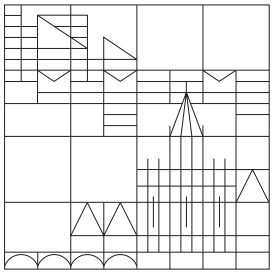
University of Konstanz

Department of Computer and Information Science



Master Thesis

Building Decentralized Applications with Data Ownership

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 :: *Prof. Dr. S. Kosub*
Advisor :: *Prof. Dr. M. Waldvogel*

Abstract

Information requires data and an economy requires a unit of account. It's important that in the *Information Economy* enabled by the internet, we own our data. Blockchain with its native token enables a unit of account. It also serves as the base layer for building a Decentralized Public Key Infrastructure (DPKI) enabling every individual to have a Self-Sovereign Identity. The concept of users owning their digital identity enables decentralized applications with Data Ownership.

This thesis explores the different concepts which make up a web application and describes how each can be decentralized using Blockchain and other peer-to-peer protocols. We analyze the state-of-the-art in decentralized applications platforms by building two *proof-of-concept* applications for file sharing, one built on Ethereum and second built on Blockstack.

The main aim of the thesis is to provide its readers with a better understanding of what Blockchain technology enables and showcase certain properties of a secure decentralized applications platform.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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.

To define a model for Data Ownership, lets looks at Land Ownership. In a land ownership model, at any given point in time, a property has a fixed Geo-location while the owner can be anywhere in the Geo-space. Conversely, In a data ownership model, at any given point in time, a user has a fixed identity while the data can be anywhere on the internet.

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.

Chapter 2 introduces the concept and technologies which serve as building blocks for decentralized applications. In Chapter 3 we learn the different types of web applications and the different concepts which make them up. We explains how each concept have evolved with the internet and how they will evolve because of Blockchains and newly emerging peer-to-peer protocols. We also describe the underlying architecture for decentralized applications. Chapter 4 explains the architecture and workings of two applications built to analyze the current state-of-the-art in decentralized applications platforms. We focus

on Ethereum and Blockstack, two popular decentralized applications ecosystem. Chapter 5 compares the two application platforms based on their architecture and features. We analyze the two applications based on how are are constructed and compare their performance using Chrome DevTools. We also compare similar protocols as used in the applications. Results are discussed in Chapter 6 and this thesis concludes with Chapter 7 where we also give a short outlook into the future.

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 Distributed Hash Table (DHT)

Distributed Hash Tables (DHTs)[5] provides a way of routing data in peer-to-peer systems. It works by enabling a lookup service similar to hash tables where (key, value) pairs are distributed across the network such that any peer can retrieve the value associated with a given key efficiently. DHTs forms a base infrastructure on top of which others decentralized applications are built. Kademlia[6] is a popular DHT which provides an efficient lookup across massive networks. S/Kademlia[7] is another DHT which extends Kademlia to protect nodes against malicious attacks.

2.4 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[8] and other cryptocurrencies help define the fifth Internet Protocol layer which enables the exchange of value as fast and efficiently as data[9].

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[10] and *Double-spending* problem[11] 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[12]. 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[13]. 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

Concepts & Design

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[9].

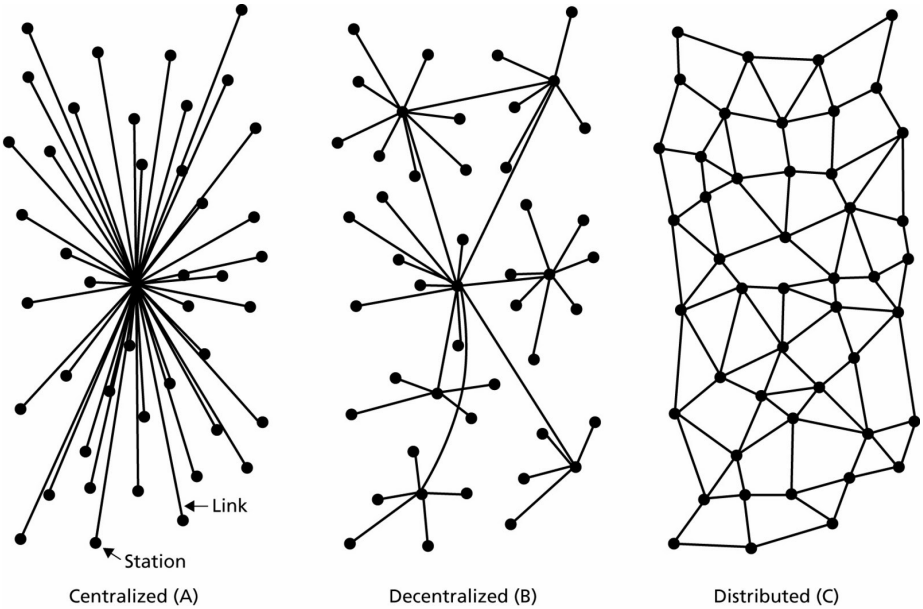


Figure 3.1: The three way of modeling web applications

Figure 3.1 shows a visual representation of three different ways of modeling web applications[14]. 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 don't have a single point of failure and are inherently fault tolerant.

3.2 Enabling Technologies

The document *Information Management: A Proposal[15]* written by *Sir Tim Berners-Lee*[1] conceived the ideas for what would become the WorldWideWeb. It's main goal was to enable information exchange between computers in an accessible way at CERN[2].

HTML[3], URI[4] and HTTP[5] were the fundamental technologies that defined the foundation of the Web. HTTP connected every computer on the planet with a common protocol. The HTTP protocol guidelines defined a set of trusted servers that translated a web address into a server address. Furthermore, HTTPS[6] added another layer of trusted servers and certificate authorities. People would host personal servers for others to connect to, and everyone owned their data[9]. As the Web evolved, applications servers[7] became the common way of interactive with the Web and the centralized model of data ownership as we know it today was born[9]. It was conceptually and programmatically easier to maintain an application server and profit from user's data that utilize it.

Blockchain is the primary technology that enables the creation of applications with a decentralized model of data ownership. It puts the users of an application in control of their data thereby enabling a more open Web, as it was originally intended[8].

The blockchain helped solve the Byzantine Generals Problem[10]. This problem describes a situation where all participating nodes in a distributed network must agree upon every message that is being transmitted between nodes, but where some of the nodes are corrupt and disseminating false information or

[1] https://www.w3.org/People/Berners-Lee/
[2] https://home.cern/
[3] https://developer.mozilla.org/en-US/docs/Web/HTML
[4] https://tools.ietf.org/html/rfc3986
[5] https://tools.ietf.org/html/rfc2616
[6] https://tools.ietf.org/html/rfc2818
[7] https://en.wikipedia.org/wiki/Application_server
[8] https://webfoundation.org/about/vision/history-of-the-web/

are unreliable. This agreement is called as **consensus**. With Bitcoin[8], decentralized consensus became possible. Agreement is achieved in the Bitcoin network by way of *proof-of-work*[9] consensus mechanism which is resistant to Sybil Attack[16]. Proof-of work is both computationally and energy expensive; other consensus mechanism such as *proof-of-stake*[10] relies on stake in the system instead of computational power.

3.3 Application Concepts

There are five concepts in a web application that have traditionally been implemented in way that puts control with a centralized entity: data, identity, value, computing and bandwidth[9]. Each of these require trust in a 3rd party - a trust which can be betrayed. Recent advancements in distributed-system technology can put users in control of these things. Below sections describes each concept in detail and shows how one can build applications in a way such that centralized control is not required.

3.4 Data

Data is the most important concept in any web application. First, let's look at how traditional web applications interact with data. Whenever, a user logs into an application, the application connects to a remote server and sends the authentication details. These details lets the server know which user is interacting with the application. Once authenticated, the user data is fetched from the remote storage and displayed to the user. All complex computations and data storage occurs on dedicated servers maintained in the cloud.

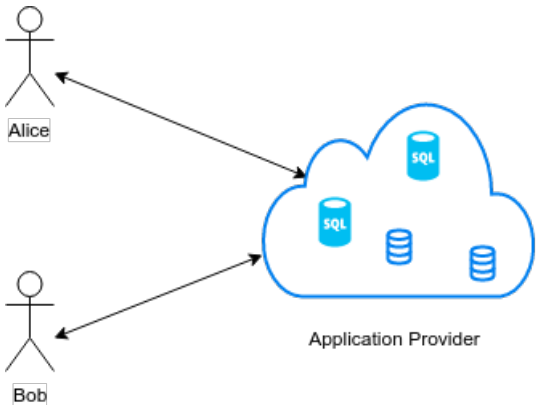


Figure 3.2: How traditional web applications work

Figure 3.2 shows how users interact with a traditional web application. Whenever, Alice wants to interact with Bob, she sends a message to the application server which then delivers the message to Bob. There is no direct path between Alice and Bob. This results in centralization where the provider acts

[9] https://en.bitcoin.it/wiki/Proof_of_work
[10] https://en.bitcoin.it/wiki/Proof_of_Stake

as an intermediary between Alice and Bob's interaction; effectively governing how data is stored and shared among them.

This model of application interaction requires that we trust the providers with out data and hope that they won't misuse or sell our data without our permission. Since revelations by Edward Snowden[17], we now know that trust can, has and will be broken as long as we entrust our data to a central entity[9]. Centralized stores of data also serve as a tool for surveillance, allowing big entities to monitor our internet behaviour without our knowledge. Cloud providers, despite having a distributed backend, are centrally owned.

Additionally, as we move from a labor-based economy to an information-based economy, data will become the primary form of value. Therefore, its important that we not only possess our data, but own it as the world evolves. An ideal solution that enables this should provide a way of storing data in a decentralized way that is robust and as trustless as possible[9].

3.4.1 Storing Data Directly in a Blockchain

This method does solves the decentralization of data as everyone who has copy of the Blockchain is storing the data but cannot alter it. The data can be encrypted such that it can only be accessed by someone having the private key. However, Blockchains were not meant for storing large amounts of data. It was designed for storing simple transactional logs, as its evident from looking at the Bitcoin Blockchain where individual records are in bytes[11] and therefore cannot hold much data. Moreover, the Blockchain architecture implies that each node in the network must store a complete copy of the Blockchain. Therefore, storing a significant amount of data on a Blockchain is both expensive and impractical.

3.4.2 Storing Data in a Distributed Hash Table

DHTs ensure data resiliency by enabling easy distribution and indexing of data. Early peer-to-peer (P2P) filesharing applications like KaZaA[18], Napster[19] and Gnutella[20] used their own versions of DHTs with a varying degree of decentralization. Some had centralized trackers to monitor the movement of data, while some had central sources from which all data must pass, leaving them with single point of failure[9].

BitTorrent[21] was the first protocol to popularize DHT for sharing of large files. As of 2013, BitTorrent has 1527 million concurrent users at any given time[22]. The BitTorrent Mainline DHT serves as a decentralized data store with centralized trackers to monitor the network. It uses a tit-for-tat strategy between seeders and leechers to maximize the bandwidth of data distribution. This makes the BitTorrent protocol the de facto method for transfer of large datasets over the web. However, using BitTorrent as a data store is not feasible as there is no incentives for nodes to store a data leading to *data impermanence*.

Along with the decentralized storage capabilities of DHT and the speed of BitTorrent's protocol, we also want data permanence. Therefore, its necessary to incentivize the nodes storing the data in some way. Moreover, we also need to ensure that the links to the data don't die, an idea which was first proposed in Project Xanadu[23].

[11] https://en.bitcoin.it/wiki/Transaction

InterPlanetary File System (IPFS)

IPFS[24] is a peer-to-peer file transfer protocol that implements these features, enabling a more permanent, decentralized Web, where links don't die and no single entity controls the data. It achieves this by combining previous peer-to-peer technologies such as DHT, BitTorrent and Git[25]. Data in the IPFS network is modeled as a *merkleDAG*[12], a simple data structure that can be conceptualized as a series of nodes connected to each other[9]. This makes IPFS a *content-addressed*[13] system allowing efficient data lookup and retrieval as it doesn't rely on a single server to access the data.

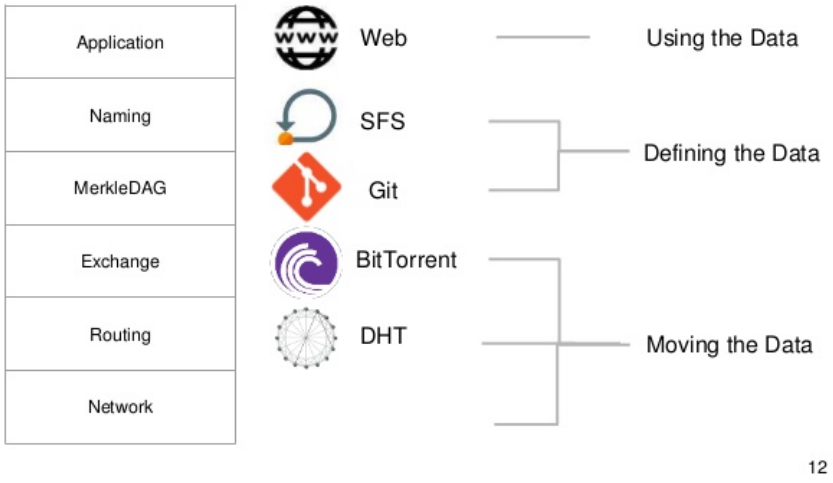


Figure 3.3: IPFS Stack

Filecoin[26], a *decentralized storage network*, combines IPFS (shown in Figure 3.3[14]) with an incentive structure that turns cloud storage into an algorithmic market effectively overcoming the limitations of BitTorrent. This market runs on a blockchain with a native protocol token, which miners earn by providing storage to clients.

3.4.3 Storing Data in a Cloud in Encrypted Containers

IPFS enables a way for decentralized storage such that it overcomes the single point of failure risk, however, the user does not control where the data is being stored. Because of this, once a data is added to the network, and is picked up by other nodes for re-hosting, it cannot be taken back as there is no way to verify information deletion[15].

[12]Similar to a Merkle tree data structure, however, they do not need to be balanced and it's non-leaf nodes can contain some data.

[13]A content-addressed storage is way to store information such that it can retrieved based on its content and not its location.

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

[15]https://github.com/ipfs/faq/issues/9

Gaia: User-Controlled Storage

Gaia[27] is a decentralized storage system that enables user-controlled private data lockers. It works by hosting data in one or more existing storage systems of user's choice. Data on Gaia is encrypted and signed by user-controlled cryptographic keys before being uploaded to a provider. Storing data using existing cloud infrastructure ensures high data availability without compromising on application performance. Further, since each data is signed by keys which a user controls, it ensures that they don't need to trust the underlying cloud providers.

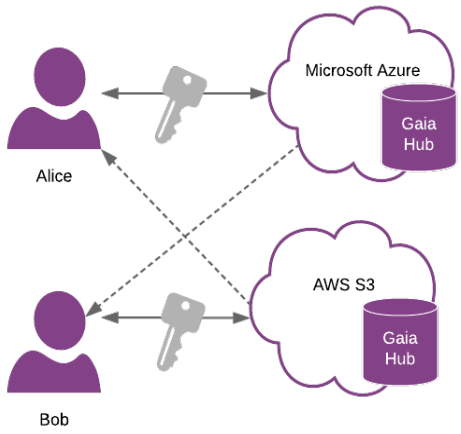


Figure 3.4: How users interact with Gaia storage[16]

Writing data to a Gaia hub involves a `POST` request along with a signed *authentication* token. This token is signed by the private key which controls the particular bucket being written to. Separate buckets are used for each application, this ensures that a given private key grants access only to a specific bucket on the Gaia Server.

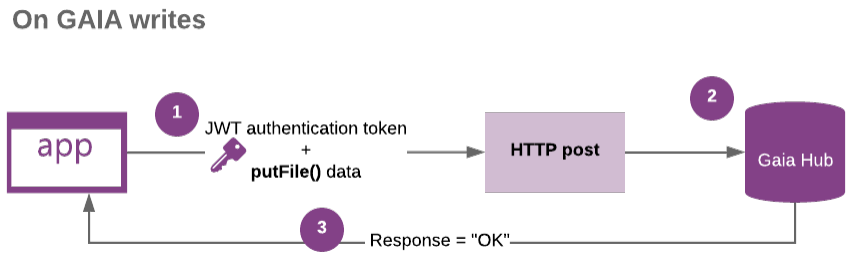


Figure 3.5: The token ensures that a write is authorized[17]

[16] https://docs.blockstack.org/storage/overview.html

Reading data from a user's Gaia hub involves a *zonefile* lookup. This zonefile is a signed JSON object containing the URLs pointing to the user's Gaia data locker. Once verified that the zonefile is signed by user's key, a standard `HTTP` request is made to fetch the requested data.

Figure 3.6 shows an overview of Gaia. Looking up data for a name works as follows:

1. Lookup the *name* in the Virtualchain to get the (*name, hash*) pair.
2. Lookup the *hash(name)* in Peer Network to get respective zonefile.
3. Get the user's Gaia URL from the zonefile and lookup the URL to connect to storage backend.
4. Fetch the requested data and verify the respective signature or hash.

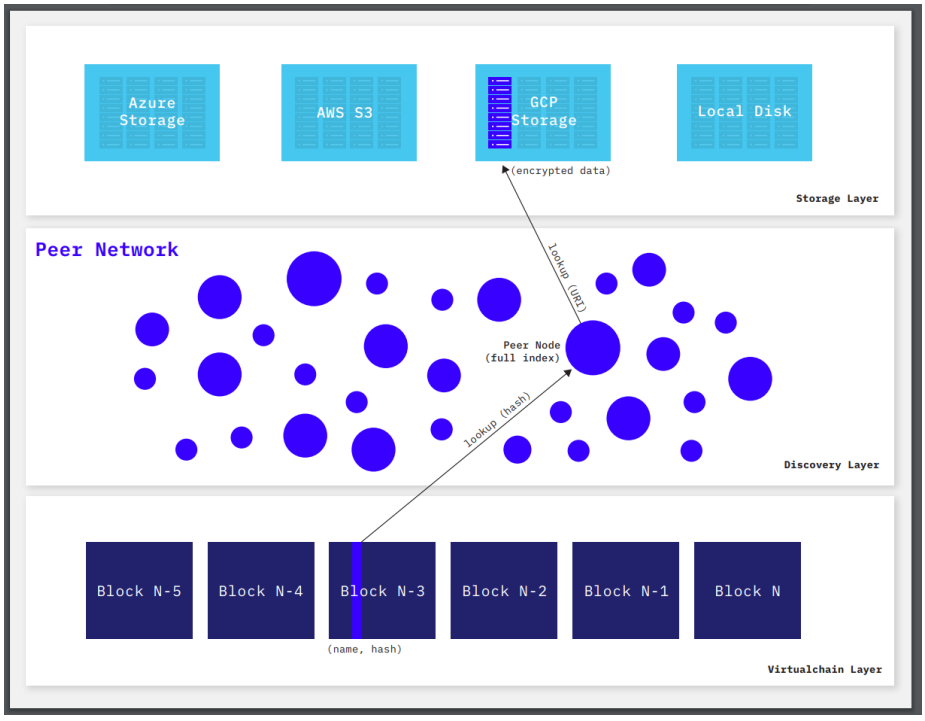


Figure 3.6: Overview of Gaia and steps for looking up data.[18]

3.5 Identity

User identities are essential to using internet applications. In order to confirm their identity, users must provide some information. This information depends on the application's authentication process. If an application maintains a user

[17] https://docs.blockstack.org/storage/authentication.html
[18] https://blockstack.org/whitepaper.pdf

database, it requires a username and a password and sometimes a second factor. If an application relies on a third-party, like Google[19] or Facebook[20] for identity management, it uses the OAuth 2.0 authentication[28] protocol, which identifies a user by generating an assertion from the identity service.

Third party identity providers implementing the OAuth[21] protocol often have their own version of implementation which leads to identity fragmentation across the web. OpenID[29] is a decentralized identity protocol that allows users to create one identity which can be carried across multiple providers. However, the user still needs to trust one of the service providers with their identity information[9].

Another way of authenticating users with an application is by using digital certificates[30]. These certificates provide a proof of ownership of a public key. Authentication and management of public keys is facilitated using a Public Key Infrastructure (PKI)[31] system. Currently, the most common approaches to PKIs are: Certificate Authorities (CAs) and PGP Web of Trust[32].

A Certificate Authority (CA) acts a trusted third party responsible for management and distribution of digital certificates for a network of users[33]. These trusted third parties introduces central control in a PKI system making them prone to single point of failure risk[34].

In PGP Web of trust, authentication is entirely decentralized; users are able to designate others as trustworthy by signing their public keys. This process generates a digital certificate containing the user's public key and signatures from entities that have deemed him trustworthy. This system does benefit from its distributed nature as there is no central control. However, PGP does not offer identity retention. There is no guarantee of consistency and nothing prevents multiple users from creating public keys for the same identity[33].

3.5.1 Decentralized Public Key Infrastructure (DPKI)

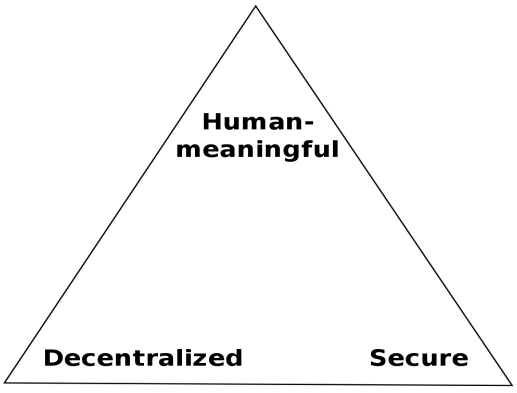


Figure 3.7: Zooko's Triangle[22]

[19] https://developers.google.com/identity/
[20] https://developers.facebook.com/docs/facebook-login/
[21] https://oauth.net/
[22] https://commons.wikimedia.org/wiki/File:Zooko%27s_Triangle.svg

The foundational precept of a DPKI is that *identities belong to the entities they represent*. This requires designing a *decentralized* infrastructure where every identity is controlled by its *principal owner* and not by some trusted third party[4].

DPKI is essentially a system to gives unique names to participants in a network protocol. These names should have three desirable traits(Zooko's triangle) namely: *Human-meaningful*, *Decentralized* and *Secure*. OpenID solved security and human-meaningfulness.

Namecoin[23] was the first working solution that satisfied all three traits of Zooko's triangle by adding decentralization. It was the first fork of Bitcoin[8] protocol designed to act as a decentralized domain name server (DNS) for *.bit* addresses. Namecoin's blockchain essentially could be used as an intermediary between a user and the service requesting user's identity[9]. It allowed a user to *register* a name by creating a blockchain transaction. This transaction included the desired name which get embedded under the */id* namespace.

3.5.2 Decentralized Identifiers (DIDs)

A DID[24] is a new type of identifier that is globally unique, resolvable with high availability and cryptographically verifiable. DIDs typically contains cryptographic key pair which enables the controller of a DID to prove control over it.

The concept of global unique decentralized identifiers is not new; Universally Unique Identifiers[25](UUIDs), also called Globally Unique Identifiers (GUIDs) were first such identifiers developed in 1980s and formally specified in RFC4122[26]. Another class of identifiers knows as persistent identifiers was standardized as Uniform Resource Names (URNs) by RFC8141[27].

However, UUIDs are not globally resolvable and URNs, if resolvable, require a central registry. Further, neither UUIDs or URNs have the ability to cryptographically verify ownership of the identifier. DIDs, on the other hand, fulfills all four requirements: persistence, global resolvability, cryptographically verifiability, and decentralization required for a self-sovereign identity[35].

Figure 3.8: The URN Specification

The DID (Figure 3.9[28]) specification follows the same pattern as the URN (Figure 3.8[29]) specification. The key difference is that with DIDs, the names-

[23] https://www.namecoin.org/
[24] https://w3c-ccg.github.io/did-spec/
[25] https://en.wikipedia.org/wiki/Universally_unique_identifier
[26] https://tools.ietf.org/html/rfc4122
[27] https://tools.ietf.org/html/rfc8141
[28] https://w3c-ccg.github.io/did-primer/did-primer-diagrams/did-format.png
[29] https://w3c-ccg.github.io/did-primer/did-primer-diagrams/urn-format.png

Figure 3.9: The DID Specification

pace component identifies a *DID method* which defines how DIDs work with a specific blockchain. The *DID method* specifications must define the format and generation of the method-specific identifier. The DID methods can also be developed for identifiers registered in federated or centralized identity management systems, thus creating a interoperability bridge between the worlds of centralized, federated and decentralized identifiers.

3.5.3 Related Work

Blockstack Authentication

Blockstack[30] is a decentralized computing network and app ecosystem that uses public-key cryptography and blockchain technology to provide its users with a DPKI system thereby making it possible to have a universal username that works across applications without the need for any passwords.

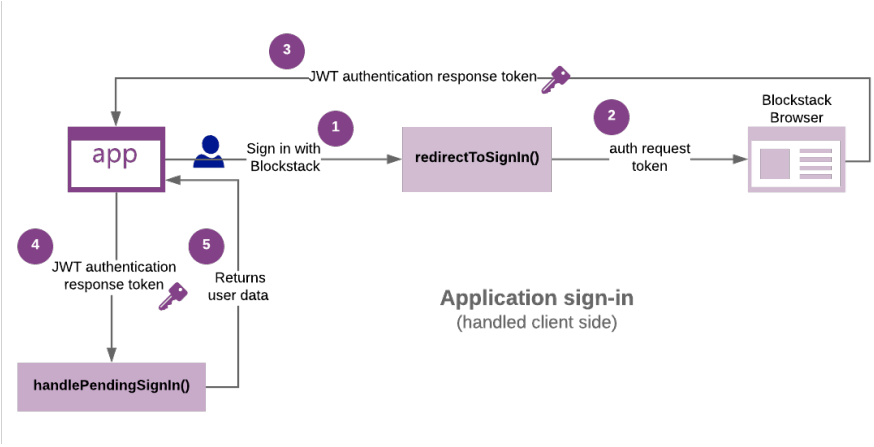


Figure 3.10: Blockstack Authentication Flow[31]

Blockstack Auth[32] is a token-based authentication[33] system. When a user signs into an application, an `authRequest` token is sent to the Blockstack Browser. Once the sign-in is approved, the Blockstack Browser responds with

[30] https://blockstack.org/
[31] https://docs.blockstack.org/storage/images/app-sign-in.png
[32] https://docs.blockstack.org/develop/overview_auth.html
[33] https://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/

an `authResponse` token. This response provides the application with enough information to generate and store authentication data. Since all usernames are registered on the blockchain, every application have an up-to-date view of all usernames and corresponding public keys. This eliminates the need for a server-side identity provider.

Figure 3.10 visualizes the authentication flow for an application using Blockstack Auth. When a user chooses to **Sign in with Blockstack**, it calls the `redirectToSignIn()` method which sends an `authRequest` token to the Blockstack Browser[34]. Upon receiving the request, the Blockstack Browser presents the user with a choice of IDs to use to sign in, as well as a list of permissions the application needs. Upon selecting an ID, the Blockstack Browser redirects back to the application with a `authResponse` token containing three pieces of information[35]:

- The user's username (or the hash of the public key if no username is set).
- An *application-specific* private key for encrypting and signing user's data. This key is deterministically generated from the user's master private key, the ID used to sign in, and the application's HTTP Origin.
- The URLs to the user private data locker (Gaia Hub[36]).

Above pieces of information instructs the application on how to find and store a user's data on their behalf.

3.6 Value

Value help us to quantify quantities of an asset in order to make them measurable[37]. It can mean money, intellectual property or securities. In traditional internet applications trust is required in a third-party (a bank or a financial institution) when transferring value between two entities. While this system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model[8].

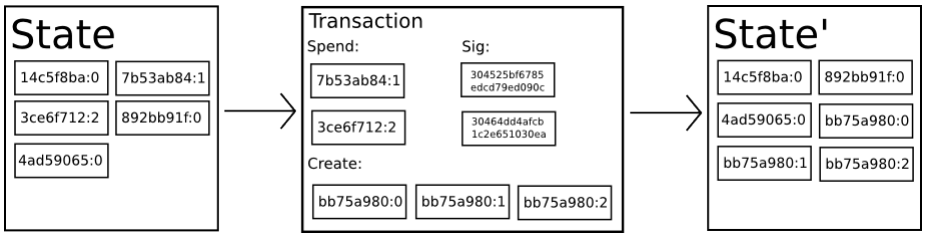


Figure 3.11: Bitcoin as a state transition system[38]

[34] https://github.com/blockstack/blockstack-browser
[35] https://blockstack.org/whitepaper.pdf
[36] https://github.com/blockstack/gaia/tree/master/hub
[37] https://medium.com/swlh/what-is-the-internet-of-values-3f14b5d35a90
[38] https://raw.githubusercontent.com/ethereumbuilders/GitBook/master/en/vitalik-diagrams/statetransition.png

Bitcoin[8] was the first peer-to-peer electronic payment system that's based on cryptographic proof instead of trust, allowing any two entities to transfer value over the internet without the need for any trusted third-party. A transaction in the Bitcoin network involves digitally signing a hash of the previous transaction and the public key of the recipient and adding these to the end of the transaction. Multiple transactions are combined to form a block and each block is linked with the previous block by it's hash and a *nonce* generated by a proof-of-work[36] algorithm. This chain of blocks forms a distributed data structure called the *blockchain*. It can be thought of as a state transition system, where a *state* consists of the ownership status of all existing bitcoins and a *state transition function* that takes a state and a transaction and outputs a new state[37].

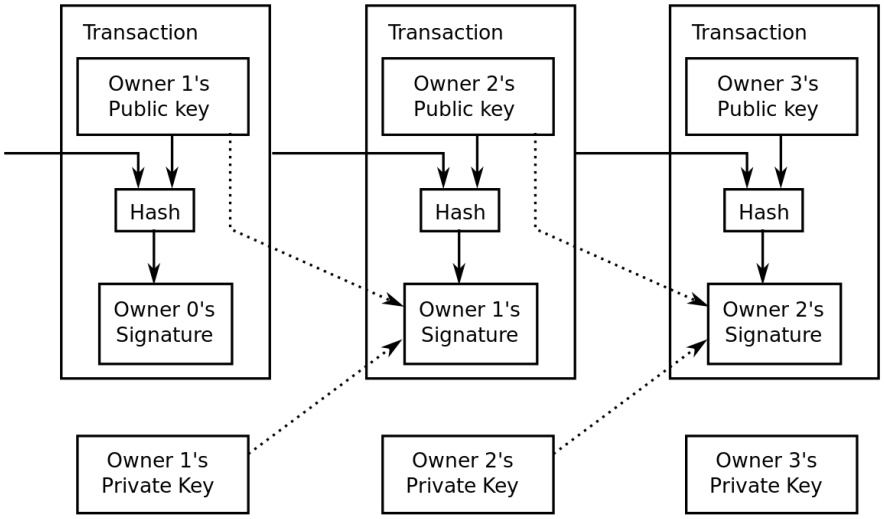


Figure 3.12: Transactions in the Bitcoin Network[39]

Blockchain, the underlying technology of Bitcoin can also be applied to build systems which can represent assets in the physical world. For example, *Secure property titles with owner authority*[40], a document written by Nick Szabo[41] describes how a blockchain-based system can be used to build a land registry. Ethereum[37], an alternative protocol with a built-in Turing-complete programming language enables such systems by allowing anyone to write *smart contracts*[38] where they can create their own arbitrary rules of ownership[42].

Ethereum also enables the creation of *tokens* which allows us to represent something specific in an application, for example, economic value, a dividend, a stake or even a voting right[43]. The two most common tokens standards in

[39] https://cdn-images-1.medium.com/max/1200/1*d9AIfrD_DOLtXfKVnadN6w.png
[40] https://nakamotoinstitute.org/secure-property-titles/
[41] https://en.wikipedia.org/wiki/Nick_Szabo
[42] https://github.com/ethereum/wiki/wiki/White-Paper
[43] https://education.district0x.io/general-topics/understanding-ethereum/the-role-of-tokens/

the Ethereum ecosystem are *ERC-20*[44]: A class of identical tokens, and *ERC-721*[45]: A class of unique tokens. These token standards can be used to tokenize ownership of any arbitrary data or even real world assets thereby enabling an *Internet of Value*[46].

3.7 Computing

Any web application needs computing resources to work. Traditional internet relied on a *client-server* model where server is usually a more powerful, better connected machine and thus provided most of the computing resources. This, however, also made server as a single bottleneck for performance and reliability[39]. As internet connectivity improved and web applications became more complex, the demand for computing resources increased.

Virtualization[47], a concept pioneered by the VM[40] operating system led to the emergence of *cloud computing*. Cloud computing satisfied the growing demand of web applications by providing an on-demand delivery of computing resources over the internet. This greatly defined the current internet architecture where applications relies on cloud services providers who provides computing resources based on different service models (Figure 3.13).

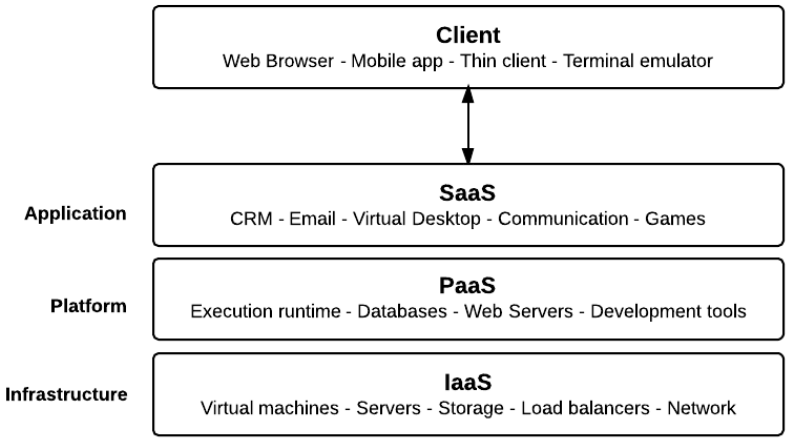


Figure 3.13: Cloud computing service models[48]

Cloud computing, however, poses privacy concerns as it requires entrusting data to information systems managed by third-party cloud service providers who has access to all of user's data at any time and could accidentally or deliberately use it for unauthorized purposes[41]. Because data from multiple entities can be stored on a large cloud servers, it poses security concerns as well. Hackers

[44] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
[45] https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
[46] https://hackernoon.com/a-vision-of-the-internet-of-value-ad187abf5826
[47] http://www.kernelthread.com/publications/virtualization/
[48] https://static.bluepiit.com/blog/wp-content/uploads/sites/2/2015/12/types-of-cloud-computing-models.png

can theoretically gain control of huge amounts of data through a single attack - a process called *hyperjacking*[49].

Blockchain can be used to create a decentralized global market where infrastructure providers, application owners and individual users can come together to trade their unused computing resources. This will also allow us to harness the computing resources of millions of personal computers and workstations connected to the public internet which have hundreds of compute cycles going unused every second[39]. Coupled with end-to-end encryption, this network of connected computers can overcome the security and privacy concerns posed by Cloud computing.

3.7.1 Related Work

Golem Network

Golem[50] aims to create a decentralized supercomputer by creating a global market for computing power. It connects computers in a peer-to-peer network, enabling every participant to share their unused resources. It uses the Ethereum blockchain for accounting, which enables direct payments between providers, application owners and individual users. Finally, it employs an *Application Registry* which is an Ethereum smart contract, to which anyone can publish their application.

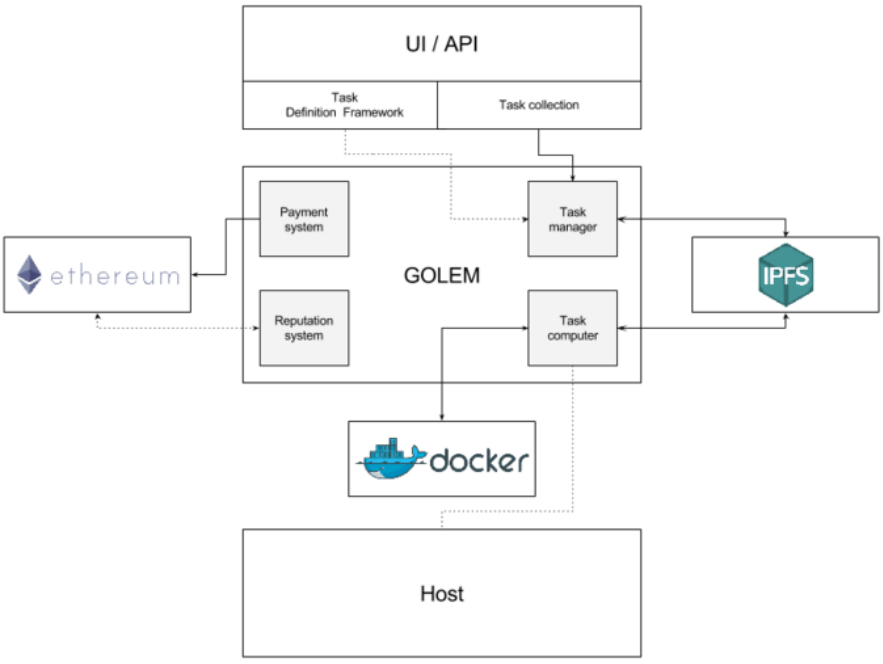


Figure 3.14: Overview of Golem Network Architecture[51]

[49] https://www.securityweek.com/deep-dive-hyperjacking
[50] https://golem.network/crowdfunding/Golemwhitepaper.pdf

3.8 Bandwidth

ISPs act as gateways between end users and the internet (Figure 3.15). They solve the "last mile" problem by connecting users to the high-speed internet and acting as a central hub for communication. However, these central gateways are also central points of failure. Governments can shut them down or ask them to blacklist certain IPs[9].

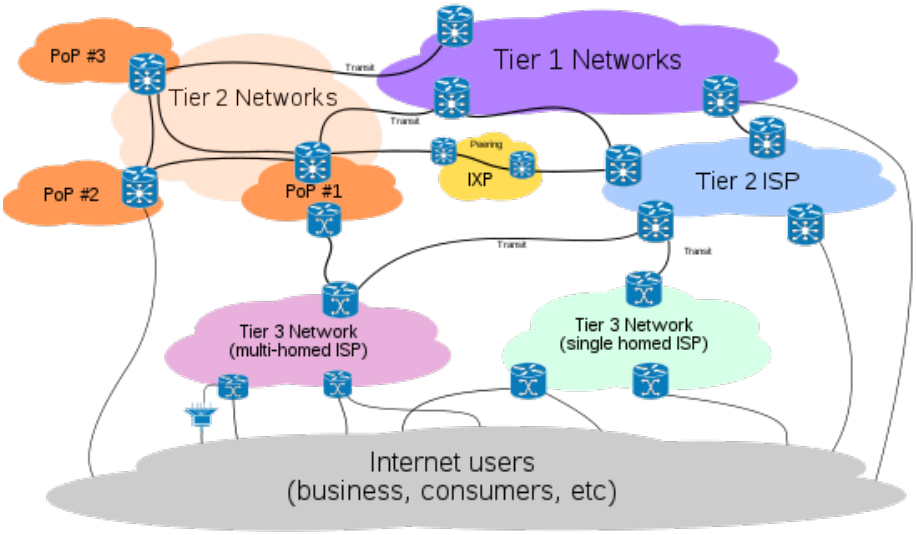


Figure 3.15: The Internet[52]

Mesh networks are the decentralized version of standards internet where users don't need to go through a central gateway. In a mesh network, nodes connect to as many adjacent nodes for efficient routing of data. However, a decentralized consensus system is required to incentivize all participants to share their bandwidth. Blockchain can used to serve as an incentive layer effectively creating a decentralized marketplace for bandwidth. Mark Nadal and Dr. Amber Cazzell describes one such protocol in their paper *A Trustless Decentralized Bandwidth Incentive*[53].

3.8.1 Related Work

NOIA Network

NOIA[54] network is a Distributed peer-to-peer content delivery network, governed by the blockchain where nodes are incentivized for providing storage and unused bandwidth to the network.

[51] Adapted from https://en.bitcoinwiki.org/wiki/Golem_Worldwide_Supercomputer
[52] https://commons.wikimedia.org/wiki/File:Internet_Connectivity_Distribution_%26_Core.svg
[53] https://web.stanford.edu/~nadal/A-Decentralized-Data-Synchronization-Protocol.pdf
[54] https://noia.network/
[55] https://docs.noia.network/noia/segment-routing--srv6-

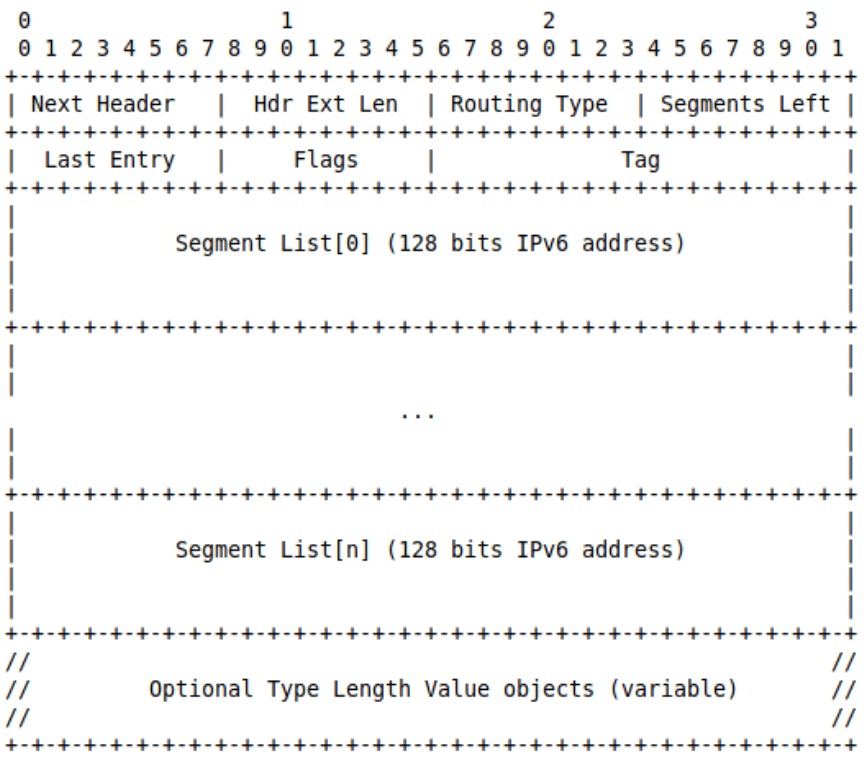


Figure 3.16: IPv6 Segment Routing Header[55]

NOIA enables a *programmable internet*[56] by leveraging IPv6 and Segment Routing (SR). IPv6 increases the address size from 32 bits to 128 bits. This increase in packet size allows NOIA to encode custom information in packet headers thus providing an identification system for nodes connected to the network. These *custom headers* (Figure 3.16) enables SR by leveraging the source routing paradigm. SR is then used to create a Software Defined Network (SDN) where nodes are identified by their segment identifiers (SIDs). SIDs are registered on the NOIA Ledger to create a network topology of SR enabled access points. A Decentralized Internet Transit Exchange (DITEX) is used for trading of access points as a Transit.

Althea Network

Althea[57] Network aims to solve the "last mile" problem by connecting a source of internet connectivity to the end users using a mesh network topology where packets are forwarded using the Babel routing protocol[42]. It creates a decentralized ISP where routers pay each other for bandwidth with cryptocurrencies. Governance in Althea network is done using a *proof-of-stake* blockchain built using the Cosmos[58] protocol.

[56] https://docs.noia.network/noia/programmable-internet
[57] https://althea.net/
[58] https://cosmos.network/
[59] Adapted from: https://althea.net/whitepaper

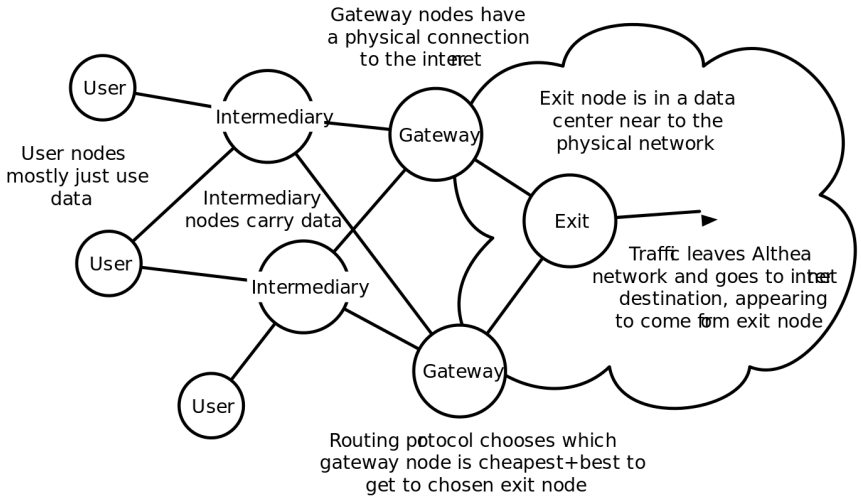


Figure 3.17: The Althea Network[59]

3.9 Application Design

Designing any web application requires understanding of the underlying architecture. This section combines all the application concepts to visualize the underlying architecture which enables decentralized applications with data ownership.

3.9.1 Decentralized Application Architecture

Self-sovereign identity is one of the most important element in decentralized applications with data ownership. The DIDs[60] specifications enables the creation of a DPKI system on top of any blockchain. A user can create a *decentralized identity* by registering their public key with a desired username by creating a blockchain transaction.

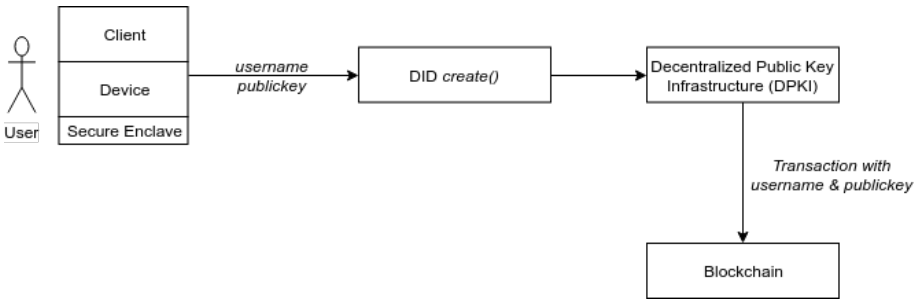


Figure 3.18: Creating a Decentralized Identity

A decentralized identity anchored on a blockchain serves two purposes. First,

[60] https://w3c-ccg.github.io/did-spec/

it gives us a cryptographic keypair which can be used by an application for encryption/decryption of data. Second, it serves as a digital wallet for secure exchange of value across the Internet using cryptocurrencies.

Decentralized applications follow a *serverless* architecture where backend services such as database, storage and authentication are provided as APIs that enable client applications to connect directly to these services. The frontend components such as business logic and application code are deployed inside a hosting environment such that the servers that run the code is completely abstracted.

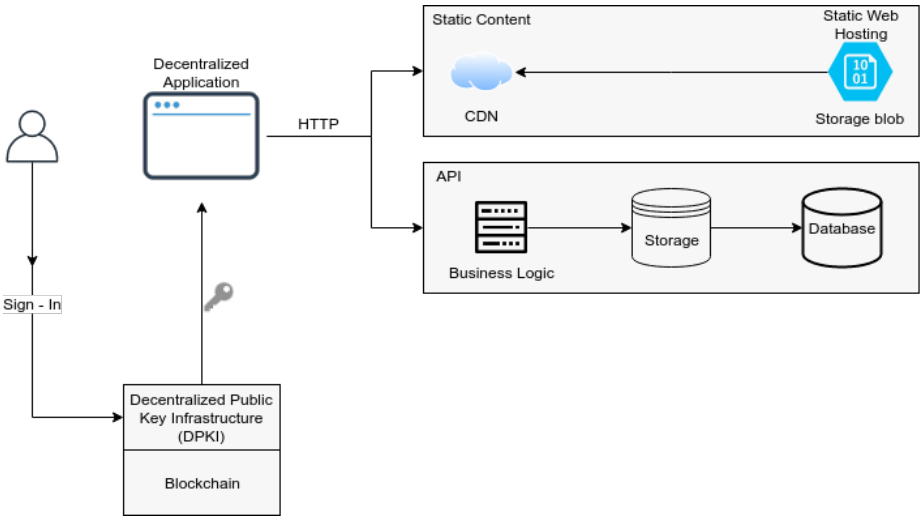


Figure 3.19: Decentralized Application Architecture

Figure 3.19 visualizes the architecture of a decentralized web application. Authentication is handled using a DPKI system. The application connects to various services like storage and database using APIs. The client application encrypts all private data using user's public key before sending it to a decentralized storage network. A database service can be used for easy indexing of data.

3.10 Summary

This chapter outlined the different concepts of a web application and showed how Blockchain enables decentralization for each of the concepts. The key concept which allows us to build decentralized applications with data ownership is *Decentralized Identity*. It enables a Web where we don't have to use username/-passwords anymore and where our data have cryptographic security.

Chapter 4

Building a Decentralized Application

4.1 Introduction

Chapter 3 focused on different application concepts and showed the underlying architecture for building decentralized applications where users are in control of their data. This chapter uses the concepts learned in the previous chapter to build two *proof-of-concept* applications for secure file sharing. This will help us analyze the current *state-of-the-art* protocols for building decentralized applications.

4.2 Problem Context

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.

4.3 Requirements

A decentralized file sharing application should have following desirable properties:

- Client-side encryption.
- Encryption keys are in user's possession.
- Users can choose the data storage location.
- Easy and secure sharing of files with other users of the application.

4.4 A File Sharing Application using Ethereum

This section describes the workings of the application *dShare-ethereum*[43] 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/

4.4.1 Use Case

Todays supply chain spans multiple geographies, but the documents involved in the industry such as delivery certificates are still in physical form. This paperwork prevents manipulations but leads to various delays across the whole chain, thus affecting everyone involved[1].

The above problem can be solved by digitizing all documents, time-stamping them using a trusted Time stamping authority (TSA) and upload them to a cloud service. However, the tools used to accomplish this solution are central services, and, therefore, suffers from data manipulations by a 3rd party.

We, therefore, need a solution that is peer-to-peer (P2P) and decentralized. Recent distributed technologies offers a means of decentralized time stamping and enables peer-to-peer storage systems that are resistant to manipulations by any 3rd party.

dShare-ethereum is built using decentralized technologies such as Bitcoin[8], Ethereum[37] and IPFS[24] such that it's capable of immutable timestamping and secure file sharing in a p2p fashion.

4.4.2 Technologies Used

Ethereum

Ethereum[37] is a blockchain platform for building decentralized applications. It allows the creation of *Smart Contracts* which serves as a backend for the application. Solidity[2] is the primary language for writing smart contracts on Ethereum.

InterPlanetary File System (IPFS)

IPFS is used as a decentralized storage for storing of files and their corresponding decryption key which are encrypted using the uploader's public key.

OriginStamp

OriginStamp[44] 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 4.1 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

[1] https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=XI912347USEN
[2] https://github.com/ethereum/solidity

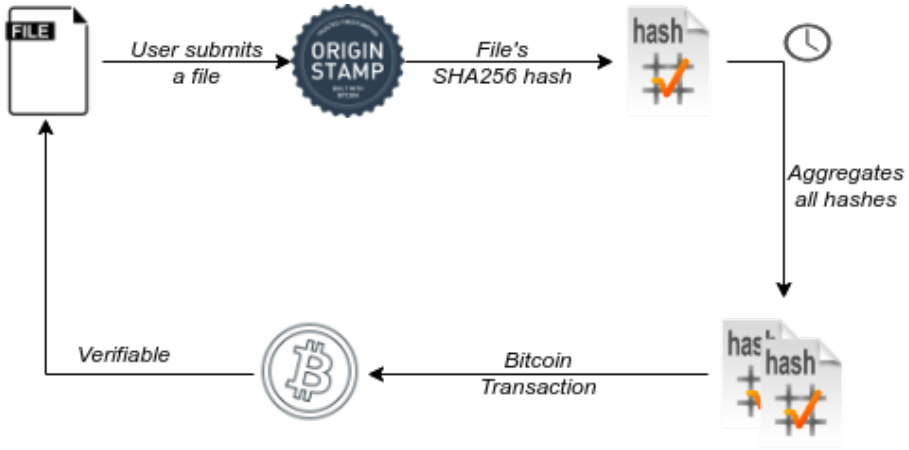


Figure 4.1: Timestamping using OriginStamp

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 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[8].

Firebase

Firebase[3] is used as a database for storing of user's public keys which is used for encryption of a file's key when it's shared with another user.

MetaMask

MetaMask[4] is an Ethereum wallet. It serves as a web3[5] provider allowing any application to interact with the Ethereum blockchain.

4.4.3 Application Architecture

Figure 4.2 visualizes the application architecture. User authentication is handled by MetaMask. For storing of files we use IPFS. Before uploading to the IPFS network, files are encrypted using AES-GCM[6] 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.

[3] https://firebase.google.com/
[4] https://metamask.io/
[5] https://web3js.readthedocs.io/
[6] https://www.aes-gcm.com/

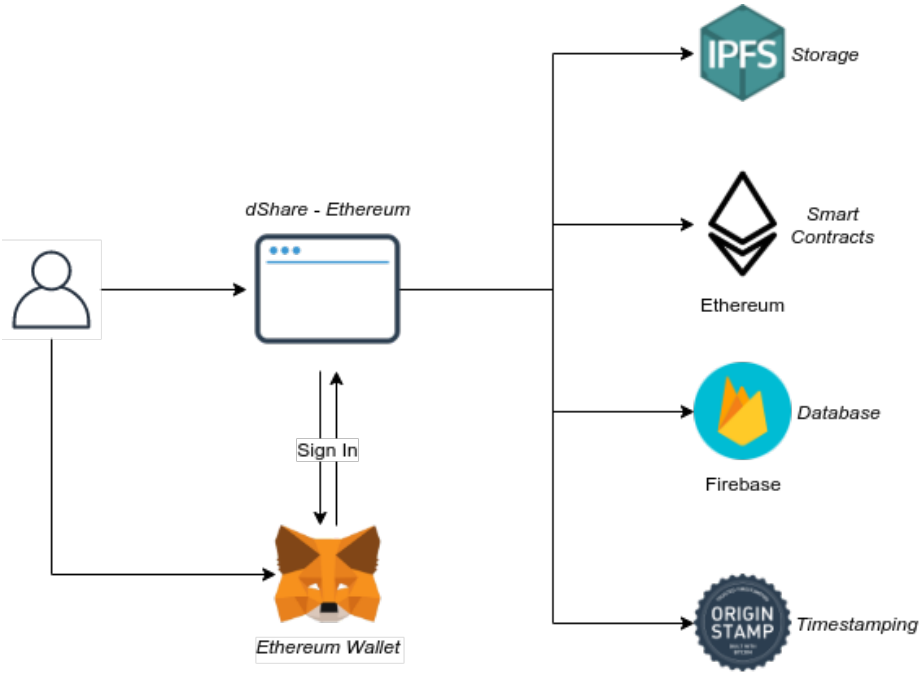


Figure 4.2: *dShare-ethereum* Architecture

The front-end of the application is built using React.js[7], a JavaScript library for building user interfaces. Solidity was used for writing smart contracts and deployed on the Ethereum test network, Rinkeby[8]. Next.js[9] was used for server-side rendering (SSR)[10], and Firebase[11] was used as a database for storing public Ethereum key of the users.

4.4.4 Working

Authentication

When a user logs into the application, she is presented with a MetaMask dialog which asks her to sign a randomly generated value. This signature is used to retrieve the public key for the selected Ethereum address. The public key is then saved to the database and user is logged in.

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 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[12] which defines the cost for a

[7] https://reactjs.org/
[8] https://www.rinkeby.io
[9] https://nextjs.org/
[10] https://nextjs.org/features/server-side-rendering/
[11] https://firebase.google.com/
[12] https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas

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. Code for both the contracts is available at https://github.com/hKedia/dShare/blob/master/ethereum/contracts/FileShare.sol

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.

File Upload

Figure 4.3 visualizes the working of the application when a user uploads a file. Reference code is available at https://github.com/hKedia/dShare/blob/master/pages/files/upload.js

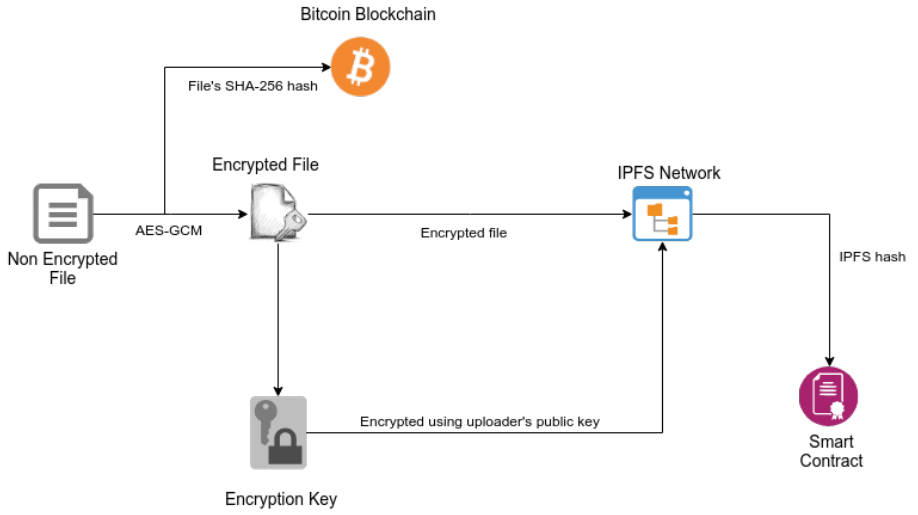


Figure 4.3: File Upload using *dshare-ethereum*

As soon as a user submits a file to be uploaded, it's SHA-256[13] hash is calculated and a timestamp is created by submitting the hash to the bitcoin blockchain using the OriginStamp API[14].

[13] https://www.movable-type.co.uk/scripts/sha256.html
[14] http://doc.originstamp.org/

Next, the file is encrypted using the SubtleCrypto[15] interface with 'AES-GCM'[16] as the encryption 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.

File Sharing

Sharing a file requires the recipient's Ethereum address and uploader's private key. Figure 4.4 visualizes the working of the application when a user shares a file with another user. Reference code is available at https://github.com/hKedia/dShare/blob/master/components/FileSharing.js

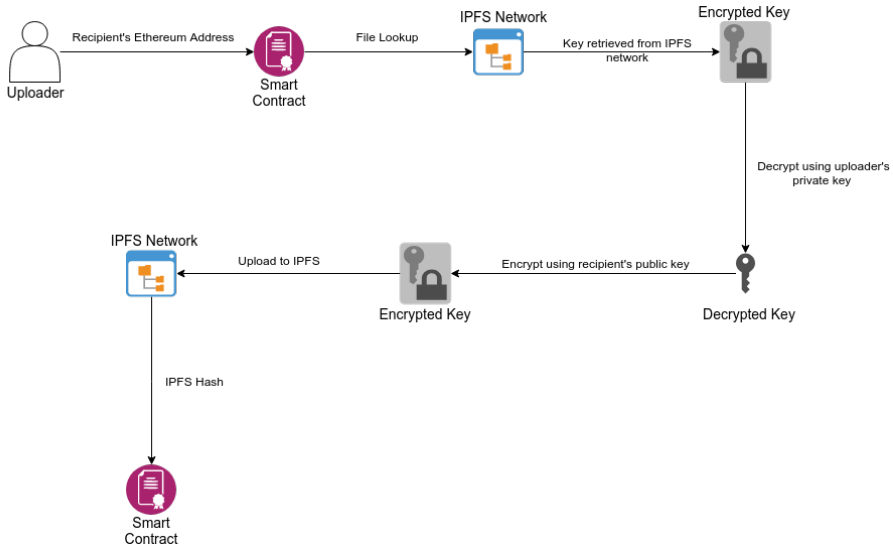


Figure 4.4: File Sharing using *dshare-ethereum*

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.

[15] https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto
[16] https://en.wikipedia.org/wiki/Galois/Counter_Mode

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)[17] 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. Reference code for file download is available at https://github.com/hKedia/dShare/blob/master/components/FileDownload.js

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. The reference code for file archiving is available at https://github.com/hKedia/dShare/blob/master/components/FileDetail.js#L71

4.4.5 Limitations

This application is built using experimental p2p technologies that is changing very rapidly. Below are some of the limitations:

- IPFS nodes treat the stored data as cache, meaning there is no guarantee the data will continue to be available. To overcome this, users should run their own IPFS nodes and *pin*[18] the data that is important.
- MetaMask, the Ethereum wallet used in the application has no function that allows us retrieve the private key of the user within the application. Therefore, whenever a user wants to share or download a file, the private key must be manually provided for decryption.
- Files can be only be shared with recipients who have logged into the application at least once.

4.5 A File Sharing Application using Blockstack

This section describes the workings of the application *dshare-blockstack*[45] built using Blockstack[19], a decentralized computing network and app ecosystem. The latest version of the application is deployed at https://dshare-blockstack.herokuapp.com/.

[17] https://tools.ietf.org/html/rfc7517
[18] https://docs.ipfs.io/guides/concepts/pinning/
[19] https://blockstack.org/

4.5.1 Use Case

dshare-blockstack focus on easy sharing of files between two users. The encryption and decryption is handled by keys generated on user's machine. It leverages the existing cloud infrastructure for storage thereby providing a rich user experience.

4.5.2 Technologies Used

Blockstack

Blockstack[27] is a platform for building highly scalable decentralized applications where users own their data. It implements a DPKI system (see Section 3.5.3) anchored on the Bitcoin blockchain. It also implements a storage system, Gaia (see Section 3.4.3) which leverages the existing cloud infrastructure providing users with private data lockers.

Radiks

"Radiks[20] is a framework for building complex and collaborative decentralized applications, using Blockstack infrastructure under the hood"[21]. It serves as a indexing server for data that is stored in Gaia. *dshare-blockstack* uses the front-end component library, *radiks.js*[22] for interacting with Radiks.

4.5.3 Application Architecture

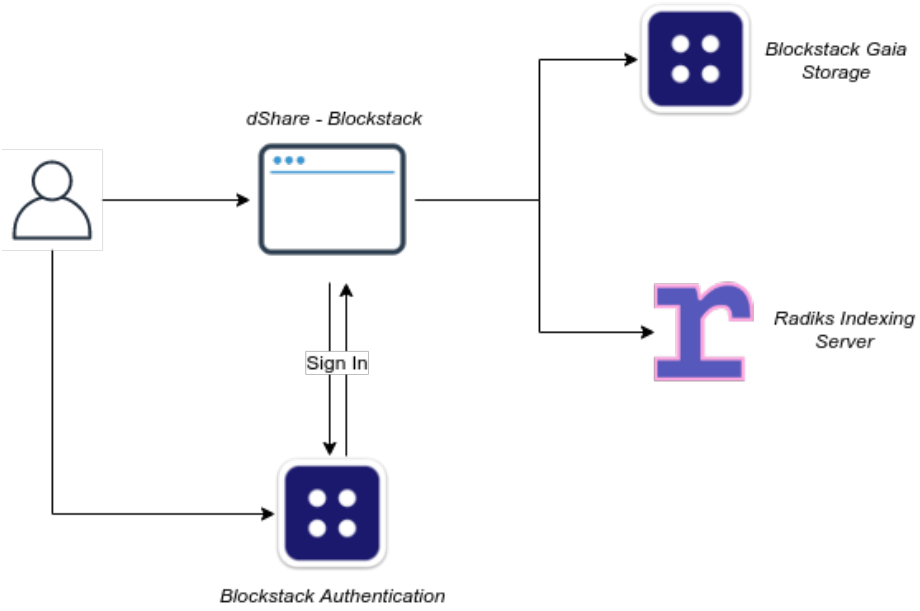


Figure 4.5: *dShare-blockstack* Architecture

[20] https://github.com/blockstack-radiks/radiks-server
[21] https://blog.blockstack.org/introducing-radiks/
[22] https://github.com/blockstack-radiks/radiks

Figure 4.5 visualizes the application architecture. User authentication is handled by the Blockstack browser[23]. Files are uploaded to the user's Gaia Hub, a private data locker in the cloud. Before uploading, files are encrypted using a symmetric key employing AES-GCM as the encryption mechanism. Sharing logic is implemented using Radiks which provides an API for creating models to represent uploaded files.

4.5.4 Working

Authentication

When a user sings into the application, she is redirected to the Blockstack browser. Here she can create a new Blockstack ID or select on of her existing IDs. See Section 3.5.3 for the authentication flow. Once authenticated, user is redirected back to the application and logged in. The public key of the user is saved under */keys* inside her Gaia hub which is later used for encrypting a file's encryption key.

File Upload

Figure 4.6 visualizes the working of the application when a user uploads a file. The reference code for the functionality is available at https://github.com/hKedia/dShare-blockstack/blob/master/pages/files/upload.js#L71.

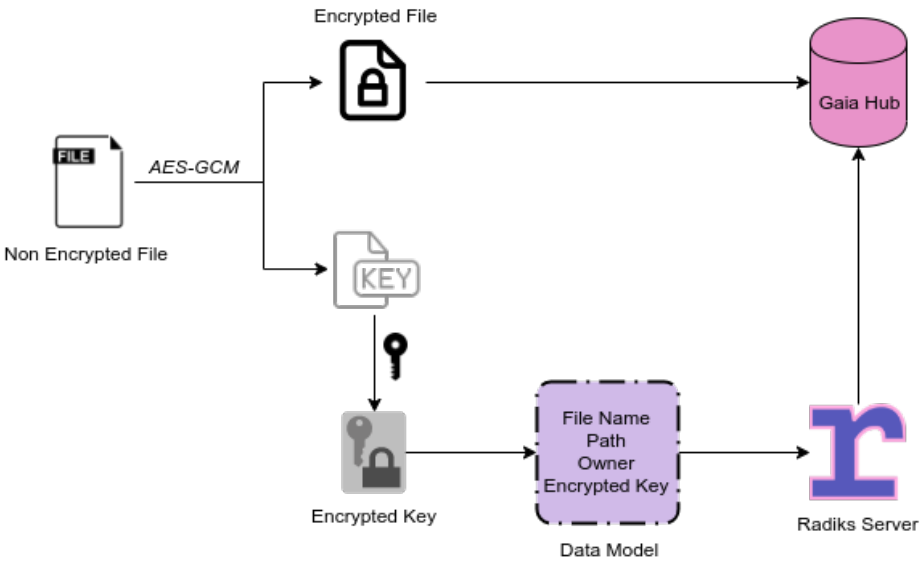


Figure 4.6: File Upload using *dShare-blockstack*

When a user submits a file to be uploaded, it gets encrypted using the SubtleCrypto interface with AES-GCM as the encryption algorithm. The encrypted data is combined with the random salt to generate a `Unit8Array` buffer. A

[23] https://browser.blockstack.org/

unique *identifier* is generated for the file path using the shortid[24] library. The `Uint8Array` buffer is then uploaded to user's Gaia hub under 'files/*identifier*'.

The *key* used to encrypt the file is first converted to JSON Web Token (JWT)[25] format and then is encrypted using the user's public key. A data model is created which includes the filename, path, owner and the encrypted key. This data model is saved to the radiks server which in turn saves the data to Gaia hub.

File Sharing

Sharing a file requires the recipient's Blockstack Id. Figure 4.7 visualizes the working of the application when a user shares a file with another user. The reference code is available at https://github.com/hKedia/dShare-blockstack/blob/master/pages/files/view.js#L93.

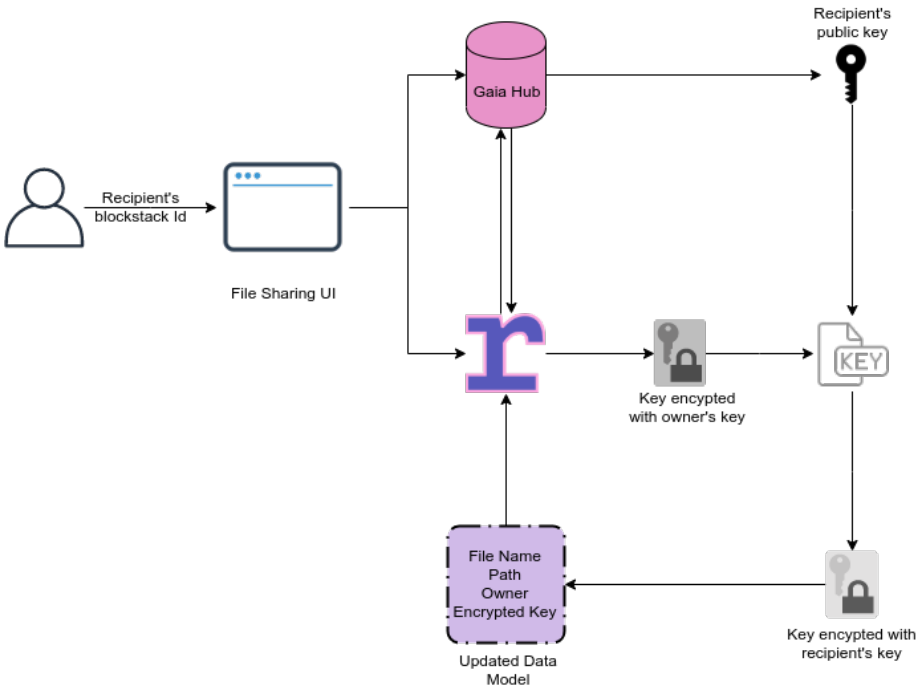


Figure 4.7: File Sharing using *dShare-blockstack*

Firstly, the recipient's public key is retrieved from the Gaia hub located under *'keys/'*. The *key* used to encrypt the file is downloaded and then decrypted using user's private key. The decrypted key is then encrypted using the recipient's public key. The data model associated with the file is updated with the recipient Blockstack ID and the newly encrypted key; and is saved to the Radiks server.

When the user stops sharing a file, the recipient id and the associated encrypted key is removed from the data model associated with the file.

[24] https://github.com/dylang/shortid
[25] https://tools.ietf.org/html/rfc7519

File Download

Downloading a file follows below steps:

- Retrieve the data model associated with the current file.
- Get the file's name, path, owner and the encryption key from the data model.
- Decrypt the encrypted *key* using user's private key.
- Retrieve the encrypted file content and convert to a `Uint8Array` buffer.
- Separate the original file content and the random salt.
- Decrypt the file using the downloaded key and save the file to user's device.

The reference code implementing file download is available at https://github.com/hKedia/dShare-blockstack/blob/master/pages/files/view.js#L40

File Deletion

Deleting a file simply involves destroying the data model associated with the file. Reference code is available at https://github.com/hKedia/dShare-blockstack/blob/master/pages/files/view.js#L83

4.5.5 Limitations

Below are some of the limitations of *dShare-blockstack*:

- To use the application, user must use a Blockstack ID with an associated username.
- The application uses the default Gaia storage which has a 25 mb file size limit.
- Files can only be shared with recipients who have logged into the application at least once.

4.6 Related Work

4.6.1 Timestamping

Trusted timestamping is a way of verifying that specific information existed at a given point in time. Blockchains enables anchoring of data utilizing cryptocurrency transactions where each transaction gets an immutable timestamp.

Proof of Existence

Proof of Existence[26] is a web-based service that implements the concept of immutable timestamping using the Bitcoin blockchain. It notarizes data in the blockchain by submitting the hash of the data in a Bitcoin transaction. Currently, the service require 0.00025BTC[27] for every certification which makes it expensive to timestamp large volumes of data.

Chainpoint

Chainpoint[28] works similarly to OriginStamp (see Section 4.4.2). The service runs on the Tierion[29] Network, providing a scalable protocol for anchoring data in a blockchain and generating blockchain receipts. These receipts are called chainpoint proofs which defines a path of operations that cryptographically links the data to one or more blockchains.

4.6.2 Storage

P2P protocols such as BitTorrent[21], Distributed Hash Tables (DHT), and Git allows us to store information distributed across multiple computers. Coupled with a Blockchain, it enables a decentralized storage market.

Sia

Sia[46] is a decentralized cloud storage system that allows its users to rent storage among peers utilizing storage contracts which are cryptographically secured by saving on a blockchain. This makes the storage contracts tamper-proof and publicly auditable.

To ensure that the storage provider holds a clients data at a given time, they constantly need to submit storage proofs. The network consensus allows automatic verification of storage proofs and enforcement of storage contracts. The availability of data is ensured using redundancy techniques such as erasure codes.

Sia uses a variant of Bitcoin blockchain for storing the contracts, and the user must use Siacoin, an ERC-20[30] token in order to transact on the Sia network.

Storj

Storj[47] works similarly to Sia. It is built on Kademlia[6] DHT, connecting peers who can transact with each other. A transaction can involve negotiation of storage contract, transfer of data, verifying remote data, download data or payments to other nodes. Each peer is capable of doing transactions independently without any human involvement.

Storj uses the Ethereum blockchain for managing its storage contracts. They are stored as versioned data structure describing the relationship between a client and a storage provider. Users must use Storjcoin, an ERC-20 token to perform transactions on the Storj network.

[26] https://poex.io/
[27] https://poex.io/prove
[28] https://chainpoint.org/
[29] https://tierion.com/
[30] https://theethereum.wiki/w/index.php/ERC20_Token_Standard

4.6.3 Blockchain

Bitcoin[8] and Ethereum[37], the two most popular public blockchains are based on *proof-of-work* (PoW)[31] consensus algorithm. PoW blockchains require a large amount of computing resources to achieve consensus. This computing power largely go wasted making PoW blockchains non-scalable and expensive to use. *Proof-of-stake* (PoS)[32] blockchains on the other hand, does not depend on computing resources but rather require a stake in the participating blockchain. This makes PoS blockchains both scalable and inexpensive to use.

Cardano

Cardano[33] is a *proof-of-stake* blockchain employing a peer reviewed academic approach. It's consensus algorithm, *Ouroboros*[48] is provably secure with rigorous security guarantees. Cardano is a multi layered blockchain. `ADA`, the internal currency of Cardano lives on the settlement layer, while control layer will run smart contracts.

Smart contracts in Cardano are written in Plutus[34], a functional smart contract language based on Haskell. It also employs a domain-specific language (DSL), Marlowe[49], for writing financial contracts.

Cardano will also implement an on-chain governance and a treasury system[50] making it a highly sustainable public blockchain.

[31] https://en.bitcoin.it/wiki/Proof_of_work
[32] https://en.bitcoin.it/wiki/Proof_of_Stake
[33] https://www.cardano.org/en/what-is-cardano/
[34] https://iohk.io/research/papers/#KQL88VAR

Chapter 5

Results & Analysis

5.1 Introduction

This chapter compares and analyzes the two *proof-of-concept* applications, *dShare-ethereum* and *dShare-blockstack* for file sharing. We also compare the underlying protocols for decentralized timestamping and decentralized storage. Finally, we compare different blockchains and outline their salient features.

5.2 Decentralized File Sharing Applications

This section compares the functionalities of the two applications and the core protocols used for building them.

5.2.1 Underlying Platform

As can be inferred from the names, 1st application is built upon Ethereum[37], while the 2nd is built upon Blockstack[27]. Analyzing the core protocols that make up the two applications will give us a better understanding in choosing which platform is better suited for building decentralized applications with data ownership. Below subsections highlights the core technical differences between Ethereum and Blockstack.[1]

	Ethereum	**Blockstack**
Architecture	Layer -1 system	Layer - 2 system
Application Storage	On - chain	Off - chain
Computation	On - chain	Off - chain
Programming Model	On - chain programs	Off - chain programs
Smart Contracts	Solidity	Clarity
Scalability	On -chain complexity makes it difficult to scale	Minimum On - chain complexity makes it easy to scale

Table 5.1: Comparing Ethereum and Blockstack

[1] https://forum.blockstack.org/t/what-is-the-difference-between-blockstack-and-ethereum/781

Protocol Architecture

Ethereum is a "layer 1" system. All the complexity and computations is handled at the blockchain layer. Things like scalability and security concerns also need to be addressed at the blockchain layer.

Blockstack is a "layer 2" system. It puts minimal logic at the blockchain layer and handles scalability and security concerns outside the blockchain by reusing existing internet protocols. This makes it blockchain agnostic. The current version of Blockstack runs on top of the Bitcoin blockchain.

Computation and Storage

In Blockstack, all computation and storage happens outside the blockchain, and blockchain is used only as a "shared source of truth" between participating nodes and clients. In Ethereum, on the other hand, all computation and most application storage happens inside blockchain itself.

Programming Model

"Blockstack's programming model is based on running off-chain programs. These programs can be written in any language". For certain use-cases on-chain smart contracts can be written in Clarity[2], a list processing (LISP) language.

"Ethereum's programming model is based on running on-chain smart contracts". These programs are written in languages like Solidity[3], which is quite new and contain multiple design flaws. Nicola et al. compiled a list of vulnerabilities in their paper *A survey of attacks on Ethereum smart contracts*[51].

Scalability of Computations

Blockstack employs the concept of *virtual chain*[13], where nodes only need to reach consensus on the shared "virtual chain" they are interacting with. This allows for a single blockchain to host multiple virtual chains. By contrast, since smart contracts run at the blockchain layer, all Ethereum nodes must process all smart contracts' computations to reach consensus. This makes running smart contracts on Ethereum quite expensive over a period of time.

5.2.2 Authentication

Both *dShare-ethereum* and *dShare-blockstack* require its users to create a key pair which is anchored on a blockchain. In case of *dShare-ethereum*, users create a cryptographic key pair using MetaMask, a crypto wallet for the Ethereum blockchain. The public key is then used to login to the application. The public Ethereum address associated with the keys is a hexadecimal string (eg. `0x931Ec0cFfD0DE326055944f4ac7c524ABe6c49fb`) and therefore is not human-meaningful.

dShare-blockstack, on the other hand, uses the Blockstack browser[4] for key pair generation. The public Blockstack ID[5] associated with a key pair is a

[2] https://docs.blockstack.org/core/smart/overview.html
[3] https://github.com/ethereum/wiki/wiki/Programming-languages-intro
[4] https://browser.blockstack.org/
[5] https://docs.blockstack.org/browser/ids-introduction.html

human-meaningful string (eg. `harshkedia.id`) and therefore provides a much richer user experience.

5.2.3 File Uploading

dShare-ethereum uses the IPFS[24] network (see Section 3.4.2) and *dShare-blockstack* uses the Gaia storage system (see Section 3.4.3) for storing of files uploaded by the users. Table 5.2[6] lists the key differences between IPFS and Gaia in terms of features.

Features	**IPFS**	**Gaia**
User controls where data is hosted	No	Yes
Data can be viewed in a web browser	Yes	Yes
Data is read/write	No	Yes
Data can be deleted	No	Yes
Data can be listed	No	Yes
Data lookups have predictable performance	No	Yes
Write permission can be delegated	No	Yes
Listing permission can be delegated	No	Yes
Supports multiple backends natively	No	Yes
Data is globally addressable	Yes	Yes
Needs a cryptocurrency to work	No	No
Data is content-addresses	Yes	No

Table 5.2: Comparing IPFS and Gaia

Every upload using *dShare-ethereum* involves a smart contract transaction. The user needs to have some ether to initiate an upload. Further, she needs to wait for the transaction to confirm before she can see the changes successfully reflected in the application. *dShare-blockstack*, on the other hand, does not need any crypto tokens as there are no smart contract transaction involved.

We compared the upload times between two applications using Chrome DevTools[7]. Keeping everything same, *dShare-blockstack* is 8 times faster than *dShare-ethereum* for a given file upload. Test data for the comparison can be found at https://github.com/hKedia/thesis/tree/master/tests.

5.2.4 File Sharing

Sharing of files using *dShare-ethereum* involves manually retrieving one's own private key from MetaMask, getting the public Ethereum address of the recipient and finally submitting a smart contract transaction. Here the data stored in the smart contract acts as the single source of truth.

Sharing in *dShare-blockstack* involves submitting a form along with the recipient's Blockstack ID. Blockstack provides functions for automatic retrieval of

[6] https://docs.blockstack.org/storage/overview.html#gaia-versus-other-storage-systems
[7] https://developers.google.com/web/tools/chrome-devtools/evaluate-performance

user's private key. It then creates a data model using radiks indexing server. All data associated with a shared file is saved in Gaia.

Comparing the share performance of the two applications using Chrome DevTools reveals that *dShare-blockstack* is 6 times faster than *dShare-ethereum* for file sharing. Test data for the comparison can be found at https://github.com/hKedia/thesis/tree/master/tests.

5.3 Decentralized Timestamping

	Scalability	Anchoring Blockchain	Timestamping Accuracy
Proof Of Existence	Not Scalable	Bitcoin	Per Block
OriginStamp	Scalable	Bitcoin	Per Time Period
Chainpoint	Scalable	Bitcoin, Ethereum	Per Time Period

Table 5.3: Comparing Decentralized Timestamping

Table 5.3 gives a comparison overview for decentralized timestamping. *Proof Of Existence* creates a Bitcoin transaction for each hash submitted by the user. Moreover, each certification costs 0.00025 BTC. These limitations make it impractical and expensive for timestamping a large volume of data.

On the other hand, both *OriginStamp* and *Chainpoint*, instead of creating a transaction for each submitted hash, concatenates the submitted hashes over a period of time and creates a single transaction with the aggregated hash. Thus providing a scalable protocol for timestamping which can handle large volume of data.

5.4 Decentralized Storage

	Sia	Storj	IPFS
Encryption	Client Side	Client Side	No Encryption by default
Storage Contracts	Yes	Yes	No
Ease of Access	Tokens Required	Tokens Required	No Tokens Required
File Sharing	No	No	Yes (Insecure)
Configurability	Low	Low	High

Table 5.4: Comparing Decentralized Storage

Table 5.4 gives a comparison overview for decentralized storage. Both *Sia* and *Storj* provide an encrypted data storage; however, current implementations do not allow for file sharing. Moreover, both require storage contracts and platform specific crypto tokens for access to the network.

IPFS, on the other hand, does not provide encryption by default. Files on the IPFS network are accessed by their hashes; thus anyone with the file's hash can access the file. There are no storage contracts involved for storing files on the IPFS network and it does not require any crypto tokens for accessing the network.

5.5 Blockchains

Table 5.5 gives a comparison overview for some public blockchains. Bitcoin[8] is the first generation blockchain which uses *proof-of-work* as its consensus algorithm. Its designed as a single layer system with a non Turing-complete scripting language[8] for transactions. It's unit of account, BTC, serves as a store of value and is used to pay for transactions. Bitcoin's Ledger uses a UTXO (unspent transaction output) based transaction model.

	Bitcoin	**Ethereum**	**Cardano**
Consensus Algorithm	Proof Of Work	Proof Of Work	Proof Of Stake
Architecture	1 - layered	1 - layered	2 - layered
Smart Contracts	No	Yes (Solidity, Vyper)	Yes (Plutus, Marlowe)
Block Time	∼10 min	∼15 sec	∼20 sec
Native Token	Bitcoin or BTC	Ether or ETH	ADA
Ledger	UTXO based	Accounts based	UTXO & Accounts based
Treasury	No	No	Yes
Governance	No	No	On - Chain

Table 5.5: Comparing Public Blockchains

Ethereum[37] is a second generation blockchain which like Bitcoin uses *proof-of-work* as its consensus algorithm. It also has a single layered architecture with a Turing-complete scripting language[9]. Ethereum serves as a platform for building decentralized applications. It's Ledger uses a Accounts based transaction model with ETH as the native currency.

Cardano, a third generation blockchain, uses *proof-of-stake* as it's consensus algorithm. The consensus algorithm, *Ouroboros*[48], is provably secure with strong security guarantees. Cardano's platform is being constructed in layers. The *settlement layer* serves as a unit of account with ADA as its native token while the *computation layer* will handle smart contract functionality. Like Ethereum, Cardano also serves as a decentralized applications platform, however, unlike Ethereum, Cardano's smart contracting language, Plutus, is based on Haskell, a functional programming language which complements the immutable nature of the blockchain. Cardano's Ledger[52] uses both UTXO and Account based transaction model.

[8] https://en.bitcoin.it/wiki/Script
[9] https://github.com/ethereum/solidity

Chapter 6

Discussion

Chapter 5 analyzed and compared the two *proof-of-concept* applications and their underlying technology. This chapter focuses on interpreting the results and showing the state-of-the-art in decentralized technologies for building applications with data ownership.

6.1 On Decentralized Application Platform

Analyzing the underlying platforms on which the two applications are built, its clear that Blockstack, due to its layered architecture is much more flexible and robust than Ethereum. Blockstack's separation of computations and storage from the blockchain layer makes it easier to scale and upgrade. It's integrated DPKI system offers its users with a self-sovereign identity. Building applications around the concept of a decentralized identity enable us to build a web where everyone owns their data.

Ethereum, too is moving towards a layered architecture with their Eth 2.0 upgrade. The upgrade is spread across three phases. The complete specifications for the upgrade can be found at `https://github.com/ethereum/eth2.0-specs`.

6.2 On Decentralized Storage

Applications with data ownership involves encrypted data where storage systems are incentivized to store data. IPFS (see Section 3.4.2) offers a content-addressed storage systems using p2p technologies like DHT, Git and BitTorrent. Gaia (see Section 3.4.3) offers mutable data storage using existing cloud infrastructure. It's clear that both IPFS and Gaia enables a decentralized storage system serving different use cases. IPFS is more suitable for applications where nature of data is immutable and where decentralization precedes performance. Gaia, on the hand, is more suitable for applications where is data is mutable and data availability is crucial.

6.3 On Blockchains

Blockchain is the key enabler for building decentralized applications. It serves as the layer for value transfer in the Internet protocol stack. As inferred from results, blockchains based on *proof-of-stake* are much more scalable and computationally inexpensive to use than blockchains based on *proof-of-work*. Since, Blockchain enables a peer-to-peer information economy, it important that its built with strong security guarantees. Also, as public blockchains don't have a single entity controlling the system, its important for a blockchain to have a treasury and a governance system.

Chapter 7

Conclusion & Outlook

With the rapidly changing technologies that enables the Information Economy, its important that individuals own their data. A Self-Sovereign identity is the most important concept that Blockchain technology enables. A Decentralized Public Key Infrastructure (DPKI) built on top of a Blockchain abstracts all the complexity associated with key management and enables everyone to own their digital identity. Building applications around the concept of user owned identities enable data ownership by giving control of data to the users who create them.

This thesis explored the different concepts which make up a web application namely Data, Identity, Value, Computation and Bandwidth; and described how each concept can be re-imagined using Blockchain and other peer-to-peer protocols.

We analyzed the state-of-the-art by building proof-of-concept applications using two of the most popular decentralized applications platform, Ethereum and Blockstack.

Ethereum, being a "layer 1" system in its current state, handles all the complexity and the computations at the blockchain layer. This makes scaling and upgrading Ethereum a relatively complex task. Moreover, the smart contract language of Ethereum, Solidity, is relatively new and contains several design flaws and is prone to various attacks.

Blockstack, being a "layer 2" system puts minimal logic at the blockchain layer and handles the complexity and computations off-chain. The design principles of Blockstack makes it relatively easy to scale and upgrade. Smart contracts on Blockstack are Turing incomplete and written in Clarity, a list processing (LISP) language.

It's common notion that decentralized applications require smart contracts, however, we demonstrate in this thesis that it's possible to build such applications without a smart contract. In fact, smart contracts are required only in certain use cases, for example, an Escrow application.

Appendix A

Acknowledgements

I would like to thank my advisor Prof. Dr. Marcel Waldvogel for his guidance and constructive discussions throughout my research. It helped me understand the principles of Distributed systems and how those principles apply to Blockchains. I am thankful to my supervisor Robert Mller who always found my ideas interesting and helped me limit my thesis scope.

I am grateful to all my friends who helped me understand Blockchain from the perspectives of Economics, Philosophy, Psychology and Mathematics. I am also grateful to authors like Yuval Noah Harari for 'Sapiens: A Brief History of Humankind' and Raoul Martinez for 'Creating Freedom: Power, Control and the Fight for Our Future' which helped me understand Human Evolution and Human Behavior. Without this holistic understanding, this thesis wouldn't have been possible.

I would also like to thank Jun.-Prof. Dr. Stephan Streuber for giving me the opportunity to work on Virtual Reality projects involving collective behavior while working on my thesis. This helped me explore my other interests in Computer Science.

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] W. Galuba and S. Girdzijauskas, "Distributed hash table," *Encyclopedia of database systems*, pp. 903–904, 2009.

[6] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[7] I. Baumgart and S. Mies, "S/kademlia: A practicable approach towards secure key-based routing," in *2007 International Conference on Parallel and Distributed Systems*. IEEE, 2007, pp. 1–8.

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

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

[10] 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.

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

[12] 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.

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

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

[15] T. J. Berners-Lee, "Information management: A proposal," Tech. Rep., 1989.

[16] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[17] G. Greenwald, *No place to hide: Edward Snowden, the NSA, and the US surveillance state*. Macmillan, 2014.

[18] N. S. Good and A. Krekelberg, "Usability and privacy: a study of kazaa p2p file-sharing," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2003, pp. 137–144.

[19] R. Ku and R. Shih, "The creative destruction of copyright: Napster and the new economics of digital technology," *U. Chi. L. Rev.*, vol. 69, p. 263, 2002.

[20] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, no. 1, pp. 50–57, 2002.

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

[22] L. Wang and J. Kangasharju, "Measuring large-scale distributed systems: case of bittorrent mainline dht," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[23] W. B. Rayward, "Visions of xanadu: Paul otlet (1868–1944) and hypertext," *Journal of the American Society for information science*, vol. 45, no. 4, pp. 235–250, 1994.

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

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

[26] J. Benet and N. Greco, "Filecoin: A decentralized storage network," *Protoc. Labs*, 2018.

[27] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16)*, 2016, pp. 181–194.

[28] D. Hardt, "The oauth 2.0 authorization framework," 2012.

[29] D. Recordon and D. Reed, "Openid 2.0: a platform for user-centric identity management," in *Proceedings of the second ACM workshop on Digital identity management*. ACM, 2006, pp. 11–16.

[30] F. A. Tycksen Jr and C. W. Jennings, "Digital certificate," Feb. 13 2001, uS Patent 6,189,097.

[31] C. Adams and S. Lloyd, *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley Professional, 2003.

[32] A. Abdul-Rahman, "The pgp trust model," in *EDI-Forum: the Journal of Electronic Commerce*, vol. 10, no. 3, 1997, pp. 27–31.

[33] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention." *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.

[34] K. Dooley, *Designing Large Scale Lans: Help for Network Designers*. " O'Reilly Media, Inc.", 2001.

[35] S. M. Smith and D. Khovratovich, "Identity system essentials," *Evemyrn, Mar*, vol. 29, p. 16, 2016.

[36] A. Back *et al.*, "Hashcash-a denial of service counter-measure," 2002.

[37] 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.

[38] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[39] J. Crowcroft, T. Moreton, I. Pratt, and A. Twigg, "Peer-to-peer systems and the grid," *draft, University of Cambridge Computer Laboratory*, 2003.

[40] R. J. Creasy, "The origin of the vm/370 time-sharing system," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[41] M. D. Ryan, "Cloud computing privacy concerns on our doorstep," *Communications of the ACM*, vol. 54, no. 1, pp. 36–38, 2011.

[42] J. Chroboczek, "The babel routing protocol," 2011.

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

[44] 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.

[45] H. Kedia, "hKedia/dShare-blockstack: First release of dShare built with Blockstack," Aug. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3359854

[46] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *Nebulous Inc*, 2014.

[47] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.

[48] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[49] P. L. Seijas and S. Thompson, "Marlowe: Financial contracts on blockchain," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2018, pp. 356–375.

[50] B. Zhang, R. Oliynykov, and H. Balogun, "A treasury system for cryptocurrencies: Enabling better collaborative intelligence," in *The Network and Distributed System Security Symposium 2019*, 2019.

[51] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts." *IACR Cryptology ePrint Archive*, vol. 2016, p. 1007, 2016.

[52] J. Zahnentferner, "Chimeric ledgers: Translating and unifying utxo-based and account-based cryptocurrencies." *IACR Cryptology ePrint Archive*, vol. 2018, p. 262, 2018.