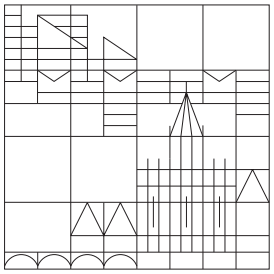
University of Konstanz

Department of Computer and Information Science



Master Thesis

Application Design for a Decentralized Web

in fulfillment of the requirements to achieve the degree of

Master of Science (M.Sc.)

Harsh Kedia

Matriculation Number :: 01/752437

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

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

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

Any dedications or other fancy stuff???

Abstract

Abstract

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.

Explaining contents of each chapter.

Explores the emerging protocols which enable a decentralize web.

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 Cloud Computing

2.5 IPv6

2.6 Segment Routing

2.7 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

Application 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].

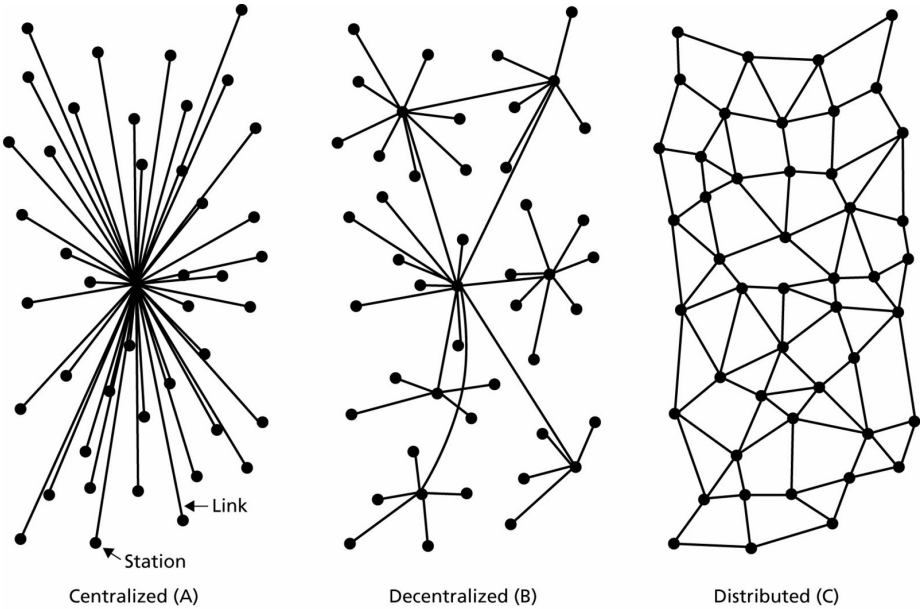


Figure 3.1: The three way of modeling web applications

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 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.

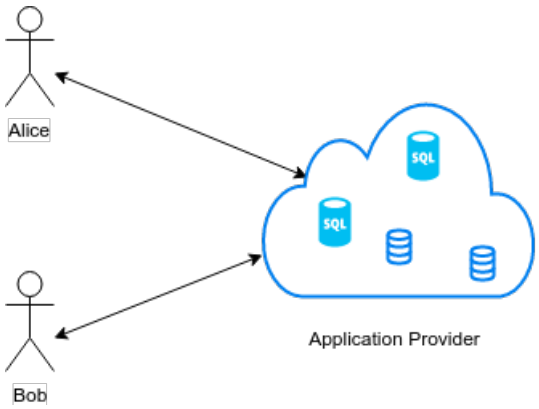


Figure 3.2: How traditional web applications work

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.

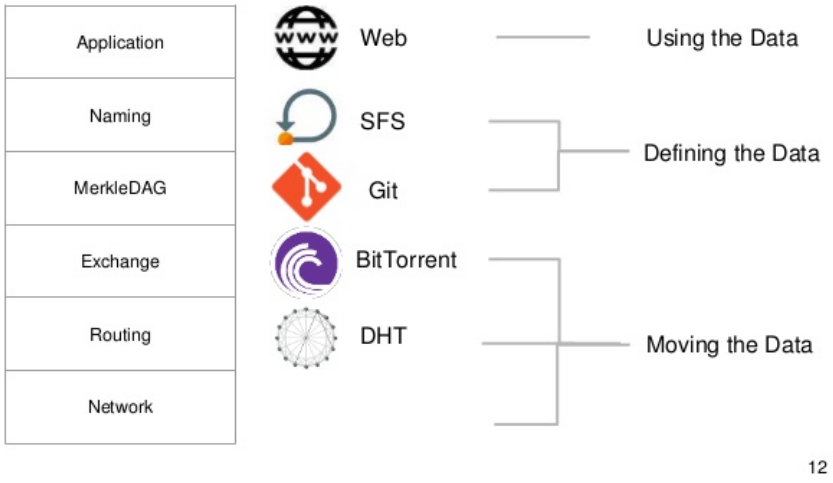


Figure 3.3: IPFS Stack

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.

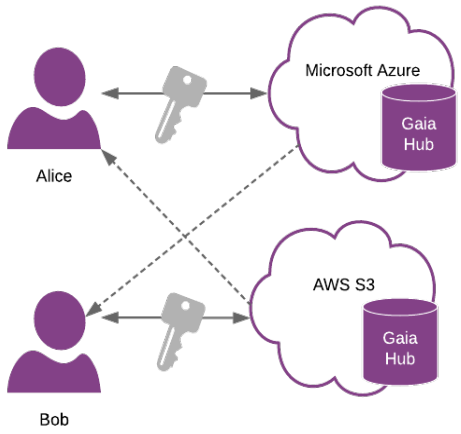


Figure 3.4: How users interact with Gaia storage[16]

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.

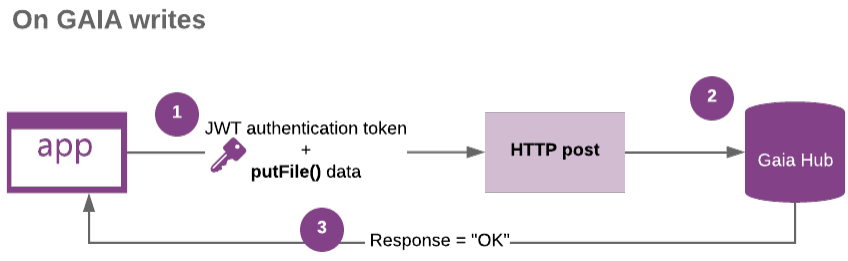


Figure 3.5: The token ensures that a write is authorized[17]

[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.

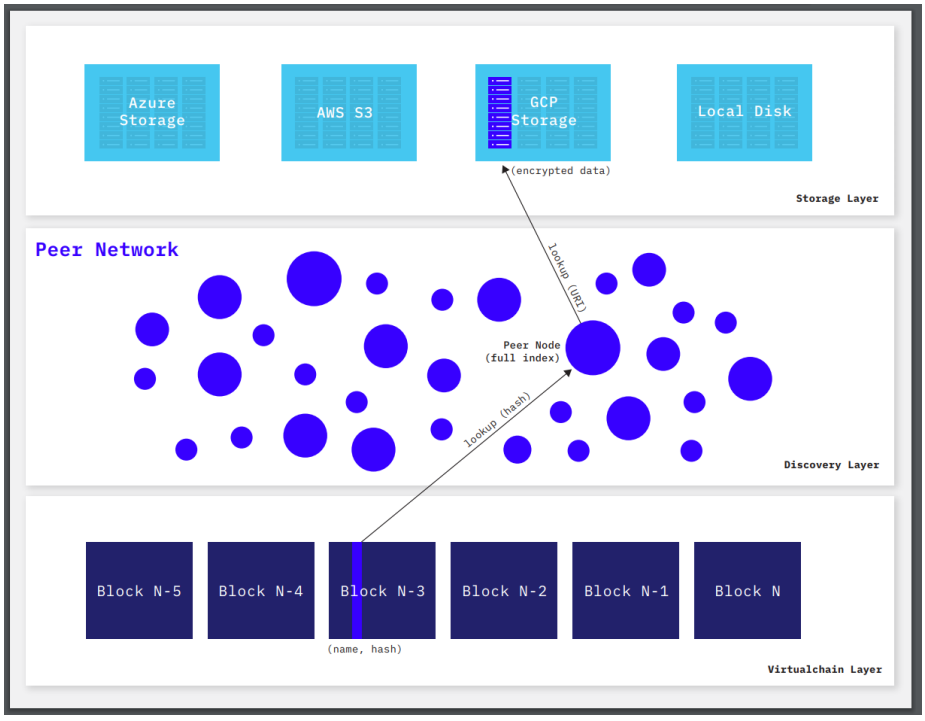


Figure 3.6: Overview of Gaia and steps for looking up data.[18]

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)

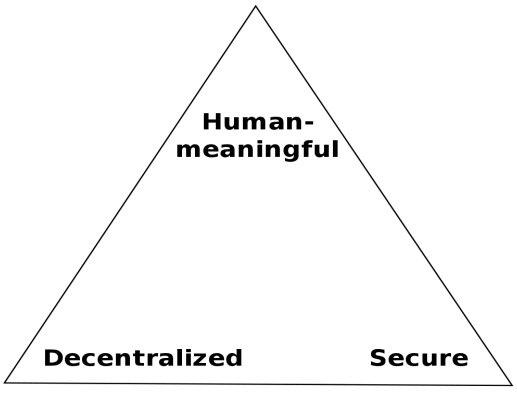


Figure 3.7: Zooko's Triangle[22]

[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.

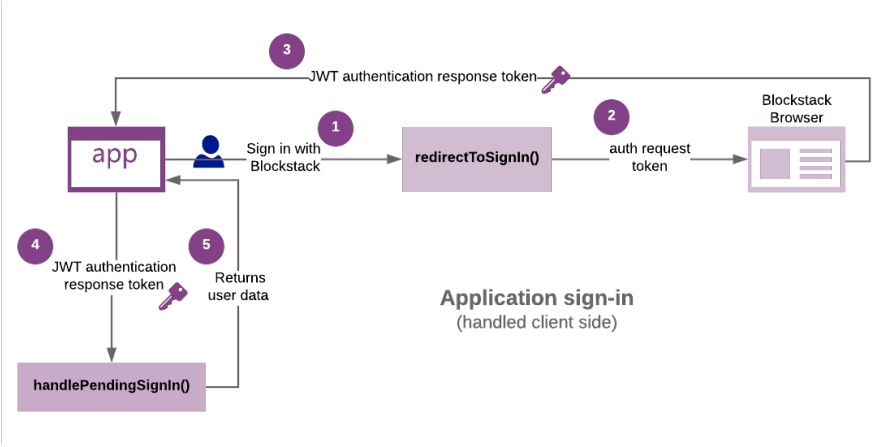


Figure 3.10: Blockstack Authentication Flow[31]

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

3.7 Computing

3.8 Bandwidth

[34] https://github.com/blockstack/blockstack-browser
[35] https://blockstack.org/whitepaper.pdf
[36] https://github.com/blockstack/gaia/tree/master/hub

Chapter 4

Example Application 1

4.1 Introduction

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

This chapter describes the workings of the application *dShare*[36] 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/

4.2 Technologies Used

4.2.1 Ethereum

Ethereum[37] is a blockchain platform for building decentralized applications. It allows the creation of *Smart Contracts*. Solidity[1] is the primary language for writing smart contracts on Ethereum.

4.2.2 InterPlanetary File System (IPFS)

IPFS[24] is a peer-to-peer file transfer protocol which enables a shared file system between all its connected peers. It achieves this by combining previous peer-to-peer systems such as Distributed Hash Tables (DHT), BitTorrent[21], and Git[25]. The data in the IPFS network are modeled as a Merkle DAG[2] thus providing a throughput storage system with content-addressed hyperlinks.

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

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

4.2.3 OriginStamp

OriginStamp[38] 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.

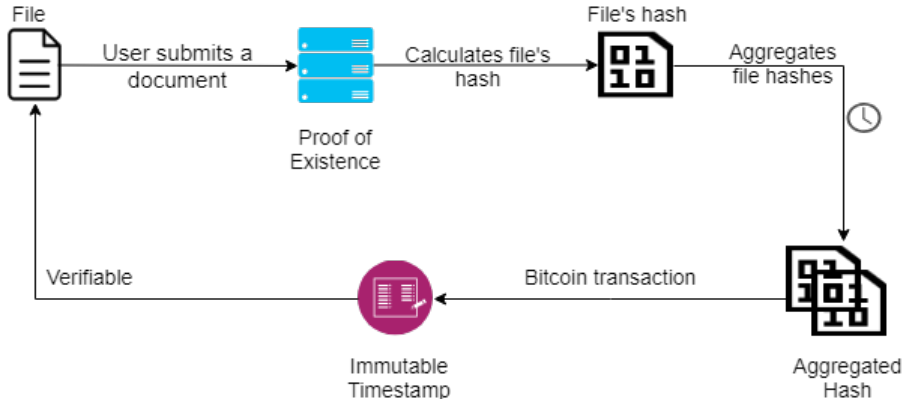


Figure 4.1: Timestamping using OriginStamp

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 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].

4.3 Implementation

For storing of files we used IPFS. Before uploading to the IPFS network, files are encrypted using AES-GCM[3] encryption mechanism. Sharing of encryption keys is facilitated using smart contracts built on Ethereum; thus files can be shared by anyone with an Ethereum address. Finally, OriginStamp is used for immutable timestamping.

The front-end of the application is built using React.js[4], a JavaScript library for building user interfaces. Solidity was used for writing smart contracts and deployed on the Ethereum test network, Rinkeby[5]. Next.js[6] was used for server-

[3] https://www.aes-gcm.com/
[4] https://reactjs.org/
[5] https://www.rinkeby.io
[6] https://nextjs.org/

side rendering (SSR)[7], and Firebase[8] was used as a database for storing public Ethereum key of the users.

4.4 Working

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

4.4.1 Smart Contract

The smart contract serves as the bridge between the front-end of the application and the Ethereum Blockchain. Data is read from and written to the blockchain 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[9] which defines the cost for a function execution in Ether. Reading from the blockchain does not require any fees.

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

FileFactory

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

File

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

4.4.2 File Upload

Figure 4.2 visualizes the working of the application when a user uploads a file.

As soon as a user submits a file to be uploaded, it's SHA-256[10] hash is calculated and a timestamp is created by submitting the hash to the bitcoin blockchain using the OriginStamp API[11].

Next, the file is encrypted using the SubtleCrypto[12] interface with 'AES-GCM'[13] as the encrypting algorithm. The encrypted data is then combined with the random salt to generate a `Uint8Array` buffer ready to be uploaded to the IPFS network.

[7] https://nextjs.org/features/server-side-rendering/
[8] https://firebase.google.com/
[9] https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas
[10] https://www.movable-type.co.uk/scripts/sha256.html
[11] http://doc.originstamp.org/
[12] https://developer.mozilla.org/en-US/docs/Web/API/SubtleCrypto
[13] https://en.wikipedia.org/wiki/Galois/Counter_Mode

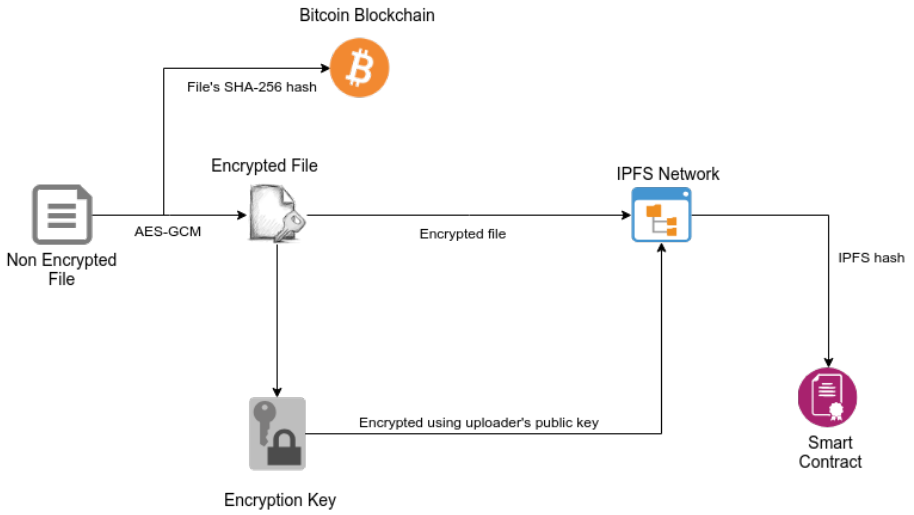


Figure 4.2: File Upload using Ethereum dApp

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.

4.4.3 File Sharing

Sharing a file requires the recipient's Ethereum address and uploader's private key. Figure 4.3 visualizes the working of the application when a user shares a file with another user.

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.

4.4.4 File Download

Downloading a file requires the user's Ethereum private key. Depending on whether the file is uploaded or shared one, corresponding function from the `File` contract is called to retrieve the file's details. The key is then decrypted using user's private key and is converted to a valid JSON web key (jwk)[14] format. The encrypted file data is then converted to a file buffer, and the original file

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

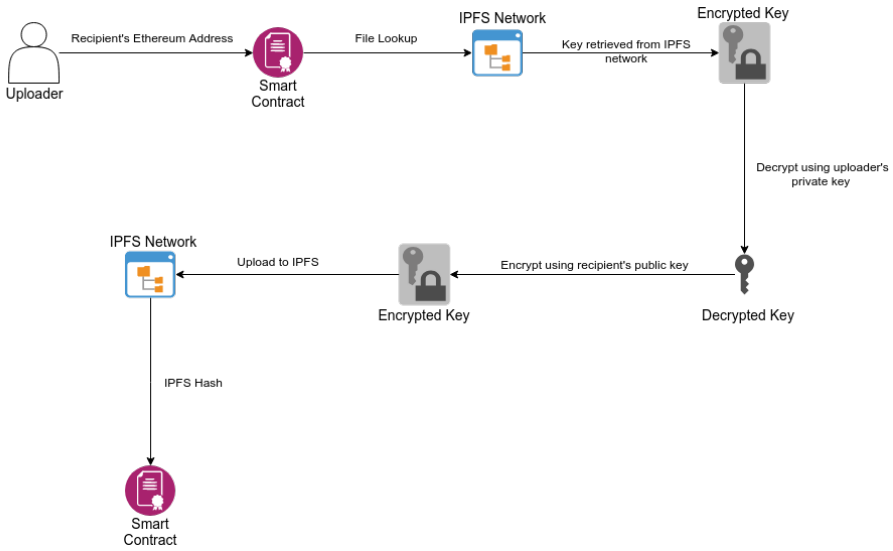


Figure 4.3: File Sharing using Ethereum dApp

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.

4.4.5 File Archiving

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

Chapter 5

Example Application 2

5.1 Introduction

5.2 Technologies Used

5.3 Implementation

5.4 Working

Chapter 6

Results

Chapter 7

Discussion

Chapter 8

Conclusion

Appendix A

Acknowledgements

Bibliography

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

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

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

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

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

[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] 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.