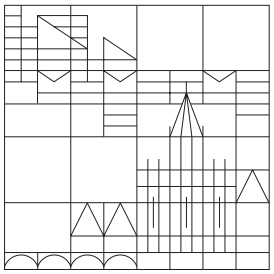
University of Konstanz

Department of Computer and Information Science



Master Thesis

Secure Smart Contracts & Scalable Decentralized Applications (dApps)

in fulfillment of the requirements to achieve the degree of

Master of Science (M.Sc.)

Harsh Kedia

Matriculation Number :: 01/752437

E-Mail :: ⟨harsh⟩.⟨kedia⟩@uni-konstanz.de

Field of Study :: Information Engineering
Focus :: *Applied Computer Science*
Topic :: *Distributed Systems*

First Assessor :: *Prof. Dr. M. Waldvogel*
Second Assessor ::
Advisor :: *Prof. Dr. M. Waldvogel*

Any dedications or other fancy stuff???

Abstract

Humans have evolved over thousands of years building systems which deals with land ownership and property rights. With the advent of Internet our lives has become more and more digital, but we have no experience in managing data ownership. It's clear that data is becoming the new currency in today's digital economy. Big tech companies understood this a long time ago and therefore offered their services free of charge in exchange of our data which they then used to generate profits, control our perception about how we see the world and also tamper with public affairs like the election. There's clearly a need to define data ownership and build systems which enable users to own their data.

With data ownership comes the question of digital identity. How can we identify ourselves over the internet? With username and passwords we can uniquely identify ourselves when using a service, but then we have to create an identity for each service we want to use. It has another drawback, i.e. our passwords are stored on a central server which is prone to hacking. There exists systems like *Google Sign-in* or *Facebook Connect* which allows us to carry our identity across multiple services but then again this identity is not owned by the user but by Google or Facebook. Therefore, there is a need for a self-sovereign identity which is owned by the user and can be verified independently by anyone.

Blockchain along with Public key cryptography allows us to build a Decentralized Public Key Infrastructure (DPKI) thereby empowering users to create self-sovereign identity. Combining self-sovereign identity with encrypted storage enable us to build systems where users own their identity as well as their data.

Blockchain also enables Smart Contracts, which are self executing code and which run when some conditions are met without requiring any intervention by any third party.

Blockchain along with smart contracts and decentralized identity allows us to build interesting applications which are more aligned with the ethos of how the Web was intended in the first place.

This thesis explores technologies like smart contracts and decentralized identity and identifies the properties of a secure decentralized application.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Context

Blockchain technology emerged in 2008 with the creation of Bitcoin, a decentralized protocol for exchanging value among peers on the internet. With the Bitcoin network, it became possible to send value across the internet without any 3rd party.

Soon later, in 2014, Ethereum was invented. It allowed us to create complex applications by writing programs in a Turing complete language. These programs called smart contracts run as they are written and once they are deployed on the blockchain, they become immutable.

Blockchain, the technology which enabled Bitcoin and Ethereum, also enabled the emergence of decentralized applications. But currently, it's not clear, what a decentralized app or dApp is? As of this writing, the most popular platform for building dApps is Ethereum. It uses solidity as it's smart contracting language. Applications built on Ethereum uses a combination of smart contracts along with a traditional web architecture. The front end of the application talks to the smart contract for interacting with the blockchain and uses traditional storage for handling large data sets.

But, do dApps need a smart contract? Is it possible to create dApps without smart contracts? Also, on what specific use cases are smart contracts required?

To explore these questions we created a decentralized file sharing dApp both on Ethereum, a 1 layered protocol and Blockstack, a 2 layered protocol.

1.2 Thesis Statement

In this thesis, we want to explore what constitutes a decentralized application and how it differs from a traditional web application. We will also explore smart contracts, their security aspects, and certain use cases where they are required as part of a decentralized application.

Based on the below metrics, we will analyze our dApp build on Ethereum and Blockstack.

- User Experience
- Scalability

- Security

Above analysis will allow us to explore questions related to smart contract security and application scalability. Results from this analysis can help us determine what constitutes a secure smart contract platform?

At the end of this thesis, we will have a clear understanding of decentralized applications, when using smart contracts makes sense and how to make secure scalable dApps.

Chapter 2

Background

2.1 Pretty Good Privacy (PGP)

PGP[1] is a encryption program which uses public-key cryptography[1] to provide cryptographic privacy and authentication for data communication. It can be also used to sign messages such that the receiver can verify both the identity of the sender and integrity of the message.

It is built upon a Distributed Web of Trust in which a user's trustworthiness is established by others who can vouch through a digital signature for that user's identity[2].

There are a number of inherent weaknesses which prevented the widespread adoption of PGP. These include the following[2]:

- Trust relationships are built on a subjective honor system.
- Only first degree relationships can be fully trusted.
- Levels of trust are difficult to quantify with actual values.
- Issues with the Web of Trust itself (Certification of Endorsement).

2.2 Public Key Infrastructure (PKI)

PKI is a system for creation, storage and distribution of digital certificates which can be used to verify ownership of a public key[3]. In today's Internet, third parties such as DNS registrars, ICANN, X.509 Certificate Authorities (CAs), and social media companies are responsible for the creation and management of online identities. Thus our online identities lie in the control of third-parties and are borrowed or rented rather than owned. This results in severe usability and security challenges[4].

There is a possible alternate approach called *decentralized public key infrastructure (DPKI)*, which returns control of online identities to the entities they belong to. By doing so, DPKI addresses many usability and security challenges that plague traditional public key infrastructure (PKI)[4].

[1] https://en.wikipedia.org/wiki/Pretty_Good_Privacy

2.3 Blockchain

The current Internet Protocol stack consists of four layers: the *Link Layer* puts data onto a wire; the *Internet Layer* routes the data; the *Transport Layer* persists the data; and the *Application Layer* provides data abstraction and delivers it to the end user in the form of applications. All four layers work seamlessly for exchanging of data, but not value. Bitcoin[5] and other cryptocurrencies help define the fifth Internet Protocol layer which enables the exchange of value as fast and efficiently as data[6].

Exchanging value across the Internet presents two challenges. First, every participant in the network must agree upon a shared state and Second, the asset being exchanged should have a clearly defined owner. These challenges are commonly referred as the *Byzantine General's* problem[7] and *Double-spending* problem[8] respectively. Blockchain, the technology underlying Bitcoin and most cryptocurrencies solved the above problems by means of decentralized consensus[2].

At a higher level, blockchains are append-only, totally-ordered, replicated logs of transactions[9]. A transaction is a signed statement that transfers the ownership of an asset from one cryptographic keypair to another. Peers[3] in the network, append new transactions by packaging them into a block and then executing a leader election protocol which determines who gets to append the next block[10]. This election protocol is determined by the underlying consensus algorithm of the blockchain. Each block contains the cryptographic hash of the previous block along with some transactional data.

[2] https://en.wikipedia.org/wiki/Consensus_(computer_science)

[3] A Node having the full copy of the blockchain.

Chapter 3

Decentralized Applications

3.1 Introduction

An Application software or *app* is a computer program designed to perform a specific set of tasks or actions for the end user. There are countless number of applications in use today and the majority of them are web applications following a centralized client-server model[6].

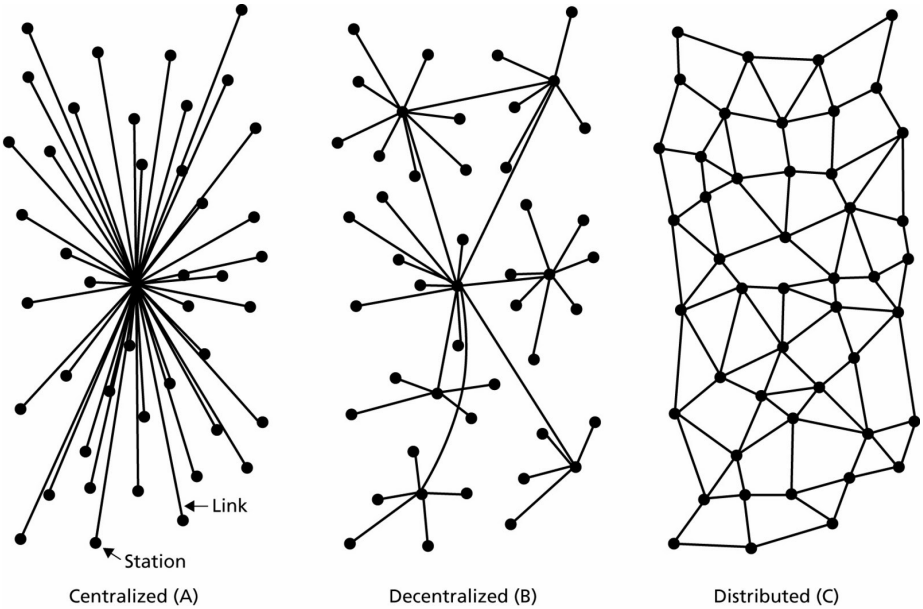


Figure 3.1: The three way of modeling web applications

Figure 3.1 shows a visual representation of three different ways of modeling web applications[11]. Here, *Centralized* and *Decentralized* refers to level of control, while *Distributed* refers to differences of location. Both centralized and decentralized systems can be distributed as well.

3.1.1 Centralized

It's currently the widespread way of building software applications. In this model a central server control the flow of information and governs the operation of individual units. Since the control is centralized, these types of systems suffer from single point of failure risk.

3.1.2 Distributed

In a Distributed model, the control still resides with a central server, however, the computation is spread across multiple nodes or servers.

3.1.3 Decentralized

In a Decentralized model, there is no central point of control as it's spread across all the servers running the application. Applications built using this model do have a single point of failure and are inherently fault tolerant.

3.2 Enabling Technologies and Concepts

3.2.1 Data

3.2.2 Identity

3.2.3 Value

3.2.4 Computing

3.2.5 Bandwidth

Chapter 4

Smart Contracts

Chapter 5

PoC 1: Ethereum dApp

5.1 Introduction

Existing applications for sharing files are central solutions and therefore suffer from single point of failure risk. Moreover, using central services for securing data means that we have to trust a 3rd party with our data thus exposing it to manipulation risks. Hence, a decentralized application is required to overcome the problems posed by a central application. With the recent developments in Blockchain technology and P2P storage, it is possible to securely store and share data without using any central server.

This chapter describes the workings of the application *dShare*[12] built using P2P technologies enabling a secure way of storing and sharing data between two individuals or entities. The latest version of the application is deployed at https://file-share-dapp.herokuapp.com/

5.2 Technologies Used

5.2.1 Ethereum

Ethereum[16] is a blockchain platform for building decentralized applications. It allows the creation of *Smart Contracts*. Solidity[1] is the primary language for writing smart contracts on Ethereum.

5.2.2 InterPlanetary File System (IPFS)

IPFS[13] is a peer-to-peer file transfer protocol which enables a shared file system between all its connected peers. It achieves this by combining previous peer-to-peer systems such as Distributed Hash Tables (DHT), BitTorrent[14], and Git[15]. The data in the IPFS network are modeled as a Merkle DAG[2] thus providing a throughput storage system with content-addressed hyperlinks.

[1] https://github.com/ethereum/solidity

[2] Merkle directed acyclic graph - similiar to a Merkle tree data structure however they do not need to be balanced and its non-leaf nodes can contain some data.

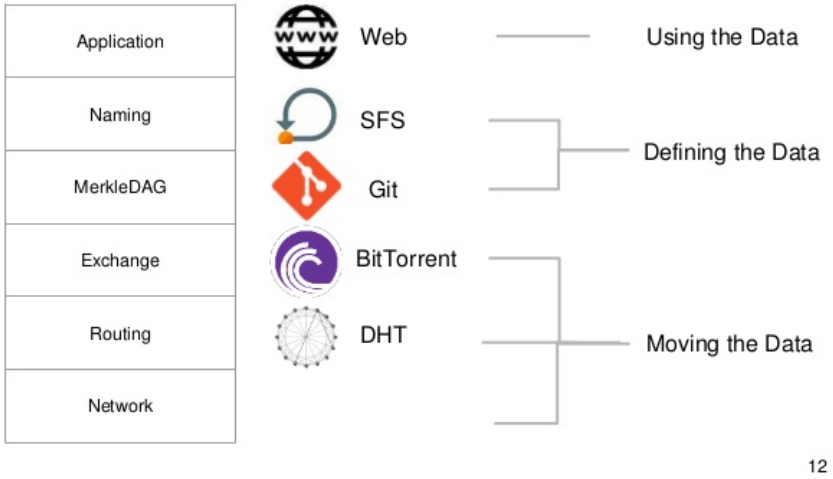


Figure 5.1: IPFS Stack

Figure 5.1[3] shows the IPFS Stack. It consists of sub-protocols, each providing a different functionality.

- Identities - node identification and verification.
- Network - connection management among peers.
- Routing - peer lookup using DHT.
- Exchange - data exchange strategies among peers.
- Objects - a content-addressed Merkle DAG.
- Files - versioned file system.
- Naming - A self-certifying mutable name system.

5.2.3 OriginStamp

OriginStamp[17] is a blockchain based system for decentralized timestamping. It uses the Bitcoin blockchain for the creation of trusted and immutable timestamps for any piece of data. Timestamps created by OriginStamp can be verified independently by anyone.

Figure 5.2 visualizes the timestamping process as implemented in OriginStamp. When a user submits a file, the hash of the data is recorded. It combines all the hashes submitted over a period of time and generates an aggregated hash. After some additional hashing and encoding operations, a Bitcoin address is created to which the smallest possible transactional amount of Bitcoins is transferred. Performing this transaction embeds the hash and the timestamp permanently to the Bitcoin blockchain. Each transaction is part of a block and

[3]Adopted from: https://image.slidesharecdn.com/ipfs-171229085327/95/ipfs-12-638.jpg?cb=1514537643

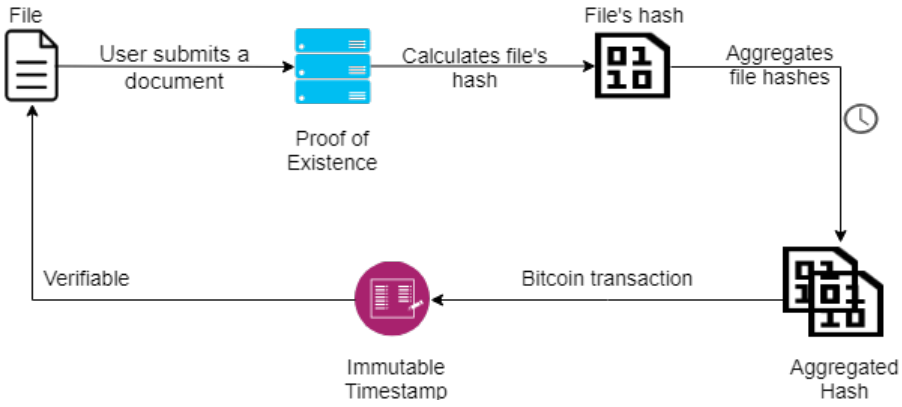


Figure 5.2: Timestamping using OriginStamp

is added to the Bitcoin blockchain by a process called mining. Since each block is linked cryptographically to the previous block, adding a new block confirms the validity of the last block. Changing the timestamp of a transaction becomes impossible once five or six subsequent blocks are mined, which requires an hour on average[5].

5.3 Implementation

For storing of files we used IPFS. Before uploading to the IPFS network, files are encrypted using AES-GCM[4] encryption mechanism. Sharing of encryption keys is facilitated using smart contracts built on Ethereum; thus files can be shared by anyone with an Ethereum address. Finally, OriginStamp is used for immutable timestamping.

The front-end of the application is built using React.js[5], a JavaScript library for building user interfaces. Solidity was used for writing smart contracts and deployed on the Ethereum test network, Rinkeby[6]. Next.js[7] was used for server-side rendering (SSR)[8], and Firebase[9] was used as a database for storing public Ethereum key of the users.

5.4 Working

This section describes the working of the different components of the application.

5.4.1 Smart Contract

The smart contract serves as the bridge between the front-end of the application and the Ethereum Blockchain. Data is read from and written to the blockchain

[4] https://www.aes-gcm.com/
[5] https://reactjs.org/
[6] https://www.rinkeby.io
[7] https://nextjs.org/
[8] https://nextjs.org/features/server-side-rendering/
[9] https://firebase.google.com/

with the help of function calls in the contract. Each function call which modifies some data requires a small fee in the form of gas[10] which defines the cost for a function execution in Ether. Reading from the blockchain does not require any fees.

The application makes use of two contracts, *FileFactory*, which acts as the factory contract for creation of new files and *File*, which represents an individual file.

FileFactory

FileFactory is the contract which is deployed on the Rinkeby test network. It has several mappings which stores the list of file contracts uploaded by a user. Whenever a user uploads a file, a function call is made to the *FileFactory* contract, which in turn deploys the *File* contract and updates the mappings for list of uploaded files and the respective uploader.

File

File contract is deployed to the blockchain whenever a file is successfully uploaded to the IPFS network using the application. Upon deployed, the constructor function is called. It takes the values passed by the *FileFactory* contract and saves the details to its *File* contract.

5.4.2 File Upload

Figure 5.3 visualizes the working of the application when a user uploads a file.

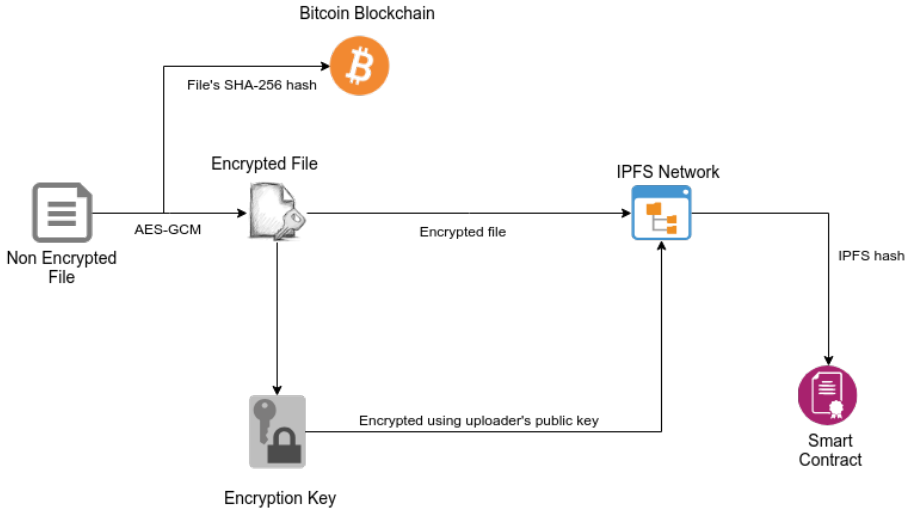


Figure 5.3: File Upload

As soon as a user submits a file to be uploaded, it's SHA-256[11] hash is calculated and a timestamp is created by submitting the hash to the bitcoin

[10] https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas
[11] https://www.movable-type.co.uk/scripts/sha256.html

blockchain using the OriginStamp API[12].

Next, the file is encrypted using the SubtleCrypto[13] interface with 'AES-GCM'[14] as the encrypting algorithm. The encrypted data is then combined with the random salt to generate a `Uint8Array` buffer ready to be uploaded to the IPFS network.

The key used to encrypt the file is converted to `JSON` and is encrypted using the uploader's Ethereum public key which is retrieved from the database. This encrypted key and the encrypted data is then uploaded to the IPFS network. Once the file is successfully uploaded, `createFile()` in the `FileFactory` contract is called which deploys a new `File` contract with all details regarding the file saved to the blockchain.

5.4.3 File Sharing

Sharing a file requires the recipient's Ethereum address and uploader's private key. Figure 5.4 visualizes the working of the application when a user shares a file with another user.

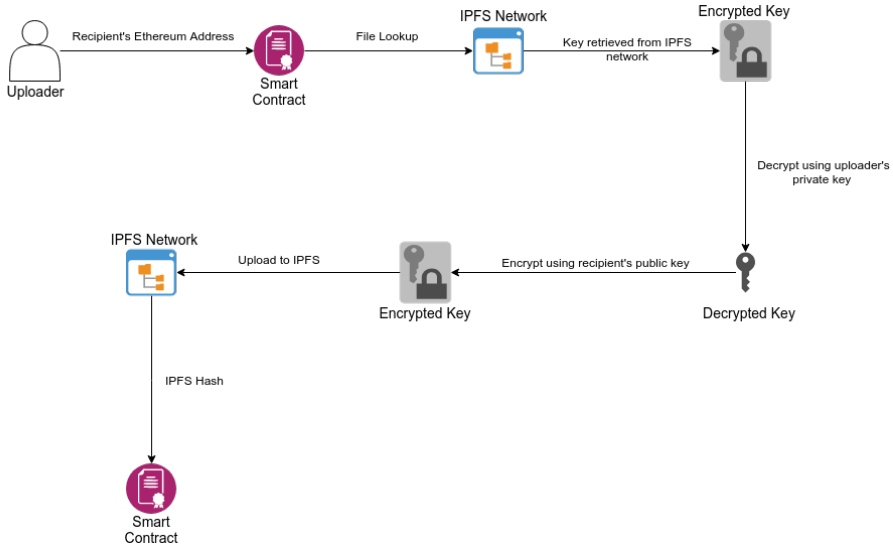


Figure 5.4: File Sharing

Firstly, the file's IPFS location is retrieved from the `File` contract. From this location, the encrypted key is download and decrypted using uploader's private key. Once decrypted, the key is again encrypted using a recipient's public key. The new encrypted key is again uploaded to the IPFS network. Finally, the IPFS location of the key is saved into the `File` contract by calling `shareFile()`.

To stop sharing a file, a function call can be made to the `File` contract with the recipient's address, which deletes the contract reference from the `recipientFiles` array.

[12] http://doc.originstamp.org/
[13] https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto
[14] https://en.wikipedia.org/wiki/Galois/Counter_Mode

5.4.4 File Download

Downloading a file requires the user's Ethereum private key. Depending on whether the file is uploaded or shared one, corresponding function from the `File` contract is called to retrieve the file's details. The key is then decrypted using user's private key and is converted to a valid JSON web key (jwk)[15] format. The encrypted file data is then converted to a file buffer, and the original file content and the random salt used for encrypting the file is retrieved. Finally, the file is decrypted and saved to the user's local storage.

5.4.5 File Archiving

Instead of deleting a `File` contract, the application provides a way to achieve files. This is also useful to keep track of archived files and restore them at a later date if required. When a file is archived, the `File` contract address is saved in an array which is later used for filtering the archived files from the UI. Restoring a file removes the `File` contract address from the archived files array.

[15] https://tools.ietf.org/html/rfc7517

Chapter 6

PoC 2: Blockstack dApp

6.1 Introduction

6.2 Technologies Used

6.3 Implementation

6.4 Working

Chapter 7

Results

Chapter 8

Discussion

Chapter 9

Conclusion

Appendix A

Acknowledgements

Bibliography

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, ser. The William Stallings books on computer and data communications technology. Prentice Hall, 1999. [Online]. Available: https://books.google.de/books?id=Dam9zrViJjEC

[2] D. Wilson and G. Ateniese, "From pretty good to great: Enhancing pgp using bitcoin and the blockchain," in *International conference on network and system security*. Springer, 2015, pp. 368–375.

[3] J. Weise, "Public key infrastructure overview," *Sun BluePrints OnLine, August*, pp. 1–27, 2001.

[4] C. Allen, A. Brock, V. Buterin, J. Callas, D. Dorje, C. Lundkvist, P. Kravchenko, J. Nelson, D. Reed, M. Sabadello *et al.*, "Decentralized public key infrastructure. a white paper from rebooting the web of trust," 2015.

[5] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[6] S. Raval, *Decentralized applications: harnessing Bitcoin's blockchain technology*. " O'Reilly Media, Inc.", 2016.

[7] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[8] U. W. Chohan, "The double spending problem and cryptocurrencies," *Available at SSRN 3090174*, 2017.

[9] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Research perspectives and challenges for bitcoin and cryptocurrencies (extended version)," *Cryptology ePrint Archive, Report 2015/452*, 2015.

[10] J. Nelson, M. Ali, R. Shea, and M. J. Freedman, "Extending existing blockchains with virtualchain," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[11] P. Baran, "On distributed communications," 1964.

[12] H. Kedia, "hKedia/dShare: First release of dShare built with Ethereum," Aug. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3359852

[13] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[14] B. Cohen, "The bittorrent protocol specification," 2008.

[15] J. Loeliger and M. McCullough, *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.", 2012.

[16] V. Buterin *et al.*, "Ethereum: A next-generation smart contract and decentralized application platform," *URL https://github. com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper*, vol. 7, 2014.

[17] T. Hepp, A. Schoenhals, C. Gondek, and B. Gipp, "Originstamp: A blockchain-backed system for decentralized trusted timestamping," *it-Information Technology*, vol. 60, no. 5-6, pp. 273–281, 2018.