed CEA system.

The existing CEA system already has user access control, limiting file access to each user's respective unit. However, users can access files from another unit by suggesting archive ID on the URL, such as http://202.XXX.XXX.XXX/XXX/arsip/edit/ArchiveID. The ID is not encrypted. Moreover, user access control is only implemented in the CEA application. The server administrator can download files directly to the server, without logging into the application.

The testing results show that the IPFS private network in the proposed system maintains the confidentiality of files. The files are only distributed to registered nodes that have the same swarm key for the private network. In addition, the server administrator cannot download files directly into the local node. The files are split, and no single node stores an entire file.

5.4. Transaction Fee (ETH) Analysis

This experiment uses several sizes, including 100 kB, 1 MB, and 10 MB, with the gas price set at 3 GWEI, 10 GWEI, and 18 GWEI, respectively. These gas prices determine how quickly a transaction will be mined. Figure 18 shows that the number of gas prices is linear with the transaction fee. The smaller the gas price, the lower the transaction fee. It does not matter how big the file size is.

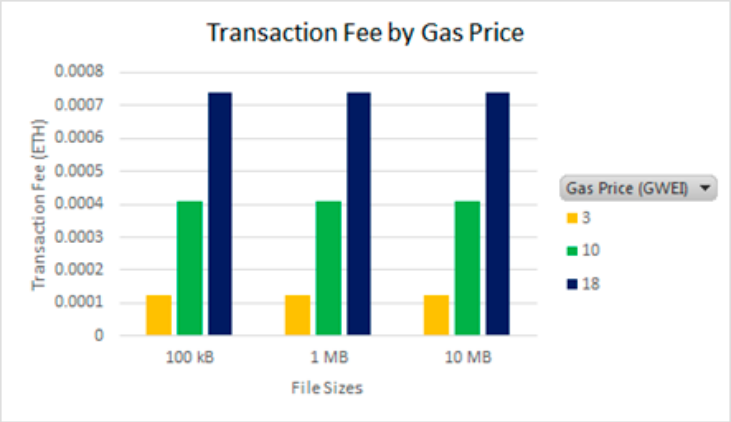


Figure 18. Evaluation of transaction fee.

Currently, the average daily upload to the CEA system is 54 transactions in each unit. Because the balance in the Ethereum private network is customizable, this experiment chooses 18 GWEI with a transaction fee of 0.00074 ETH. The estimation of the yearly transaction fee in daily work is 12,960 transactions * 0.00074 ETH, which is equal to 9.5904 ETH, which is equal to 1787.27 USD. According to Indonesian Government Regulation Number 15 [17], in 2018, the monthly fee for bandwidth 100 Mbps in BPPT was 101,690,000 IDR, or equal to 7222.15 USD. It is 86,665.8 USD per year. Using the proposed system is more cost-effective than using the existing system. In addition, the transaction fee in the proposed system is only charged if users upload or download a file in the CEA system.

5.5. System Performance Analysis

In this section, the measurements focus on latency, bandwidth usage, and throughput. Latency is measured by sending ten packets in ping command. As seen in Figure 19, the round-trip time (RTT) in the existing CEA system is consistent. The RTT is between 9 and 15 ms, with an average latency of 11 ms. In the proposed CEA system, the RTT fluctuates. Because of network congestion, the sixth response goes up to 12 ms. Then, it drops to 2 ms. However, the average latency is 3 ms, which is less than the existing CEA system. Transferring data by using the proposed CEA system is faster than doing so in the existing CEA system.

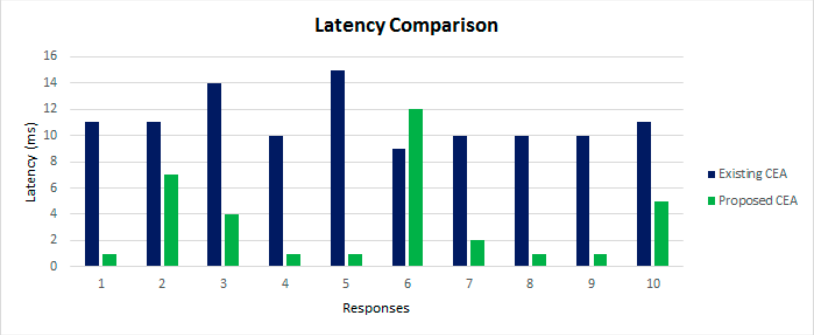


Figure 19. Latency comparison.

The measurement of bandwidth usage refers to the upload and download operations in the proposed CEA system. We use the file sizes of 100 kB, 1 MB, and 10 MB, multiplying by 54 as the average number of transactions in the existing CEA system. As seen in Figure 20, the upload operation consumes more bandwidth than does the download operation. The highest bandwidth usage for uploading 540 MB is 0.084 kB/s, whereas downloading 540 MB is only 0.024 kB/s. Neither uploading nor downloading consumes much bandwidth.

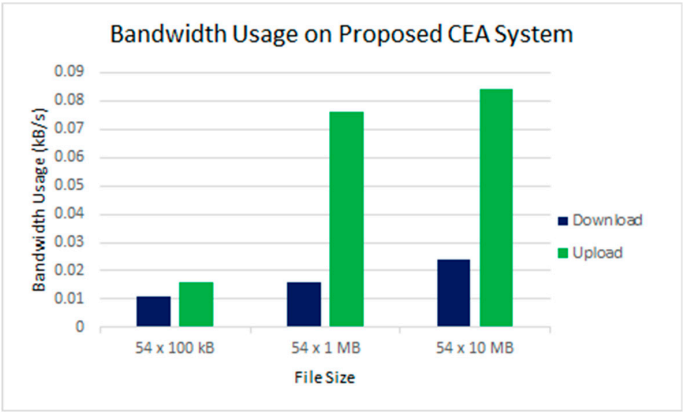


Figure 20. Bandwidth usage.

Furthermore, this paper measures the bandwidth capacity of the proposed CEA system, using speedtest-cli [18]. The proposed CEA system handles 16.64 Mbits/s for downloading and 11.20 Mbits/s for uploading. In addition, iPerf [19] is used to measure the approximate packet size that can be transferred. It shows that the proposed CEA system can transfer a 25.7 Mbyte packet size with an average of 15.5 Mbits/s during 13.9 s. The throughput supports the proposed CEA system for uploading and downloading a maximum packet size of 540 MB.

6. Conclusions

This paper introduced a prototype for the CEA system that integrates an IPFS private network, Ethereum private network (CEA Network), and Decentralized CEA (CEA DApp). The results of the functionality evaluation show files are well distributed among the nodes in the IPFS private network, where all nodes participate in providing data. CEA DApp offers a solution to the single point of failure issue and improves data availability. The nodes without the right swarm key cannot connect to the IPFS private network. This feature ensures data confidentiality, because the files cannot be distributed to the IPFS public network.

Moreover, the hashes of files are stored as transactions on CEA Network. The transactions are unchangeable, digitally signed with Ethereum accounts, timestamped, and knowable by all participants on CEA Network.

The transaction fee estimation shows that the proposed CEA system is more cost-effective than the existing system. The performance evaluation shows that the proposed CEA system has low latency and does not consume much bandwidth. In addition, the throughput results support the bandwidth usage for the maximum transaction estimation.

This system is still a prototype, and improvements are needed. In the future, the scheme of this system needs to be enhanced by implementing archives' lifecycle and involving an off-chain, e-Archive database to store user accounts and transaction logs.

Author Contributions: Supervision, H.J.; Writing—original draft, I.P.; Writing—review & editing, M.E. and H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program, through the National Research Foundation of Korea (NRF-2018R1D1A1B07050380), funded by the Ministry of Education (NRF-), and Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIP) (No.2018-0-00539, Development of Blockchain Transaction Monitoring and Analysis Technology).

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

References

1. National Archives of Indonesia. Ketentuan-Ketentuan Pokok Kearsipan. 1971. Available online: http://www.anri.go.id/assets/download/82UU-Nomor-7-Tahun-1971-Tentang-Ketentuan-Pokok-Kearsipan.pdf (accessed on 12 August 2019).
2. Benet, J. IPFS-Content Addressed, Versioned, P2P File System (Draft 3). Available online: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf (accessed on 14 August 2019).
3. Wang, S.; Ouyang, L.; Yuan, Y.; Ni, X.; Han, X.; Wang, F.Y. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Trans. Syst. Man Cybern. Syst.* **2019**, *49*, 2266–2277. [CrossRef]
4. Alkhudhayr, F.; Alfarraj, S.; Aljameeli, B.; Elkhdiri, S. Information Security: A Review of Information Security Issues and Techniques. In Proceedings of the 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 1–3 May 2019.
5. Ramkumar, M. A blockchain based framework for information system integrity. *China Commun.* **2019**, *16*, 1–17. [CrossRef]
6. Rajalakshmi, A.; Lakshmy, K.V.; Sindhu, M.; Amritha, P. A Blockchain and IPFS based framework for secure research record keeping. *Int. J. Pure Appl. Math.* **2018**, *119*, 1437–1442.
7. Hasan, H.R.; Salah, K. Proof of Delivery of Digital Assets Using. *IEEE Access* **2018**, *6*, 65439–65448. [CrossRef]

8. Yatskiv, V.; Yatskiv, N.; Bandrivskyi, O. Proof of Video Integrity Based on Blockchain. In Proceedings of the 2019 9th International Conference on Advanced Computer Information Technologies (ACIT), Ceske Budejovice, Czech Republic, 5–7 June 2019.
9. Magrahi, H.; Omrane, N.; Senot, O.; Jaziri, R. NFB: A Protocol for Notarizing Files over the Blockchain. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018.
10. Xu, Q.; Song, Z.; Moh Gong, S.R.; Li, Y. Building an Ethereum and IPFS-Based Decentralized Social Network System. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018.
11. Infura.io. Ethereum & IPFS APIs. Available online: https://infura.io/ (accessed on 3 September 2019).
12. Ray, S. What Is a DApp? Available online: https://towardsdatascience.com/what-is-a-dapp-a455ac5f7def (accessed on 25 August 2019).
13. Aung, Y.N.; Tantidham, T. Ethereum-based Emergency Service for Smart Home System: Smart Contract Implementation. In Proceedings of the 2019 21st International Conference on Advanced Communication Technology (ICACT), Pyeongchang, Gangwon-do, South Korea, 17–20 February 2019.
14. Metamask.io. Metamask. Available online: https://metamask.io/ (accessed on 9 September 2019).
15. Ethereum Community. Account Management. Available online: https://ethereum-homestead.readthedocs.io/en/latest/account-management.html#accounts (accessed on 2 September 2019).
16. Iavor, M. Compiling and Smart Contracts: ABI Explained. Available online: https://www.sitepoint.com/compiling-smart-contracts-abi/ (accessed on 15 October 2019).
17. BPK (Badan Pemeriksa Keuangan). Jenis Dan Tarif Atas Jenis Penerimaan Negara Bukan Pajak Yang Berlaku Pada Badan Pengkajian Dan Penerapan Teknologi. Available online: https://peraturan.bpk.go.id/Home/Details/99184/pp-no-51-tahun-2018 (accessed on 29 October 2019).
18. Martz, M. Speedtest-cli 2.1.2. Available online: https://pypi.org/project/speedtest-cli/ (accessed on 2 December 2019).
19. Linode. Network Throughput Testing with iPerf. Available online: https://www.linode.com/docs/networking/diagnostics/install-iperf-to-diagnose-network-speed-in-linux/ (accessed on 12 November 2019).