A

Major Project

On

Efficient Local Secret Sharing For Distributed Blockchain Systems

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

BY

Allabi Shaik (177R1A05M8)

Prathyusha Maganti (177R1A05M0)

Suresh Karthala (187R5A0507)

Under the Guidance of

G.Latha

(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)Recognized Under Section 2(f) & 12(B) of the UGC Act.1956,
Kandlakoya (V), Medchal Road, Hyderabad-501401.

2016-2020

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

This is to certify that the project entitled "**EFFICIENT LOCAL SECRET SHARING FOR DISTRIBUTED BLOCKCHAIN SYSTEM**" being submitted by **ALLABI SHAIK (177R1A05M8), PRATHYUSHA MAGANTI (177R1A05M0) & SURESH KARTHALA (187R5A0507)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering of the Jawaharlal Nehru Technological University Hyderabad, during the year 2020-2021. It is certified that they have completed the project satisfactorily.

INTERNAL GUIDE
G. Latha
Associate Professor

DIRECTOR
Dr. A. Raji Reddy

HOD
Dr. K. Srujan Raju

EXTERNAL EXAMINER

Submitted for viva voce Examination held on ____________________

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We take this opportunity to express my profound gratitude and deep regard to my guide

G.Latha, Assoc. Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) Coordinators: **Mr.B.Ramji and Mr.J.Narasimha rao** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to the Head of the Department **Dr. K. Srujan Raju** for providing excellent infrastructure and a nice atmosphere for completing this project successfully.

We are obliged to our Director **Dr. A. Raji Reddy** for being cooperative throughout the course of this project. We would like to express our sincere gratitude to our Chairman Sri.
Ch. Gopal Reddy for his encouragement throughout the course of this project

The guidance and support received from all the members of **CMR TECHNICAL CAMPUS** who contributed and who are contributing to this project, was vital for the success of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity thank our family for their constant encouragement without which this assignment would not be possible. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project

ALLABI SHAIK (177R1A05M8)
PRATHYUSHA MAGANTI(177R1A05M0)
SURESH KARTHALA(187R5A0507)

ABSTRACT

Blockchain systems store transaction data in the form of a distributed ledger where each peer is to maintain an identical copy. Blockchain systems resemble repetition codes, incurring high storage cost. Recently, distributed storage block chain (DSB) systems have been proposed to improve storage efficiency by incorporating secret sharing, private key encryption, and information dispersal algorithms. However, the DSB results in significant communication cost when peer failures occur due to denial of service attacks. In this letter, we propose a new DSB approach based on a local secret sharing (LSS) scheme with a hierarchical secret structure of one global secret and several local secrets. The proposed DSB approach with LSS improves the storage and recovery communication costs. Blockchain systems have created a new environment of business transactions and self-regulated cryptocurrencies. However, blockchain works on the premise that every peer stores the entire ledger of transactions in the form of a hash chain, even though they are meaningless to peers that are not party to the transaction

LIST OF FIGURES

LIST OF SCREENSHOTS

1.INTRODUCTION

1.INTRODUCTION

1.1 PROJECT SCOPE

The project titled as “Efficient Local Secret Sharing for Distributed Blockchain System’s” scope is to store data efficiently and to secure data from attacks. In this we use DSB approach based on a local secret sharing (LSS) scheme . LSS efficiently incorporates local secrets and global secrets into a hierarchical secret sharing scheme Networks for this purpose.

1.2 PROJECT PURPOSE

To reduce storage space Distributed Blockchain storage was introduced where all nodes will share a part of data instead of storing entire data or keys by using SHAMIR SECRET KEY algorithm. Due to this secret sharing technique storage space gets reduced but if any node is attacked or down then we cannot share or reconstruct original data and to reconstruct original data distributed nodes has to find out all nodes who have this shares which can cause lots of communication cost. To overcome this Local Secret Sharing (LSS) scheme was introduced.

1.3 PROJECT FEATURES

In Local Secret Sharing (LSS) hash code will be considered as common data which require by all nodes for verification so hash code will be stored as a GLOBAL PEERS or NODES. Private keys will be considered as local data and not require by all nodes, so it will store in LOCAL PEERS. If any node is down or hacked then node will recover data by connecting to local nodes.

2.SYSTEM ANALYSIS

2.SYSTEM ANALYSIS

2.1 PROBLEM DEFINITION

The problems of Blockchain and Distributed Blockchain Systems like storage space and recovery costs will be solved using Local Secret Sharing (LSS).It is achieved by considering hash code as common data which require by all nodes for verification so hash code will be stored as a GLOBAL PEERS or NODES. Private keys will be considered as local data and not require by all nodes, so it will store in LOCAL PEERS. If any node is down or hacked then node will recover data by connecting to local nodes.

2.2 EXISTING SYSTEM

In existing Blockchain system each node was storing complete data '527166' but in distributed Blockchain each node will store only its share such as 52 in one node and attack or down then we cannot get it share and we cannot reconstruct original data and to reconstruct original data distributed nodes has to find out all nodes who have this shares which can cause lots of communication cost. In distributed Blockchain another disadvantage is it will store hash code and share part in each node which can still require some storage space.

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Security issues are more.
- Need more storage space.
- The cost of recovering data is more
- More Chances for loss of data

To avoid all these limitations and make the working more accurately the system needs to be computerized.

2.3 PROPOSED SYSTEM

In this paper, we introduce Local Secret Sharing (LSS) where hash code will be consider as common data which require by all node for verification so hash code will be store as GLOBAL Private keys will be consider as local data and not require by all nodes so it connecting to local nodes. So in propose work hash code and private keys will store in different LOCAL and GLOBAL nodes so storage cost will be some more reduce and node can recover data from its local neighbours then communication cost will also be reduce.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

- Low Cost.
- Much Security.
- The Cost of Recovery is Less.
- No Data Loss Occurs.

2.4 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements,

- Windows 7, 8 and 10
- Linux
- Mac OS
- Chrome OS

2.5 HARDWARE REQUIREMENTS:

The hardware requirements are the requirements of a hardware device. Most hardware only has operating system requirements or compatibility.

- Processor - pentinum-IV
- Speed - 1.1 Ghz
- Ram - 256MB(min)
- Hard Disk - 20GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse
- Monitor - SVGA

2.6 LANGUAGES AND TECHNOLOGIES USED

- Python
- Blockchain Systems

3.ARCHITECTURE

3.ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture describes about the process of the project.

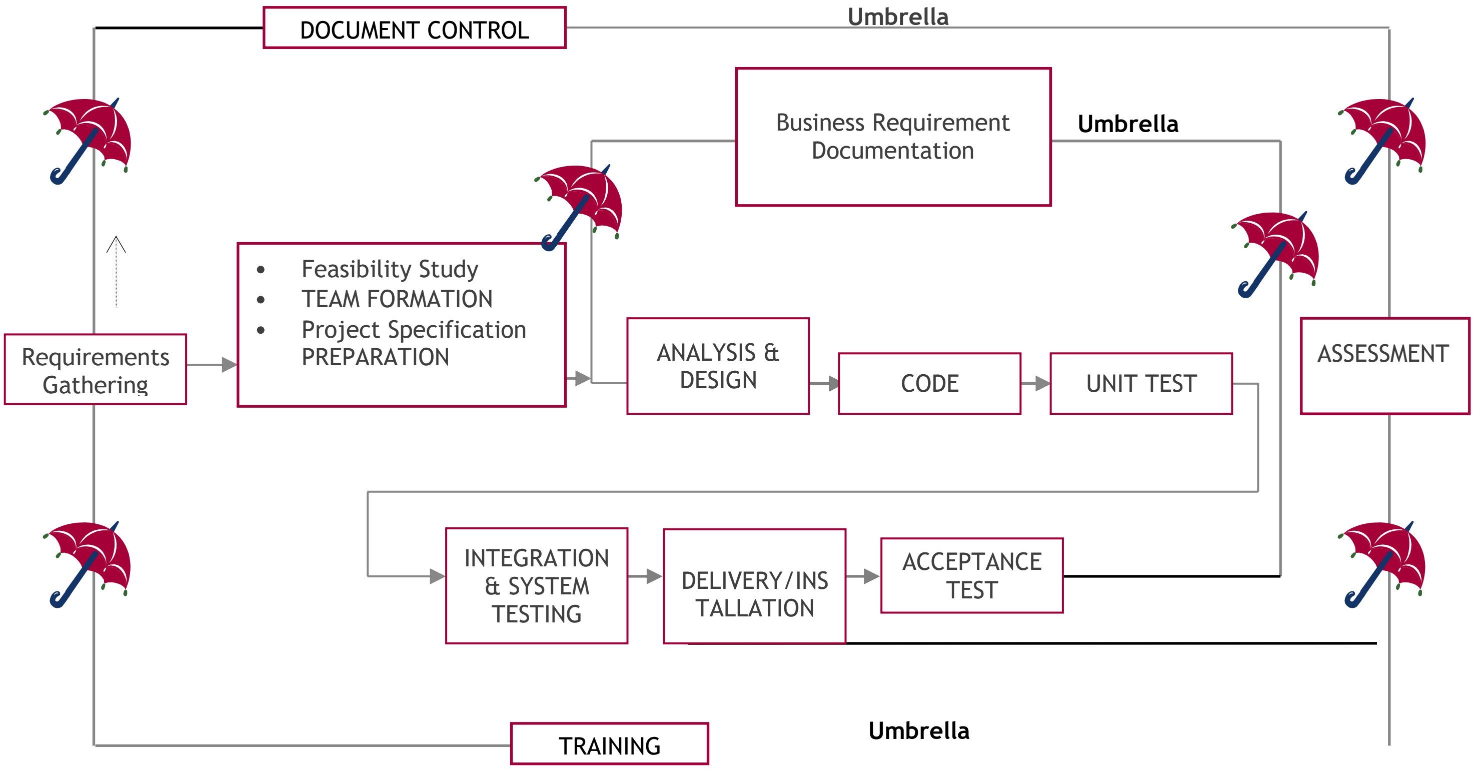
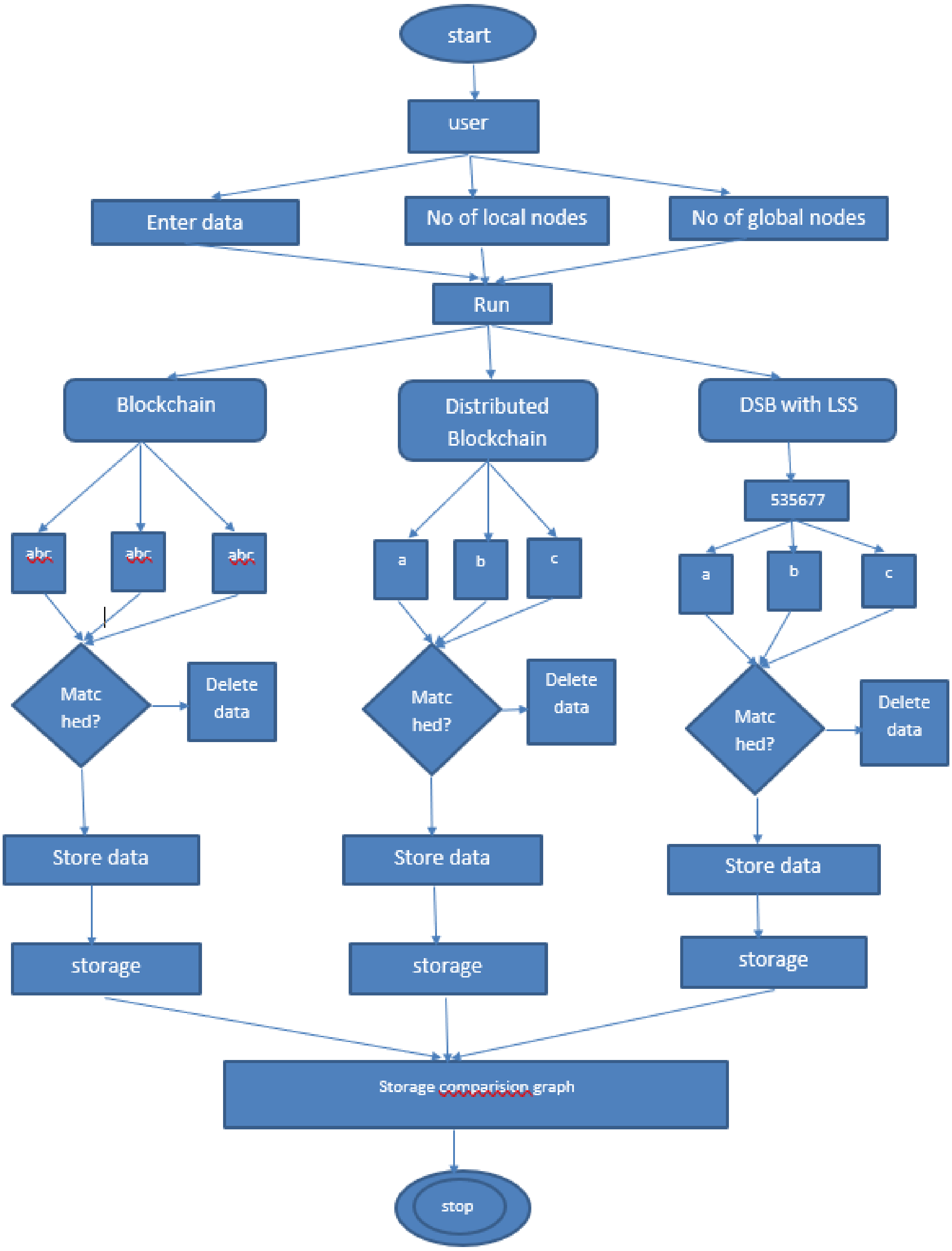


Fig. 3.1 Project Architecture

3.2 PROCESS FLOW DIARGRAM:



3.3 Algorithm:

Given partition $\mathcal{A} = \{A_1, \ldots, A_{\frac{n}{r+1}}\}$:

Set hash value $W^{(t)}$ as global secret $s^{(t)} = a_{0,0} \in \mathbb{F}_q$.
Generate a random vector $\mathbf{a}^{(t)}_{\backslash a_{0,0}} \in \mathbb{F}_q^{k-1}$.
Compute local secrets $s_l^{(t)}$ for $l \in [1, \frac{n}{r+1}]$ and $(f_{\mathbf{a}}^{(t)}(\alpha_i))$
    for $i \in [1, n]$ by Alg. 2.
Distribute and store $(f_{\mathbf{a}}^{(t)}(\alpha_i))$ into n peers.
for $l = 1$ to $\frac{n}{r+1}$ **do**
    Encrypt $B^{(t)}$ with $s_l^{(t)}$ as $\mathbf{m}_l^{(t)} = \Phi(B^{(t)}; s_l^{(t)})$.
    Encode $\mathbf{m}_l^{(t)}$ into $\mathbf{c}_l^{(t)}$ by $(r+1, r)$ coding.
    Distribute and store $\mathbf{c}_l^{(t)}$ among peers in A_l.
end for

3.2 MODULES

1.Enter Data :

The Enter Data is the initial module in our project. In this we have to enter some data to encrypt and store in blockchain.

2. Number Of Global And Local Peers :

Then enter number of global peers to store hash code and then local peers will store private keys share.

3. Run Traditional Blockchain :

In traditional Blockchain we don't have any secret sharing scheme so we are not showing any data related to sharing.

4. Run Distributed Storage Blockchain(DSB) :

The Distributed Storage Blockchain is one of the module in our project. After run the DSB in the screen we get selected text first showing previous and Current block hash codes and then displaying encrypted data then displaying decrypted data with the help of private keys and then displaying DSB storage cost.

5. Run Distributed Storage Blockchain With LSS :

Run Distributed Storage Blockchain with LSS to share secret hash code in global peer and private keys in local peers and then calculate storage cost.

6. Storage Comparison Graph :

In Storage Comparison Graph x-axis represents technique name and y-axis represents storage cost for each technique.

3.3 USE CASE DIAGRAM

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different **use** cases in which the user is involved.

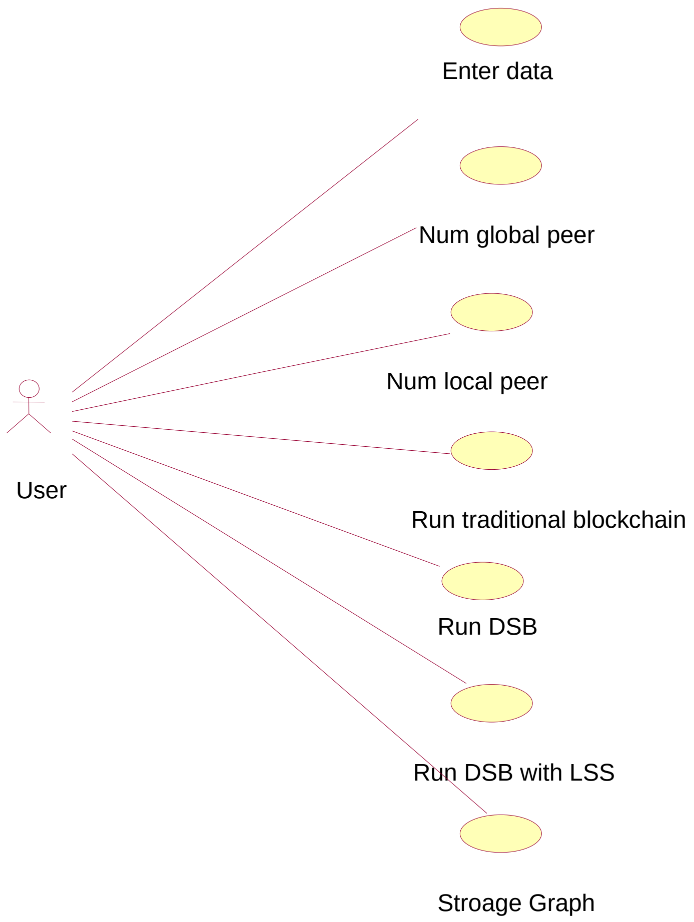


Fig. 3.3 Use Case Diagram

3.4 CLASS DIAGRAM

Class Diagram is a collection of classes and objects.

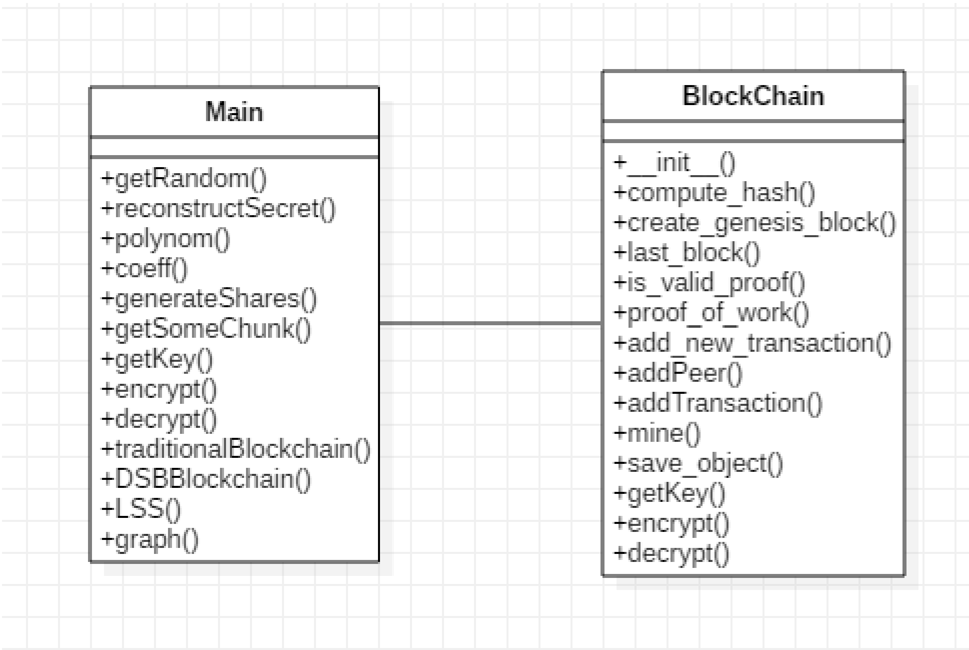


Fig. 3.4 Class Diagram

3.5 SEQUENCE DIAGRAM

The user gives the image to the trained model and the system performs the task to give the desired output

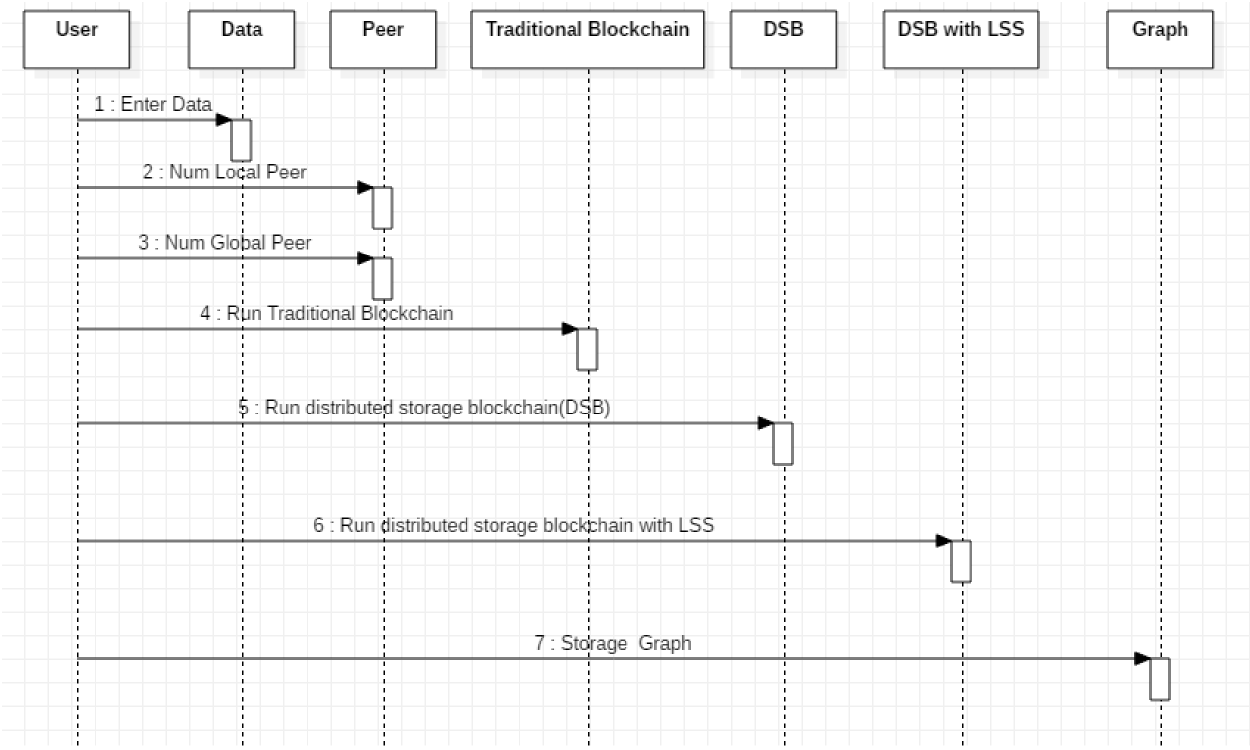


Fig.3.5 Sequence Diagram

3.6 ACTIVITY DIAGRAM

It describes about flow of activity states.

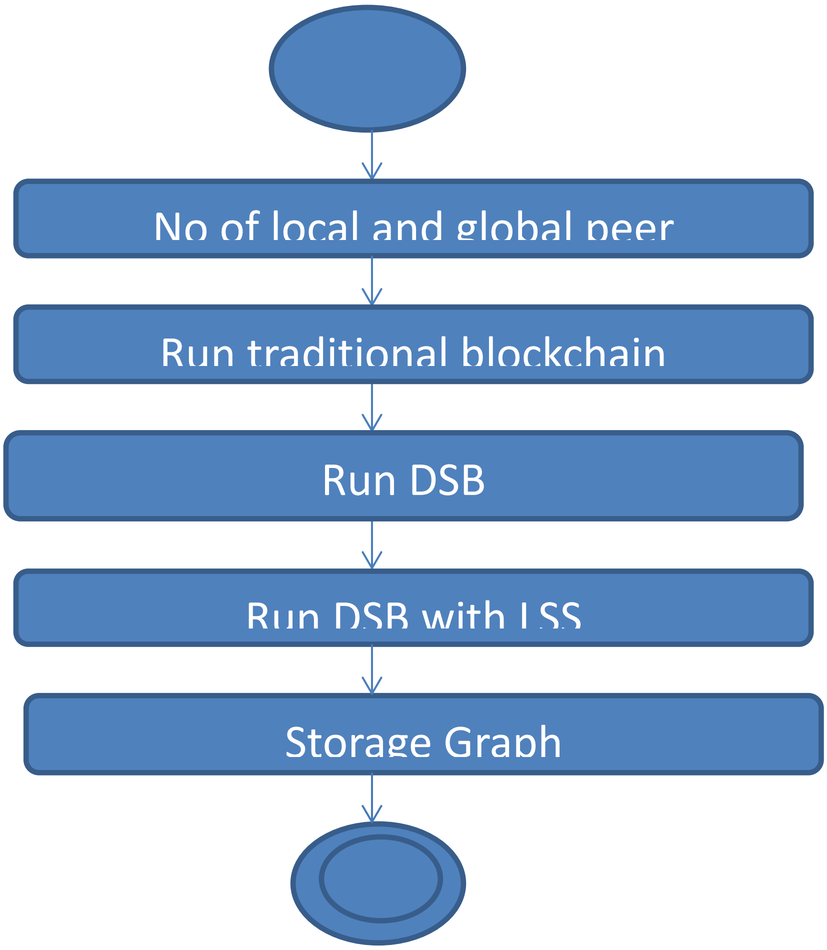


Fig. 3.6 Activity Diagram

4.IMPLEMENTATION

4.IMPLEMENTATION

Blockchain.py:

```
from hashlib import sha256
import json
import time
import pickle
from datetime import datetime
import random
import pyaes, pbkdf2, binascii, os, secrets
import base64
class Block:
    def __init__(self, index, transactions, timestamp, previous_hash):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.nonce = 0
    def compute_hash(self):
        block_string = json.dumps(self.__dict__, sort_keys=True)
        return sha256(block_string.encode()).hexdigest()
class Blockchain:
    # difficulty of our PoW algorithm
    difficulty = 2 #using difficulty 2 computation
:
```

```
  def __init__(self)
  self.unconfirmed_transactions = []
    self.chain = []
    self.create_genesis_block()
    self.peer = []
    self.translist = []
def create_genesis_block(self): #create genesis block
    genesis_block = Block(0, [], time.time(), "0")
    genesis_block.hash = genesis_block.compute_hash()
    self.chain.append(genesis_block)
@property
def last_block(self):
    return self.chain[-1]
def add_block(self, block, proof): #adding data to block by computing new and previous hashes
    previous_hash = self.last_block.hash
    if previous_hash != block.previous_hash:
        return False
    if not self.is_valid_proof(block, proof):
        return False
    block.hash = proof
    #print("main "+str(block.hash))
    self.chain.append(block)
    return True
```

```
    def is_valid_proof(self, block, block_hash): #proof of work
        return (block_hash.startswith('0' * Blockchain.difficulty) and block_hash == block.compute_hash())
def proof_of_work(self, block): #proof of work
         block.nonce = 0


        computed_hash = block.compute_hash()
        while not computed_hash.startswith('0' * Blockchain.difficulty):
            block.nonce += 1
            computed_hash = block.compute_hash()
        return computed_hash
    def add_new_transaction(self, transaction):
        self.unconfirmed_transactions.append(transaction)
    def addPeer(self, peer_details):
        self.peer.append(peer_details)
    def addTransaction(self,trans_details): #add transaction
        self.translist.append(trans_details)
    def mine(self):#mine transaction
        if not self.unconfirmed_transactions:
            return False
        last_block = self.last_block
        new_block = Block(index=last_block.index + 1,
                          transactions=self.unconfirmed_transactions,
                          timestamp=time.time(),
                          previous_hash=last_block.hash)
```

```
proof = self.proof_of_work(new_block)

      self.add_block(new_block, proof

      self.unconfirmed_transactions = []

      return new_block.index

     def save_object(self,obj, filename)

    with open(filename, 'wb') as output:

        pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)

  def getKey(self): #generating key with PBKDF2 for AES

      password = "s3cr3t*c0d3"

      passwordSalt = '76895'

      key = pbkdf2.PBKDF2(password, passwordSalt).read(32)

      return key

  def encrypt(self,plaintext): #AES data encryption

      aes=pyaes.AESModeOfOperationCTR(self.getKey(),
pyaes.Counter(31129547035000047302952433967654195398124239844566322884172163637846056248223
))

      ciphertext = aes.encrypt(plaintext)

      return ciphertext

  def decrypt(self,enc): #AES data decryption

      aes=pyaes.AESModeOfOperationCTR(self.getKey(),
pyaes.Counter(31129547035000047302952433967654195398124239844566322884172163637846056248223
))

      decrypted = aes.decrypt(enc)

      return decrypted

if __name__== "__main__":

  blockchain = Blockchain()
```

```
for i in range(0,30):
    count = 'IoT'+str(i)+","+str(random.randint(2,58))
        print(count)
    enc = blockchain.encrypt(count)
    enc = str(base64.b64encode(enc),'utf-8')
    blockchain.addPeer(enc)
    if len(blockchain.peer) > 10:
        for i in range(len(blockchain.peer)):
            if len(blockchain.chain) == 15:
                blck = blockchain.chain.pop(0)
                with open('remove.txt', 'wb') as output:
                    pickle.dump(blck, output, pickle.HIGHEST_PROTOCOL)
            blockchain.add_new_transaction(blockchain.peer[i])
            blockchain.mine()
        blockchain.save_object(blockchain,'BC_DB.txt')
        blockchain.peer.clear()
with open('BC_DB.txt', 'rb') as input:
     blockchain = pickle.load(input)
            for i in range(len(blockchain.chain)):
                b = blockchain.chain[i]
                data = b.transactions[0]
                data = base64.b64decode(data)
                decrypt = blockchain.decrypt(data)
            print(str(decrypt.decode("utf-8"))+""+str(b.previous_hash)+"=”"+str(b.index)+""+str(b.hash)+"
"+str(datetime.fromtimestamp(b.timestamp)))
```

Main.py:

```
from tkinter import messagebox

from tkinter import *

from tkinter import simpledialog

import tkinter

import matplotlib.pyplot as plt

import numpy as np

import pyaes, pbkdf2, binascii, os, secrets

import base64

import array

import random

from math import ceil

from decimal import *

import mmap

from Blockchain import *

main = Tk()

main.title("Efficient Local Secret Sharing for Distributed Blockchain Systems")

main.geometry("1300x1200")

global field_size

field_size = 10**5

global existing_storage

global dsb_storage

global lss_storage

dictKey = {}
```

```
 def getRandom():
        return random.randrange(11234,998765)
def reconstructSecret(shares):
    # Combines shares using
    # Lagranges interpolation.
    # Shares is an array of shares
    # being combined
    sums, prod_arr = 0, []
    for j in range(len(shares)):
        xj, yj = shares[j][0],shares[j][1]
        prod = Decimal(1)
        for i in range(len(shares)):
            xi = shares[i][0]
            if i != j: prod *= Decimal(Decimal(xi)/(xi-xj))
        prod *= yj
        sums += Decimal(prod)
    print(Decimal(sums))
    return int(round(Decimal(sums),0))
def polynom(x,coeff):
    # Evaluates a polynomial in x
    # with coeff being the coefficient
    # list
    return sum([x**(len(coeff)-i-1) * coeff[i] for i in range(len(coeff))])
```

```
def coeff(t,secret):
# degree t-1 whose constant = secret'''
   coeff = [random.randrange(0, field_size) for _ in range(t-1)]
   coeff.append(secret)
   return coeff
def generateShares(n,m,secret):
   cfs = coeff(m,secret)
   shares = []
   for i in range(1,n+1):
      r = random.randrange(1, field_size)
      shares.append([r, polynom(r,cfs)])
   return shares
def getSomeChunk(filename, start, len):
   fobj = open(filename, 'r+b')
   m = mmap.mmap(fobj.fileno(), 0)
   return m[start:start+len]
def getKey():
   password = "s"
   passwordSalt = '7'
   key = pbkdf2.PBKDF2(password, passwordSalt).read(32)
   return key
def encrypt(plaintext,key): #AES data encryption
   aes=pyaes.AESModeOfOperationCTR(key,
pyaes.Counter(31129547035000047302952433967654195398124239844566322884172163637846056248223
))
```

```
  ciphertext = aes.encrypt(plaintext)

    return ciphertext

def decrypt(enc,key): #AES data decryption

    aes=pyaes.AESModeOfOperationCTR(key,
pyaes.Counter(31129547035000047302952433967654195398124239844566322884172163637846056248223
))

    decrypted = aes.decrypt(enc)

    return decrypted

def traditionalBlockchain():

    global existing_storage

    text.delete('1.0', END)

    data = tf1.get().strip()

    peers = int(tf2.get().strip())

    local = int(tf3.get().strip())

    key = getKey()

    secret = getRandom()

    dictKey[secret] = key

    enc = encrypt(data,key)

    enc = str(base64.b64encode(enc),'utf-8')

    blockchain = Blockchain()

    blockchain.add_new_transaction(enc)

    hash = blockchain.mine()

    b = blockchain.chain[len(blockchain.chain)-1]

    bdata = b.transactions[0]

    data = base64.b64decode(bdata)
```

```
    decrypts = decrypt(data,key)
    text.insert(END,'Private Key : '+str(key)+"\n")
    text.insert(END,'Blockchain Storage : '+str(b.transactions)+"\n")
    text.insert(END,'Previous block hash code : '+str(b.previous_hash)+"\n")
    text.insert(END,'Current block hash code : '+str(b.hash)+"\n")
    text.insert(END,'Decrypted Data : '+str(decrypts.decode("utf-8"))+"\n")
    existing_storage = (len(str(b.previous_hash)) + len(b.hash) + len(key) + len(bdata))
    print(str(len(str(b.previous_hash))) +" "+ str(len(b.hash)) +" "+ str(len(key))+" "+ str(len(bdata)))
    existing_storage = local * peers * existing_storage
    existing_storage = existing_storage / 1000
    text.insert(END,'Traditional Blockchain Storage : '+str(existing_storage))
def DSBBlockchain():
    global dsb_storage
    text.delete('1.0', END)
    data = tf1.get().strip()
    peers = int(tf2.get().strip())
    local = int(tf3.get().strip())
    key = getKey()
    secret = getRandom()
    dictKey[secret] = key
    enc = encrypt(data,key)
    enc = str(base64.b64encode(enc),'utf-8')
    blockchain = Blockchain()
    blockchain.add_new_transaction(enc)
```

```
hash = blockchain.mine()
 key = getKey()
 enc = encrypt(data,key)
 t,n = 5, 7
 secret = getRandom()
 text.insert(END,'Original Private Key : '+str(secret)+"\n")
 dictKey[secret] = key
 shares = generateShares(n, t, secret)
 text.insert(END,'Shares generated from private key : '+str(shares)+"\n")
 length = len(shares)
 share1 = ''
 share2 = ''
 num_block = length / 2
 j = 0
 for i in range(len(shares)):
     if j < num_block:
         value = str(shares[i])
         value = value[1:len(value)-1]
         value = value.split(",")
         value[0] = value[0].strip()
         value[1] = value[1].strip()
         share1+=value[0]+","+value[1]+" "
         j = j + 1
```

```
  elif j >= num_block:
        value = str(shares[i])
value = value[1:len(value)-1]
        value = value.split(",")
        value[0] = value[0].strip()
        value[1] = value[1].strip()
        share2+=value[0]+","+value[1]+" "
        j = j + 1
combine_share = []
first = share1.strip().split(" ")
second = share2.strip().split(" ")
for i in range(len(first)):
    arr = first[i].split(",")
    f = int(arr[0])
    s = int(arr[1])
    temp = [f,s]
    combine_share.append(temp)
for i in range(len(second)):
    arr = second[i].split(",")
    f = int(arr[0])
    s = int(arr[1])
    temp = [f,s]
    combine_share.append(temp)
print(combine_share)
```

```
    pool = random.sample(combine_share, t)
    original = reconstructSecret(pool
    pool = random.sample(combine_share, t)
    original = reconstructSecret(pool)
    text.insert(END,'\nCombining shares from all shares to generate private key : '+str(combine_share)+"\n")
    text.insert(END,"Reconstructed private key :"+str(original)+"\n")
    key = dictKey.get(original)
    b = blockchain.chain[len(blockchain.chain)-1]
    bdata = b.transactions[0]
    data = base64.b64decode(bdata)
    decrypts = decrypt(data,key)
    text.insert(END,'Previous block hash code : '+str(b.previous_hash)+"\n")
    text.insert(END,'Current block hash code : '+str(b.hash)+"\n")
    text.insert(END,'Blockchain Storage : '+str(b.transactions)+"\n")
    text.insert(END,'Decrypted Data : '+str(decrypts.decode("utf-8"))+"\n")
    dsb_storage = (len(str(b.previous_hash)) + len(b.hash) + len(key))
    print(str(len(str(b.previous_hash))) +" "+ str(len(b.hash)) +" "+ str(len(key)))
    dsb_storage = local * peers * dsb_storage
    dsb_storage = dsb_storage / 1000
    text.insert(END,'DSB Blockchain Storage : '+str(dsb_storage)
def LSS():
    global lss_storage
    text.delete('1.0', END)
```

```
data = tf1.get().strip()
peers = int(tf2.get().strip())
local = int(tf3.get().strip())
key = getKey()
secret = getRandom()#getting random secret key
dictKey[secret] = key
enc = encrypt(data,key)#encrypting data
enc = str(base64.b64encode(enc),'utf-8')
blockchain = Blockchain()
blockchain.add_new_transaction(enc) #creating blockchain object and storing encrypted data
hash = blockchain.mine() #mining the transaction
t,n = 5, 7
text.insert(END,'Original Private Key : '+str(secret)+"\n")
shares = generateShares(n, t, secret) #generating secret
text.insert(END,'Shares generated from private key : '+str(shares)+"\n")
length = len(shares)
share1 = ''
share2 = ''
num_block = length / 2
j = 0
for i in range(len(shares)): #distributing shares to all peers in loop
    if j < num_block:
        value = str(shares[i])
        value = value[1:len(value)-1]
```

```
        value = value.split(",")
        value[0] = value[0].strip()
         value[1] = value[1].str
    s = int(arr[1])
    temp = [f,s]
    combine_share.append(temp) #storing all collected shares in arrat
print(combine_share)
pool = random.sample(combine_share, t)#combining sll shares
original = reconstructSecret(pool) #reconstructing original secret
text.insert(END,'\nCombining shares from all shares to generate private key : '+str(combine_share)+"\n")
text.insert(END,"Reconstructed private key :"+str(original)+"\n")
key = dictKey.get(original) #getting key by giving original share
b = blockchain.chain[len(blockchain.chain)-1]
bdata = b.transactions[0]
data = base64.b64decode(bdata)
decrypts = decrypt(data,key)#decrypting data using private key recover from all shares
text.insert(END,'Previous block hash code : '+str(b.previous_hash)+"\n")
text.insert(END,'Current block hash code : '+str(b.hash)+"\n")
text.insert(END,'Blockchain Storage : '+str(b.transactions)+"\n")
text.insert(END,'Decrypted Data : '+str(decrypts.decode("utf-8"))+"\n")
global_storage = (len(str(b.previous_hash)) + len(b.hash)) #calculating storage cost
local_storage = len(key)
lss_storage = (global_storage * peers) + (local_storage * local)
```

```
lss_storage = lss_storage / 1000
    text.insert(END,'DSB with LSS Blockchain Storage : '+str(lss_storage))
def graph():
    height = [existing_storage,dsb_storage,lss_storage]
    bars = ('Traditional Blockchain Storage','DSB Storage','DSB with LSS Storage')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()
font = ('times', 15, 'bold')
title = Label(main, text='Efficient Local Secret Sharing for Distributed Blockchain Systems')
title.config(bg='mint cream', fg='olive drab')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)
font1 = ('times', 13, 'bold')
ff = ('times', 12, 'bold')
l1 = Label(main, text='Enter Data : ')
l1.config(font=font1)
l1.place(x=50,y=100)
tf1 = Entry(main,width=40)
tf1.config(font=font1)
tf1.place(x=230,y=100)
l2 = Label(main, text='Num Global Peer : ')
```

```
l2.config(font=font1)
l2.place(x=50,y=150)
tf2 = Entry(main,width=40)
tf2.config(font=font1)
tf2.place(x=230,y=150 )
l3 = Label(main, text='Num Local Peer : ')
l3.config(font=font1)
l3.place(x=50,y=200)
tf3 = Entry(main,width=40)
tf3.config(font=font1)
tf3.place(x=230,y=200)
traditionalButton = Button(main, text="Run Traditional Blockchain", command=traditionalBlockchain)
traditionalButton.place(x=20,y=250)
traditionalButton.config(font=ff)
dsbButton = Button(main, text="Run Distributed Storage Blockchain (DSB)", command=DSBBlockchain)
dsbButton.place(x=270,y=250)
dsbButton.config(font=ff)
lssButton = Button(main, text="Run Distributed Storage Blockchain with LSS", command=LSS)
lssButton.place(x=610,y=250)
lssButton.config(font=ff)
graphButton = Button(main, text="Storage Comparison Graph", command=graph)
graphButton.place(x=980,y=250)
graphButton.config(font=ff)
font1 = ('times', 13, 'bold')
```

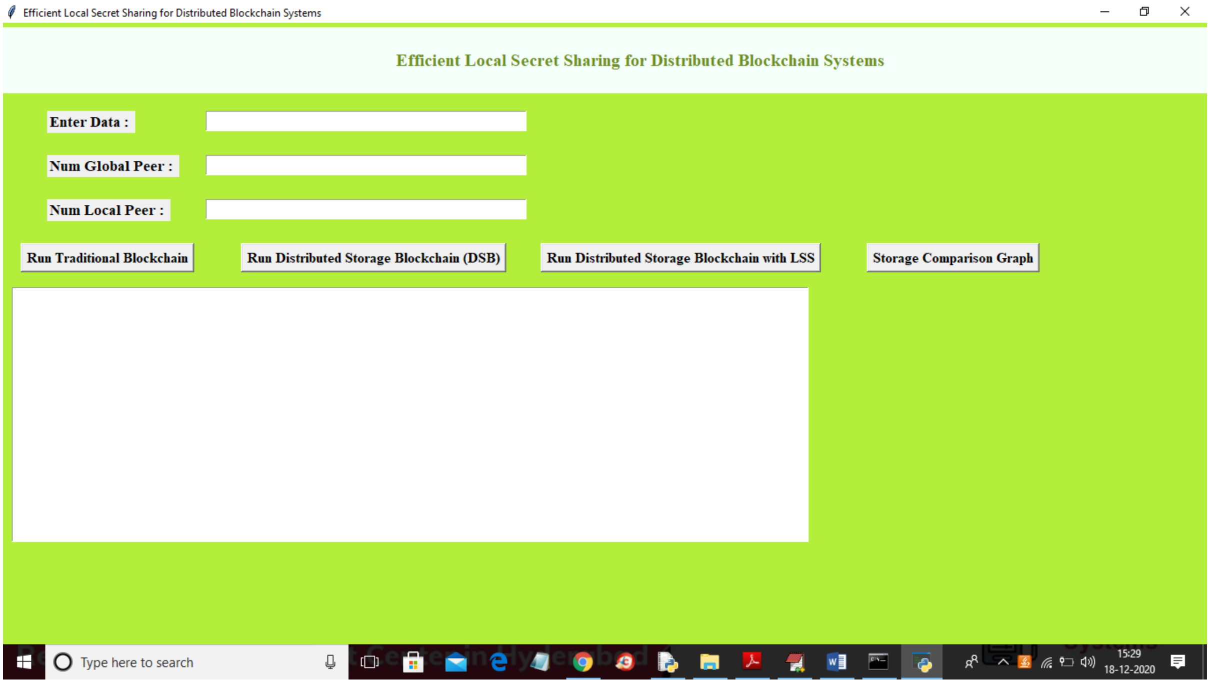
```
text=Text(main,height=15,width=100)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=300)
text.config(font=font1)
main.config(bg='OliveDrab2')
main.mainloop()
```

-

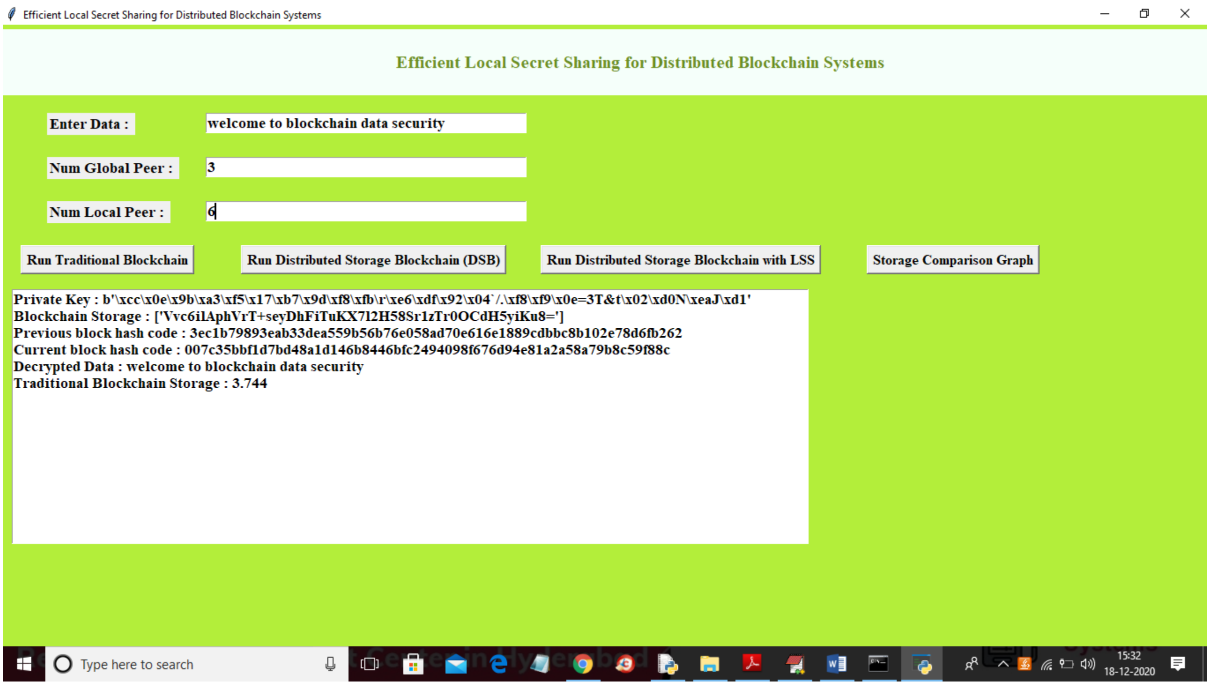
5.SCREENSHOTS

5.SCREENSHOTS

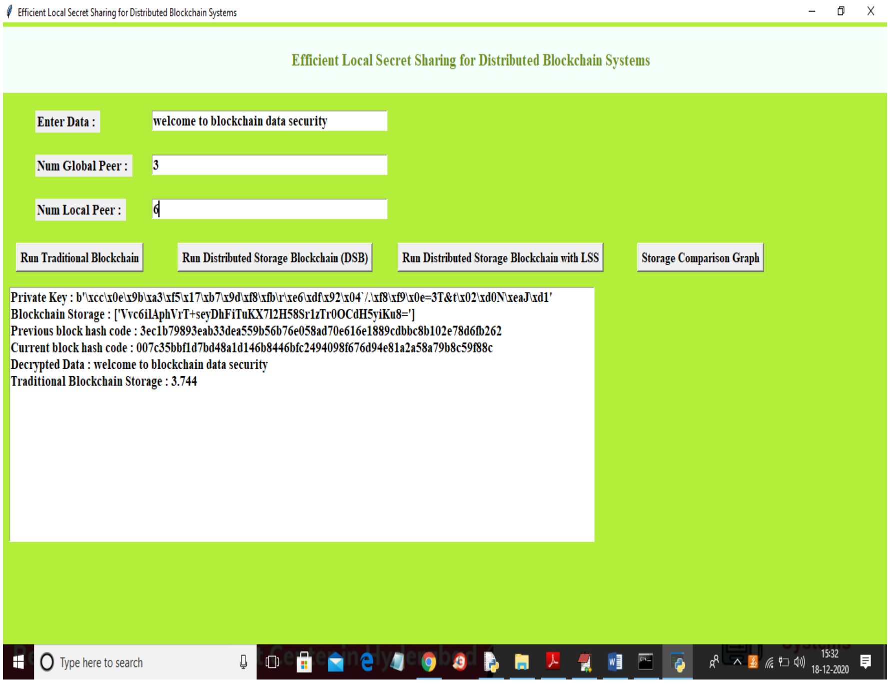
5.1 OUTPUT SCREEN



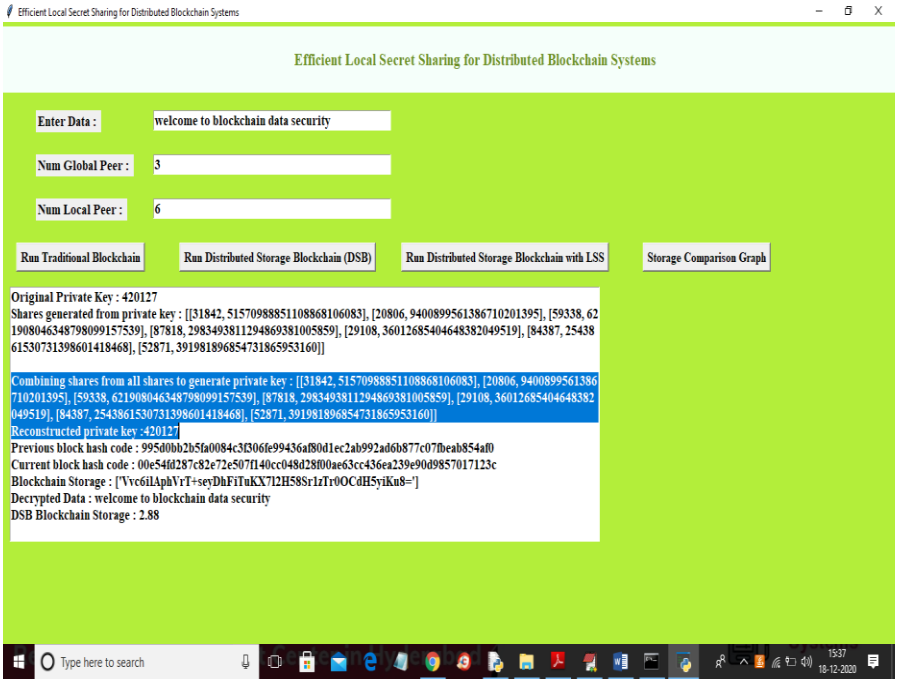
5.2 ENTERING INPUTS:



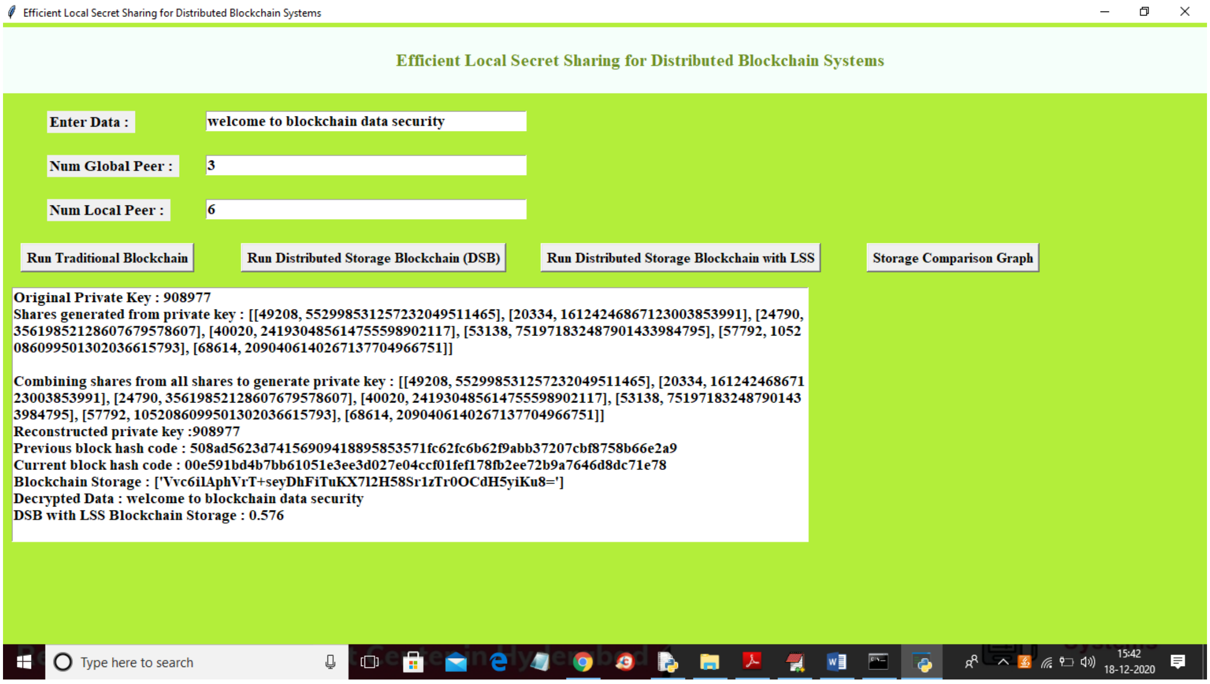
5.3 TRADITIONAL BLOCKCHAIN STORAGE



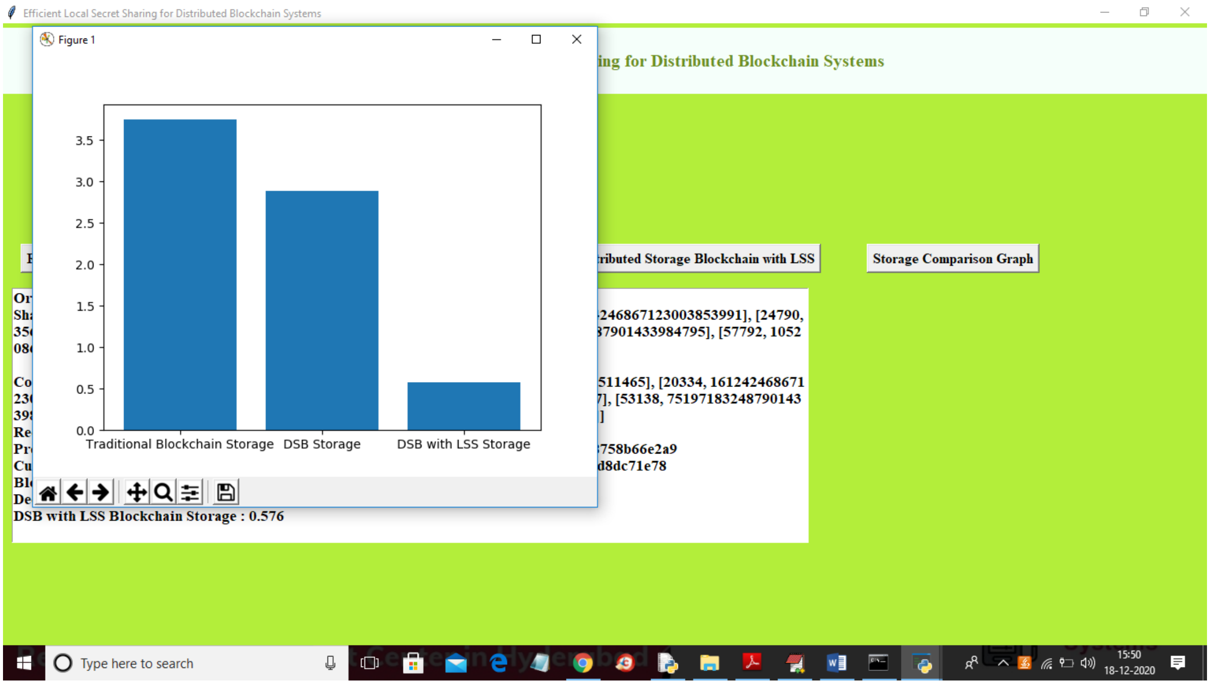
5.4 DISTRIBUTED BLOCKCHAIN STORAGE



5.5 DISTRIBUTED BLOCKCHAIN WITH LSS STORAGE



5.6 STORAGE COMPARISION GRAPH



6.TESTING

6.TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes.

6.3 TESTCASES

Id	Test Case Name	Test Case Desc.	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		
01	Enter Data	Test whether the data enter or not	If the data may not entered	We cannot do further process	The data entered successfully	High	High
02	Num global peer	Verify either the gobal peer numbers entered or not	Without enter the number of global peer	We cannot save the private keys to share	global peer number entered successfully	High	High
03	Num local peer	Verify either the local peer numbers entered or not	Without enter the number of local peer	We cannot save the private keys to share	Local peer number entered successfully	High	High
04	Run traditional blockchain	Verify either the traditional blockchain will run or not	Without having the data	We cannot run the traditional blockchain	The traditional blockchain Run successfully	High	High
5	Run distributed storage blockchain(DSB)	Verify either the distributed storage blockchain(DSB) will run or not	Without having the data	We cannot run the distributed storage blockchain(DSB)	The distributed storage blockchain(DSB) run successfully	High	High

6	Run distributed storage blockchain with LSS	Verify either the distributed storage blockchain with LSS will run or not	Without having the data	We cannot run the distributed storage blockchain with LSS	The distributed storage blockchain with LSS run successfully	High	High
7	Stroage Graph	Verify either the storage Graph is displayed or not	Without saving the data	The storage cost Comparison Graph is not displayed	The stotage cost Comparison Graph is displayed successfully	High	High

7.CONCLUSION

7.CONCLUSION

In this project, we have proposed a new DSB scheme based on LSS. The proposed scheme reduces the storage and recovery communication costs. communication. So we can conclude that DSB with LSS consume less storage space compared to all other techniques.

8.BIBLIOGRAPHY

8.BIBILOGRAPHY

8.1References

- K. Croman et al., “On scaling decentralized blockchains,” in Proc. Int. Conf. Financial Cryptogr.Data Secur., Aug. 2016, pp. 106–125. (pg.1)
- R. K. Raman and L. R. Varshney, “Distributed storage meets secret sharing on the blockchain,” in Proc. Inf. Theory Appl. Workshop (ITA), Feb. 2018.(pg.2-3)
- H. Krawczyk, “Secret sharing made short,” in Proc. Annu. Int. Cryptol. Conf., Jan. 1994, pp. 136–146. (pg 4-8)
- Shamir, “How to share a secret,” Commun. ACM, vol. 22, no. 11, pp.612–613, Nov. 1979. (pg 9-15)
- Pannetrat and R. Molva, “Efficient multicast packet authentication,” in Proc. Netw. Distrib. Syst. Security Symp. (NDSS), Feb. 2003. (pg 16-20)
- Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” IEEE Trans. Inf. Theory, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.(pg 21-27)
- Tamo and A. Barg, “A family of optimal locally recoverable codes,” IEEE Trans. Inf. Theory, vol. 60, no. 8, pp. 4661–4676, Aug. 2014. (pg 28-32)
- Krawczyk, “Secret sharing made short,” in Proc. Annu. Int. Cryptol. Conf., Jan. 1994, pp. 136–146. (pg 33-36)
- R. K. Raman and L. R. Varshney, “Dynamic distributed storage for blockchains,” in Proc. IEEE Int. Symp. Inf. Theory (ISIT), Jun. 2018,(pg 37-38)