**A Block chain based secure cloud files sharing scheme with fine grained access control**

**CLIENT-SIDE**

**Blockchain.py**

#import datetime

import time

importhashlib

importjson

from flask import Flask, jsonify, request

import requests

fromurllib.parse import urlparse

# Building a Blockchain

classBlockchain:

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")

defcreate\_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

defget\_previous\_block(self):

returnself.chain[-1]

defproof\_of\_work(self, previous\_proof):

new\_proof = 1

check\_proof = False

whilecheck\_proof is False:

hash\_operation = hashlib.sha256(str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

ifhash\_operation[:4] == '0000':

check\_proof = True

else:

new\_proof += 1

returnnew\_proof

def hash(self, block):

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

return hashlib.sha256(encoded\_block).hexdigest()

defis\_chain\_valid(self, chain):

previous\_block = chain[0]

block\_index = 1

whileblock\_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()

ifhash\_operation[:4] != '0000':

return False

previous\_block = block

block\_index += 1

return True

defadd\_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

defreplace\_chain(self):

network = self.nodes

longest\_chain = None

max\_length = len(self.chain)

for node in network:

response = requests.get(f'http://{node}/get\_chain')

ifresponse.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

iflongest\_chain:

self.chain = longest\_chain

return True

return False

**SERVER-SIDE**

**Server.py**

importos

importurllib.request

importipfshttpclient

frommy\_constants import app

importpyAesCrypt

from flask import Flask, flash, request, redirect, render\_template, url\_for, jsonify

fromflask\_socketio import SocketIO, send, emit

fromwerkzeug.utils import secure\_filename

import socket

import pickle

fromblockchain import Blockchain

import requests

socketio = SocketIO(app)

blockchain = Blockchain()

defallowed\_file(filename):

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

defappend\_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()

defdecrypt\_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'])

defencrypt\_file(file\_path, file\_key):

pyAesCrypt.encryptFile(file\_path, file\_path + ".aes", file\_key, app.config['BUFFER\_SIZE'])

defhash\_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']

returnfile\_hash

defretrieve\_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)

returnsaved\_file

@app.route('/')

def index():

returnrender\_template('index.html')

@app.route('/home')

def home():

returnrender\_template('index.html')

@app.route('/upload')

def upload():

returnrender\_template('upload.html' , message = "Welcome!")

@app.route('/download')

def download():

returnrender\_template('download.html' , message = "Welcome!")

@app.route('/connect\_blockchain')

defconnect\_blockchain():

is\_chain\_replaced = blockchain.replace\_chain()

returnrender\_template('connect\_blockchain.html', chain = blockchain.chain, nodes = len(blockchain.nodes))

@app.errorhandler(413)

defentity\_too\_large(e):

returnrender\_template('upload.html' , message = "Requested Entity Too Large!")

@app.route('/add\_file', methods=['POST'])

defadd\_file():

is\_chain\_replaced = blockchain.replace\_chain()

ifis\_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.')

ifrequest.method == 'POST':

error\_flag = True

if 'file' not in request.files:

message = 'No file part'

else:

user\_file = request.files['file']

ifuser\_file.filename == '':

message = 'No file selected for uploading'

ifuser\_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'

iferror\_flag == True:

returnrender\_template('upload.html' , message = message)

else:

returnrender\_template('upload.html' , message = "File succesfully uploaded")

@app.route('/retrieve\_file', methods=['POST'])

defretrieve\_file():

is\_chain\_replaced = blockchain.replace\_chain()

ifis\_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.')

ifrequest.method == 'POST':

error\_flag = True

ifrequest.form['file\_hash'] == '':

message = 'No file hash entered.'

elifrequest.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!"

iferror\_flag == True:

returnrender\_template('download.html' , message = message)

else:

returnrender\_template('download.html' , message = "File successfully downloaded")

# Getting the full Blockchain

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

defget\_chain():

response = {'chain': blockchain.chain,

'length': len(blockchain.chain)}

returnjsonify(response), 200

@socketio.on('connect')

defhandle\_connect():

print('Client connected')

print(request)

@socketio.on('add\_client\_node')

defhandle\_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')

defhandle\_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')

defhandle\_disconnect():

print('Client disconnected')

print(request)

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

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

**My\_constants.py**

from flask import Flask

UPLOAD\_FOLDER = 'F:/secure file sharing/uploads'

DOWNLOAD\_FOLDER = 'F:/secure file sharing/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'] = set(['txt', 'pdf', 'png', 'jpg', 'jpeg', 'gif'])

app.config['BUFFER\_SIZE'] = 64 \* 1024

app.config['MAX\_CONTENT\_LENGTH'] = 16 \* 1024 \* 1024