Health Care System - Register

Team

- Sai Manideep Bandaru (2019A7PS0016H)
- S. Varshith Reddy (2018B3A70022H)
- Aryan Surana (2018B4A70937H)

Problem Statement:

Conventional database systems are managed by powerful stakeholders and institutions, having power and reason to force their hand. Databases support data manipulation operations like create, read, update, and delete, which compromises privacy and security. With conventional health databases there always exists the risk of a single point of failure. This problem is solved by using cryptographic solutions and usage of blockchain technology.

Usage of Blockchain

Intro to Blockchain

Blockchain is a peer-to-peer, decentralized database management system. It only supports create and read functions i.e, it is very difficult to meddle with the data or records. It improves security and privacy using asymmetric cryptography.

Solution Using Blockchain

A profile is created by mining into the chain thus to access the profile , a key has to be generated with respect to the block it occupies. To view the profile , he simply needs to enter a cryptic SHA256 name of the patient and the key of the mine . This improves security and privacy. Whenever there is an update , the edited or updated new profile is allocated a new block by mining again into the chain randomly thus it secures the previous data to.User only has to remember the latest displayed key.The detailed usage of blockchain will be explained in further details below.

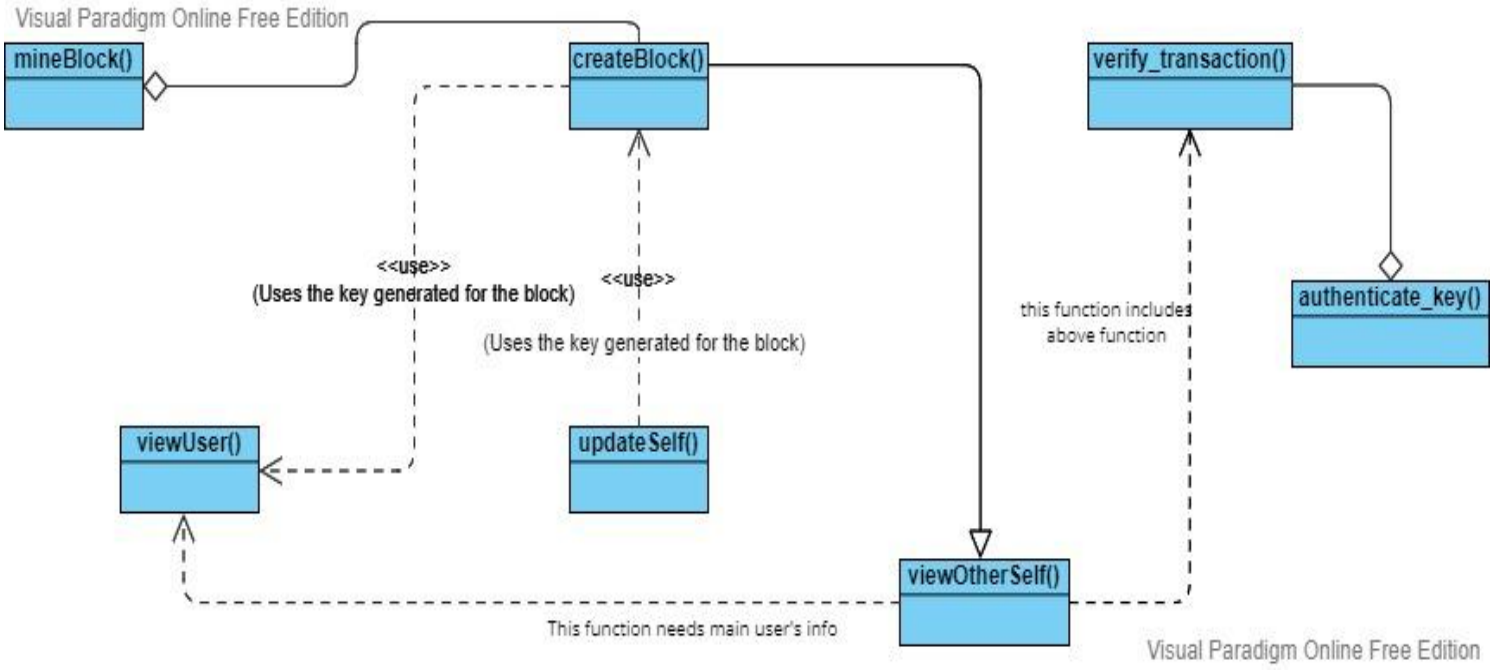
Zero Knowledge Proof

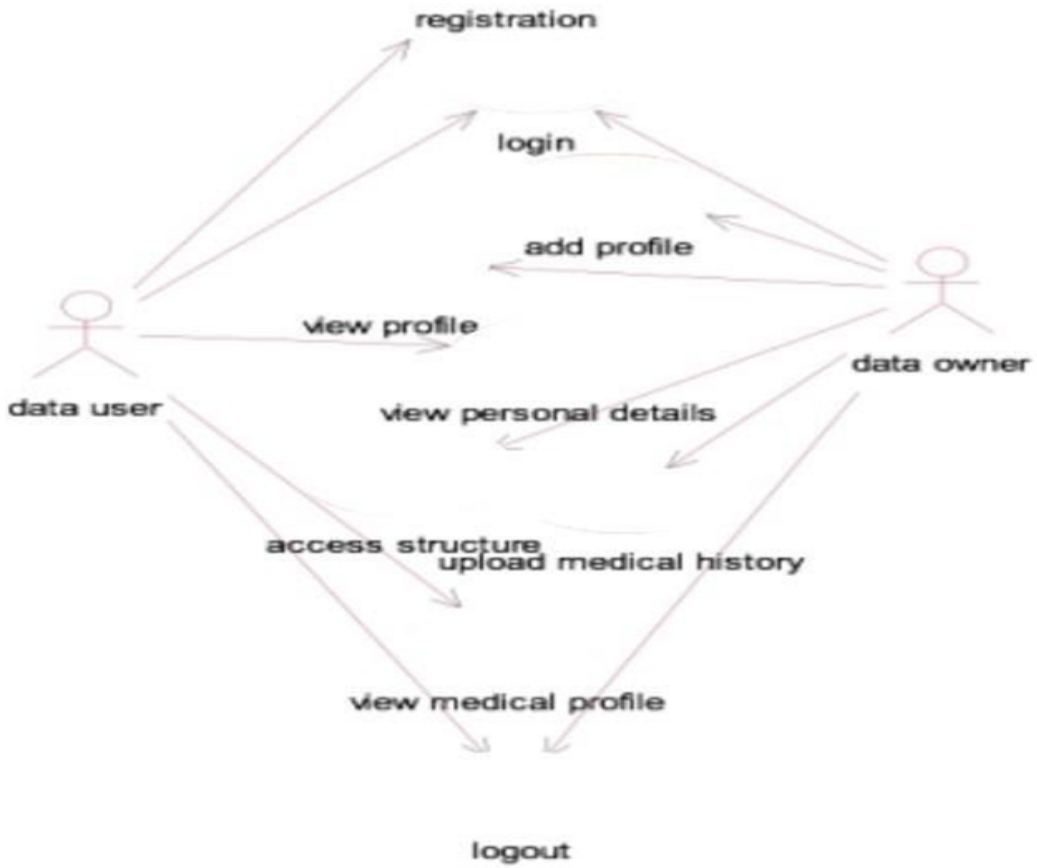
This has two people – prover and verifier – to prove that the user has the patient's encrypted name. This is proved by executing a probability based algorithm, by taking inputs from both prover and verifier. The prover is asked to enter a large prime number. Then the prover is asked to enter a random number between 2 and prime number-1. The verifier then is asked to enter a boolean (0/1) value. The verifier then matches the values and gives the user access to the medical details of the patient.

Functions used in the Register and UML Class Diagram

Few main functions which we used are :

1. createBlock():
2. mineBlock():
3. verify_transaction():
4. updateSelf():
5. viewSelf():
6. viewOtherSelf():
7. authenticate_key():


Visual Paradigm Online Free Edition
mineBlock()
createBlock()
verify_transaction()
authenticate_key()
<<use>>
(Uses the key generated for the block)
<<use>>
(Uses the key generated for the block)
this function includes above function
viewUser()
updateSelf()
viewOtherSelf()
This function needs main user's info
Visual Paradigm Online Free Edition


registration
login
add profile
view profile
data user
data owner
view personal details
access structure
upload medical history
view medical profile
logout

Code Implementation

1-createblock():

```
def createBlock(obj):
    '''
    Takes input from input file or console.
    The block is mined.
    '''
    global usr_count

    f=open("input.txt","r")
    data_list=f.readlines()
    f_list=[]
    for x in data_list:
        if x!= '\n':
            f_list.append(x)
    i=usr_count
    usr_name = f_list[4*i].strip('\n')
    usr_med_rep = f_list[4*i+1].strip('\n')
    usr_permitted = f_list[4*i+2].strip('\n')
    usr_age = f_list[4*i+3].strip('\n')


    '''
    usr_name = f_list[4*i].strip('\n')
    usr_med_rep = f_list[4*i+1].strip('\n')
    usr_permitted = f_list[4*i+2].strip('\n')
    usr_age = f_list[4*i+3].strip('\n')
    '''


    if (i==0):
        prev_hash="0"
    else:
        prev_hash=obj[i-1].curr_hash

    usr_name_en=(hl.sha256(usr_name.encode())).hexdigest()
    print(usr_name," : ",usr_name_en)


    obj[i].constructor(prev_hash,usr_name, usr_name_en, usr_age, usr_med_rep, usr_permitted)
    det_str=obj[i].prev_hash+obj[i].name+str(obj[i].age)+obj[i].med_rep+obj[i].permitted_users

    concat_list[i]=det_str
    obj[i].curr_hash=obj[i].mineBlock(det_str, 2)
    current_hash[i]=obj[i].curr_hash

    i+=1
    usr_count=i
```

2-mineBlock():

```
def mineBlock(self, in_str, diff_level):
    '''
    Mines the block using the input data string.
    Puzzle difficulty level is set.
    Puzzle is solved.
    Hashed string is returned.
    '''
    print("In mine: ")
    nonce = 0
    src_str=in_str
    src_str+=str(nonce)
    src_str=(hl.sha256(src_str.encode())).hexdigest()
    tgt=""
    for i in range(diff_level):
        tgt+=str(0)

    while (src_str[:diff_level] != tgt):
        src_str=in_str
        nonce+=1
        src_str+=str(nonce)
        print("Mining: ", nonce)
        src_str=(hl.sha256(src_str.encode())).hexdigest()
    print("Key (nonce value): ", nonce)
    print("Block mined successfully! \n")
    return src_str
```

3-updateSelf():

```
def updateSelf(obj):
    global usr_count
    key=input("Enter your key to update:\n")
    key_authen,ind=authenticate_key(obj, key)
    if(key_authen==False):
        print("Invalid key! Please try again _/\_\n")
    else:
        usr_in=input("Update health condition(s)? (y/n)\n")
        if(usr_in=="y" or usr_in=="Y"):
            report_up=input("Enter updated health condition(s):\n")
        else:
            report_up=obj[ind].med_rep

        usr_in=input("Update age? (y/n)\n")
        if(usr_in=="y" or usr_in=="Y"):
            age_up=input("Enter updated age:\n")
        elif(usr_in=="n"or usr_in=="N"):
            age_up=obj[ind].age

        usr_in=input("Update permitted names? (y/n)\n")
        if(usr_in=="y" or usr_in=="Y"):
            permitted_names_up=input("Enter updated permissable names:\n")
        else:
            permitted_names_up=obj[ind].permitted_users

        i=usr_count
        prev_hash=obj[i-1].curr_hash

        obj[i].constructor(prev_hash, obj[ind].ac_name ,obj[ind].name, age_up, report_up, permitted_names_up)
        det_str=obj[i].prev_hash+obj[i].name+str(obj[i].age)+obj[i].med_rep+obj[i].permitted_users
        concat_list[i]=det_str
        obj[i].curr_hash=obj[i].mineBlock(det_str, 2)
        current_hash[i]=obj[i].curr_hash
        print("Details updated\n")
        i+=1
        usr_count=i
```

4-viewSelf():

```
def viewSelf(obj):
    key=input("Enter your key to view details:\n")

    key_authen,ind_match=authenticate_key(obj, key)
    if(key_authen==False):
        print("Invalid key! Please try again _/\_\n")
    else:
        print("........Data.........")
        print("Name: ", obj[ind_match].ac_name)
        print("Age: ", obj[ind_match].age)
        print("Health condition(s): ", obj[ind_match].med_rep)
        print("Permitted users: ", obj[ind_match].permitted_users)
```

5-viewOtherUser():

This function uses verify_transaction() and authenticate_key() functions and the respective images are below.

```
def viewOtherUser(obj):
    name_match_index=-1

    patient_name=input("Enter patient's encrypted name:\n")

    for l in range(usr_count):
        if(patient_name==obj[l].name):
            name_match_index=l

    if(name_match_index==-1):
        print("Invalid key! Please try again _/\_\n")
    else:
        acc_names=(obj[name_match_index].permitted_users).split(",")
        print("*Shown for testing purposes*\nPermitted names: ",acc_names)

        usr_name=input("Enter your name:\n")
        if usr_name not in acc_names:
            print("Sorry, your name is not in permitted list")
        else:
            print("Executing zero knowledge proof")
            en_int=name_encoder(usr_name)
            status=False
            while(status!=True):
                prime,random=prover_input()
                ver_in=verifier_input()
                status=obj[name_match_index].verify_transaction(en_int,ver_in,prime,random)

            if status==True:
                print("........Data.........")
                print("Name: ", obj[name_match_index].ac_name)
                print("Age: ", obj[name_match_index].age)
                print("Health condition(s): ", obj[name_match_index].med_rep)
                print("Permitted users: ", obj[name_match_index].permitted_users)
```

6-verify_transaction():

```
def verify_transaction(self,enc_int,ver_in,p,r):
    print("Transaction Verification in Progress: ")
    gen_prime=generator(p)
    b=1
    h,val1,val2=1,1,1
    usr_b = ver_in
    s=(r+ver_in*enc_int)%(p-1)

    for k in range(enc_int):
        b=b*gen_prime
        b%=p
        print("[---|---]")

    for k in range(r):
        h*=gen_prime
        h%=p

    for i in range(s):
        val1*=gen_prime
        val1%=p

    val2 = (h*(pow(b,usr_b)))%p

    if (val1==val2):
        return True
    else:
        return False
```

7-authenticate_key():

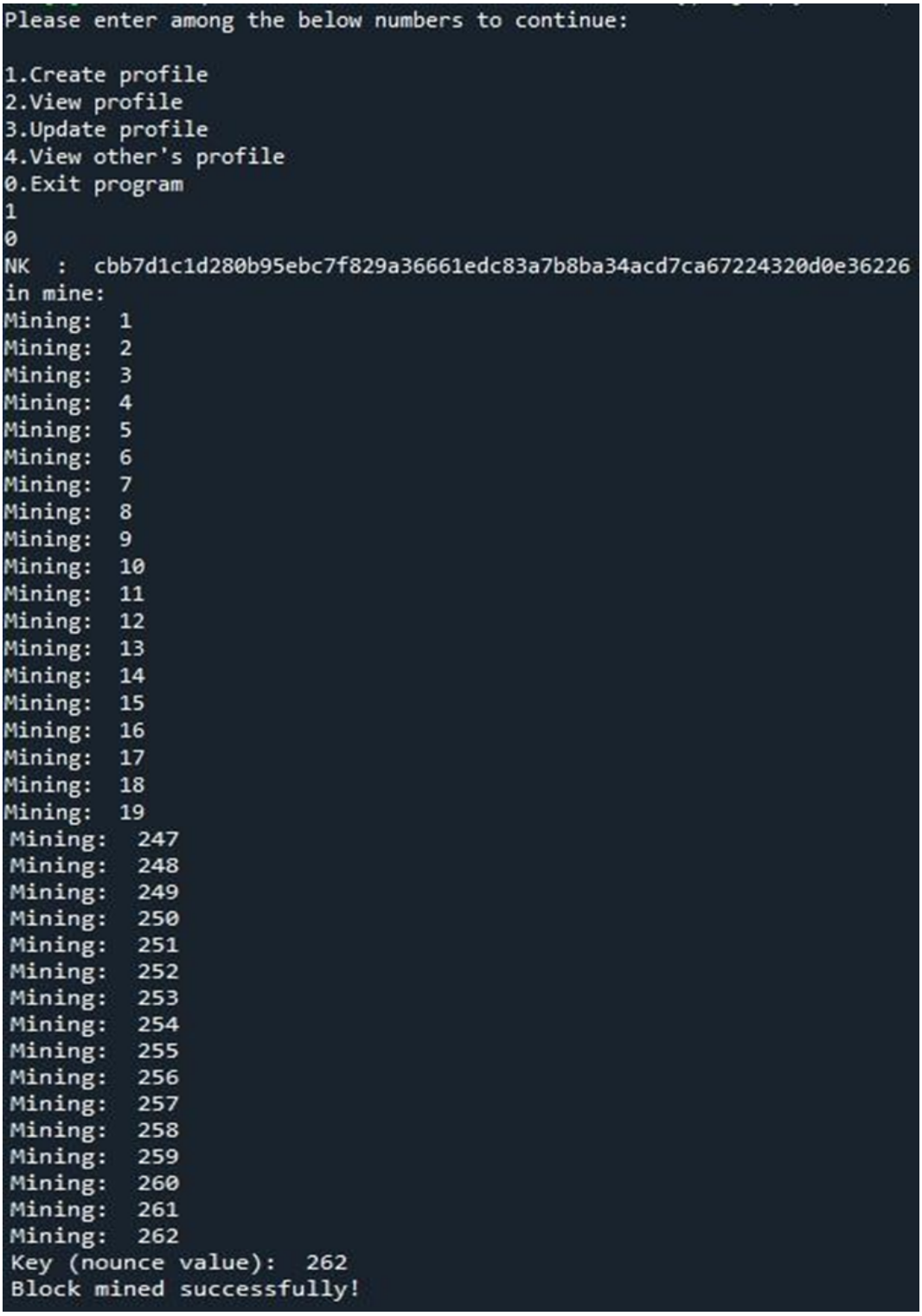
```
def authenticate_key(obj, key):
    key_flag=False
    for k in range(usr_count):
        det_str=concat_list[k]+str(key)

        if((hl.sha256(det_str.encode())).hexdigest()==current_hash[k]):
            hash_match_index=k
            key_flag=True
            break

    for l in range(usr_count):
        if(obj[l].name==obj[hash_match_index].name):
            name_match_index=l

    if(hash_match_index!=name_match_index or key_flag==False):
        return False, None
    else:
        return True, hash_match_index
```

Working Screenshots



```
Please enter among the below numbers to continue:

1.Create profile
2.View profile
3.Update profile
4.View other's profile
0.Exit program
2

Enter your key to view details:
262
........Data.........
Name:  NK
Age:  20
Health condition(s):  Cold,Cough
Permitted users:  Punith,Manoj
```

```
Please enter among the below numbers to continue:

1.Create profile
2.View profile
3.Update profile
4.View other's profile
0.Exit program
3

Enter your key to update:
262

Update health condition(s)? (y/n)
y

Enter updated health condition(s):
Headache,Leg cramps

Update age? (y/n)
n

Update permitted names? (y/n)
n
in mine:
Mining:  1
Mining:  2
Mining:  3
Mining:  4
Mining:  5
Mining:  6
Mining:  7
Mining:  8
Mining:  9
Mining:  10
```

```
Please enter among the below numbers to continue:

1.Create profile
2.View profile
3.Update profile
4.View other's profile
0.Exit program
4

Enter patient's encrypted name:
b3e82ca42bbc2b5cd7ead94206edcc078a1f8d681201f7ccd4251a3904fa0bcb


Enter your name:
NK
Executing zero knowledge proof

Hey Prover!
Enter a prime number:
11

Hey Prover!
Enter a random number b/w 2 and(prime -1)
7

Hey Verifier!
Enter a random bit 0 or 1
1
Verifying transaction:
Inside gcd() 2

Inside gcd() 2
```

```
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
[_|_]
.........Data.........
Name:  Punith
Age:  22
Health condition(s):  Flu,Cold
Permitted users:  NK
```