A Preliminary Study of Distributed Storage, Sharing and Computing with IPFS and Blockchain

by

Soumya Pal

(Under the Direction of WenZhan Song)

Abstract

In the current world, data is generated in many ways and forms. Blockchain has been used for an incentive mechanism for sharing of data as incentives might make users willing to share their data. It is also quick and anonymous, thus protecting the identity. IPFS, a decentralized storage, is used to store the data for a web3 framework however, IPFS stores the data without encrypting it. Blockchain has been leveraged to add an encryption mechanism and so, the files stored on IPFS are encrypted thus adding a layer of security. Access control has also been implemented for the files. A system was designed which combines blockchain and IPFS to encrypt the data stored on IPFS and to facilitate quick and anonymous data sharing while having access control for the files. The system is also combined with Golem network to execute python codes on a decentralized network.

Index words: Blockchain, Ethereum, Smart Contracts, IPFS, Data sharing, Data trading

A Preliminary Study of Distributed Storage, Sharing and Computing with IPFS and Blockchain

by

Soumya Pal

B.Tech., KIIT University, India, 2019

A Thesis Submitted to the Graduate Faculty of the

University of Georgia in Partial Fulfillment of the Requirements for the Degree

Master of Science

Athens, Georgia

2022

A Preliminary Study of Distributed Storage, Sharing and Computing with IPFS and Blockchain

by

Soumya Pal

Major Professor: Wenzhen Song

Committee: Kyle Johnsen

Jin Ye

Electronic Version Approved:

Ron Walcott

Vice Provost for Graduate Education and Dean of the Graduate School

The University of Georgia

May 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data is the core of most applications and websites and just being on the internet generates data based on the user and usage. Data can be in many forms such as medical data, personal data, business data, sensor data and more. However, people and industries are not willing to share their data for privacy concerns and due to the lack of compensation for their data. Thus, a mechanism that allows people to share their data without revealing their identity along with getting incentives for it might make them more likely to share their data. This is where blockchain comes in which helps protect the identity of the data owners and buyers along with an incentive mechanism. This data sharing will allow those who need the data for research, to train or test machine learning algorithms or for any other uses will be able to get it by paying for it. Blockchain is a potential solution to this as blockchain protects the privacy of the user along with quick transaction speed. Sharing of data needs to be done while protecting the privacy of both the data owner and data requester along with compensating the data owner for sharing their data. The data being sold can be stored on a decentralized database such as IPFS. IPFS however, does not encrypt the data stored on it. Thus, an encryption mechanism on top of it will go a long way in making the file sharing more secure. This has been achieved with the help of blockchain which also provides a way to sell data without revealing any user identifiable information. Using blockchain, an access control mechanism has also been added such that, once a file is bought by an user, they can use it at a later time without buying it again. Thus, this paper investigates a way to combine blockchain with IPFS in a way that allows the

files to be encrypted before storing it on IPFS and access control has also been implemented for the files. Blockchain by default allows selling/buying without revealing the users identity as only the wallet address is required to make transactions.

1.1 Blockchain

Blockchain, as the name suggests is a chain of blocks. Each block contains a block header, a hash, and some transactions. The block header contains the block number and hash of previous block along with other necessary information. A hash is the result of a complex mathematical function which always yield a different value. The hash of a block is the hash of the whole block except the hash itself. Each transaction contains a timestamp, transaction hash and other information as required. Once a transaction is recorded on the blockchain it is practically very difficult to modify it. That is because making even a small change will change the hash of each block and thus changing the hash of every subsequent block which is computationally very expensive thus making blockchain secure.

Blockchain being secure by nature has great potential in the field of IoT as it can help make the network secure. Research has been carried out to investigate them and to create or propose such frameworks. Somy et al., 2019 worked on an ownership preserving AI marketplace using Blockchain. In this model, data owners (DO) own data that they wish to monetize, model Owners (MO) are those who can implement sophisticated machine learning and AI models but lack the data to train and test these models and Cloud Owners (CO) who are the cloud service providers and want to sell their storage and GPU computing resources for training the data models. Model owners are looking to monetize their models and thus do not want to lose ownership of their models during the training process. The system proposed by Somy et al., 2019 has two distinct phases for training so that both data and model privacy is maintained. In the first data distribution phase, the data off DO is divided into parts and sent to a set of CO so that no single CO has all the data and this sending of data is recorded on the blockchain. The second phase is the collaborative training phase where an MO develops a model based on the sample data shared by the DO and tries to obtain permission to train on the full dataset. The system then uses federated learning to

train the model of the MO on the data as the data is distributed across several COs. However, in federated learning every CO would get access to the final trained model once training is complete. To avoid this the MO used homomorphic encryption to encrypt the model and the training takes place in rounds. The MO shares the running version of the encrypted model with all COs which trains the model with their subset of the data and returns the result to the MO. Since MOs get access to partially trained models at the end of every round, they can potentially learn characteristics of data stored by other COs. To avoid this, during each round, a random set of COs holding a subset of the dataset is utilized for training. This way while all data subsets are eventually utilized for training over several rounds, it becomes difficult for MOs to decipher the data characteristics of individual COs. In this way integrity of data ownership is preserved. This system focuses on network logistics and the monetization part has not been studied by them. The data owner would need to be online while sharing his data in case of a request as the data is not stored on an centralized server or distributed storage. Blockchain technology is currently a hype in many sectors, including the energy sector. (Richter et al., 2018) worked on combining blockchain with the local electricity market (LEM). A LEM is a local community market that allows households to trade electricity generated locally over a local platform. LEMs integrated with blockchain allows the creation of a decentralized market without an intermediary. (Dambrot, 2018) presented their work where genetic data is stored on blockchain along with other features on it. ReGene also combines blockchain and genomics, where in an a software backup/restore model is used on the blockchain genome data. The purpose of the software is to have a method method for resurrecting a prior genomic structure to express the corresponding previous phenotype. ReGene's thus combines secure, permanent and unchangeable genomic data with the ability to reverse expression errors and thus restore to a former error-free phenotype state.

1.2 Blockchain, Internet of Things and Edge computing

An Internet of Things system was proposed by (Roman & Ordieres-Mere, 2019) which collects, sends, stores, and publishes relevant data using a Raspberry Pi as the smart sensor. Distributed Ledger Technolo-

gies (DLT) such as BigchainDB is used to store data in a blockchain-like and IOTA to publish a statistic data summary respectively, along with storing the data streams in a SQL database. The collection, storage, and publication of the data is free and almost instantaneous. For their use case, (Roman & Ordieres-Mere, 2019) used SenseHat sensors to ensure proper management of frozen food. The SenseHat sensor allows the collection of several measurements such as; temperature, humidity, pressure, magnetic field, accelerations and orientation. Environmental data about the condition of the storage was transmitted through TCP/IP protocols, including different channels as GPRS, 4G, 5G, to a remote server. MQTT protocol was used for the communication between the raspberry pi and the server. A statistic summary of the collected data was elaborated, and published periodically to Distributed Technology where interested parties can have instant, direct, and reliable access.

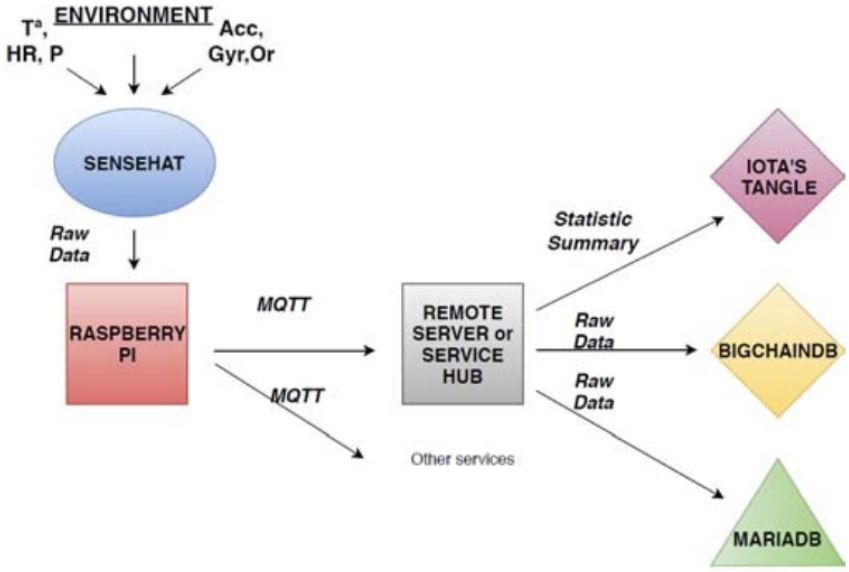

Figure 1.1: Architecture of their design

The statistical summary was stored on IOTA which is the distributed ledger, where as the stream data was stored on MariDB and BigChainDB (which is a distributed storage). Statistical summary was uploaded to IOTA every 15 minutes.

An architecture for Access Control of Electronic Health Records using Hybrid Blockchain-Edge technology was proposed by (Guo et al., 2019). The hybrid architecture proposed facilitates access control

of EHR data by combining blockchain and edge node. In the architecture, a blockchain-based controller was used to manage identity and access control policies which also served as a tamper-proof log of access events and off-chain edge nodes were used to store the EHR data and apply policies specified in the Abbreviated Language For Authorization (ALFA) to enforce attribute-based access control for EHR data along with the blockchain-based access control logs. The entities which are a part of the architecture as stated by (Guo et al., 2019) are:

- Patient: A patient is a user who owns their EHR data. A patient can specify the access policies for the EHR data they owns.
- Healthcare provider: A healthcare provider (e.g., doctor and nurse) is a user who requires access to EHR data owned by patients. A healthcare provider needs to seek access authorizations from patients to access the patients data.
- Smart sensor/imaging equipment: Smart sensor is a device which collects EHR data from patients and sends it to the edge node. Imaging equipment such as X- ray, CT scan, MRI, and ultrasound, generate EHR data from patients.
- EHR data: EHR data is the information owned by patients, and can be accessed by authorized healthcare providers.
- Edge node: An edge node is a computing and storage device which stores EHR data and imposes attribute-based access control policies.
- Blockchain: Blockchain is used as the controller of the architecture to manage access control policies and to maintain a tamper-proof access log.

Smart sensors and imaging equipment collect generated EHR data from the patient and uploads them to the edge node. The edge node then applies ABAC policies to enforce access permissions for the EHR data and returns once, a self-destructing url to the patient with the address of the EHR on the edge node. The same patient then needs to register with the Hyperledger Composer Fabric blockchain and define the

Access Control List (ACL) policy to decide on access permissions for the healthcare providers. After this, Doctors/nurses can send an access request through the smart contract for accessing their EHR data, where it will check the identity information against ACL policy. If the condition is satisfied, the smart contract will return the corresponding url address to locate the edge node where the EHR data is stored. Finally, Doctor or nurses gets access to EHR data if they satisfy the requirement of ABAC policies enforced on the EHR data.

(Desai et al., 2019) developed a blockchain based auction system which leverages both private and public blockchains. Sensitive bids were opened on a private blockchain where only the auctioneer or the parties that are running the auction, knew of the bids and the results were announced on a public blockchain for everyone to see. Any party caught cheating can be monetarily punished using public smart contracts. The architecture consists of an auctioneer, participants of the auction and voters who vote if a contract breach is suspected and also verify if the information posted by the auctioneer on the public blockchain is accurate. The private blockchain is hosted by the auctioneer who also publishes the result to the public blockchain. Thus, the auctioneer acts as a mediator between the public and private blockchains. Auction participants can also join in as voters. Three smart contracts are used which are the auction contract, the PublicDeclare contract and the CongressFactory contract. The public blockchain hosts the two smart contracts the PublicDeclare cthe CongressFactory contract. The former is used to declare the highest and second highest bidders along with their respective bids of the auction and the latter is used in case of a breach if any occurs. The PublicDeclare contract is used to declare the result to the public blockchain.

Chapter 2

Background and Related works

In this chapter we will take a look into blockchain, other related components and techniques used.

2.1 Web 3.0

2.1.1 Evolution of the web

The internet or web has undergone a lot of changes over the years, and its applications today are very different from its early days. The evolution of the web is often partitioned into three separate stages: web1.0, web2.0, and web3.0. As stated by Dabit Dabit, 2021, web1.0 was the first iteration of the web where the participants were mostly consumers of content while the creators were developers who build websites that contained information to serve mainly in the form of text or image. Web1.0 lasted approximately from 1991 to 2004.

Web1.0 involved sites serving static content instead of dynamic HTML. A static file system was used to server data and content rather than from databases, and the sites did not have much to interact with on them. Web1.0 can be considered as the read-only web.

The web in the current form as experienced by most users, is what is known as web2.0 or just web2 (Dabit, 2021). Web2 can be thought of as the interactive and social web. In web2 it's not required to be

a developer to participate in the creation process as there are applications that allow users to become a creator quite easily. For example videos, images or even a thought one has can be shared with the world. Other users have the ability to view it along with commenting on it and sharing with other people. Web2 is simple, and because of its simplicity it helps more people around the world to become creators. The web in its current form is great in many ways, but it has areas of improvement. Most companies, instead of going for some sustainable model of growth that can be sustained in a somewhat organic manner, often have to decide between two paths: advertisements or selling personal data. Many web2 companies like Google, Facebook, Twitter among others, who have more data can recommend targeted personalized ads for their users. This results in more clicks and thus more ad revenue for the company. The exploitation and centralization of user data is core of how the web in present day and that's how it is engineered to function.

Web2 applications repeatedly experience data breaches (Dabit, 2021). In web2, one does not have any control over their data or how it is stored. All of the data that users share knowingly or unknowingly is then owned and controlled by the companies in charge of these platforms.

Web3.0 tries to solve these issues by providing a new architecture for the web (Dabit, 2021). There are a few fundamental differences between web2 and web3, with decentralization being at its core.

Web3 enhances the internet with a few other added characteristics. Web3 is:

- Verifiable
- Trustless
- Self-governing
- Permissionless
- Distributed and robust
- integrated with native built-in payments

In web3, developers don't usually build and deploy applications that are served from a single server or that store their data in a single database, which are usually hosted on and managed by a single cloud provider (Dabit, 2021). Instead, web3 applications either run on blockchains, decentralized networks of many peer to peer nodes (servers), or a combination of the two that forms a cryptoeconomic protocol. Decentralized storage systems are used for storing the data. These apps are often referred to as dapps (decentralized apps). To create a stable and secure decentralized network, network participants (developers) are given incentives and they compete to provide the highest quality services to users who use the service.

Cryptocurrency plays a big role in many of the web3 protocols (Dabit, 2021). It provides a financial incentive (tokens) to users who are willing to participate in creating, governing, contributing to, or improving one of the projects. These protocols may offer a variety of different services like computing, storage, bandwidth, identity, hosting, and other web services commonly provided by cloud providers in the past. Users can earn money by participating in the protocol in various ways, at both technical and non-technical levels.

Consumers of the service usually pay to use the protocol () similarly to how they would pay a cloud provider like AWS today. Except in web3, the money goes directly to the network participants. In this, like in many forms of decentralization, often unnecessary and inefficient intermediaries are removed..

Many web infrastructure protocols such as Filecoin, Livepeer, Arweave, and The Graph issue utility tokens which govern how the protocol functions. These tokens also reward participants at many levels of the network. Even native blockchain protocols like Ethereum operates in this manner.

Web3 however has it's limitations as well (Richards, 2021). Some current limitations are:

- Scalability – transactions are slower on web3 because they're decentralized. Changes to state, like a payment, need to be processed by a miner and propagated throughout the network.
- UX – interacting with web3 applications can require extra steps, software, and education. This can be a hurdle to adoption.
- Accessibility – the lack of integration in modern web browsers makes web3 less accessible to most users.

- Cost – most successful dApps put very small portions of their code on the blockchain as it's expensive.

Identity in web3 is different from how it works in web2. Most of the time in web3 apps, identities are tied to the wallet address of the user interacting with the application thus only the wallet address is shared. Unlike web2 authentication methods which use OAuth or email + password that almost always require users to use sensitive and/or personal information, wallet addresses are completely anonymous unless the user ties their own identity to it publicly. The users identity is also seamlessly transferred across apps if they use the same wallet across multiple dApps, thus helping them build up their reputation over time.

2.1.2 Web2 vs Web3

The following table provides side by side differences between web2 and web3 (Richards, 2021).

Table 2.1: Web2 vs Web3

Web 2.0	Web 3.0
Company owner can censor any account or content(such as twitter can censor tweets)	In Web3 content cannot be censored because control is decentralized
Payment service may decide to not allow payments for certain types of work	Web3 payment apps require no personal data and can't prevent payments
Servers for gig-economy apps could go down and affect worker income	Web3 servers are made to not go down as they use decentralized networks (eg Ethereum) of 1000s of computers as their back-end

2.2 Blockchain

Blockchain made its first appearance with bitcoin (Nakamoto, 2008) where its main use was as a ledger to keep track of transactions. Trying to intergrate blockchain technologies into many fields is an ongoing work since then. A blockchain in simple terms, is a chain of blocks. A blockchain network consists of nodes called peers and each peer has a copy of the blockchain. This adds an extra layer of security. Along with that, blockchain protects the privacy of the user. After a blockchain transaction is successful it is made available to the public for anyone to see. However, transactions do not contain any identifiable information about

Table 2.2: Centralized vs Decentralized

Centralized	Decentralized
Low network diameter (all participants are connected to a central authority) and information propagates quickly, as propagation is handled by a central authority with huge computational resources.	The furthest participants on the network might potentially be many edges away from each other. Information broadcast from one side of the network might take a long time to reach the other side.
Usually higher performance (higher throughput, fewer total computational resources expended) and easier to implement.	Usually lower performance (lower throughput, more total computational resources expended) and more complex to implement.
In the event of conflicting data, resolution is clear and easy: the central authority has the power to decide.	A protocol (mostly complex) is needed for dispute resolution, if peers make conflicting claims about the state of data which participants are meant to be synchronized on.
Single point of failure: malicious actors may be able to take down the network by targeting the central authority.	No single point of failure: network can still function even if a considerable proportion of participants are attacked/taken out.
Coordination among network participants is much easier, and is handled by a central authority. Central authority can compel network participants to adopt upgrades, protocol updates, etc., with very little friction.	Coordination is often difficult, as no single agent has the final say in network-level decisions, protocol upgrades and more. In the worst case, network is prone to fracturing when there are disagreements about protocol changes.
Central authority can censor data, potentially cutting off parts of the network from interacting with the rest of the network.	Censorship is much harder, as information has many ways to propagate across the network.
Participation in the network is controlled by the central authority.	Anyone can participate in the network; there are no "gatekeepers." Ideally, the cost of participation is very low.

the parties that have made the transaction. Rather it just shows the wallet addresses of the involved parties. A hash is used to identify each block in a blockchain for which the SHA256 cryptographic hash is used. Each block also has a reference to it's previous block by the hash of the previous block in its header, thus creating a chain of blocks as seen in (Buterin, 2013).

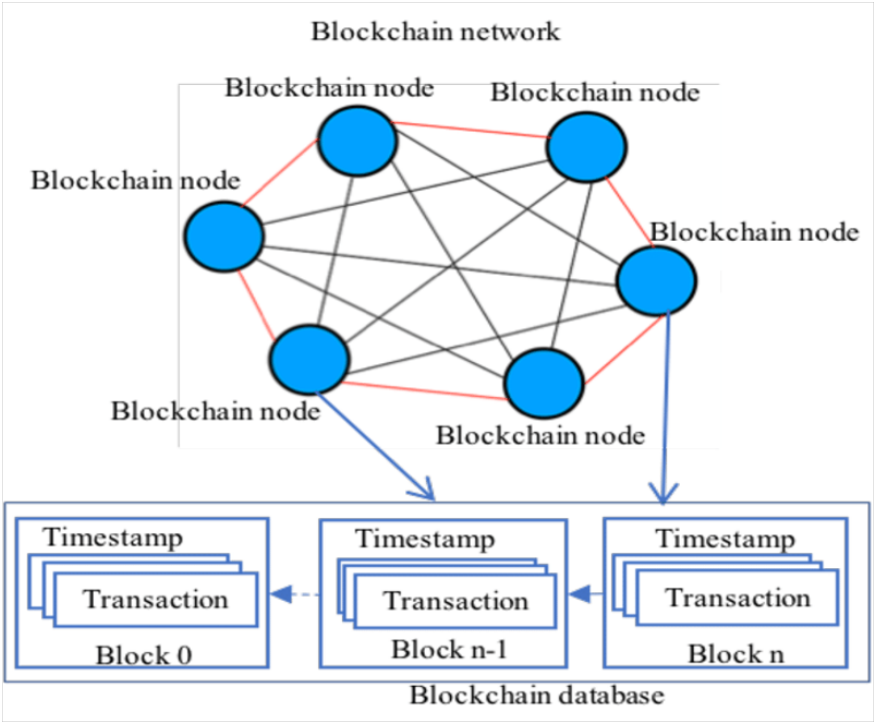

Figure 2.1: Blockchain Network (Lantz & Cawrey, 2020)

Table 2.3: Block Structure

Size of block	Field	Description
4 Bytes	Block Size	The size of the block after this field
80 bytes	Block Header	Meta data about the block
1-9 Bytes	Transaction Counter	How many transactions follow
Variable	Transactions	Transactions in this block

2.2.1 Block header

The block header consists of 3 metadata sets:

- Reference to the previous block by it's hash

- Difficulty, timestamp and nonce which are related to the mining competition
- A Merkle tree root data structure is used to efficiently summarize the transactions in the block

The digital fingerprint of a block,as stated by Lantz and Cawrey, 2020, is its cryptographic hash, which is also the primary identifier, generated by hashing the block header twice using the SHA256 algorithm. The 32-byte hash obtained is termed as the block hash but is more actually the hash of the block header, as only the block header is used to generate it. For example, 000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce2 is the block hash of the first bitcoin block generated. The block hash identifies a block uniquely and unambiguously and can be independently obtained from any node by simply hashing the block header.

Table 2.4: Block Header

Size	Field	Description
4 bytes	Version	Version number to track software/protocol updates
32 bytes	Hash of the previous block	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle root	A hash of the root of this block's transaction merkle tree
4 Bytes	Timestamp	Creation time of this block
4 bytes	Difficulty	The proof-of-work algorithm difficulty for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

2.2.2 Merkle Tree

A merkle tree is used to store the summary of all transactions in a blockchain (Lantz & Cawrey, 2020). A merkle tree or binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large datasets. In computer science, the term "tree" is used to describe a branching data structure, but these trees are usually displayed upside down with the "root" at the top and the "leaves" at the bottom of a diagram, as seen in figure 2.2. Merkle trees are binary trees which contain cryptographic hashes. Merkle trees are used to summarize all the transactions of a block. It produces a digital fingerprint of the entire set of transactions along with providing a very efficient process to verify whether a transaction is to be included in a block. A Merkle tree is constructed by recursively hashing pairs of nodes until only

one hash is obtained which is known as the root, or merkle root. The cryptographic hash algorithm used is mainly SHA256 applied twice, otherwise known as double-SHA256.

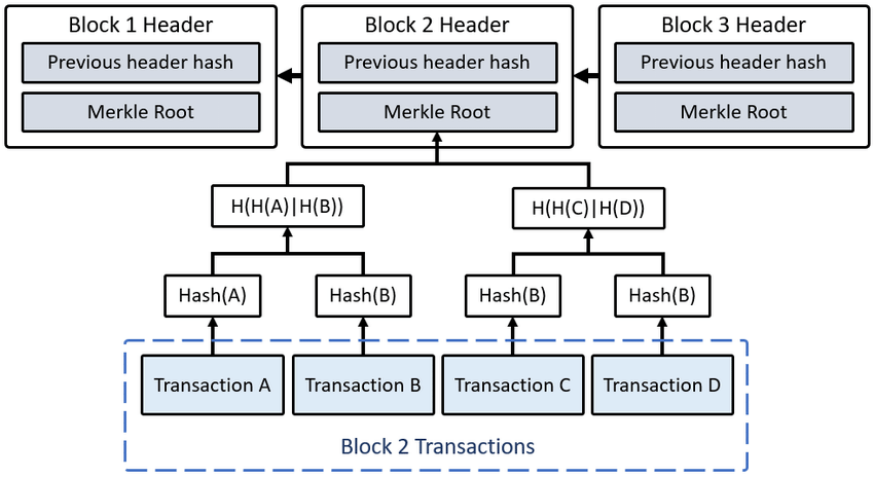

Figure 2.2: Merkle Tree of a Blockchain (Lantz & Cawrey, 2020)

2.2.3 Mining and Consensus

In the context of blockchain technology, mining is the process of adding transactions to the distributed public ledger of existing transactions, known as the blockchain (Lantz & Cawrey, 2020). The term is best known for its association with bitcoin, though other technologies using blockchain technologies employ mining. Bitcoin mining rewards people who run mining operations with more bitcoins. Mining involves hashing a block of transactions that cannot be easily forged, protecting the integrity of the entire blockchain without the need for a central system. Mining is typically done on a dedicated computer, as it requires a fast CPU, as well as higher electricity usage and more heat generated than typical computer operations. The main incentive for mining is that users who choose to use a computer for mining are rewarded for doing so. Consensus in a distributed network refers to the problem where the nodes must come to an agreement while in the presence of faulty or malicious nodes. It also leads to a Byzantine Generals problem (Lamport et al., 2019). It is mainly a communication failure problem as in how can a system, receiving the data be certain that they received the correct information. This lead to the Byzantine fault tolerance (BFT) which aims to solve the problem of reaching a consensus when nodes can generate

arbitrary data. BFT states that it can guarantee the safety and make sure progress is made in the system provided that no more than

$$\lfloor (n-1) \div 3 \rfloor \tag{2.1}$$

nodes are faulty over the lifetime of the system where n is the total number of nodes (Castro, Liskov, et al., 1999). BFT can typically handle up to 33% of the nodes being faulty. Consensus mechanisms such as proof of work (PoW) and proof of stake (PoS) are the most common ones. As stated in the bitcoin paper the PoW system works by searching for a value that when hashed has a hash starting with a number of zero bits (Nakamoto, 2008). This is achieved by adding a nonce until the resulting hash starts with the requisite number of bits. Once the nonce has been identified, proof of work is satisfied and cannot be changed without doing the same work for it and for every subsequent block again which is computationally very expensive. Proof of Stake was introduced in the PeerCoin cryptocurrency which is a hybrid design where PoW is used for initial coin mining and PoS is used for the network security (King & Nadal, 2012). In PoS, the age of each coin is taken into consideration and is termed as coin-days which increases each day the coin is held. Once the coin or coins are spent the coin-day resets to zero. According to PoS the amount a validator need to contribute to mint a new block is given by

$$proofofhash < coins \times age \times target \tag{2.2}$$

The proofofhash as stated in BlackCoin Vasin, 2014 is a obfuscation sum that depends on a stake, the unspent output and the current time. Coins are the number of coins a miner has spent for mining privileges, age being the age of the coins that have been spent and target is the required amount of coins as specified, through a difficulty adjustment process by the network. Cardano's Ouroboros protocol is a modified version of the PoS protocol to include additional security measures to ensure persistance and liveliness in the system (Kiayias et al., 2017). In this a delegation process is included to elect the stakeholder and a snapshot is taken of current stakeholders in what is know as an epoch. In each epoch, a new stakeholder is elected by a subset of the stakeholders who randomly decide who the stakeholders will be in the next

epoch. Stellar Consensus Protocol (SCP) is another decentralized protocol where the nodes within the network do not need to trust the entire network but rather have the ability to choose which nodes they trust (Mazieres, 2015). The group is referred to as a quorum slice and a quorum is a set of nodes sufficient to reach an agreement. A quorum slice is a subset of a quorum which convinces one particular node of agreement. Proof of Importance (PoI) which is used by the NEM network is another such protocol. They have their own underlying cryptocurrency NEM. PoI works by having accounts own the NEM cryptocurrency. An account must hold atleast 10,000 NEM to be eligible for importance calculation. The importance is calculated by the amount of NEM held, the rank of the account within the network, a weighting factor, and two constants determined by the NEM network. Currently PoW is the most common with some trying to adopt PoS or other protocols. Ethereum is trying to switch to PoS from PoW as their consensus protocol.

2.3 Ethereum

The idea of Ethereum is to create an alternative protocol for building decentralized applications, providing a different set of trade offs that might be useful for a large class of decentralized applications (Buterin, 2013). It gives particular emphasis to situations where rapid development time, security for small and rarely used applications, and the ability to efficiently interact between different applications, are important. Ethereum achieves this by building the abstract foundation layer which is a blockchain with a built-in Turing-complete programming language, Solidity. Solidity allows anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.

2.3.1 Ethereum State

In bitcoin, the ledger can be considered as a state transition system where the "state" consists of the ownership status of all existing bitcoins and a state transition function which takes a state and a trans-

action and outputs a new state as the result (Wackerow, 2022). The Ethereum state transition function, APPLY(S,TX) -> S' can be defined as:

- Verify if the transaction is well-formed (that is, has the right number of values), the signature is valid, and the nonce matches the nonce on the sender's account. If not, return an error.
- The transaction fee is calculated as STARTGAS * GASPRICE, and the senders address is determined from the signature. Tthe fee amount is reduced from the senders account balance and the senders nonce is incremented. In case of insufficient funds, an error is returned.
- Initialize GAS = STARTGAS, and reduce a certain quantity of gas per byte to pay for the bytes in the transaction.
- The transaction value is transferred from the senders account to the receivers account. The receivers account is created if it does not exist. In case the receiving account is a contract, execute the contracts code either till completion or until the execution runs out of gas.
- In case the value transfer fails due to the sender not having enough money, or if the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.
- In case of other failures, refund the fee and all remaining gas to the sender, and send the fee paid for gas consumed to the miner.

2.3.2 Code Execution

The code for Ethereum smart contracts is written in a low-level, stack-based bytecode language, known as "Ethereum virtual machine code" or "EVM code". The code is a series of bytes, where each byte represents an operation. Code execution in Ethereum is an infinite loop that repeatedly performs the operation at the current program counter (starting from zero) and then increments the program counter by one, until

the end of the code or in case of an error or STOP or if RETURN instruction is detected. The operations have access to three types of space to store data:

- The stack, which is a last-in-first-out container where values can be pushed and popped
- Memory, which is an infinitely expandable byte array
- The contract's long-term storage in form of a key/value store. Storage persists for the long term, unlike stack and memory, which reset after computation ends,

The value, sender and data of the incoming message, as well as block header data can be accessed by the code which can also return a byte array of data as an output. The formal execution model of EVM code is pretty simple. When the Ethereum virtual machine is active, its computational state can be defined by the tuple (block-state, transaction, message, code, memory, stack, pc, gas), where block-state is the global state which contain all accounts and includes balances and storage of the accounts. At the beginning of each execution round, the current instruction is found by taking the pc-th byte of the code (or 0 if pc >= len(code)), and each instruction has a definition on how it affects the tuple. A basic implementation of Ethereum takes a few hundred lines of code. However, there are many ways to optimize Ethereum virtual machine execution via just-in-time compilation.

2.3.3 Smart Contracts

A smart contract is a program that runs on the Ethereum (Buterin, 2013) blockchain. It resides at a specific address on the block and is a collection of code (functions) and data (it's state). Smart contracts are a type of Ethereum account which means they have a balance and can send transactions over a network. They are deployed to the blockchain and run as programmed without any external interference. Users can interact with the smart contract by submitting transactions from their Ethereum accounts which will execute a function that's defined on the smart contract. Joshua, 2022 states that like a regular contract, smart contracts can have rules which can be automatically enforced via code. Once they are deployed they cannot be deleted and interactions with them are irreversible. Smart contracts are permission-less and

can be written and deployed by anyone. A certain gas fee is required to deploy the smart contract to the blockchain as deploying a contract is technically a transaction. Following are developer-friendly languages for writing smart contracts for Ethereum:

- Solidity
- Vyper

```
pragma solidity 0.8.7;

contract VendingMachine {

    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
    // 1. set the deploying address as the owner of the contract
    // 2. set the deployed smart contract's cupcake balance to
100
    constructor() {
        owner = msg.sender;
        cupcakeBalances[address(this)] = 100;
    }

    // Allow the owner to increase the smart contract's cupcake
balance
    function refill(uint amount) public {
        require(msg.sender == owner, "Only the owner can
refill.");
        cupcakeBalances[address(this)] += amount;
    }

    // Allow anyone to purchase cupcakes
    function purchase(uint amount) public payable {
        require(msg.value >= amount * 1 ether, "You must pay at
least 1 ETH per cupcake");
        require(cupcakeBalances[address(this)] >= amount, "Not
enough cupcakes in stock to complete this purchase");
        cupcakeBalances[address(this)] -= amount;
        cupcakeBalances[msg.sender] += amount;
    }
}
```

Figure 2.3: Example smart contract for a vending machine

Smart contracts, as Joshua, 2022 states, cannot send HTTP requests and thus they alone cannot get information about "real-world" events and that is by design. Relying on external information could jeopardise consensus, which is important for security and decentralization. Oracles can be used to get

around this. Another limitation of smart contracts is the maximum contract size. A smart contract will run out of gas if a contract is more than 24KB.

2.3.4 dApp

A decentralized application (dApp), as talked about by Svantes, 2022 is an application that is built on a decentralized network instead of a single source of control. It combines smart contracts with a front-end user interface. For Ethereum, smart contracts are accessible and transparent; similar to open APIs. Thus a dApp can even include a smart contract written by someone else.

The back-end code of a dApp, runs on a decentralized peer-to-peer network in contrast to normal apps where the backend code runs on centralized servers (Svantes, 2022). The front-end user interfaces of a dApp can be coded in any language (just like an app) to make calls to its back-end. Decentralized storage such as IPFS can be used to host the front-end. The back-end of a dapp is basically a smart contract. As stated bu Svantes, 2022, some feature of dApp are:

- Decentralized - dApps operate on Ethereum, an open public decentralized platform where no one person or group has control
- Deterministic - dApps perform the same function irrespective of the environment in which they get executed
- Turing complete - dApps can perform any action given the required resources
- Isolated - dApps are executed in a virtual environment known as Ethereum Virtual Machine (EVM) so that if the smart contract has a bug, it won't hamper the normal functioning of the blockchain network

A smart contract is a code that resides on the Ethereum blockchain and gets executed exactly as programmed (Svantes, 2022). Once smart contracts are deployed to the network they can't be changed. dApps are said to be decentralized because they are controlled by the logic on the contract and not by an

individual or company (Svantes, 2022). The contracts needs to be designed carefully and tested thoroughly before deploying them. Benefits of a dApp are:

- Zero downtime: Once a smart contract is deployed to the blockchain, the network can always serve clients who want to interact with the contract. Malicious actors, thus, are unable to launch denial-of-service attacks targeted towards individual dApps.
- Privacy: Real-world identity is not required to deploy or interact with a dApp.
- Resistance to censorship: A single entity in the network cannot stop users from submitting transactions, deploying dApps, or reading data from the blockchain.
- Data integrity: Use of cryptography makes data stored on the blockchain immutable and indisputable. Malicious users cannot forge transactions or other data that is already public.
- Trust-less computation/verifiable behavior: Smart contracts if written correctly are guaranteed to execute in expected ways, without the need of a central trust authority.

While dApps are have their advantages they also have their disadvantages. Some disadvantages are (Svantes, 2022):

- dApps can be harder to maintain because the code and data published to the blockchain are harder to modify. It's hard for developers to make updates to their dApps (or the underlying data stored by a dApp) once they are deployed - even if bugs or security risks are identified in an old version. A newer version of code can be redeployed and be redirected to but the old version cannot be deleted or modified.
- Performance overhead – There is a huge performance overhead, and scaling is really hard. To achieve the level of security, integrity, transparency, and reliability that Ethereum aspires to, every node runs and stores every transaction. On top of this, proof-of-work takes time as well. A back-of-the-envelope calculation puts the overhead at something like 1,000,000x that of standard computation currently.

- Network congestion – When one dApp uses too many computational resources, the entire network gets backed up. Currently, the network can only process about 10-15 transactions per second; if transactions are being sent in faster than this, the pool of unconfirmed transactions can quickly explode.

- User experience – It may be harder to engineer user-friendly experiences because the average end-user might find it too difficult to set up a tool stack necessary to interact with the blockchain in a truly secure fashion.

- Centralization – User-friendly and developer-friendly solutions built on top of the base layer of Ethereum might end up looking like centralized services anyway. For example, such services may store keys or other sensitive information server-side, serve a front-end using a centralized server, or run important business logic on a centralized server before writing to the blockchain. Centralization eliminates many (if not all) of the advantages of blockchain over the traditional model.

2.4 IPFS

The InterPlanetary File System (IPFS) is a decentralized, peer to peer file sharing protocol or file system(Benet, 2014). It tries to connect all computing devices within the same system of files. IPFS is similar to the web in some ways, but it can be viewed as a single BitTorrent swarm, exchanging objects within one Git repository. IPFS provides a high throughput block storage content-addressed model, with content-addressed hyper links (Benet, 2014). The result is a generalized Merkle DAG. It is a data structure on which it is possible to build versioned file systems, blockchains, and also a permanent web. IPFS combines a distributed hashtable, an incentivized block exchange, and a self-certifying namespace. In IPFS nodes do not need to trust each other and have no single point of failure.

IPFS is a distributed file system which combines successful ideas from previous peer-to-peer systems, such as DHTs, BitTorrent, Git, and SFS. IPFS aims to simplify, and connect proven techniques into a single cohesive system (Benet, 2014). It provides a new platform for writing and deploying applications,

and a new system for distributing and version controlling large data. IPFS has the potential to improve the web. IPFS is a peer-to-peer network where no nodes are privileged. IPFS objects are stored in local storage of nodes. Nodes can connect to each other and exchange objects. These objects are files and other data structures. The IPFS Protocol is a stack of sub-protocols responsible for different functionality:

- Identities: Manages node identity generation and verification
- Network: Manages connections to other nodes using several underlying network protocols.
- Routing: Stores information to locate specific peers and objects. Responds to both local and remote queries. The default is a DHT.
- Exchange: A novel block exchange protocol(BitSwap) that manages efficient block distribution.
- Objects: A Merkle DAG of content-addressed immutable objects with links. It is used to represent arbitrary data structures, such as file hierarchies and communication systems.
- Files: Versioned file system hierarchy which is inspired by Git.
- Naming: A self-certifying mutable name system.

IPFS uses content based addressing, termed as CID to server the files. A CID is the hash that is returned once the file is uploaded to the IPFS network. Thus, a CID fro IPFS looks like a hash and has the same length no matter the size of the content. IPFS however, does not encrypt the file that is stored on it. Anyone who has the CID of the file can get access to the file even if they acquire the CID by malicious means. Thus having an added layer of security for IPFS in the form of encrypted files would go a long way in making it more secure and better. This can be achieved by combining it with blockchain in such a way that the file being stored is encrypted and gets decrypted before being used or shared with the user. This way, both the CID of the encrypted file and the decryption key of the file need not be shared with the requester.

Chapter 3

System Architecture and Design

This chapter discusses architecture and its implementation and working. A dApp was created using the Ethereum blockchain and IPFS that allows users to share files with users who are looking for data for their algorithms. The files can be of any kind including csv data files. The decentralized storage system IPFS is used to store the data files that are uploaded to be share. This way the data owner need not be online to share their data. The design was created using javascript, HTML and CSS, Solidity (for smart contracts) and nodejs for the server side. By leveraging the Ethereum smart contract, the file is encrypted and also access control has also been added such that once a user buys a file they can use it at a later time without buying it again.

3.1 Architecture Design and Diagram

In the architecture shown in Figure 3.1 , users or data owners (DO) can sell a file which can be anything including data a file. They will be able to set a price for the file in ETH when uploading the file and it gets added to the web dApp to be bought by other users. Adding a file for sale involves sending the information as a transaction which incurs a gas fee that the data owners have to pay to have their file for sale on the web dApp. Once the file is on the web dApp other users can buy the file. This might discourage malicious

users from uploading a file as it requires them to pay for it before it can be added for sharing/sale. [1] For buying the file, users need to pay the price of the file along with a gas fee. The gas fee is to pay the miners who will add it to the blockchain ledger to keep track of who is selling it and who is buying it. The file that is uploaded by the data owner is stored on IPFS and thus the data owner does not have to be online at all times. IPFS stores the file and returns a CID which is a hash to access the file. However, this CID is not shared with the buyer but rather is used to fetch the data to the back-end server. It has many parts such as the frontend of the web dApp built using javascript, html and css, smart contracts written in solidity, which allows interaction with the blockchain to store and fetch information, IPFS is where the data is stored and a back-end where the data is fetched to, decrypted and run through a python file. Users adding their files need not have any identifiable information as only the wallet address is required for all transactions. The wallet address is similar to a hash and does not contain any user identifiable information. Thus, it is difficult to figure out the owner and buyer of files. This may make users more likely to share their data in exchange for incentives. The files that are stored on IPFS are encrypted. IPFS does not encrypt files by default, however this was achieved by using smart contracts which facilitates the encryption of the files stored on IPFS and allows easy decryption.

Another aspect of the dApp is Access control. It is implemented such that once the user buys a file they can use it at a later time again without buying it and the same file can be bought by multiple users. The file is also not directly handed over to the user but rather the encrypted file is fetched to a back-end server, followed by decryption and then a python file is run through it. The output of the python code can then be returned back to the user. Since the file is not handed over to the user the decryption key is not required to be shared with them either. The flow of work according to the architecture diagram is:

1. Users (Data Owners) add a file to share/sell along with setting its price and adding some information such as it's name and small description.

2. The file is encrypted using AES (Tate, 2001) encryption mechanism and then uploaded to IPFS, the distributed storage.

[1] https://github.com/soumyapal96/plata

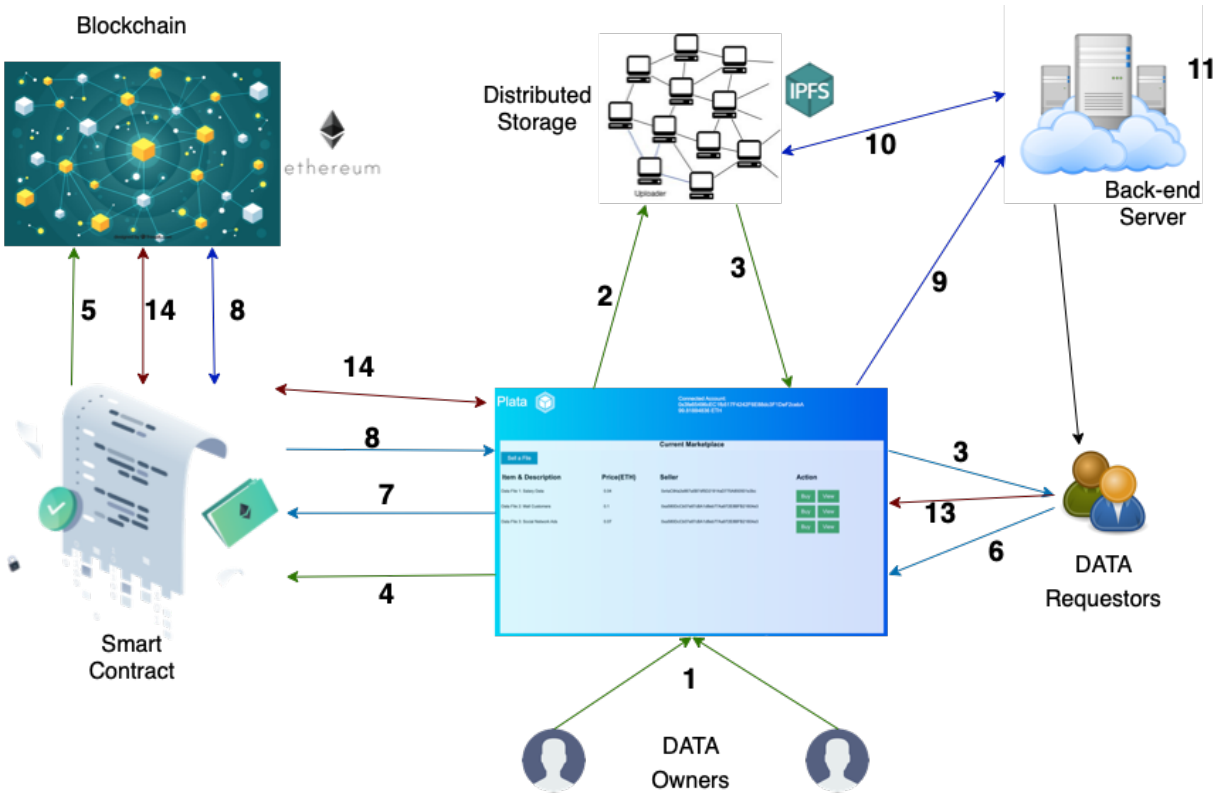

Figure 3.1: Architecture with IPFS

3. The CID of the file is then returned to be stored on the blockchain.

4. The smart contract is then evoked to store the information, such as the name, price, CID, and encryption key, on the blockchain. The addition of information to the blockchain is a transaction and thus costs a gas fee.

5. The information is then stored on the blockchain.

6. When a data requester buys a data file, they buy it by paying the price of the file along with the gas fee.

7. If the transaction is successful then the smart contract gets invoked to fetch the information for the corresponding file and to record the transaction.

8. The blockchain records the transaction of buying the file and returns the information to the web UI client side

9. The hash of the encrypted file and the encryption key is then send to a server

10. The server requests the encrypted file from IPFS with the help of the CID. IPFS on getting this request sends the encrypted file to the server.

11. This encrypted file is then decrypted with the help of the decryption key that the server received before. After decryption of the file it is then run through a python code.

12. The result of the python code can then be returned to the user. Thus the user never gets the file or the encryption key.

13. When a user requests to view a file that they have previously purchased, then after paying only the gas fee, the smart contract is invoked.

14. The smart contract checks if the user has purchased the file before and if they have then a transaction of viewing the file gets recorded on the blockchain. The file information is again fetched from the blockchain and sent to the web UI (client side).

15. Once the client side receives the information from the blockchain, the same process of sending the information to the server, file retrieval and running through a python code occurs.

16. The result can again be returned to the user.

[2]

3.1.1 The smart Contracts

The smart contracts have been written in the language Solidity. Market.sol is the main contract with two other contracts, Ownable.sol and Migrations.sol being used with the main contract. Ownable.sol

[2] https://github.com/soumyapal96/plata

makes sure the file uploaded is owned by the user who is uploading it and they get compensated when someone buys their uploaded file. Market.sol, as the main contract, has different user defined variables and functions. Article is a user defined variable or structure declared which has a few different data types

```
contract Market is Ownable {
    // Custom types
    struct Article {
        uint id;
        address payable seller;
        address buyer;
        string name;
        string decryptkey;
        uint256 price;
        string hashvalue;
        address[] ACL;
    }

    // State variables
    mapping(uint => Article) public articles;
    uint articleCounter;
    address seller;
    address buyer;
    string name;
    string decryptkey;
    uint256 price;
    string hashvalue;
    address[] ACL;
```

Figure 3.2: User defined variables of Smart Contract

in it as seen in Figure 3.2. Payable is a solidity keyword which denotes that the seller can be paid for their file. The 'payable' keyword results is cryptocurrency getting transferred to their wallet on a successful transaction. The smart contract is invoked to store the hash or CID of the file that gets uploaded to IPFS. ACL is an array that allows access control of files. Users who buy the file get added to the access control and then can use the file again at a later time. ACL, thus, is an array of wallet addresses. The sellArticle function as seen in Figure 3.3 is used to store information about the files on the blockchain. It stores

```
// sell an article
function sellArticle(string memory _name, string memory _decryptkey, uint256 _price, string memory _hashvalue) public {
    // a new article
    articleCounter++;

    // store this article
    articles[articleCounter] = Article(
        articleCounter,
        msg.sender,
        address(0),
        _name,
        _decryptkey,
        _price,
        _hashvalue,
       ACL
    );
```

Figure 3.3: Function to sell a File

information such as, the wallet address of the owner, price, hash, decryption key, name and description of file, on the blockchain. The adding of file for sale is a transaction which costs gas fee to be paid by the file seller. On successful recording of the data to the block or ledger a event log is emitted. This is used to send the information to the front-end of the web dApp as blockchains and smart contracts do not support https requests which is by design. Figure 3.4 depicts the function buyArticle to buy a file. It performs a few checks when a file is requested to be bought such as, if the requestor is the owner of the file as they cannot buy their own file. It also checks for any changes that might have occurred to the price of the file during during network transmission of information by unethical means by some network error then the file cannot be bought. The access control list (ACL) is update by adding the buyer's wallet address to it so they can use it again in the future. Once these checks are validated the the encrypted file that is stored on IPFS is fetched and decrypted followed by being used to run through a python file. The checkAccess function as in figure 3.5 is used to check if the user requesting to view the file bought the file before or not. If they bought the file, then the ACL would contain their wallet address and on successful checking, the CID and decryption key is fetched from the blockchain. Following this the CID is used to fetch the file to the back-end server and the key is used to decrypt it before being used to run through a python code.

```
// buy an article
function buyArticle(uint _id) public payable {

    // check that there is at least one article
    require(articleCounter > 0, "There should be at least one article");

    // check that the article exists
    require(_id > 0 && _id <= articleCounter, "Article with this id does not exist");

    // retrieve the article
    Article storage article = articles[_id];

    //buyer is added to the ACL
    article.ACL.push(msg.sender);

    // the owner cannot buy their own article
    require(article.seller != msg.sender, "Seller cannot buy his own article");

    // check that the price received sent corresponds to the article price
    require(article.price == msg.value, "Value provided does not match price of article");

    // store buyer's information
    article.buyer = msg.sender;

    // transfer the price to the owner
    article.seller.transfer(msg.value);

    // trigger the event
    emit LogBuyArticle(_id, article.seller, article.buyer, article.name, article.decryptkey, article.price, article.hashvalue, article.AC

}
```

Figure 3.4: Function to buy a File

```
function checkAccess(uint _id) public {
    uint flag = 0;
    Article storage article = articles[_id];
    for (uint j = 0; j < article.ACL.length; j++) {
        if (article.ACL[j] == msg.sender) {
            flag = 2;
            break;
        } else {
            flag = 1;
        }
    }
    if (flag == 2) {
     emit LogBuyArticle(_id, article.seller, article.buyer, article.name, article.decryptkey, article.price, article.hashvalue, article.ACL);
    } else {
     emit LogNoAccess("Need to purchase first");
    }
```

Figure 3.5: checkAccess function for ACL

3.2 Truffle and Ganache

As stated by (Rajeevan, 2019) Truffle is a development environment which is useful for testing framework for blockchains using the Ethereum Virtual Machine (EVM). It makes development of Ethereum based apps easier. Truffle provides the following advantages:

- It has built-in compilation and deployment of smart contracts.

- Contracts are tested automatically to allow fast development.
- It allows easy migration of smart contracts to the blockchain, in other words contract deployment.
- It can deploy smart contracts to multiple private and public blockchain networks.
- NPM is used for package management, while following the ERC190 standard.
- It provides an interactive console for communicating directly with contracts.
- It allows execution of scripts in a Truffle environment.

Ganache, as also stated by (Rajeevan, 2019), provides a personal blockchain for fast Ethereum distributed application development. Ganache can be used for the entire development cycle, as it allows development and deployment of smart contracts. It lets developers test dApps in a safe and deterministic environment. Ganache has both a UI and CLI. Ganache UI is a desktop application which supports both Ethereum and Corda technology. Ganache-cli which is the command-line tool, is available for Ethereum development. The test blockchain was generated with the help of Ganache. Ganache provides test wallets with ETH

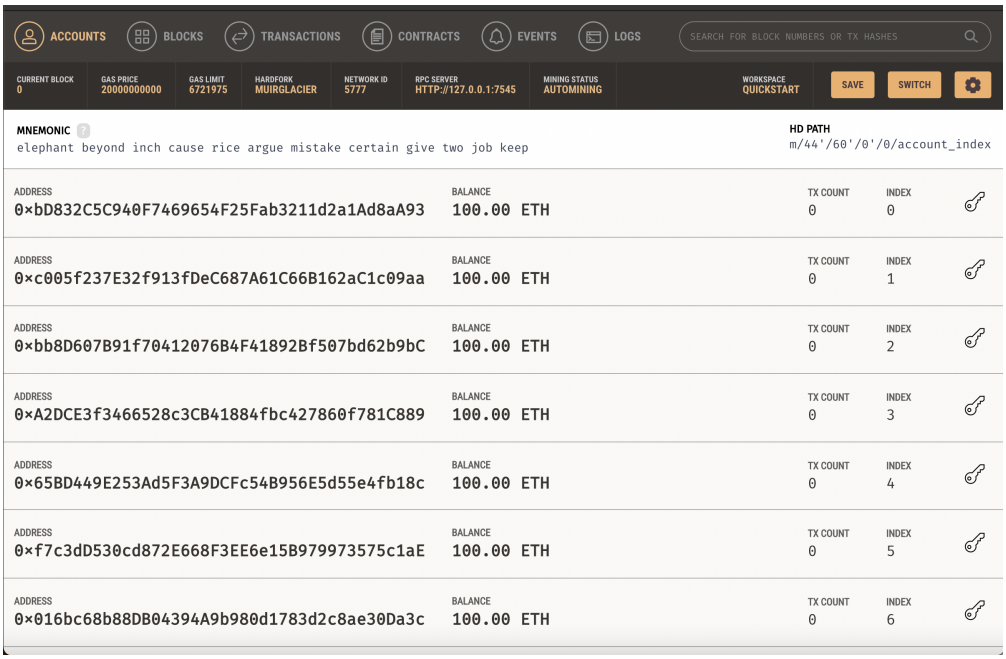

Figure 3.6: The Ganache GUI

to be used for development and testing of applications. The Ganache GUI as seen in Figure 3.6 provides many wallet addresses (10 each time a new chain is created) or accounts with test ETH that can be used for testing applications and/or smart contracts. Truffle was used to compile and deploy the smart contracts to the ganache test blockchain network. Deploying a contract is technically a transaction and costs a gas fee. The transactions from deploying the smart contracts to the blockchain is shown in Figure 3.7 The

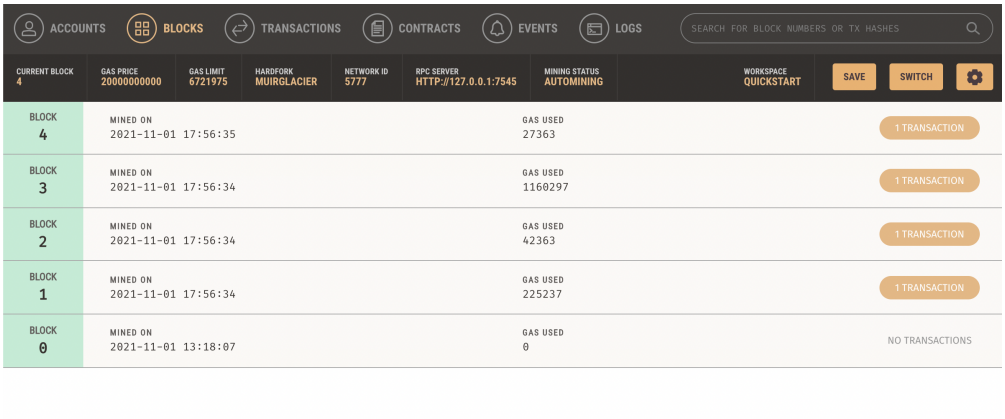

Figure 3.7: Result of contract deployment to Ganache and the transactions

JavaScript files in the migration folder are the codes that deploy the contracts in the contracts folder to the blockchain. Two contracts are deployed to the smart contracts which are the market.sol being the main contract and the Migrations.sol which keep track of the number of contracts deployed to the blockchain.

3.3 The Web dApp

A front-end is required for the users to interact with and view the files that are being sold. The front-end allows for an UI which can call the smart contracts from within it. It was created using JavaScript along with html and css. Users who want to sell data can add their file and the price with some other information as seen in figure 3.9. Once the file is added it gets uploaded to IPFS and the hash can be seen by the user who is uploading the file. After this a transaction, with cost equal to a gas fee, is made to store the information on the blockchain. The metamask wallet is used for making the transactions.

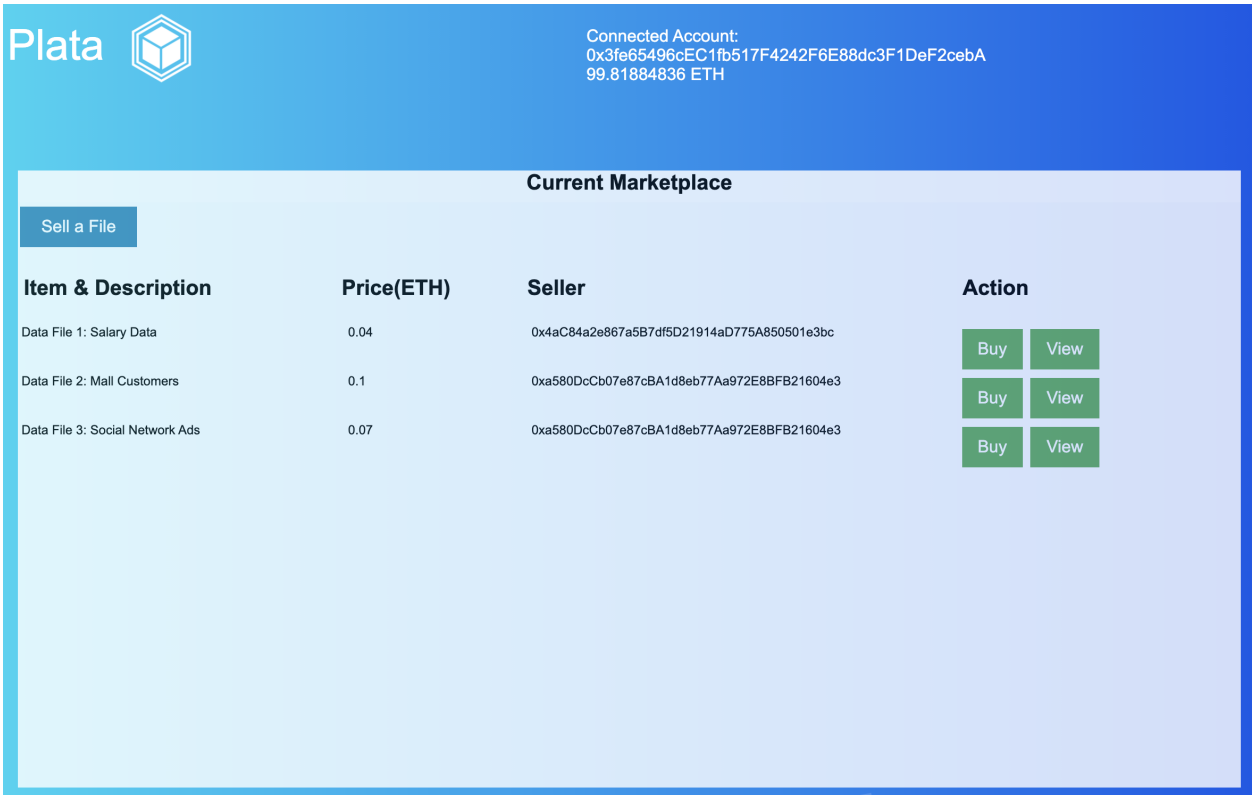

Figure 3.8: The front-end where the added file can be bought

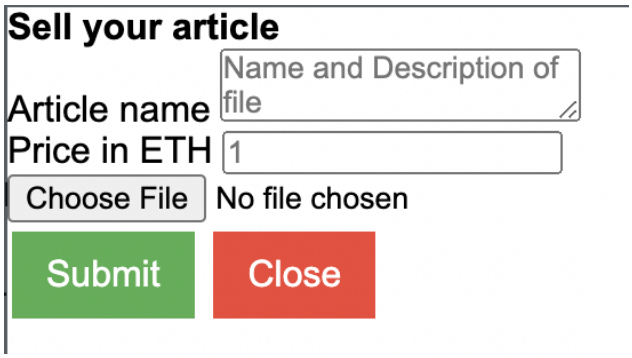

Figure 3.9: Frontend User Interface to upload a file

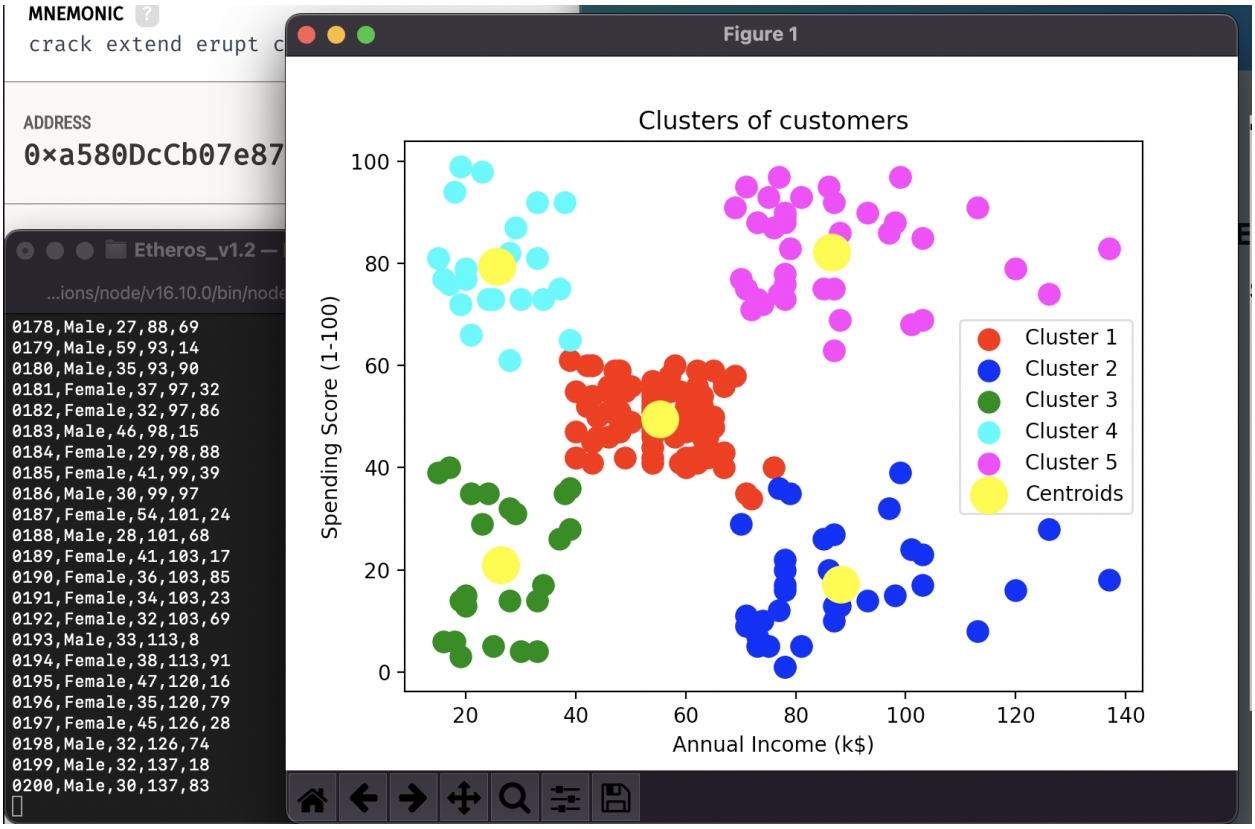

Figure 3.10: Result of python code after decryption of file

A buyer can buy or view the files from the front-end. They can click the buy button as seen in figure 3.8 which will automatically open metamask with the quoted price. On successful transfer of the price in ETH, the file is fetched from IPFS to a back-end server and run through a python file on the back-end server. The transaction to buy a file also includes a gas fee which needs to be paid by the buyer. Once they have bought the file they can use it later as well to run through python codes. The view button can be used to use the file in future. This essentially is a form of access control where one can use the file in the future if you have bought it. In other words buying the file gives access to use it multiples times without buying it again. Users who have not bought the file cannot use it without buying it first. The result can then be returned to the user.

The back-end server is written using nodejs. Once the back-end receives the CID of the file and the decryption key, it requests IPFS for the file corresponding to the CID and fetches it to the server. After the file is fetched, it gets decrypted and a python code is run through it after decryption. The result of the python code can be seen in figure 3.10 and the result can then be returned to the user who bought or requested to view the file, if already bought before.

3.4 Metamask

Metamask, as Rajeevan, 2019 said, is a crypto wallet which can be used with websites based on Ethereum. MetaMask allows users to manage accounts and their keys in many ways while isolating them from the site context as it opens in a new window. It supports hardware wallets as well. Metamask does not collect any user identifiable information and uses a password and seed phrase for recovery. It is an improvement over storing user keys on a central server which may result in mass account thefts. It is convenient for development. As stated by Rajeevan, 2019, it involves interacting with an Ethereum API that identifies the users web3-compatible browsers, and whenever a transaction request is made with signatures such as eth-sendTransaction or eth-signTypedData, MetaMask automatically prompts the user to pay using metamask accounts. MetaMask comes pre-loaded with fast connections to the Ethereum blockchain and test networks and can also connect to local Ethereum based blockchain such as the one created by ganache.

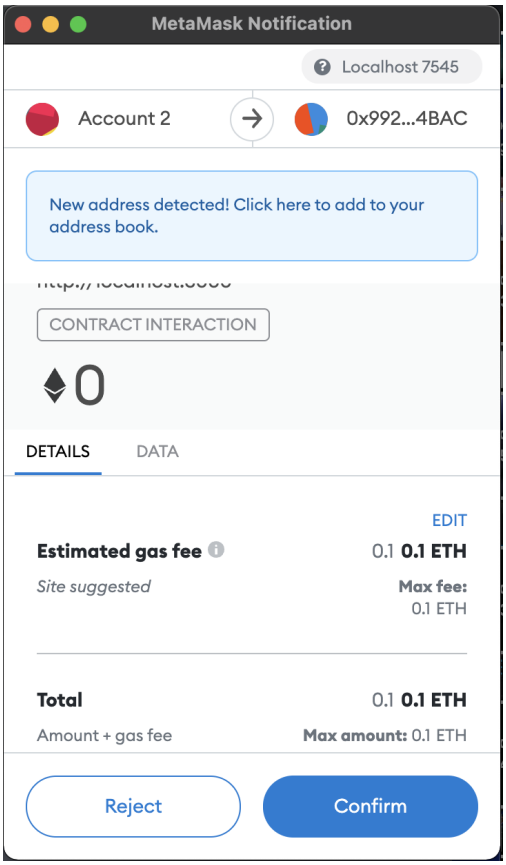

Figure 3.11: Metamask Transaction for selling/adding the file requiring gas fee

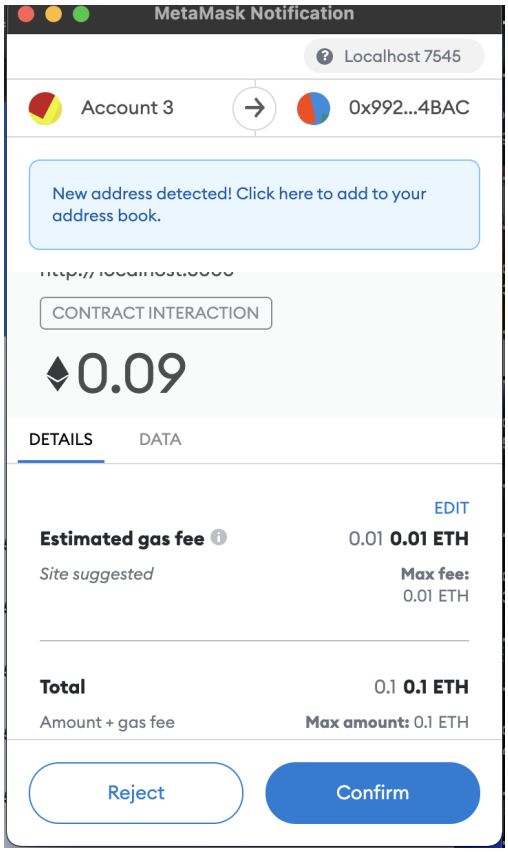

Figure 3.12: Metamask transaction to buy the file

Transaction for adding a file to the webapp is seen in figure 3.11 and figure 3.12 shows the transaction for buying a file.

3.5 Decentralized computing network Golem

Golem network is a decentralized supercomputer which aims to create a global market for computing power (Zawistowski et al., 2016). Golem connects computers in a peer-to-peer network, enabling both application owners and individual users ("requestors") to rent resources of other users' ("providers") machines. For this paper a simple hash cracker was used for testing the framework. In hash cracker, the word corresponding to the given has is found by checking it against a list of words in a dictionary. Here files containing a hash is uploaded to the webapp to be sold and the file is stored encrypted on IPFS. After the file is bought, it is fetched from IPFS and is run through a python code to crack it. This code is run on the decentralized network of Golem computing. Running a tast on Golem computing requires a fee to be paid for using the computing resources of the users who share their computing resources. The result of Golem after it is executed is seen in figure 3.14

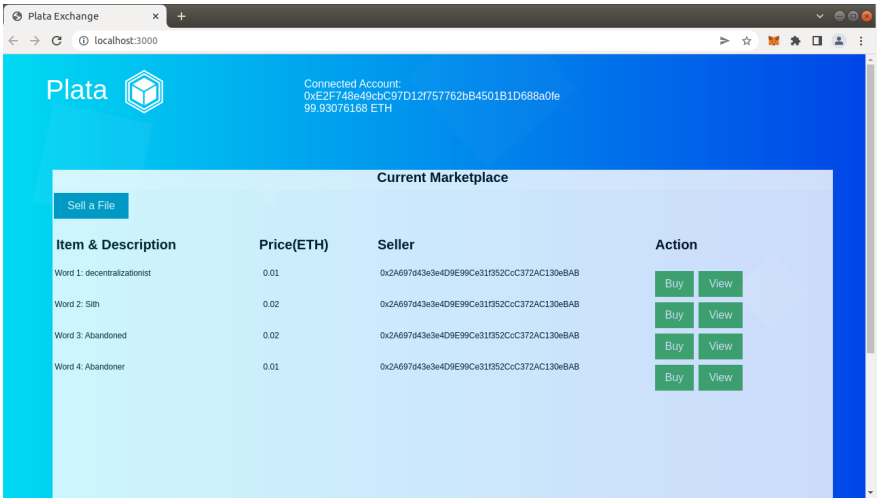

Figure 3.13: The different Hash files on the front-end

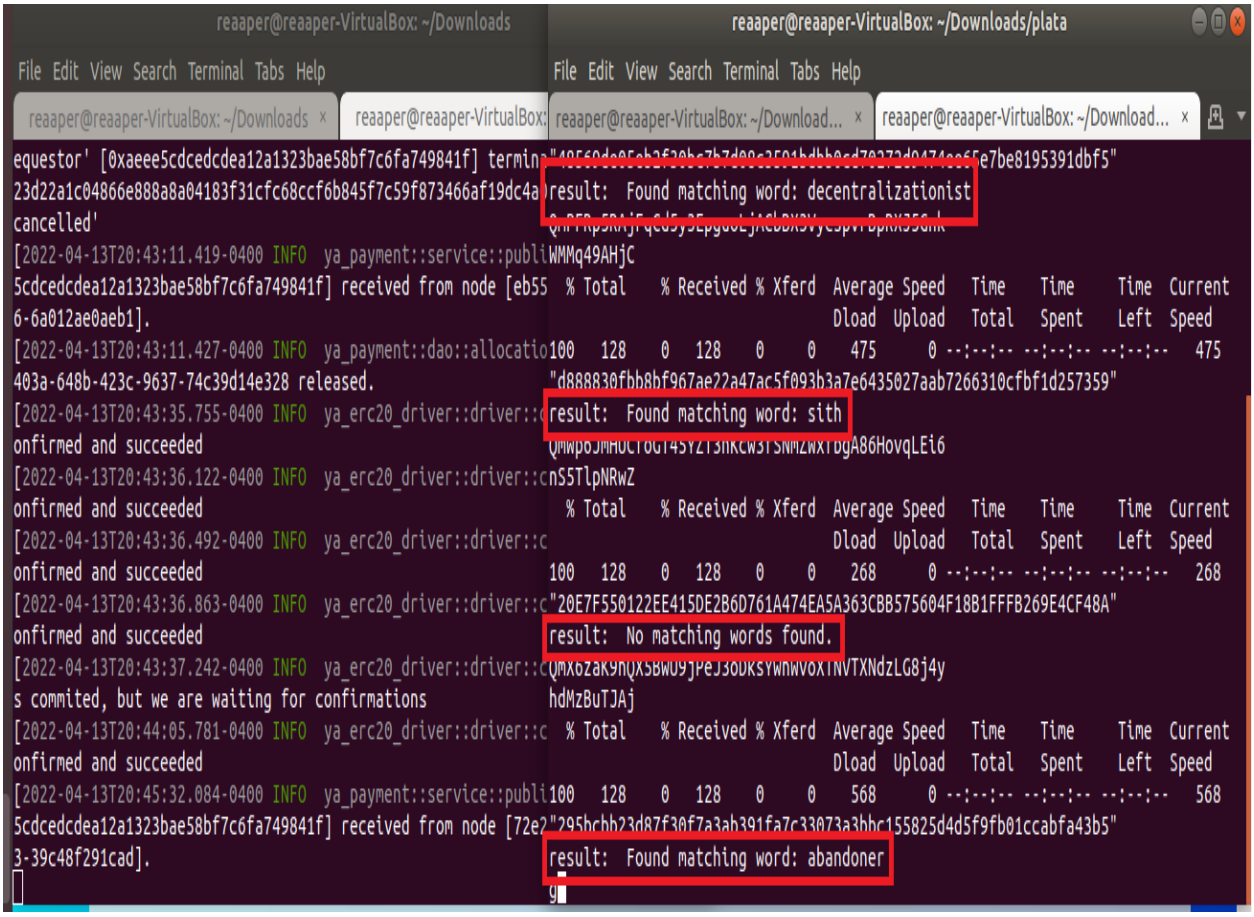

Figure 3.14: Result of hash cracker after running on Golem network is on the right terminal. The left terminal shows the Golem network as it executes the python code

Chapter 4

Conclusion and Future Work

A working model was created using Ethereum as the blockchain and IPFS as the decentralized storage where users can add files to sell or others can buy the file by paying for it in crypto currency. The users sell or buy data using their crypto wallet address and requires no user identifiable information. The data being stored on IPFS is encrypted so having the CID won't be of much use as it will only see the encrypted data. Access control has also been introduced with it where once the data is bought it can be run through a python code multiple times as the data is not directly handed over to the buyer. This adds a layer of security with the encryption and an added feature of access control. Most of the technologies used such as blockchain, IPFS and other related technologies are in the alpha stage with changes taking place quite frequently. This work can be improved in many ways to make it more secure and to add more layers of security during sharing of sensitive data. The data processing after fetching the file takes place on a central server which can be made decentralized by running on a decentralized network instead like Golem network or iExec. Blockchain technology has potential to be used in different scenarios for data sharing as it only requires the use of a wallet as identity and stored every record of transaction on an immutable ledger which is the blockchain.

Bibliography

Benet, J. (2014). Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*.

Buterin, V. (2013). Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper.

Castro, M., Liskov, B. et al. (1999). Practical byzantine fault tolerance. *OSDI*, *99*(1999), 173–186.

Dabit, N. (2021). *What is web3? the decentralized internet of the future explained*. https://www.freecodecamp.org/news/what-is-web3/

Dambrot, S. M. (2018). Regene: Blockchain backup of genome data and restoration of pre-engineered expressed phenotype. *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 945–950.

Desai, H., Kantarcioglu, M., & Kagal, L. (2019). A hybrid blockchain architecture for privacy-enabled and accountable auctions. *2019 IEEE International Conference on Blockchain (Blockchain)*, 34–43.

Guo, H., Li, W., Nejad, M., & Shen, C.-C. (2019). Access control for electronic health records with hybrid blockchain-edge architecture. *2019 IEEE International Conference on Blockchain (Blockchain)*, 44–51.

Joshua. (2022). *Introduction to smart contracts*. https://ethereum.org/en/developers/docs/smart-contracts/

Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. *Annual International Cryptology Conference*, 357–388.

King, S., & Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, *19*(1).

Lamport, L., Shostak, R., & Pease, M. (2019). The byzantine generals problem. *Concurrency: The works of leslie lamport* (pp. 203–226).

Lantz, L., & Cawrey, D. (2020). *Mastering blockchain*. O'Reilly Media.

Mazieres, D. (2015). The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, *32*.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.

Rajeevan, A. (2019). *Ethereum tools*. https://arunrajeevan.medium.com/ethereum-tools-1e729ecbb05e

Richards, S. (2021). *Web2 vs web3*. https://ethereum.org/en/developers/docs/web2-vs-web3/

Richter, B., Mengelkamp, E., & Weinhardt, C. (2018). Maturity of blockchain technology in local electricity markets. *2018 15th International Conference on the European Energy Market (EEM)*, 1–6.

Roman, V., & Ordieres-Mere, J. (2019). Iot blockchain technologies for smart sensors based on raspberry pi. *IEEE 11th international conference on service-oriented computing and applications IoT*, 216–220.

Somy, N. B., Kannan, K., Arya, V., Hans, S., Singh, A., Lohia, P., & Mehta, S. (2019). Ownership preserving ai market places using blockchain. *2019 IEEE International Conference on Blockchain (Blockchain)*, 156–165.

Svantes, F. (2022). *Introduction to dapps*. https://ethereum.org/en/developers/docs/dapps/

Tate, S. R. (2001). The advanced encryption standard.

Vasin, P. (2014). Blackcoin's proof-of-stake protocol v2. *URL: https://blackcoin. co/blackcoin-pos-protocol-v2-whitepaper. pdf*, *71*.

Wackerow, P. (2022). *Ethereum virtual machine (evm)*. https://ethereum.org/en/developers/docs/evm/

Zawistowski, J., Janiuk, P., & Regulski, A. (2016). The golem project–crowdfunding. whitepaper. *Golem,(Nov. 201)*, *28*.