**ABSTRACT**

Blockchain is arguably one of the most significant and disruptive technologies that came into existence since the inception of the Internet. It's the core technology behind Bitcoin and other crypto-currencies that drew a lot of attention in the last few years.As its core, a blockchain is a distributed database that allows direct transactions between two parties without the need of a central authority. This simple yet powerful concept has great implications for various institutions such as banks, governments and marketplaces, just to name a few. Any business or organization that relies on a centralized database as a core competitive advantage can potentially be disrupted by block chain technology.

.

**CHAPTER 1**

**INTRODUCTION**

**1.1 Introduction**

A Blockchain is a vastly distributed, global, open, shared ledger or database that run on millions of devices, recording all transactions happening between any two parties. It does this efficiently and in a verifiable manner, essentially acting as a secure storage for not just information, but anything of value.In simple terms, a Blockchain is a decentralized network for value exchange much which acted as a decentralized network for information exchange. The blockchain is considered to be a foundational technology that no single organization owns or controls.

**1.2 Problem Statement**

Intermediaries today are largely centralized systems This means that they are immensely vulnerable to attacks and failures. Centralized systems, by design, have a single point of failure.This presents tremendous opportunities for malicious players to stage large scale attacks.

**CHAPTER 2**

**Literature suvery**

**[1] Kabi, O.R., Franqueira, V.N.L.: Blockchain-Based Distributed Marketplace. In: Abramowicz W., Paschke A. (eds) Business Information Systems Workshops. BIS 2018. Lecture Notes in Business Information Processing, vol 339. Springer, Cham (2018).**

Developments in Blockchain technology have enabled the creation of smart contracts; i.e., self-executing code that is stored and executed on the Blockchain. This has led to the creation of distributed, decentralised applications, along with frameworks for developing and deploying them easily. This paper describes a proof-of-concept system that implements a distributed online marketplace using the Ethereum framework, where buyers and sellers can engage in e-commerce transactions without the need of a large central entity coordinating the process. The performance of the system was measured in terms of cost of use through the concept of ‘gas usage’. It was determined that such costs are significantly less than that of Amazon and eBay for high volume users. The findings generally support the ability to use Ethereum to create a distributed on-chain market, however, there are still areas that require further research and development.

**[2] Özyilmaz, K.R., Doğan M., Yurdakul A.: IDMoB: IoT Data Marketplace on Blockchain. In: Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, 2018, pp. 11-19 (2018)**.

Today, Internet of Things (IoT) devices are the powerhouse of data generation with their ever-increasing numbers and widespread penetration. Similarly, artificial intelligence (AI) and machine learning (ML) solutions are getting integrated to all kinds of services, making products significantly more "smarter". The centerpiece of these technologies is "data". IoT device vendors should be able keep up with the increased throughput and come up with new business models. On the other hand, AI/ML solutions will produce better results if training data is diverse and plentiful. In this paper, we propose a blockchain-based, decentralized and trustless data marketplace where IoT device vendors and AI/ML solution providers may interact and collaborate. By facilitating a transparent data exchange platform, access to consented data will be democratized and the variety of services targeting end-users will increase. Proposed data marketplace is implemented as a smart contract on Ethereum blockchain and Swarm is used as the distributed storage platform

**[3] Cruz, J. P., Kaji, Y., Yanai N.: RBAC-SC: Role-Based Access Control Using Smart Contract. In: IEEE Access Volume 6, 12240-12251 (2018).**

The role-based access control (RBAC) framework is a mechanism that describes the access control principle. As a common interaction, an organization provides a service to a user who owns a certain role that was issued by a different organization. Such trans-organizational RBAC is common in face-toface communication but not in a computer network because it is difficult to establish both the security that prohibits malicious impersonation of roles and the flexibility that allows small organizations to participate and users to fully control their own roles. In this paper, we present a role-based access control using smart contract (RBAC-SC), a platform that makes use of Ethereum’s smart contract technology to realize a transorganizational utilization of roles. Ethereum is an open blockchain platform that is designed to be secure, adaptable, and flexible. It pioneered smart contracts, which are decentralized applications that serve as “autonomous agents” running exactly as programmed and are deployed on a blockchain. The RBAC-SC uses smart contracts and blockchain technology as versatile infrastructures to represent the trust and endorsement relationship that are essential in RBAC and to realize a challenge-response authentication protocol that verifies a user’s ownership of roles. We describe the RBAC-SC framework, which is composed of two main parts, namely, the smart contract and the challenge-response protocol, and present a performance analysis. A prototype of the smart contract is created and deployed on Ethereum’s Testnet blockchain, and the source code is publicly available.

**[4] Xia, Q., Sifah, E.B., Asamoah, K.O., Gao, J., Du, X., Guizani, M.: MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain. In: IEEE Access, vol. 5, pp. 14757-14767, (2017).**

Recent years have witnessed a paradigm shift in the storage of Electronic Health Records (EHRs) on mobile cloud environments, where mobile devices are integrated with cloud computing to facilitate medical data exchanges among patients and healthcare providers. This advanced model enables healthcare services with low operational cost, high flexibility, and EHRs availability. However, this new paradigm also raises concerns about data privacy and network security for e-health systems. How to reliably share EHRs among mobile users while guaranteeing high-security levels in the mobile cloud is a challenging issue. In this paper, we propose a novel EHRs sharing framework that combines blockchain and the decentralized interplanetary file system (IPFS) on a mobile cloud platform. Particularly, we design a trustworthy access control mechanism using smart contracts to achieve secure EHRs sharing among different patients and medical providers. We present a prototype implementation using Ethereum blockchain in a real data sharing scenario on a mobile app with Amazon cloud computing. The empirical results show that our proposal provides an effective solution for reliable data exchanges on mobile clouds while preserving sensitive health information against potential threats. The system evaluation and security analysis also demonstrate the performance improvements in lightweight access control design, minimum network latency with high security and data privacy levels, compared to the existing data sharing models.

**[5] Azaria, A., Ekblaw, A., Vieira, T.,Lippman, A.: MedRec: Using Blockchain for Medical Data Access and Permission Management. In: 2nd International Conference on Open and Big Data (OBD) pp. 25-30, (2016).**

Years of heavy regulation and bureaucratic inefficiency have slowed innovation for electronic medical records (EMRs). We now face a critical need for such innovation, as personalization and data science prompt patients to engage in the details of their healthcare and restore agency over their medical data. In this paper, we propose MedRec: a novel, decentralized record management system to handle EMRs, using blockchain technology. Our system gives patients a comprehensive, immutable log and easy access to their medical information across providers and treatment sites. Leveraging unique blockchain properties, MedRec manages authentication, confidentiality, accountability and data sharing- crucial considerations when handling sensitive information. A modular design integrates with providers' existing, local data storage solutions, facilitating interoperability and making our system convenient and adaptable. We incentivize medical stakeholders (researchers, public health authorities, etc.) to participate in the network as blockchain “miners”. This provides them with access to aggregate, anonymized data as mining rewards, in return for sustaining and securing the network via Proof of Work. MedRec thus enables the emergence of data economics, supplying big data to empower researchers while engaging patients and providers in the choice to release metadata. The purpose of this short paper is to expose, prior to field tests, a working prototype through which we analyze and discuss our approach.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1 EXISTING SYSTEM**

Considering worst-case channel uncertainties, we investigate the robust secure beam forming design problem in multiple-input-single-output (MISO) full-duplex, two-way secure communications. Our objective is to maximize worst-case sum secrecy rate under weak secrecy conditions and individual transmit power constraints. Since the objective function of the optimization problem includes both convex and concave terms, we propose to transform convex terms into linear terms. We decouple the problem into four optimization problems and employ the alternating optimization algorithm to obtain the locally optimal solution. Simulation results demonstrate that our proposed robust secure beamforming scheme outperforms the nonrobust one. It is also found that when the regions of channel uncertainties and the individual transmit power constraints are sufficiently large, because of self-interference, the proposed two-way robust secure communication is proactively degraded to one-way communication.

**3.2 PROPOSED SYSTEM**

To solve the double-spending problem, Satoshi proposed a public ledger, i.e., Bitcoin’s blockchain to keep track of all transactions in the network. Distributed: The ledger is replicated across a number of computers, rather than being stored on a central server. Any computer with an internet connection can download a full copy of the blockchain.Cryptographic: Cryptography is used to make sure that the sender owns the bitcoin that she's trying to send, and to decide how the transactions are added to the blockchain. Immutable: The blockchain can be changed in append only fashion. In other words, transactions can only be added to the blockchain but cannot be deleted or modified. Uses Proof of Work (PoW): A special type of participants in the network called miners compete on searching for the solution to a cryptographic puzzle that will allow them to add a block of transactions to Bitcoin’s blockchain. This process is called Proof of Work and it allows the system to be secure (more on this later).

**3.3 HARWARE REQUIREMENTS**

Processor : Intel Pentium Dual Core 2.00GHz

Hard disk : 500 GB

RAM : 8 GB (minimum)

**3.4 SOFTWARE REQUIREMENTS**

* Python 3.6.4 Version

**3.5 SOFTWARE SPECIFICATION**

**3.5.1 Block Chain**

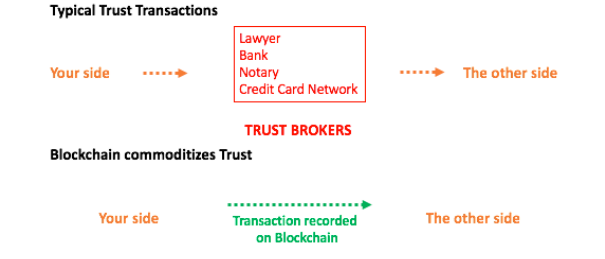
Block chain is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system. A blockchain is essentially a digital ledger of transactions that is duplicated and distributed across the entire network of computer systems on the blockchain.

**Uses of Block Chain**

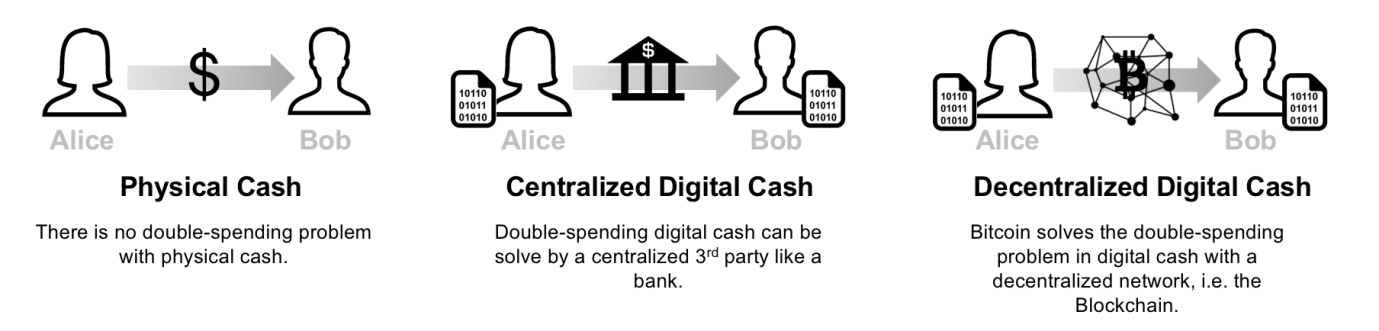
Blockchain technology can be integrated into multiple areas. The primary use of blockchains today is as a [distributed ledger](https://en.wikipedia.org/wiki/Distributed_ledger) for [cryptocurrencies](https://en.wikipedia.org/wiki/Cryptocurrency" \o "Cryptocurrency), most notably [bitcoin](https://en.wikipedia.org/wiki/Bitcoin" \o "Bitcoin). There are a few operational products maturing from [proof of concept](https://en.wikipedia.org/wiki/Proof_of_concept) by late 2016. Businesses have been thus far reluctant to place blockchain at the core of the business structure. Although businesses have been reluctant to fully implement blockchain, many have begun testing the technology and are conducting low-level implementation to gauge its effects on organizational efficiency.

In 2019, it was estimated that around $2.9 billion were invested in blockchain technology, which represents an 89% increase from the year prior. Additionally, the International Data Corp has estimated that corporate investment into blockchain technology will reach $12.4 billion by 2022.[[55]](https://en.wikipedia.org/wiki/Blockchain#cite_note-Forbes2-55) Furthermore, According to [PricewaterhouseCoopers](https://en.wikipedia.org/wiki/PricewaterhouseCoopers) (PwC), the second-largest professional services network in the world, blockchain technology has the potential to generate an annual business value of more than $3 trillion by 2030. PwC's estimate is further augmented by a 2018 study that they have conducted, in which PwC surveyed 600 business executives and determined that 84% have at least some exposure to utilizing blockchain technology, which indicts a significant demand and interest in blockchain technology.

Individual use of blockchain technology has also greatly increased since 2016. According to statistics in 2020, there were more than 40 million blockchain wallets in 2020 in comparison to around 10 million blockchain wallets in 2016.



**Blockchain Transcations**

****

**Types of Block Chain:**

Currently, there are at least four types of blockchain networks

* public blockchains
* private blockchains
* hybrid blockchains.
* Sidechains

### Public blockchains

A public blockchain has absolutely no access restrictions. Anyone with an [Internet](https://en.wikipedia.org/wiki/Internet) connection can send [transactions](https://en.wikipedia.org/wiki/Financial_transaction) to it as well as become a [validator](https://en.wikipedia.org/wiki/Validator" \o "Validator) (i.e., participate in the execution of a [consensus protocol](https://en.wikipedia.org/wiki/Consensus_(computer_science))). Usually, such networks offer [economic incentives](https://en.wikipedia.org/wiki/Incentive) for those who secure them and utilize some type of a [Proof of Stake](https://en.wikipedia.org/wiki/Proof-of-stake) or [Proof of Work](https://en.wikipedia.org/wiki/Proof-of-work_system) algorithm.

Some of the largest, most known public blockchains are the bitcoin blockchain and the Ethereum blockchain.

### Private blockchains

A private blockchain is permissioned. One cannot join it unless invited by the network administrators. Participant and validator access is [restricted](https://en.wikipedia.org/wiki/Closed_platform). To distinguish between open blockchains and other peer-to-peer decentralized database applications that are not open ad-hoc compute clusters, the terminology [Distributed Ledger](https://en.wikipedia.org/wiki/Distributed_Ledger) (DLT) is normally used for private blockchains.

### Hybrid blockchains

A hybrid blockchain has a combination of centralized and decentralized features. The exact workings of the chain can vary based on which portions of centralization decentralization are used.

### Sidechains

A sidechain is a designation for a blockchain ledger that runs in parallel to a primary blockchain. Entries from the primary blockchain (where said entries typically represent [digital assets](https://en.wikipedia.org/wiki/Digital_asset)) can be linked to and from the sidechain; this allows the sidechain to otherwise operate independently of the primary blockchain (e.g., by using an alternate means of record keeping, alternate [consensus algorithm](https://en.wikipedia.org/wiki/Consensus_(computer_science)), etc.)

**3.5.2 ANACONDA**

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. Anaconda Distribution is used by over 6 million users, and it includes more than 250 popular data science packages suitable for Windows, Linux, and MacOS.

Python is a high-level programming language devised by Guido van Rossum & first released in 1991. It’s the most popular coding language used by software developers to build, control, manage and for testing. It is also an interpreter which executes Python programs. The python interpreter is called python.exe on Windows.

**Python Packages**

Packages or additional libraries help in scientific computing and computational modelling. In Python, the packages are not the part of the Python standard library. Few major packages are –

numpy (NUMeric Python): matrices and linear algebra

scipy (SCIentific Python): many numerical routines

matplotlib: (PLOTting LIBrary) creating plots of data

sympy (SYMbolic Python): symbolic computation

pytest (Python TESTing): a code testing framework

Together with a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages.

This work environment, Anaconda is used for scientific computing, data science, statistical analysis, and machine learning. The latest version of Anaconda 5.0.1 is released in October 2017.The released version 5.0.1 addresses some minor bugs and adds useful features, such as updated R language support. All of these features weren’t available in the original 5.0.0 release.

This package manager is also an environment manager, a Python distribution, and a collection of open source packages and contains more than 1000 R and Python Data Science Packages.

**IPYTHON NOTEBOOKS**

IPython is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history. IPython provides the following features:

* Interactive shells (terminal and Qt-based).

• A browser-based notebook interface with support for code, text, mathematical expressions, inline plots and other media.

• Support for interactive data visualization and use of GUI toolkits.

• Flexible, embeddable interpreters to load into one's own projects.

• Tools for parallel computing.

IPython is based on an architecture that provides parallel and distributed computing. IPython enables parallel applications to be developed, executed, debugged and monitored interactively. Hence, the I (Interactive) in IPython.[3]This architecture abstracts out parallelism, which enables IPython to support many different styles of parallelism[4]including:

With the release of IPython 4.0, the parallel computing capabilities have been made optional and released under the ipyparallel python package.IPython frequently draw from SciPy stack[5] libraries like NumPy and SciPy, often installed alongside from one of many Scientific Python distributions. IPython provide integration some library of the SciPy stack like matplotlib, like inline graph when in used with the Jupyter notebook. Python libraries can implement IPython specific hooks to customize object Rich object display. SymPy for example implement rendering of Mathematical Expression as rendered LaTeX when used within IPython context.

Other features:

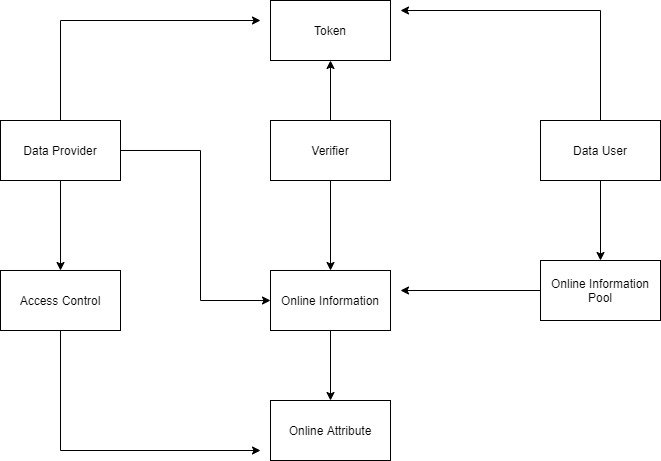
IPython also allows non-blocking interaction with Tkinter, PyGTK, PyQt/PySide and wxPython (the standard Python shell only allows interaction with Tkinter). IPython can interactively manage parallel computing clusters using asynchronous status call-backs and/or MPI. IPython can also be used as a system shell replacement. Its default behaviour is largely similar to Unix shells, but it allows customization and the flexibility of executing code in a live Python environment. Using IPython as a shell replacement is less common and it is now recommended to use Xonsh which provide most of the IPython feature with better shell integrations.

**CHAPTER 4**

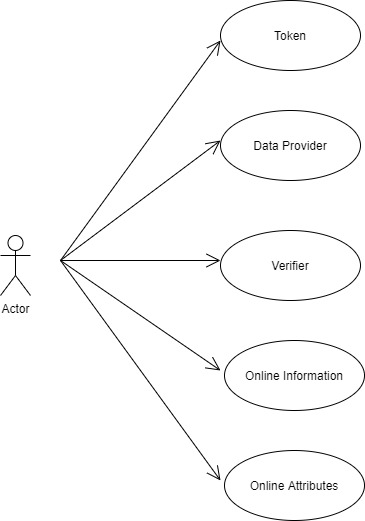
**ARCHITECTURE**

**4.1 SYSTEM ARCHITECTURE**

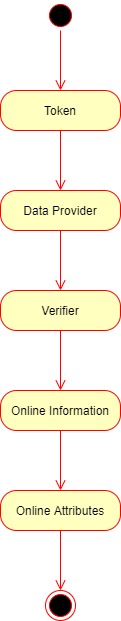
System architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

****

**4.2 Usecase Diagram**

****

**4.3 Activity Diagram**

****

**CHAPTER 5**

**SYSTEM MODULE**

**5.1 MODULE**

* **Process**
* **Attributes**
* **Hash Algorithm**
  1. **MODULE DESCRIPTION**

**Process**

* **Step 1 (one-time effort):** Create a bitcoin wallet. For a person to send or receive bitcoins, she needs to create a bitcoin wallet.
* A bitcoin wallet stores 2 pieces of information: A private key and a public key. The private key is a secret number that allows the owner to send bitcoin to another user, or spend bitcoins on services that accept them as payment method. The public key is a number that is needed to receive bitcoins.
* The public key is also referred to as bitcoin address (not entirely true, but for simplicity we will assume that the public key and the bitcoin address are the same). Note that the wallet doesn’t store the bitcoins themselves. Information about bitcoins balances are stored on the Bitcoin’s blockchain.
* **Step 2: Create a bitcoin transaction**: If Alice wants to send 1 BTC to Bob, Alice needs to connect to her bitcoin wallet using her private key, and create a transaction that contains the amount of bitcoins she wants to send and the address where she wants to send them (in this case Bob’s public address).
* **Step 3:** Broadcast the transaction to Bitcoin’s network. Once Alice creates the bitcoin transaction, she needs to broadcast this transaction to the entire Bitcoin’s network.
* **Step 4:** Confirm the transaction. A miner listening to Bitcoin’s network authenticates the transaction using Alice's public key, confirms that Alice has enough bitcoins in her wallet (in this case at least 1 BTC), and adds a new record to Bitcoin’s Blockchain containing the details of the transaction.
* **Step 5:** Broadcast the blockchain change to all miners. Once the transaction is confirmed, the miner should broadcast the blockchain change to all miners to make sure that their copies of the blockchain are all in sync.

**Attributes:**

* **Transactions:** List of transactions that will be added to the next block.
* **Chain:** The actual blockchain which is an array of blocks.
* **Nodes:** A set containing node urls. The blockchain uses these nodes to retrieve blockchain data from other nodes and updates its blockchain if they're not in sync.
* **node\_id:** A random string to identify the blockchain node.

The Blockchain class also implements the following methods:

* register\_node(node\_url): Adds a new blockchain node to the list of nodes.
* **verify\_transaction\_signature(sender\_address, signature, transaction):** Checks that the provided signature corresponds to transaction signed by the public key (sender\_address).
* **submit\_transaction(sender\_address, recipient\_address, value, signature):** Adds a transaction to list of transactions if the signature verified.
* **create\_block(nonce, previous\_hash):** Adds a block of transactions to the blockchain.
* **hash(block):** Create a SHA-256 hash of a block.
* **proof\_of\_work():** Proof of work algorithm. Looks for a nonce that satisfies the mining condition.
* **valid\_proof(transactions, last\_hash, nonce, difficulty=MINING\_DIFFICULTY):** Checks if a hash value satisfies the mining conditions. This function is used within the proof\_of\_work function.
* **valid\_chain(chain):** checks if a bockchain is valid.
* **resolve\_conflicts():** Resolves conflicts between blockchain's nodes by replacing a chain with the longest one in the network.

**Hash Algorithm**

* In this project, we compared the MD5 ash algorithm with ShA256 hash algorithm for transaction and we are going to find the performance of these two hash function
* This application generates two encoding cryptographic keys for viewing the private post and public posts.
* The key is generated by using the cryptographic algorithm.
* The user post undergoes hash encryption using the md5 algorithm and sha256 algorithm hashing functions.
* This project shows about blockchain which is a linked list of transaction which contains data and hash pointer to the previous block in the block chain
* Hash function is used to map data in blocks. The values returned by hash function are called hash values, hash codes, etc.
* Block  chain is a growing list of records called blocks which are linked using cryptography.
* Each block contains a cryptographic hash of the previous blocks and also transaction data.
* This technology can be implemented anywhere in health care, medicine, insurance, etc Here we are implementing this technology in our daily life activity which is in social now for high-security transaction of post between users.

**CHAPTER 6**

**SYSTEM DESIGN**

from collections import OrderedDict

import binascii

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

import requests

from flask import Flask, jsonify, request, render\_template

class Transaction:

def \_\_init\_\_(self, sender\_address, sender\_private\_key, recipient\_address, value):

self.sender\_address = sender\_address

self.sender\_private\_key = sender\_private\_key

self.recipient\_address = recipient\_address

self.value = value

def \_\_getattr\_\_(self, attr):

return self.data[attr]

def to\_dict(self):

return OrderedDict({'sender\_address': self.sender\_address,

'recipient\_address': self.recipient\_address,

'value': self.value})

def sign\_transaction(self):

"""

Sign transaction with private key

"""

private\_key = RSA.importKey(binascii.unhexlify(self.sender\_private\_key))

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode('utf8'))

return binascii.hexlify(signer.sign(h)).decode('ascii')

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('./index.html')

@app.route('/make/transaction')

def make\_transaction():

return render\_template('./make\_transaction.html')

@app.route('/view/transactions')

def view\_transaction():

return render\_template('./view\_transactions.html')

@app.route('/wallet/new', methods=['GET'])

def new\_wallet():

random\_gen = Crypto.Random.new().read

private\_key = RSA.generate(1024, random\_gen)

public\_key = private\_key.publickey()

response = {

'private\_key': binascii.hexlify(private\_key.exportKey(format='DER')).decode('ascii'),

'public\_key': binascii.hexlify(public\_key.exportKey(format='DER')).decode('ascii')

}

return jsonify(response), 200

@app.route('/generate/transaction', methods=['POST'])

def generate\_transaction():

sender\_address = request.form['sender\_address']

sender\_private\_key = request.form['sender\_private\_key']

recipient\_address = request.form['recipient\_address']

value = request.form['amount']

transaction = Transaction(sender\_address, sender\_private\_key, recipient\_address, value)

response = {'transaction': transaction.to\_dict(), 'signature': transaction.sign\_transaction()}

return jsonify(response), 200

if \_\_name\_\_ == '\_\_main\_\_':

from argparse import ArgumentParser

parser = ArgumentParser()

parser.add\_argument('-p', '--port', default=8080, type=int, help='port to listen on')

args = parser.parse\_args()

port = args.port

app.run(host='127.0.0.1', port=port)

Frontend

from collections import OrderedDict

import binascii

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

import hashlib

import json

from time import time

from urllib.parse import urlparse

from uuid import uuid4

import requests

from flask import Flask, jsonify, request, render\_template

from flask\_cors import CORS

MINING\_SENDER = "THE BLOCKCHAIN"

MINING\_REWARD = 1

MINING\_DIFFICULTY = 2

class Blockchain:

def \_\_init\_\_(self):

self.transactions = []

self.chain = []

self.nodes = set()

#Generate random number to be used as node\_id

self.node\_id = str(uuid4()).replace('-', '')

#Create genesis block

self.create\_block(0, '00')

def register\_node(self, node\_url):

"""

Add a new node to the list of nodes

"""

#Checking node\_url has valid format

parsed\_url = urlparse(node\_url)

if parsed\_url.netloc:

self.nodes.add(parsed\_url.netloc)

elif parsed\_url.path:

# Accepts an URL without scheme like '192.168.0.5:5000'.

self.nodes.add(parsed\_url.path)

else:

raise ValueError('Invalid URL')

def verify\_transaction\_signature(self, sender\_address, signature, transaction):

"""

Check that the provided signature corresponds to transaction

signed by the public key (sender\_address)

"""

public\_key = RSA.importKey(binascii.unhexlify(sender\_address))

verifier = PKCS1\_v1\_5.new(public\_key)

h = SHA.new(str(transaction).encode('utf8'))

return verifier.verify(h, binascii.unhexlify(signature))

def submit\_transaction(self, sender\_address, recipient\_address, value, signature):

"""

Add a transaction to transactions array if the signature verified

"""

transaction = OrderedDict({'sender\_address': sender\_address,

'recipient\_address': recipient\_address,

'value': value})

#Reward for mining a block

if sender\_address == MINING\_SENDER:

self.transactions.append(transaction)

return len(self.chain) + 1

#Manages transactions from wallet to another wallet

else:

transaction\_verification = self.verify\_transaction\_signature(sender\_address, signature, transaction)

if transaction\_verification:

self.transactions.append(transaction)

return len(self.chain) + 1

else:

return False

def create\_block(self, nonce, previous\_hash):

"""

Add a block of transactions to the blockchain

"""

block = {'block\_number': len(self.chain) + 1,

'timestamp': time(),

'transactions': self.transactions,

'nonce': nonce,

'previous\_hash': previous\_hash}

# Reset the current list of transactions

self.transactions = []

self.chain.append(block)

return block

def hash(self, block):

"""

Create a SHA-256 hash of a block

"""

# We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes

block\_string = json.dumps(block, sort\_keys=True).encode()

return hashlib.sha256(block\_string).hexdigest()

def proof\_of\_work(self):

"""

Proof of work algorithm

"""

last\_block = self.chain[-1]

last\_hash = self.hash(last\_block)

nonce = 0

while self.valid\_proof(self.transactions, last\_hash, nonce) is False:

nonce += 1

return nonce

def valid\_proof(self, transactions, last\_hash, nonce, difficulty=MINING\_DIFFICULTY):

"""

Check if a hash value satisfies the mining conditions. This function is used within the proof\_of\_work function.

"""

guess = (str(transactions)+str(last\_hash)+str(nonce)).encode()

guess\_hash = hashlib.sha256(guess).hexdigest()

return guess\_hash[:difficulty] == '0'\*difficulty

def valid\_chain(self, chain):

"""

check if a bockchain is valid

"""

last\_block = chain[0]

current\_index = 1

while current\_index < len(chain):

block = chain[current\_index]

#print(last\_block)

#print(block)

#print("\n-----------\n")

# Check that the hash of the block is correct

if block['previous\_hash'] != self.hash(last\_block):

return False

# Check that the Proof of Work is correct

#Delete the reward transaction

transactions = block['transactions'][:-1]

# Need to make sure that the dictionary is ordered. Otherwise we'll get a different hash

transaction\_elements = ['sender\_address', 'recipient\_address', 'value']

transactions = [OrderedDict((k, transaction[k]) for k in transaction\_elements) for transaction in transactions]

if not self.valid\_proof(transactions, block['previous\_hash'], block['nonce'], MINING\_DIFFICULTY):

return False

last\_block = block

current\_index += 1

return True

def resolve\_conflicts(self):

"""

Resolve conflicts between blockchain's nodes

by replacing our chain with the longest one in the network.

"""

neighbours = self.nodes

new\_chain = None

# We're only looking for chains longer than ours

max\_length = len(self.chain)

# Grab and verify the chains from all the nodes in our network

for node in neighbours:

print('http://' + node + '/chain')

response = requests.get('http://' + node + '/chain')

if response.status\_code == 200:

length = response.json()['length']

chain = response.json()['chain']

# Check if the length is longer and the chain is valid

if length > max\_length and self.valid\_chain(chain):

max\_length = length

new\_chain = chain

# Replace our chain if we discovered a new, valid chain longer than ours

if new\_chain:

self.chain = new\_chain

return True

return False

# Instantiate the Node

app = Flask(\_\_name\_\_)

CORS(app)

# Instantiate the Blockchain

blockchain = Blockchain()

@app.route('/')

def index():

return render\_template('./index.html')

@app.route('/configure')

def configure():

return render\_template('./configure.html')

@app.route('/transactions/new', methods=['POST'])

def new\_transaction():

values = request.form

# Check that the required fields are in the POST'ed data

required = ['sender\_address', 'recipient\_address', 'amount', 'signature']

if not all(k in values for k in required):

return 'Missing values', 400

# Create a new Transaction

transaction\_result = blockchain.submit\_transaction(values['sender\_address'], values['recipient\_address'], values['amount'], values['signature'])

if transaction\_result == False:

response = {'message': 'Invalid Transaction!'}

return jsonify(response), 406

else:

response = {'message': 'Transaction will be added to Block '+ str(transaction\_result)}

return jsonify(response), 201

@app.route('/transactions/get', methods=['GET'])

def get\_transactions():

#Get transactions from transactions pool

transactions = blockchain.transactions

response = {'transactions': transactions}

return jsonify(response), 200

@app.route('/chain', methods=['GET'])

def full\_chain():

response = {

'chain': blockchain.chain,

'length': len(blockchain.chain),

}

return jsonify(response), 200

@app.route('/mine', methods=['GET'])

def mine():

# We run the proof of work algorithm to get the next proof...

last\_block = blockchain.chain[-1]

nonce = blockchain.proof\_of\_work()

# We must receive a reward for finding the proof.

blockchain.submit\_transaction(sender\_address=MINING\_SENDER, recipient\_address=blockchain.node\_id, value=MINING\_REWARD, signature="")

# Forge the new Block by adding it to the chain

previous\_hash = blockchain.hash(last\_block)

block = blockchain.create\_block(nonce, previous\_hash)

response = {

'message': "New Block Forged",

'block\_number': block['block\_number'],

'transactions': block['transactions'],

'nonce': block['nonce'],

'previous\_hash': block['previous\_hash'],

}

return jsonify(response), 200

@app.route('/nodes/register', methods=['POST'])

def register\_nodes():

values = request.form

nodes = values.get('nodes').replace(" ", "").split(',')

if nodes is None:

return "Error: Please supply a valid list of nodes", 400

for node in nodes:

blockchain.register\_node(node)

response = {

'message': 'New nodes have been added',

'total\_nodes': [node for node in blockchain.nodes],

}

return jsonify(response), 201

@app.route('/nodes/resolve', methods=['GET'])

def consensus():

replaced = blockchain.resolve\_conflicts()

if replaced:

response = {

'message': 'Our chain was replaced',

'new\_chain': blockchain.chain

}

else:

response = {

'message': 'Our chain is authoritative',

'chain': blockchain.chain

}

return jsonify(response), 200

@app.route('/nodes/get', methods=['GET'])

def get\_nodes():

nodes = list(blockchain.nodes)

response = {'nodes': nodes}

return jsonify(response), 200

if \_\_name\_\_ == '\_\_main\_\_':

from argparse import ArgumentParser

parser = ArgumentParser()

parser.add\_argument('-p', '--port', default=5000, type=int, help='port to listen on')

args = parser.parse\_args()

port = args.port

app.run(host='127.0.0.1', port=port)

**CHAPTER 7**

**TESTING**

**7.1 TESTING TECHNIQUES**

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

**WHITE BOX TESTING**

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

* Flow graph notation
* Kilometric complexity
* Deriving test cases
* Graph matrices Control

**BLACK BOX TESTING**

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

**SOFTWARE TESTING STRATEGIES:**

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

Testing begins at the module level and works “outward” toward the integration of the entire computer based System.

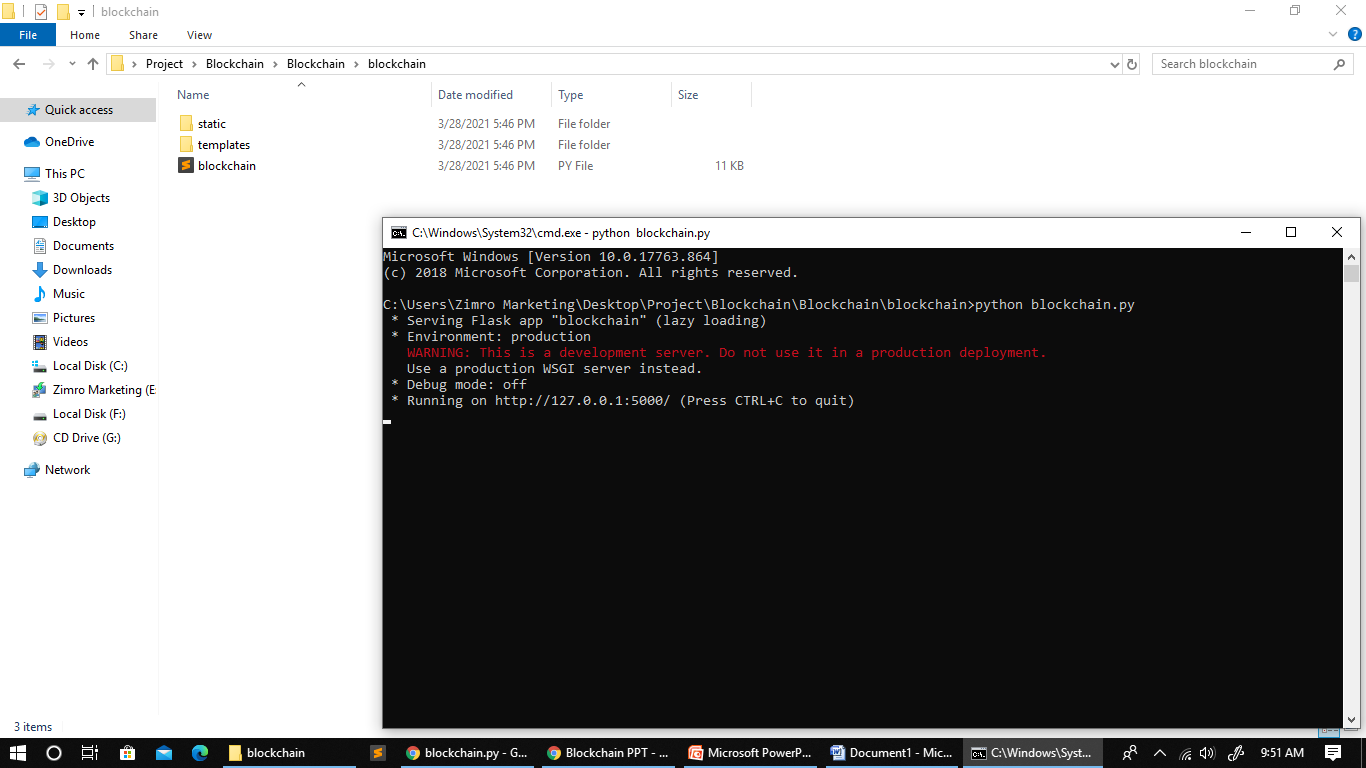
**CHAPTER 8**

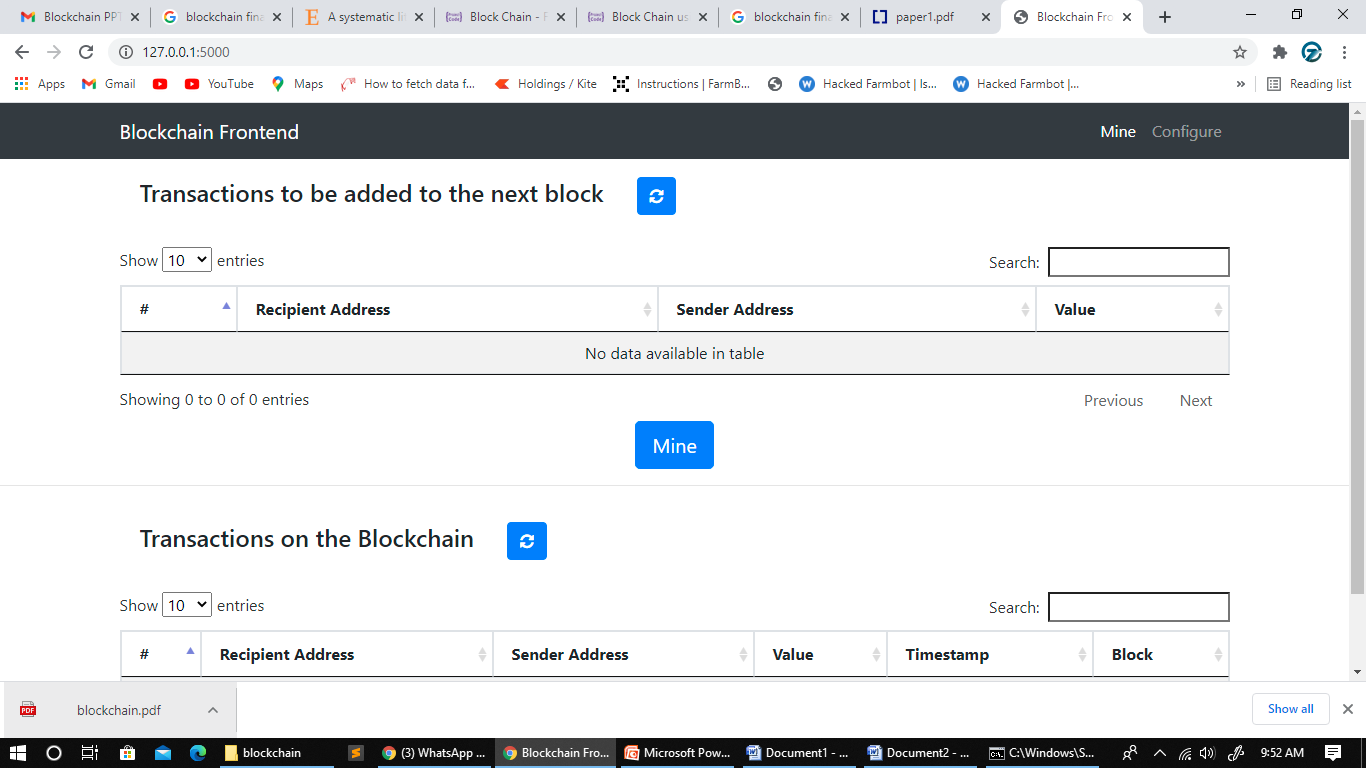
**CONCLUSION**

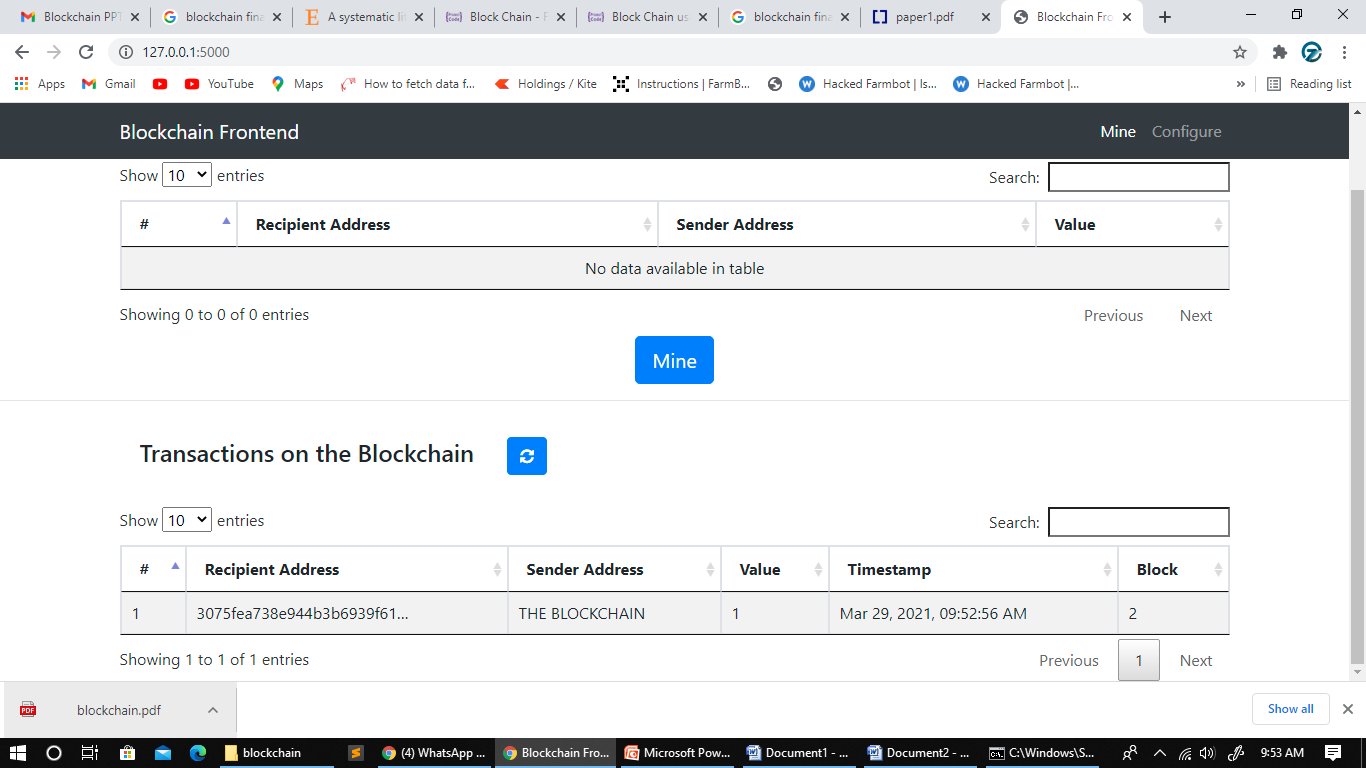
Crypto currencies such as Bitcoin, on the other hand, although popular, may take some time to be adopted as a primary medium of exchange, replacing fiat currencies. We have already seen how Bitcoin standard is seen in comparison to the Gold standard, without some of the disadvantages of the latter. To become the future currency of the world, it needs to get the nod of governments and policy makers. It also needs to make itself technologically much safer to vulnerable attacks. These are as much a problem of public perception as they are about the economic soundness. In the truly exciting times ahead, we only have to wait and watch how Bitcoins and Blockchain play out and where they live up to their potential and promises.

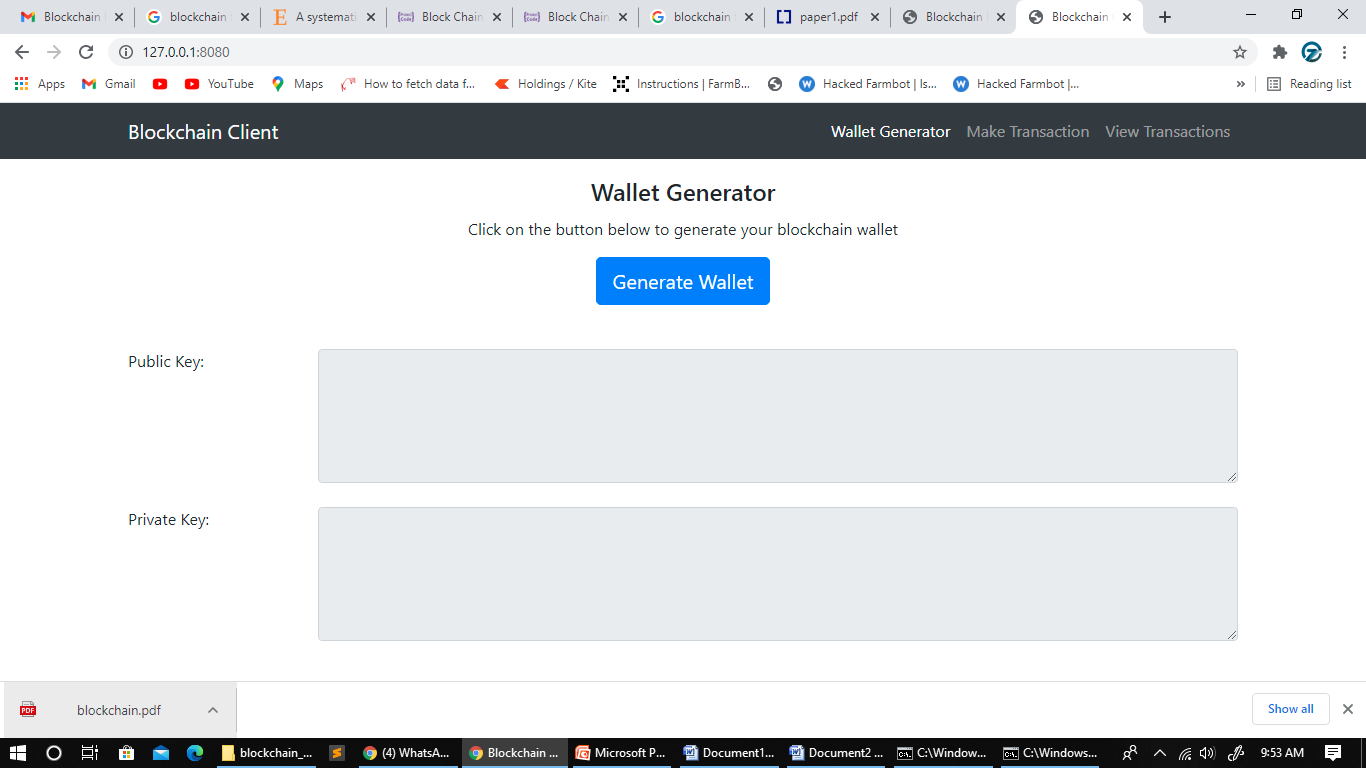
**CHAPTER 9**

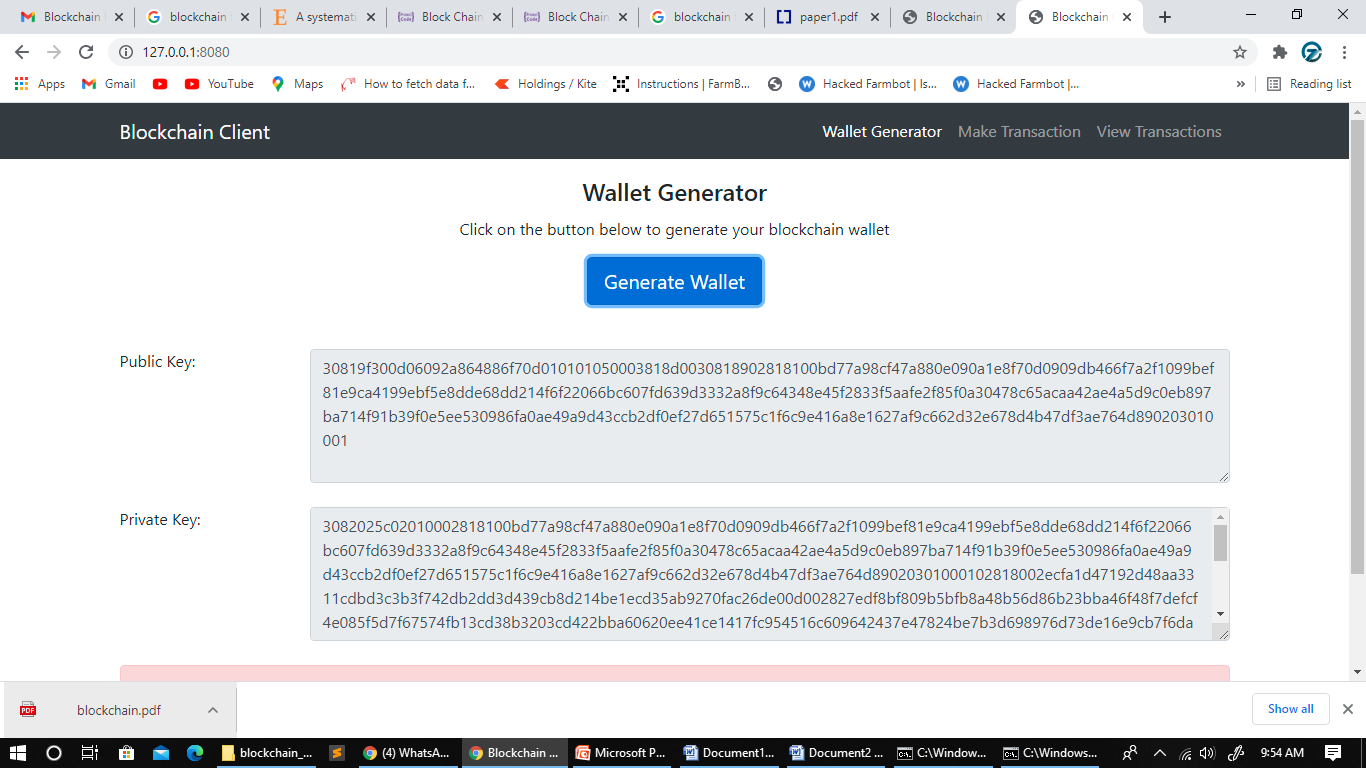
**SCREENSHOT**





****

****

****

**REFERENCE**

1. Kitchenham, B., Mendes, E., Travassos, G. H.: Cross versus within-company cost estimation studies: A systematic review. In: Software Engineering, IEEE Transactions on, vol. 33, no. 5, pp. 316–329 (2007).

2. Turhan, B., Mendes, E.: A Comparison of Cross-Versus Single-Company Effort Prediction Models for Web Projects. In: 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014. 285-292. 10.1109/SEAA.2014.41 (2014).

3. Idri, A., Amazal, F.,Abran, A.: Analogy-based software development effort estimation: A systematic mapping and review. In: Information and Software Technology. 58. 10.1016/j.infsof.2014.07.013 (2014).

4. Desharnais, J.M.: Analyse statistique de la productivitie des projects informatique a partie de la technique des point des function, PhD thesis, Univ. of Montreal (1989).

5.Desharnaisdataset,http://promise.site.uottawa.ca/SERepository/datasets/desharnais.arff, last accessed 06.05.2019.

6. ISBSG, International Software Benchmarking standards Group, <http://www.isbsg.org>.

7. Boehm, B.W., Software Engineering Economics, Prentice Hall PTR, New Jersey (1981).

8.COCOMOdataset,http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff, last accessed 06.05.2019.

9. QSM Software Project Database, https://www.qsm.com/resources/qsm-database, last accessed 23.06.2019.

10. QSM SLIM-Estimate, https://www.qsm.com/tools/slim-estimate, last accessed 23.06.2019.

11. QSM SLIM-Metrics, https://www.qsm.com/tools/slim-metrics, last accessed 23.06.2019.

12. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008), https://bitcoin.org/bitcoin.pdf, last accessed 06.05.2019.

13. Infographic: A Look at Blockchain Technology [Online Image]. Retreived 16.06.2019 from <https://burniegroup.com/infographic-a-look-at-blockchain-technology>

14. Ethereum. Blockchain App Platform. https://ethereum.org , last accessed 06.05.2019.

15. Hyperledger. Blockchain Technologies for business. https://www.hyperledger.org/wp-content/uploads/2018/08/HL\_Whitepaper\_IntroductiontoHyperledger.pdf, last accessed 17.06.2019.