Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Web Based Decentralized File Sharing Using Encryption

J COMPONENT PROJECT

CSE3502 – Information Security Management

by -

Team Members

Aryaman Jaisinghani	**19BCE0853**
Akshat Shukla	**19BCE0872**
Mohit Chotani	**19BCE0893**
Vaibhav	**19BCE0903**

Prof. Raja S P

Table of Contents

1. Abstract **3**

2. Introduction **4**

2.1 Problem Statement 5

3.2 Objective 5

3.3 Motivation 5

3.Techniques Identified **6**

4. Methodology **8**

5. Flowchart **10**

6. Performance Analysis **10**

7. Literature Survey **13**

8. Implementation **15**

8. Output **29**

9. Future Scope **34**

10. Conclusion **34**

11. References **35**

ABSTRACT

Communication over web, internet and through Social media are some of the popular ways through which the users communicate with each other via the use of messages and sharing other sorts of files and media.These media and files are sent from the source to the destination effortlessly. But then there is no guarantee that these shared media or the data shared are not intruded by the intruder in between the source and the destination. Encryption is one of the techniques for providing security for the data that has been sent by the sender. There are different algorithms used in order to achieve the encryption of the data. To address this issue we aim at developing a secure file transfer system. In this project we will be creating a decentralized system where files can be shared securely. By integrating the blockchain technology, we will make sure that the files shared on the network cannot be tampered or corrupted and the complete history of all the files shared will be stored giving users a sense of security.

The project will be implemented in real time with great ease and users can easily rely on this application after using it. This project will be developed using Python Language.

Keywords: Cryptography, Blockchain, IPFS, Encryption, Symmetric Key Encryption and Asymmetric Key Encryption, SHA 256, AES, Public-Key

INTRODUCTION

Security is one of the most vital concerns in any field, especially today in the current scenario when there is an extensive rise in the usage of the internet, people tend to require secure transmission and secure ways of having end to end encryption. Thus, an effective text encryption algorithm is needed for achieving an immense amount of privacy. In order to provide privacy, we propose a safe file sharing encryption system by utilizing decentralized blockchain architecture. By making use of Blockchain decentralized property as well as peer to peer network advantage files can be shared using a safe key and which can then be used in the AES algorithm to perform the encryption and decryption operations on the shared file.

Data Share is a service that allows users to send and receive sensitive messages, files, images and documents securely, with end-to-end encryption, using just their web browsers. Encryption takes place in the sender's web browser, and decryption takes place in the recipient's web browser, via a decentralized system. Only encrypted information is sent through servers. Unencrypted information and the keys used to encrypt/decrypt this information never leave the sender's web browser or the recipient's web browser. Senders may send information to a recipient through the service, without registering for the service, simply by knowing the recipient’s username on the service.

Data Share is an attempt to achieve true end-to-end encryption by implementing a decentralized platform for message encryption. This service is available to everyone with internet access and requires almost no technical background for its usage. In addition, senders are able to send files and messages to recipients without the need for senders to be set up on the service, simply by knowing a unique username, address or URL for a recipient and the shared private key. Its main aim is to enable secure messaging for everyone.

Secure: As the algorithm is used the encryption and decryption will be secure.

Confidentiality: This application can help in sending the confidential documents and folders without any attacks.

Attacks: This application can help in prevention of any types of attacks with great ease.

Ease: This application is easy to use.

PROBLEM STATEMENT

It is common for individuals to have a need to exchange information in a secure and confidential manner. For example, accountants, attorneys, and medical professionals frequently have a need to send and receive confidential information to and from their clients and the information shared required to be stored for both parties. Because standard SMTP email does not provide a means for encrypting information from the sender's end to the recipient's end of an email transmission, users have sought solutions to use instead of SMTP email, or in conjunction with SMTP email, for exchanging information confidentially and securely. Also, not all people are familiar with message encryption and its techniques. People who are familiar know that it isn't a trivial task and chances of wrong implementation is high.

OBJECTIVE

Our Objective is to build and implement a secure decentralized file sharing system, so called DATA SHARE which is a method of securely exchanging confidential data over a peer-to-peer network, and thus create a Platform for Secure Data Transfer.

MOTIVATION

The main motivation behind this project is to develop a secure file and data transfer technique that could provide end to end encryption establishing a peer to peer network. We can utilize the shared key to perform encryption and decoding of the text utilizing AES algorithm technique.

TECHNIQUES IDENTIFIED

In order to solve the above issue we looked into various techniques that could be used to solve the problem, and we came across some of the key exchange algorithms that are mentioned below**:**

SHA-256 Hashing Algorithm

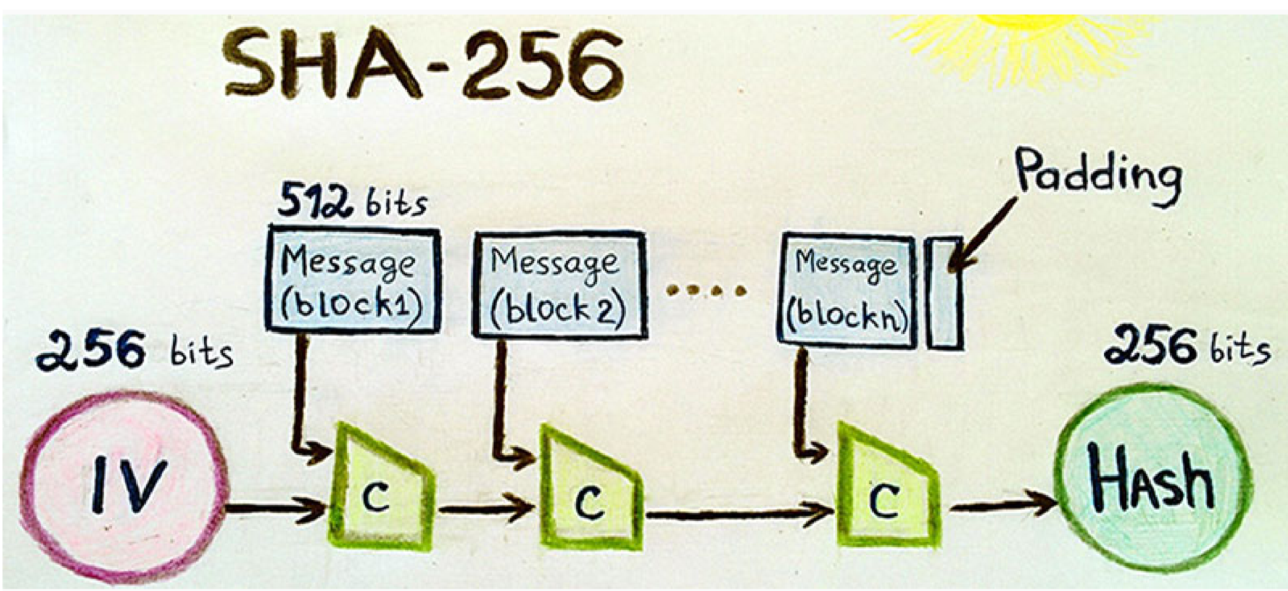
We will be using the SHA-256 algorithm to generate a unique hash of the entire block that is used by the corresponding blocks to form the chain (via the previous hashes). IPFS as well uses this algorithm to generate the hash of the shared file. The SHA-256 hashing algorithm is employed because of the following advantages:

One-way:- Once the hash is generated, we can't revert to the original data from the hash.

Deterministic:- For a particular input, the hash generated, always remains the same i.e. the same input always gives the same hash.
Quick computation of the hash.

Avalanche-effect:- Even a slight change in the input will bring about a large change in the final hash, making it untraceable

Withstand collisions:- There is a very rare chance that the hash generated for two different inputs will be the same. Think of it as a human fingerprint!



AES ENCRYPTION:

It is an encryption which requires a secret key to encrypt as well as decrypt the messages.In this encryption both the sender and receivers use the same secret key generated. It provides the safety by encrypting using AES algorithm during the exchange using the above technique mentioned.

Unlike its predecessor DES, AES does not rely on a Feistel network, but instead it is based on a substitution-permutation network. It can be implemented in hardware as well as software. The AES algorithm is more robust against hacking because it uses longer key sizes like 128, 192, and 256 bits. In order to encrypt the file data we will be using AES Encryption. It is a form of symmetric, cryptographic encryption that depends on a shared key between the sender and receiver to access any file (here the file key) . If we do not employ the AES encryption, any connected user to the blockchain can access the file hash and thus, the shared file using the hash, directly from the IPFS. Using the AES Encryption, we encrypt the file using the file key of the uploader. Thus, if any user tries to download the file directly from the IPFS, all they get is a non-readable file. Thus, only users with a valid file key can access the readable file contents, thereby enhancing the security of the blockchain and the file contents.As a result, it is a very safe and secure protocol.

METHODOLOGY

1) CREATING THE BLOCKCHAIN

i) Block Structure

In Data Share, a single block in a blockchain has the following structure:

The Block contains :

Block number: Simply displays the index number of the block. Block 0 refers to the genesis block.

Timestamp: This field indicates as to when the block was created and added to the blockchain.

Proof : Also called a nonce, it stands for "number only used once," which is a number added to a hashed—or encrypted—block in a blockchain that, when rehashed, meets the difficulty level restrictions i.e by varying the proof we can vary the hash generated so that a new block can be created.

Previous hash: This field represents the hash of the previous block. The hash of the entire block is generated using the SHA-256 hashing algorithm. This field creates a chain of blocks and is the main element behind blockchain architecture's security.

Sender: The person who uploads the file enters his identity proof or name when he uploads the file.

Receiver: Displays who the intended receiver shall be.

Hash of the file shared: The uploaded file is first encrypted using the AES encryption mechanism and subsequently using the SHA-256 hashing algorithm when it is uploaded to ipfs. The hash, then received from the IPFS (Interplanetary File System)after the encryption is the hash of the shared file which is added to the block.

ii) Creating Peer to Peer Network

In order to create a peer to peer network (p2p) for the blockchain to function, all the connected nodes must be in the same network. Only those users who are connected to the blockchain's p2p network will have access to the blockchain's data. This p2p network will be created using Socket Programming. Using socket programming, the list of connected nodes gets updated as soon as a new user gets connected or disconnected to the network and the updated list is broadcasted to the whole p2p network. As soon as all the connected nodes get the updated list of the nodes in the network, the consensus protocol works smoothly whenever a new block is added or the blockchain gets updated. This is how peer to peer networks will work.

iii) File Key

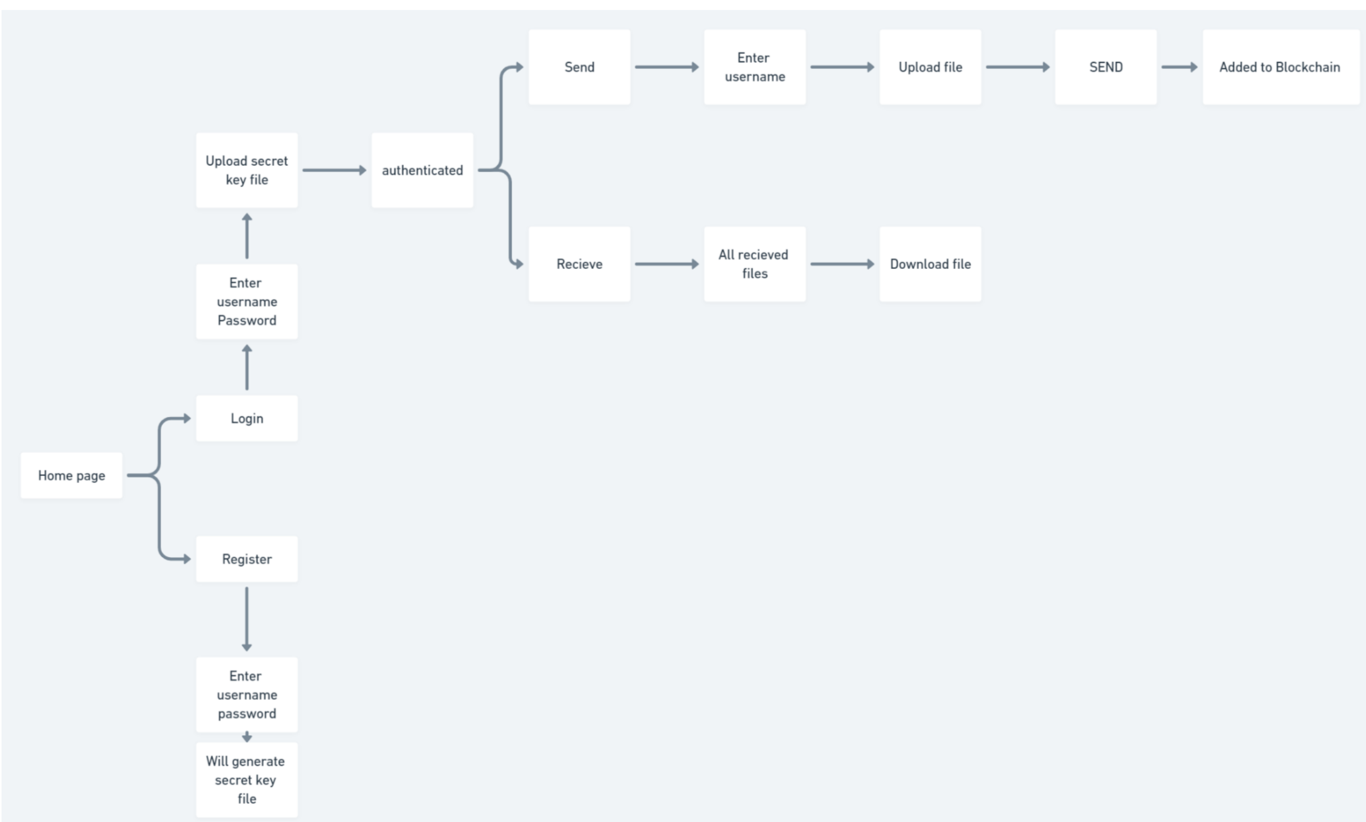
There will be a unique key/password shared between the sender and the receiver of the shared file to increase the security of the file(s) on the blockchain network.
The file key will be further used to encrypt the file using AES encryption before uploading it to the IPFS network. The uploader will have to share the key only with the intended receiver(s) so he/she can download the file. The type of files that can be uploaded are .pdf , .png , .jpeg and .txt. As of now the size of the file that can be uploaded to the network is limited to 16 Megabytes.

Further in order to download the file the receiver who will be having the valid file key shared by the sender will be only able to download the shared file from the blockchain to his/her local computer. The file key here is used to decrypt: the AES encrypted file downloaded from the IPFS network so that the file can be interpretable. Make sure you enter the correct file key and hash for a successful download.

2) INTEGRATING WITH IPFS

Our blockchain relies on IPFS for keeping it lightweight and scalable. If the files were stored directly on the blockchain, it would render the blockchain very heavy and inefficient. Combining IPFS and blockchain, we get to access the IPFS's power of decentralized storage and enhance the blockchain's security and accessibility. Instead of storing the file directly on the blockchain, we store the files on the IPFS network while the blockchain stores only the file' hash. Each file will have a unique hash as IPFS employs the SHA-256 hashing algorithm. Thus, the file is stored in a secure decentralized network and is easily accessible through the blockchain. The file can be retrieved using its generated hash easily. Hence IPFS eliminates the bottleneck of storing entire files on the blockchain.

FLOWCHART:



PERFORMANCE ANALYSIS

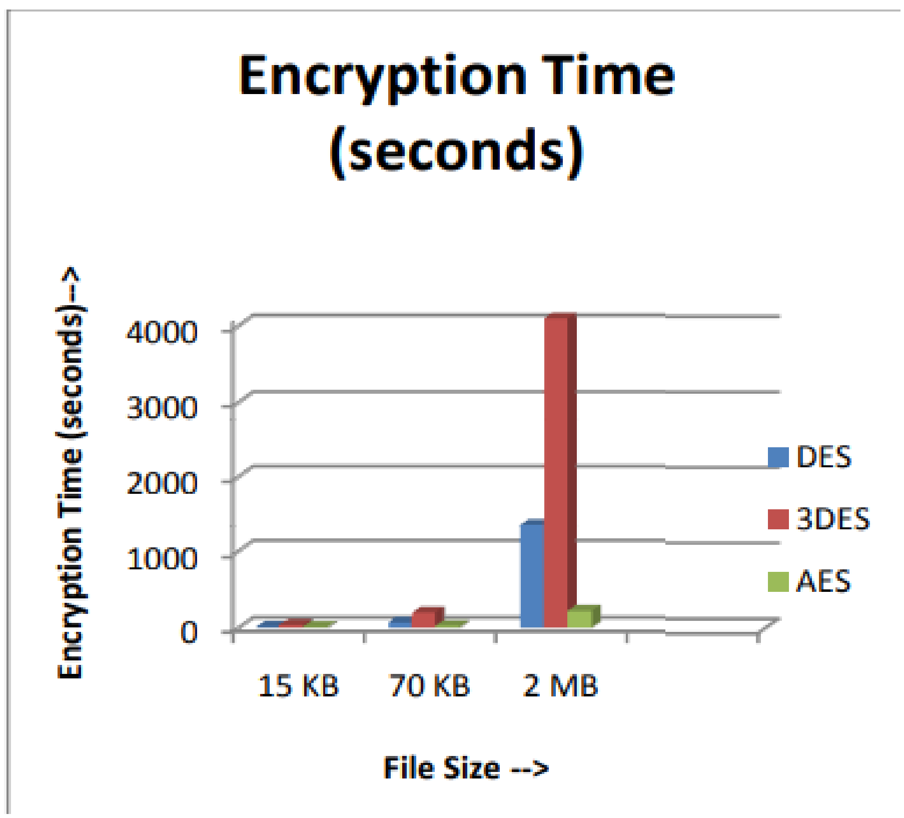
The various performance metrics that are evaluated include

Encryption Time - The encryption time is the time that an encryption algorithm takes to produce a cipher text from a plaintext. It is measured in seconds.

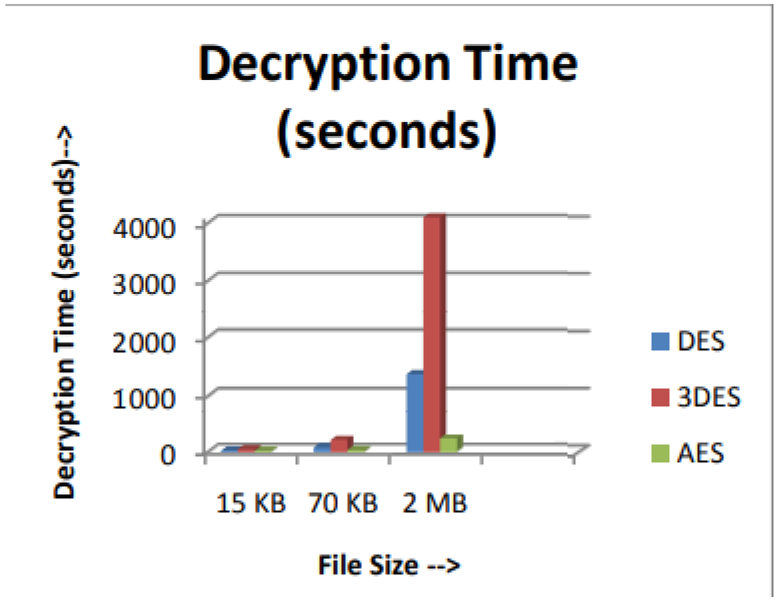
Decryption Time - The decryption time is the time that a decryption algorithm takes to produce a plaintext from a cipher text. It is measured in seconds.

Throughput - The throughput of an encryption or decryption scheme defines the speed of encryption. The throughput of the encryption can be calculated as in equation
Throughput = Tp (Kilobytes) / Et (Second) where Tp: Total plain text (Kilobytes) Et: Encryption time (second)

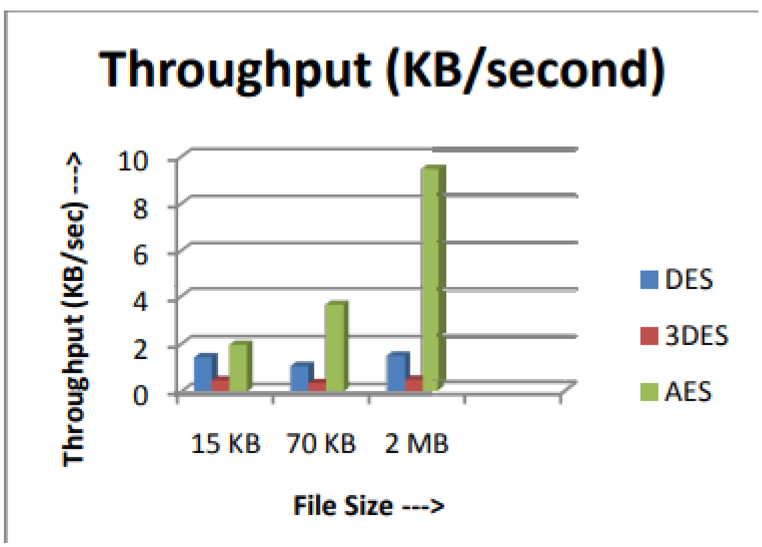
Memory Utilization - The Memory Utilization defines how much memory is being consumed by the process while doing encryption or decryption. It is measured in Kilobytes (KB).



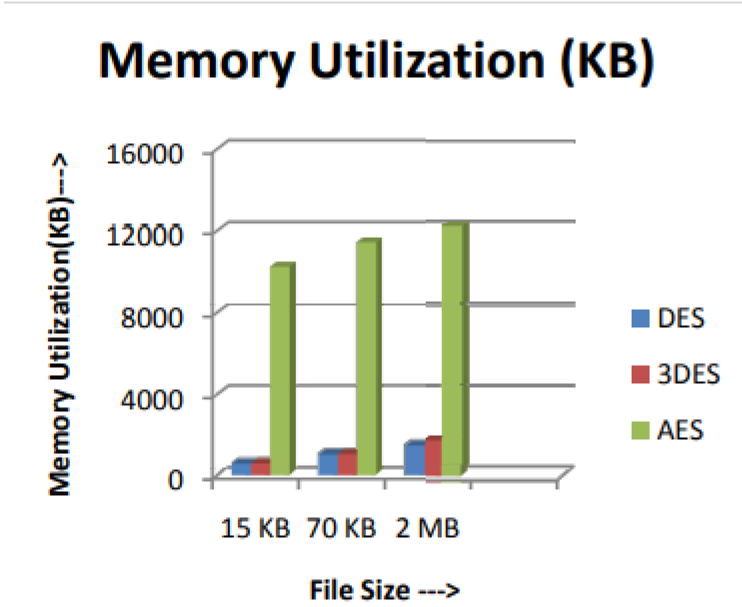
Comparison of Encryption Time for various algorithms



Comparison of Decryption Time for various algorithms



Comparison of Throughput for various algorithms



Comparison of Throughput for various algorithms

Analysis: It can be analyzed that for text files of different sizes, AES algorithm takes less encryption and decryption time compared to other algorithms. Throughput varies inversely to the encryption or decryption time. Thus it is more in case of the AES algorithm. Memory utilization of DES is less than the other algorithms.

LITERATURE SURVEY :

1. Performance Study of Enhanced SHA-256 Algorithm.

Authors: Gowthaman A, M Sumathi
International Journal of Applied Engineering Research

Description:
This paper looks into optimization techniques in new VLSI architecture for the SHA-256 hash functions. It combines different techniques namely rescheduling of Carry Select Adder and Non linear Pseudo code random generator for improving the functions in an inner loop of hashing algorithm.
The proposed system can be suitable to achieve less area utilization, improved security and fastest data throughput and higher performing frequency.

2. File Encryption, Decryption Using AES Algorithm in Android Phone.

Authors: Suchita Tayde , Asst. Prof. Seema Siledar
International Journal of Advanced Research in Computer Science and Software Engineering

Description
AES can be implemented on various platforms, especially in small devices like android mobile phones. In this paper the Advanced Encryption Standard algorithm has been used to overcome low size and slow speed as compared to DES, 3DES, Blowfish, RSA algorithms. This paper shows successful implementation of file and image encryption as well as decryption. The user experiences faster file encryption and decryption. Showing us that AES encryption and decryption algorithms run faster in android phone as compared to other algorithms. This application used here guarantees secure end to end transfer of data without any corruption of data.

3. Advanced Encryption Standard (AES) Algorithm to Encrypt and Decrypt Data.

Authors: Ako Muhamad Abdullah
Cryptography and Network Security, 16, 1-11

Description
Internet communication is playing an important role in transferring large amounts of data in various fields.Cryptography is one of the most significant and popular techniques to secure the data from attackers by using two vital processes that is Encryption and Decryption.

This paper gives an overview of the AES algorithm and explains several crucial features of this algorithm in detail. Paper revolves around the comparison of the AES algorithm with different cryptographic techniques like DES, 3DES, Blow fish etc. According to the result of the research done by the author, AES performs better than DES, 3DES.

4. High Securing Cover-File of Hidden Data Using Statistical Technique and AES Encryption Algorithm.

Authors: A. A. Zaidan, Anas Majeed, and B. B. Zaidan
World Academy of Science Engineering and Technology (WASET), 54, 468-479.

Description
The paper proposes that the system can be summarized as hiding the password or any information beyond the end of an executable file so there is no function or routine in the operating system to extract it.

The proposed system in the paper aims to hide information (data file) in any execution file (EXE) and to detect the hidden file and implementation of a steganography system is done which embeds information in an execution file. (EXE) files have been investigated. The system tries to find a solution to the size of the cover file and make it undetectable by anti-virus software. The system includes two main functions; first is the hiding of the information in a Portable Executable File (EXE), through the execution of four process (specify the cover file, specify the information file, encryption of the information, and hiding the information) and the second function is the extraction of the hiding information through three process (specify the steno file, extract the information, and decryption of the information).

5. Performance evaluation of several symmetric key algorithms

Authors: D. A. F. Saraiva, V. R. Q. Leithardt, D. de Paula, A. S. Mendes, G. V. González, P. Crocker(2019)

Description

This paper compares the performance of several symmetric key algorithms, among them AES, RC6, and Twofish, all in GCM mode. All supported key sizes were tested (128, 192, and 256 bits). However, only encryption and decryption times were measured. The tests were made in a laptop with an Intel CPU and in an emulated ARMv7-a CPU. The emulation was run on the same laptop. We verified that AES had the best execution times for the Intel device due to hardware acceleration, but in the emulated ARMv7-a CPU, RC6 had the best results.

IMPLEMENTATION:

CODE:

Blockchain.py

```
# import datetime

import time

import hashlib

import json

from flask import Flask, jsonify, request

import requests

from urllib.parse import urlparse




# Building a Blockchain



class Blockchain:



   def __init__(self):

       self.chain = []



               self.create_block(proof=1,  previous_hash='0',  sender='N.A',
receiver='N.A', file_hash='N.A')  ##########

       self.nodes = set()

       self.nodes.add("127.0.0.1:5111")
```

```
    def create_block(self, proof, previous_hash, sender, receiver, file_hash):

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

                 'timestamp': str(time.strftime("%d %B %Y , %I:%M:%S %p",
time.localtime())),


                 'proof': proof,

                 'previous_hash': previous_hash,

                 'sender': sender,  #########

                 'receiver': receiver,  #########

                 'shared_files': file_hash}


        self.chain.append(block)

        return block


    def get_previous_block(self):

        return self.chain[-1]


    def proof_of_work(self, previous_proof):

        new_proof = 1

        check_proof = False

        while check_proof is False:

                 hash_operation = hashlib.sha256(str(new_proof ** 2 -
previous_proof ** 2).encode()).hexdigest()
```

```
            if hash_operation[:4] == '0000':
                check_proof = True
            else:
                new_proof += 1
        return new_proof

    def hash(self, block):
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def is_chain_valid(self, chain):
        previous_block = chain[0]
        block_index = 1
        while block_index < len(chain):
            block = chain[block_index]
            if block['previous_hash'] != self.hash(previous_block):
                return False
            previous_proof = previous_block['proof']
            proof = block['proof']
            hash_operation = hashlib.sha256(str(proof ** 2 - previous_proof **
2).encode()).hexdigest()
            if hash_operation[:4] != '0000':
                return False
            previous_block = block
```

```
            block_index += 1
        return True

    def add_file(self, sender, receiver, file_hash):



        previous_block = self.get_previous_block()
        index = previous_block['index'] + 1

        previous_proof = previous_block['proof']
        proof = self.proof_of_work(previous_proof)
        previous_hash = self.hash(previous_block)
        self.create_block(proof, previous_hash, sender, receiver, file_hash)
        return index

    def replace_chain(self):
        network = self.nodes
        longest_chain = None
        max_length = len(self.chain)
        for node in network:
            response = requests.get('http://{}/get_chain'.format(node))

            if response.status_code == 200:
```

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

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

            if length > max_length and self.is_chain_valid(chain):

                max_length = length

                longest_chain = chain

    if longest_chain:

        self.chain = longest_chain

        return True

    return False

```

Server.py

```
import os

import urllib.request

import ipfshttpclient

from main_server.my_constants import app

import pyAesCrypt

from flask import Flask, flash, request, redirect, render_template, url_for,
jsonify

from flask_socketio import SocketIO, send, emit

from werkzeug.utils import secure_filename

import socket

import pickle

```

```
from main_server.blockchain import Blockchain



socketio = SocketIO(app)

blockchain = Blockchain()


def allowed_file(filename):

        return  '.'  in  filename  and  filename.rsplit('.',  1)[1].lower()  in
app.config['ALLOWED_EXTENSIONS']


def append_file_extension(uploaded_file, file_path):

    file_extension = uploaded_file.filename.rsplit('.', 1)[1].lower()

    user_file = open(file_path, 'a')

    user_file.write('\n' + file_extension)

    user_file.close()


def decrypt_file(file_path, file_key):

    encrypted_file = file_path + ".aes"

    os.rename(file_path, encrypted_file)

            pyAesCrypt.decryptFile(encrypted_file,   file_path,     file_key,
app.config['BUFFER_SIZE'])


def encrypt_file(file_path, file_key):
```

```
    pyAesCrypt.encryptFile(file_path, file_path + ".aes", file_key,
app.config['BUFFER_SIZE'])


def hash_user_file(user_file, file_key):
    encrypt_file(user_file, file_key)
    encrypted_file_path = user_file + ".aes"
    client = ipfshttpclient.connect('/dns/ipfs.infura.io/tcp/5001/https')
    response = client.add(encrypted_file_path)
    file_hash = response['Hash']
    return file_hash


def retrieve_from_hash(file_hash, file_key):
    client = ipfshttpclient.connect('/dns/ipfs.infura.io/tcp/5001/https')
    file_content = client.cat(file_hash)
    file_path = os.path.join(app.config['DOWNLOAD_FOLDER'], file_hash)
    user_file = open(file_path, 'ab+')
    user_file.write(file_content)
    user_file.close()
    decrypt_file(file_path, file_key)
    with open(file_path, 'rb') as f:
        lines = f.read().splitlines()
        last_line = lines[-1]
    user_file.close()
    file_extension = last_line
```

```
    saved_file = file_path + '.' + file_extension.decode()

    os.rename(file_path, saved_file)

    print(saved_file)

    return saved_file


@app.route('/')

def index():

    return render_template('index.html')


@app.route('/home')

def home():

    return render_template('index.html')


@app.route('/upload')

def upload():

    return render_template('upload.html', message ="Welcome!")


@app.route('/download')

def download():

    return render_template('download.html', message ="Welcome!")


@app.route('/connect_blockchain')

def connect_blockchain():
```

```
    is_chain_replaced = blockchain.replace_chain()
             return   render_template('connect_blockchain.html',   chain   =
blockchain.chain, nodes = len(blockchain.nodes))


@app.errorhandler(413)
def entity_too_large(e):
      return  render_template('upload.html',  message  ="Requested  Entity  Too
Large!")


@app.route('/add_file', methods=['POST'])
def add_file():

    is_chain_replaced = blockchain.replace_chain()

    if is_chain_replaced:
          print('The nodes had different chains so the chain was replaced by
the longest one.')
    else:
        print('All good. The chain is the largest one.')

    if request.method == 'POST':
        error_flag = True
        if 'file' not in request.files:
            message = 'No file part'
```

```
        else:
            user_file = request.files['file']
            if user_file.filename == '':
                message = 'No file selected for uploading'

            if user_file and allowed_file(user_file.filename):
                error_flag = False
                filename = secure_filename(user_file.filename)
                file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
                user_file.save(file_path)
                append_file_extension(user_file, file_path)
                sender = request.form['sender_name']
                receiver = request.form['receiver_name']
                file_key = request.form['file_key']

                try:
                    hashed_output1 = hash_user_file(file_path, file_key)
                    index = blockchain.add_file(sender, receiver, hashed_output1)
                except Exception as err:
                    message = str(err)
                    error_flag = True
                    if "ConnectionError:" in message:
```

```
                        message = "Gateway down or bad Internet!"


            else:

                error_flag = True

                  message = 'Allowed file types are txt, pdf, png, jpg, jpeg,
gif'


        if error_flag == True:

            return render_template('upload.html', message = message)

        else:

             return render_template('upload.html', message ="File succesfully
uploaded")


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

def retrieve_file():


    is_chain_replaced = blockchain.replace_chain()


    if is_chain_replaced:

          print('The nodes had different chains so the chain was replaced by
the longest one.')

    else:

        print('All good. The chain is the largest one.')
```

```
if request.method == 'POST':

    error_flag = True

    if request.form['file_hash'] == '':
        message = 'No file hash entered.'
    elif request.form['file_key'] == '':
        message = 'No file key entered.'
    else:
        error_flag = False
        file_key = request.form['file_key']
        file_hash = request.form['file_hash']
        try:
            file_path = retrieve_from_hash(file_hash, file_key)
        except Exception as err:
            message = str(err)
            error_flag = True
            if "ConnectionError:" in message:
                message = "Gateway down or bad Internet!"

    if error_flag == True:
        return render_template('download.html', message = message)
    else:
```

```
                        return  render_template('download.html',  message  ="File
successfully downloaded")


# Getting the full Blockchain

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

def get_chain():

    response = {'chain': blockchain.chain,

                'length': len(blockchain.chain)}

    return jsonify(response), 200


@socketio.on('connect')

def handle_connect():

    print('Client connected')

    print(request)


@socketio.on('add_client_node')

def handle_node(client_node):

    print(client_node)

    blockchain.nodes.add(client_node['node_address'])

    emit('my_response', {'data': pickle.dumps(blockchain.nodes)}, broadcast =
True)


@socketio.on('remove_client_node')

def handle_node(client_node):
```

```
    print(client_node)

    blockchain.nodes.remove(client_node['node_address'])

    emit('my_response', {'data': pickle.dumps(blockchain.nodes)}, broadcast =
True)


@socketio.on('disconnect')

def handle_disconnect():

    print('Client disconnected')

    print(request)


if __name__ == '__main__':

    socketio.run(app, host = '127.0.0.1', port= 5111)
```

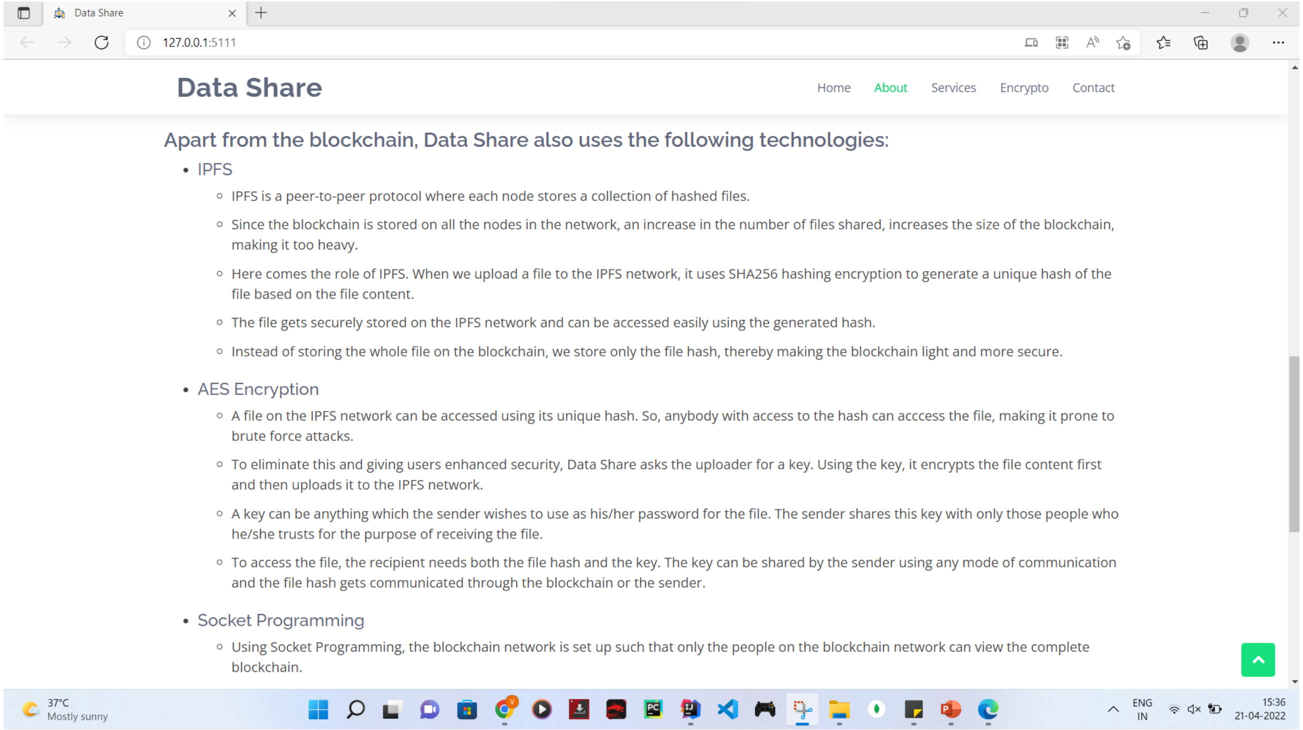
My_constants.py

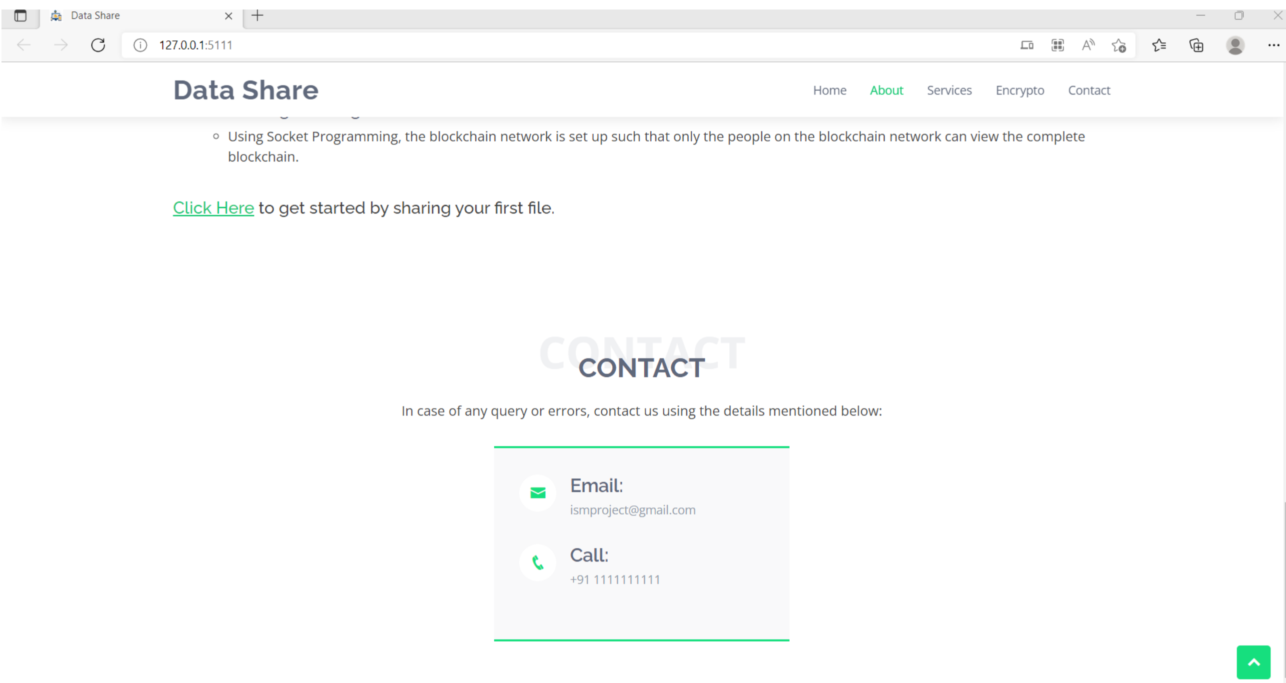
```
from flask import Flask

UPLOAD_FOLDER                                                                  =
'D:/Blockchain-based-Decentralized-File-Sharing-System-using-IPFS/main_server
/uploads'

DOWNLOAD_FOLDER                                                                =
'D:/Blockchain-based-Decentralized-File-Sharing-System-using-IPFS/main_server
/downloads'

app = Flask(__name__)

app.secret_key = "secret key"

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

app.config['DOWNLOAD_FOLDER'] = DOWNLOAD_FOLDER
```

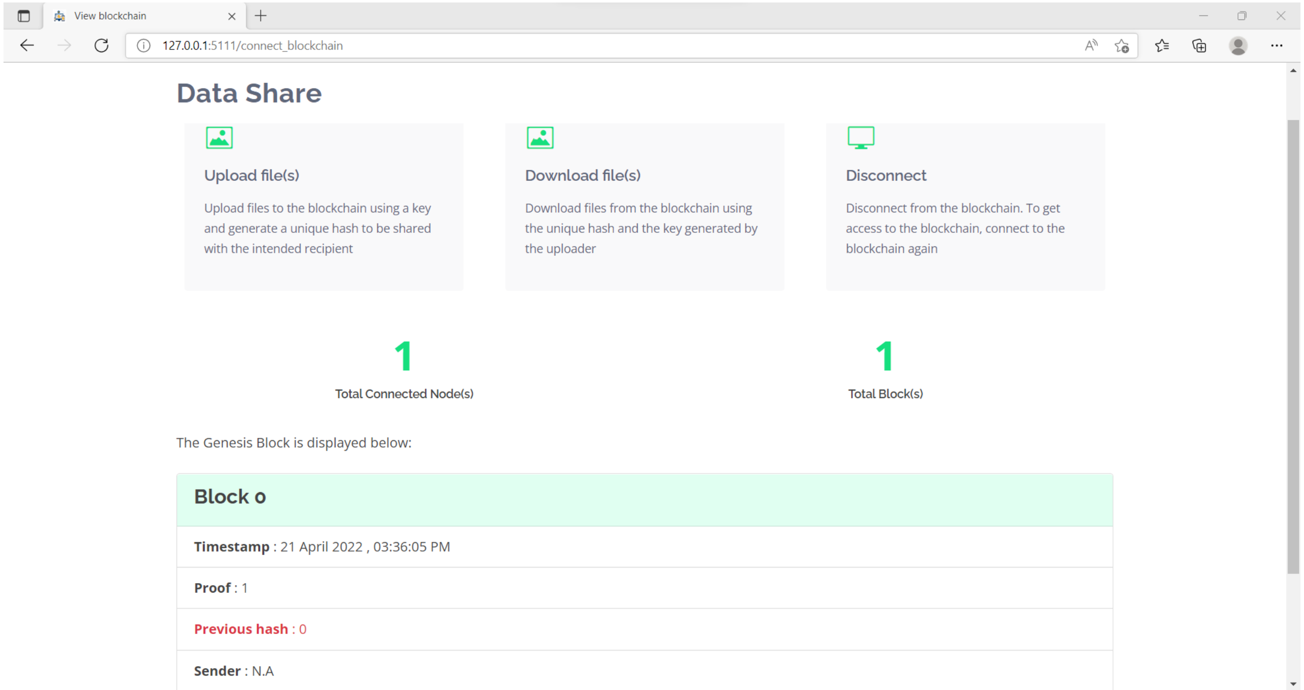
```
app.config['ALLOWED_EXTENSIONS'] = {'txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'}

app.config['BUFFER_SIZE'] = 64 * 1024

app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024
```

OUTPUT:

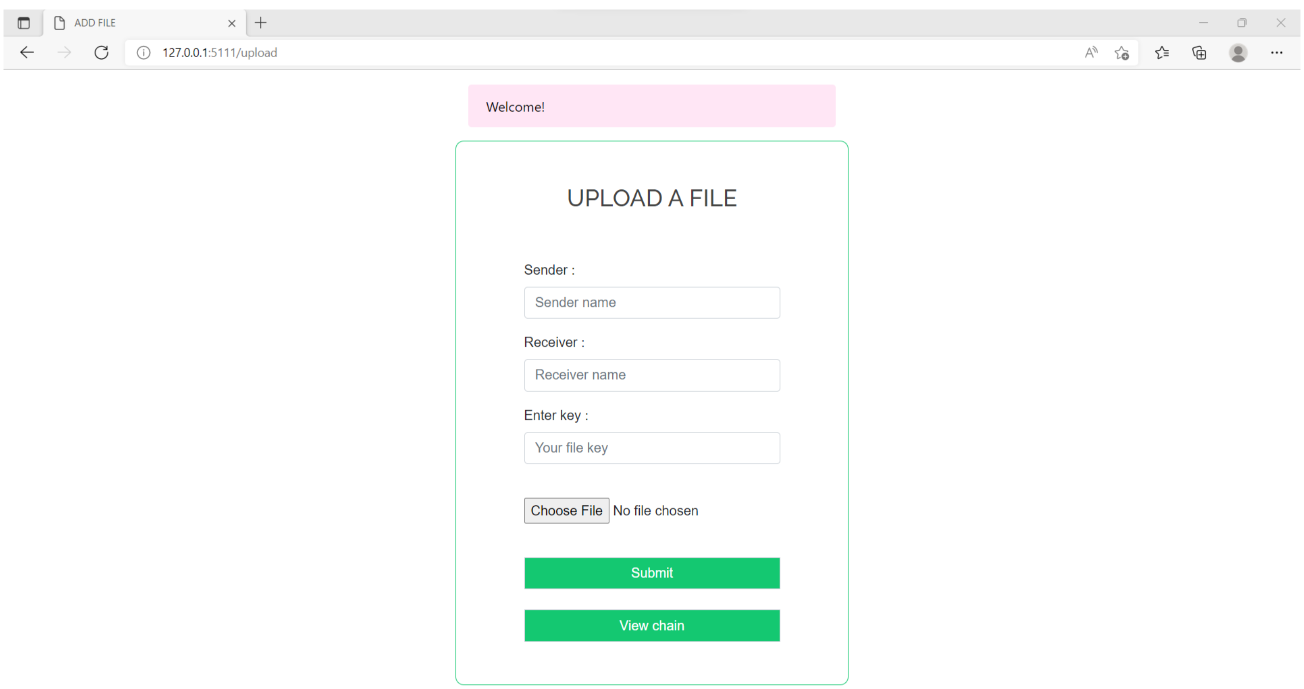
Index Page


Data Share
127.0.0.1:5111
Data Share
Home
About
Services
Encrypto
Contact
Apart from the blockchain, Data Share also uses the following technologies:
IPFS
IPFS is a peer-to-peer protocol where each node stores a collection of hashed files.
Since the blockchain is stored on all the nodes in the network, an increase in the number of files shared, increases the size of the blockchain, making it too heavy.
Here comes the role of IPFS. When we upload a file to the IPFS network, it uses SHA256 hashing encryption to generate a unique hash of the file based on the file content.
The file gets securely stored on the IPFS network and can be accessed easily using the generated hash.
Instead of storing the whole file on the blockchain, we store only the file hash, thereby making the blockchain light and more secure.
AES Encryption
A file on the IPFS network can be accessed using its unique hash. So, anybody with access to the hash can acccess the file, making it prone to brute force attacks.
To eliminate this and giving users enhanced security, Data Share asks the uploader for a key. Using the key, it encrypts the file content first and then uploads it to the IPFS network.
A key can be anything which the sender wishes to use as his/her password for the file. The sender shares this key with only those people who he/she trusts for the purpose of receiving the file.
To access the file, the recipient needs both the file hash and the key. The key can be shared by the sender using any mode of communication and the file hash gets communicated through the blockchain or the sender.
Socket Programming
Using Socket Programming, the blockchain network is set up such that only the people on the blockchain network can view the complete blockchain.
37°C
Mostly sunny
ENG
IN
15:36
21-04-2022

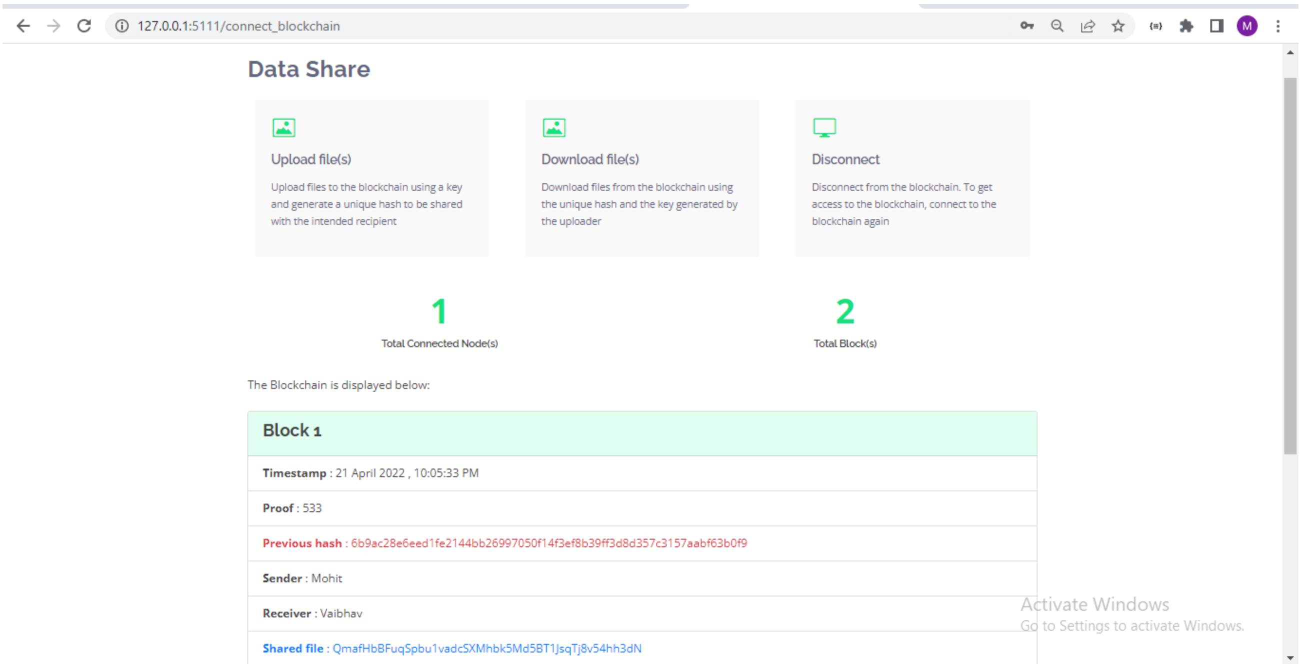

Data Share
127.0.0.1:5111
Data Share
Home
About
Services
Encrypto
Contact
Using Socket Programming, the blockchain network is set up such that only the people on the blockchain network can view the complete blockchain.
Click Here to get started by sharing your first file.
CONTACT
In case of any query or errors, contact us using the details mentioned below:
Email:
ismproject@gmail.com
Call:
+91 1111111111



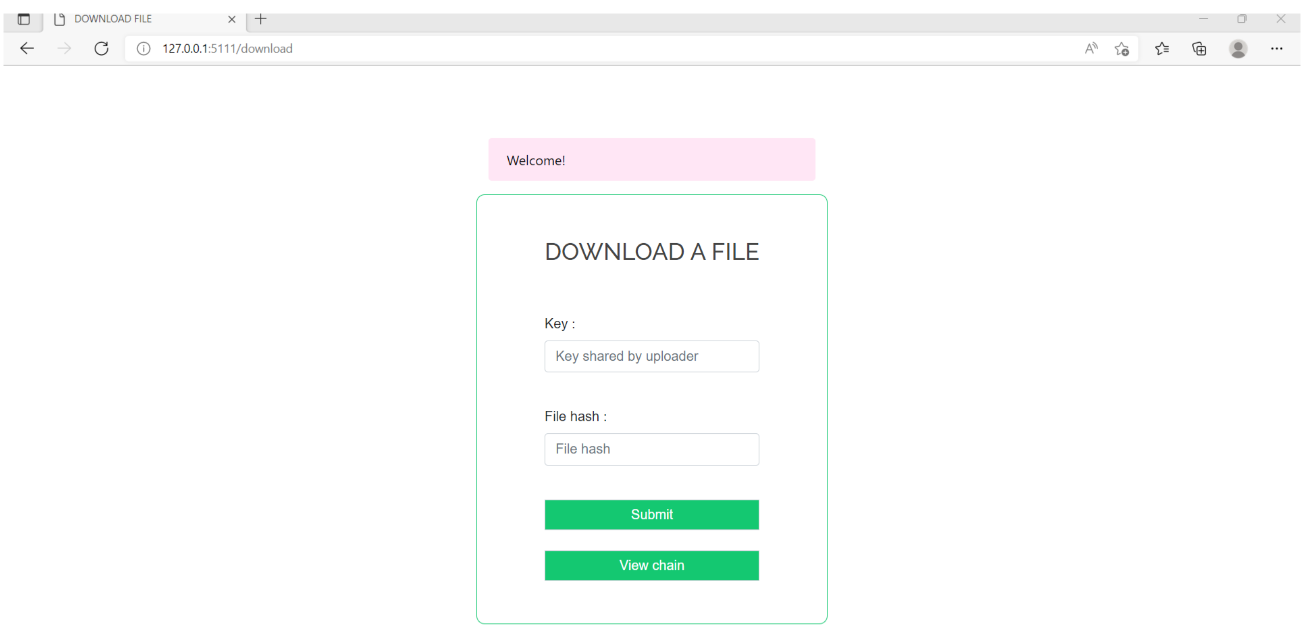
Connecting to Blockchain



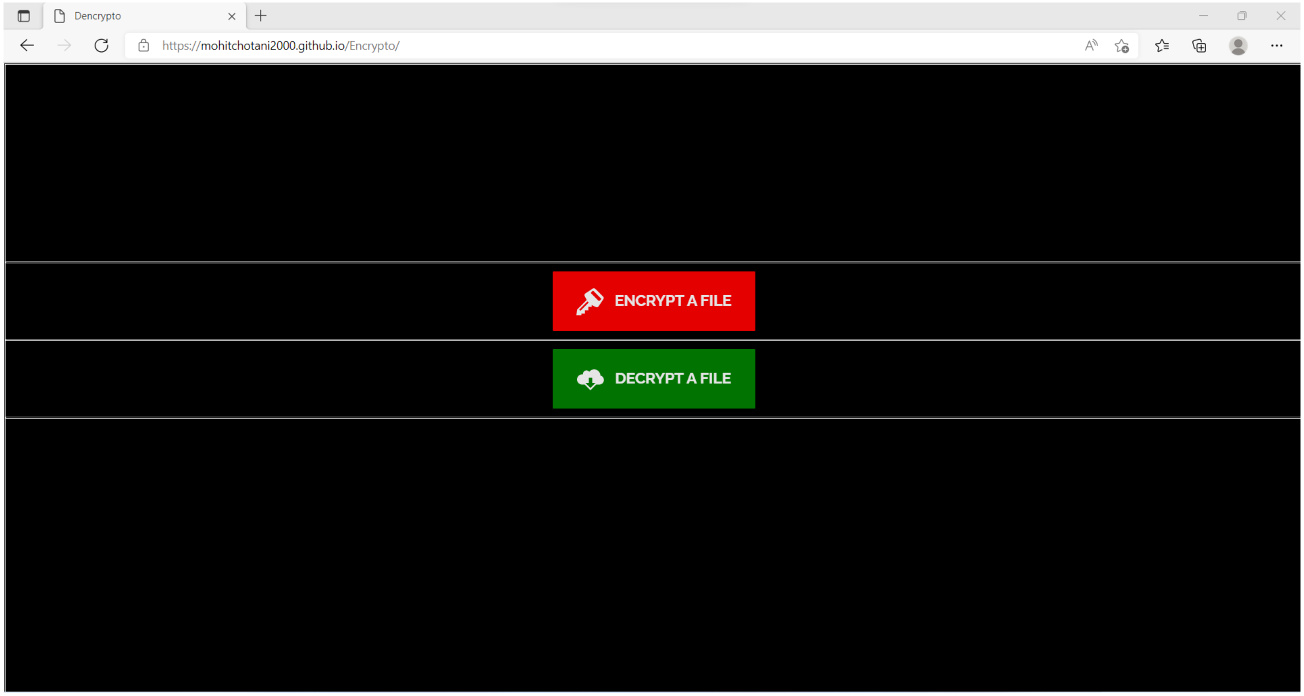
Uploading a File



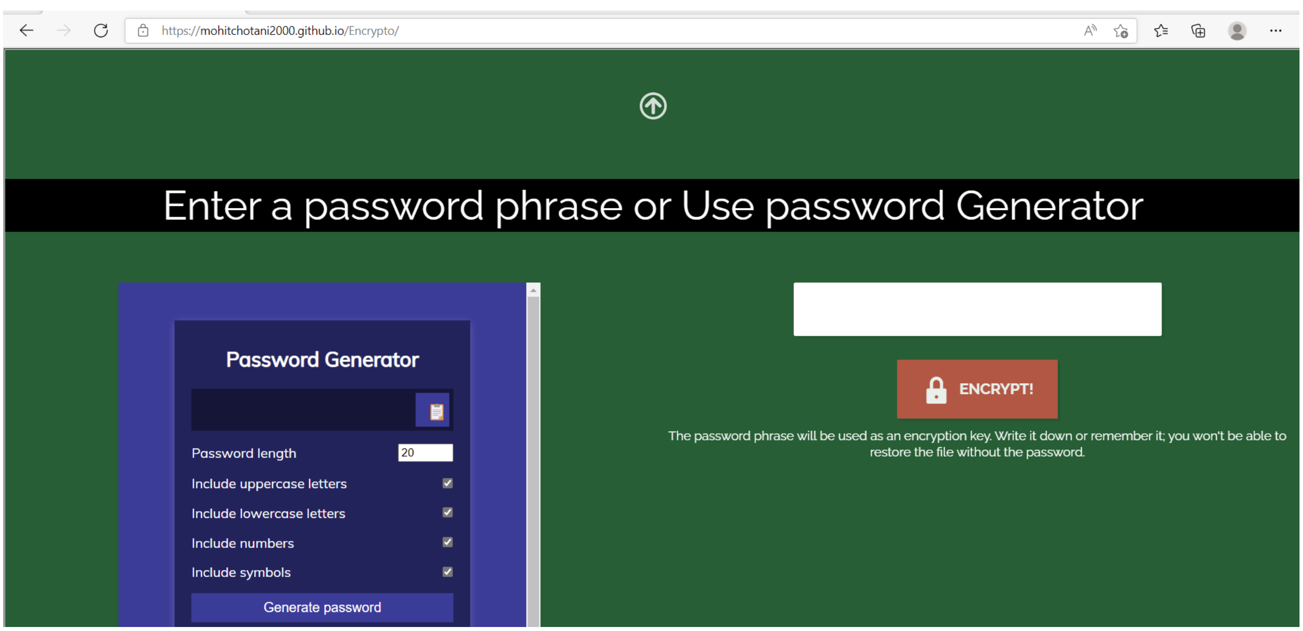
File uploaded and new Block Created



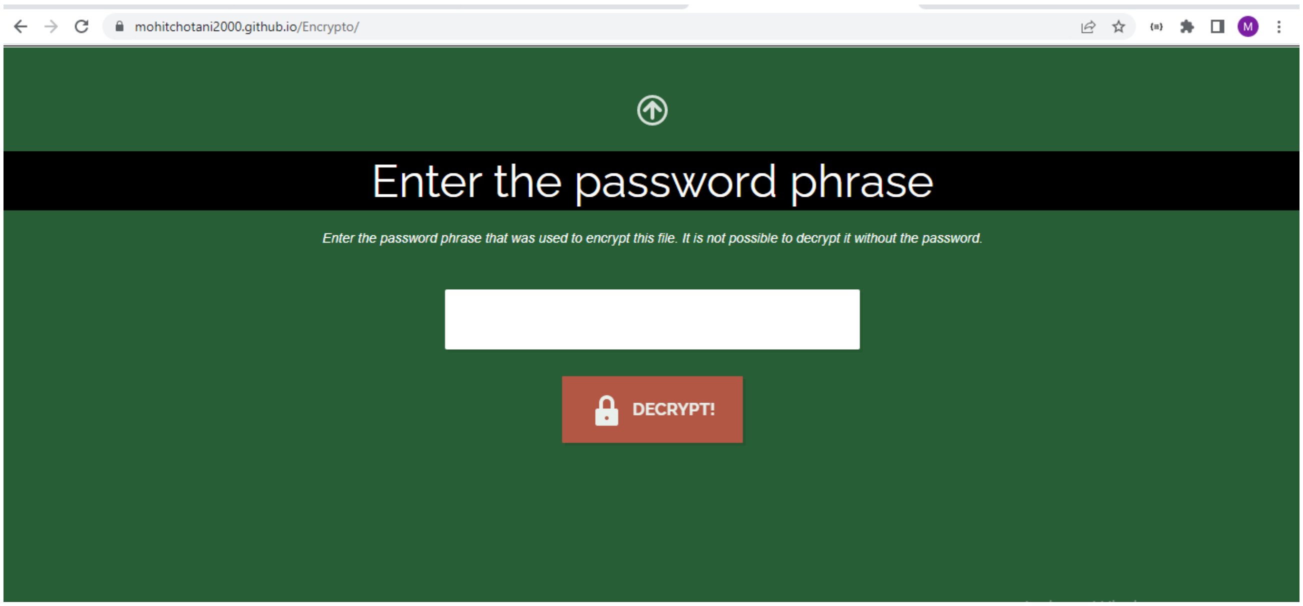
Downloading a File



Encrypto Landing Page



Encrypting a File using Key Generator



Decrypting the encrypted File using the key

CONCLUSION:

Blockchain can be set up to operate in a variety of ways, using different mechanisms to secure a consensus on transactions, seen only by authorized users.

Blockchain depends on scalability and does not work well if the file size is too big, but when combined with IPFS, it could overcome this disadvantage and help redefine the way we interact with information and identity.

Blockchain holds enormous potential in the future. This technology will not only save your time and money but it will revolutionize many industries.

FUTURE SCOPE:

As the security is growing day by day attackers are also being more cognizant. Each security schema has some weak points i.e. if an attacker knows them then he can bypass security. So to make the system more secure we can work on the weakness of the system and can further enhance the security.

The project Data Share currently runs on a local network but it can be made to function on any public network with web hosting, thereby making it more scalable.

Upon proper exploration, blockchain could prove to be a boon for leading industries like digital advertising, cybersecurity, supply chain management, networking and forecasting.

REFERENCES:

[1] Balakrishnan, Bhargav, 2010. Three Tier Encryption Engineering and Algorithm for Secure File Transfer. Computer Applications (ICCEA), 2010 Second International Conference on. Vol. 2. IEEE, 2010.

[2] Wheeler, David J. And Needham, Roger M. TEA, a Tiny Encryption Algorithm Computer Laboratory, Cambridge University, England, November 1994.

[3] Tayde, Suchita and Seema Siledar, 2015. File Encryption, Decryption Using AES Algorithm in Android Phone. International Journal of Advanced Research in Computer Science and Software Engineering, 5.5.

[4] Feldman M, Lai K, Stoica I, Chuang J (2004) Robust incentive techniques for peer-to-peer networks. In: Proceedings of the 5th ACM conference on Electronic commerce. ACM, pp. 102–111

[5] https://ieeexplore.ieee.org/abstract/document/9159961

[6] Haddi FL, BenchaÃ´rba M (2015) A survey of incentive mechanisms in static and mobile p2p systems. J Netw Comput Appl 58:108–118

[7] File Encryption, Decryption Using AES Algorithm in Android Phone.Suchita Tayde , Asst. Prof. Seema Siledar. International Journal of Advanced Research in Computer Science and Software Engineering.

[8] Rijmen, Vincent and Joan Daemen, 2001. Advanced encryption standard. Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, pp: 19-22.

[9] Dhongade, Sudhir, et al., 2015. An Efficient Certificateless Encryption for Secure Data Sharing Over the Network Using AES-128 and AES-256..