BLOCKCHAIN SECURITY

- **OSAMA SAGHHER --BCS211074**
- **ALISHBA NAEEM --BCS211051**
- **OBAID ULLAH --BCS211055**

25Jan,2022

—

OOP

—

SIR SIRAJ RATHORE

Contents

Abstract.

A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending.

1.Intro

Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

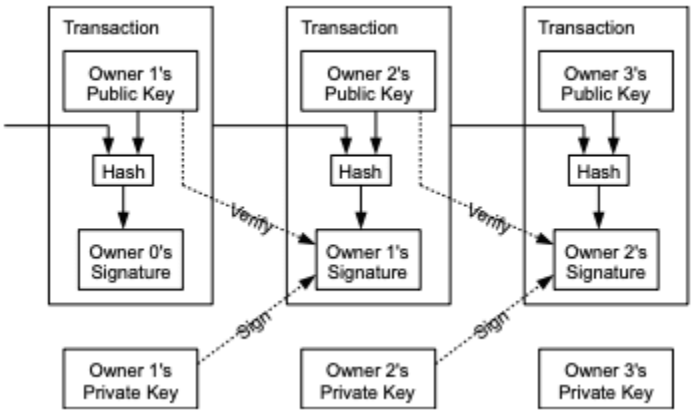

Figure 1:Transactions Architecture

2. Chain the blocks

The blockchain is supposed to be a collection of blocks, we can store all the blocks in a **tuple** due to the immutability, but that might not be enough. The solution is chaining of blocks, every block has the hash value of its previous block the point is what about the first block in the blockchain, well the first block is created by Owner of the block chain, known as the genesis block.

Now if someone tries to modify the content of a block, then the block ends up to new position in the blockchain hence its previous block hash is changed, this can be observed by the nodes.

There are in total 7 functions in this program,

1. Let's look at our class's, we have two class's. One class represent the variables, like amount, sender, receiver, etc. Other class hams function's

```
class Node{
    public:
    int ammount;
    string sender;
    string receiver;
    double id;
    string timestamp;
    string hash;
    Node *next;

    Node()//default constructor
    {
        ammount=0;
        sender="Owner";
        receiver="Fixed by Owner";
        timestamp="NULL";
        id=rand() % 100;//Unique id for each block
        next=NULL;
    }

    Node(int n,string s, string r, string time )//constructor
    {
        ammount=n;
        sender=s;
        receiver =r;
     //   int t = stoi(time);
        timestamp=time;
        id=rand() % 100;//Unique id for each block
        next=NULL;
    }
};
class Blockchain : public Node
{
    private:
    Node *listptr,*temp;
    //ptr is used to point at the current node and temp is used for temporary usages such as
iterations , ammount storage for a small time.
    public:
    Blockchain()//
    {
```

```
        listptr=NULL;
        temp=NULL;
        //initializing the values to NULL , to avoid segmentation error
    }
    void Create(int , string, string);
    void Display();
    void HashDisplay();
    int Isempty();
    int len();
    void Insert();
    void Search();
    void ProofofWork();
    void Mining();
    void Correctblock(Blockchain, Blockchain);
}lk1,lk2 ;
```

2. Now lets go into our Create function : We pass parameter's to the function

```
void Blockchain :: Create(int num, string sender , string receiver)
{
    time_t my_time = time(NULL);
    timestamp=ctime(&my_time);

    Node *newnode = new Node(num,sender,receiver,timestamp);
    if(listptr == NULL)
        listptr = temp = newnode;
    else
        {
            temp -> next = newnode;//temp currently points towards current block
            temp = temp->next;
        }

}
```

3. We now move on Display function :

```
void Blockchain:: Display()
{
    Node * current_node = listptr ;         /*    temp points to first node*/
    if ( Isempty()==0 )
    {
            cout<<"List is empty";
    }
    else
    {
       cout<<"\n";
        while (current_node != NULL) //iterate till it reaches last block
        {
```

```
            cout <<"\n |\t"<< current_node->ammount <<" Sent to "<<current_node->receiver<<" by
"<<current_node->sender<<" ,Transaction ID "<<current_node->id<<" ,at "<<current_node-
>timestamp<<" \t| "<< " ---> ";
            current_node = current_node->next;         /*    move temp to the next node */
        }
        cout<< "NULL";
        cout<<"\n";
    }
}
```

4.We now move onto Hash Display Function , We generally only work with hash value in blockchain , but for verification and simplicity , i had used normal display.

```
void Blockchain:: HashDisplay()
{
    Node * current_node = listptr ;         /*    temp points to first node*/
    if ( Isempty()==0 )
    {
            cout<<"List is empty";
    }
    else
    {
        std::hash<string> shashVal;
        string final =current_node->sender+current_node->receiver;//hash value of Sender and
receiver
        std::hash<int> ihashVal;
        int fin=current_node->id;//hash value of id
       cout<<"\n";
        while (current_node != NULL) //iterate till it reaches last block
        {
            cout <<"\n |\t"<< current_node->ammount <<" Hash value of transaction
"<<shashVal(final)+ihashVal(fin)<< " ,at time "<<current_node->timestamp<<" \t| "<< " ---> ";
            current_node = current_node->next;         /*    move temp to the next node */
        }
        cout<< "NULL";
        cout<<"\n";
    }
}
```

5.len function is used ahead , to find the longest blockchain to add our block in

```
int Blockchain ::len()
{
    int count=0;
    Node * current_node = listptr ;         /*    temp points to first node*/
    if ( Isempty()==0 )
    {
            cout<<"List is empty";
    }
    else
    {
        while (current_node != NULL)
        {
            count+=1;
            current_node = current_node->next;         /*    move temp to the next node */
        }
    }
    return count;
}
```

6.Search Function is Used to find a transaction in blockchain using transaction ID

```
void Blockchain :: Search()
{
    int val;
    cout<<"\n Enter  ID  to search  in List \n";
    cin>>val;
    int find=0;
    int count=0;
    Node *current_node=listptr;
    while (current_node->next!=NULL)
    {
        if (current_node->id==val)
        {
            find=1;
            break;
        }
        count+=1;
        current_node=current_node->next;

    }
    if(find ==1)
         cout<<"\n Transaction is  present at "<<count;
     else
         cout<<"\n Transaction not present";
}
```

7. Proof of work is done by checking some kind of pattern , in the blocks of a particular Blockchain , such as maybe in Blockchain A , the first two values of hash's of each block are 00

```
//Assuming that there are two Blockchains , one starts with 1 while another one starts with zero
assuming
```

```
    Node * current_node = listptr ;         /*    temp points to first node*/
    if ( Isempty()==0 )
    {
            cout<<"List is empty";
    }
    else
    {
       cout<<"\n";
        while (current_node != NULL)
        {
            std::hash<string> shashVal;
            string final =current_node->sender+current_node->receiver;
            if (final[0]==1)
            {
                cout<<"Block A";
            }
            else if (final[1]==0)
            {
                cout<<"Block B";
            }
            else
                cout<<"Block no pattern";
                break;
        }
    }
}
```

8.Mining this process is done by nodes in a blockchain , to verify transaction , this process takes care of Double spending issue , as once transaction is verified then only is it added on the blockchain

```
void Blockchain :: Mining()
{
    /* Mining means transaction verification from the node server ,
        Now this can't be done in Data Structures , This is the double spending problem
        What the best way we can do in DS as of now is put it in an order

        Assuming that two transactions happen from the same sender named ash ,
        We put those two transactions in an order to check if he has balance to do these two
transfer or not ,
        most importantly that he is not sending the same  money to both the receivers

        Assuming that they both have 100 rupees right now in their accounts , So they can't make a
transaction more than 100

    */
    Node *current_node=listptr;
    string senderMin;
    cout<<"\n Enter  Sender name  to check  \n";
    cin>>senderMin;
    int find=0;
    int count=0;
```

```
    while (current_node->next!=NULL)
    {
        if (current_node->sender==senderMin)
        {
            if(current_node->ammount <100 )
            {
                count+=1;
                cout<<"Not enough credit , required credit: \t"<<current_node->ammount -100;
            }
            break;
        }
        current_node=current_node->next;

    }
    if (count==0)//If transactions is valid the
    {
            Node *newnode = new Node(current_node->ammount,current_node->sender,current_node-
>receiver,current_node->timestamp);

    }


    Correctblock(lk1,lk2);
}
```

9.Now when the transaction is verified , now the question is Which blockchain should we add the block to , the block is added to the longest Blockchain on the node , this is done by Correctblock function.

```
void Blockchain::Correctblock(Blockchain l1, Blockchain l2)
{
    int x,y;
    x=l1.len();
    y=l2.len();
    if(x>y)
    {
        cout<<"Blockchain A is longer hence , block will be added into ";
        l1.Display() ;
    }
    else
    {
        cout<<"Blockchain B is longer hence , block will be added into ";
        l2.Display();

    }

}
```

OUTPUT:


Welcome to Blockchain implementation by Hridyesh Press
1.Add Blocks in block chain
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
6.Proof of Work
7.Mining
1
Enter the ammount ,
Sender Digital signature
Receiver Digital signature
89
ash
misty
To continue press 0
0
Press
1.Add Blocks in block chain
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
6.Proof of Work
7.Mining
2
| 100 Sent to Choice of Owner by Owner ,Transaction ID 15 ,at Thu Apr 2 00:52:22 2020
| --->
| 89 Sent to misty by ash ,Transaction ID 93 ,at Thu Apr 2 00:52:29 2020
| ---> NULL
To continue press 0
0
Press
1.Add Blocks in block chain
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
6.Proof of Work
7.Mining
3
| 100 Hash value of transaction 14779909488103273175 ,at time Thu Apr 2 00:52:22 2020
| --->
| 89 Hash value of transaction 4295105400813960203 ,at time Thu Apr 2 00:52:29 2020
| ---> NULL
To continue press 0
0
Press
1.Add Blocks in block chain
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
6.Proof of Work
7.Mining
6
Block no pattern
To continue press 0
0
Press
1.Add Blocks in block chain
2.Display the blocks without hash function
3.Display the HASH Blocks
4.Length of Block chain
5.Search Transaction
6.Proof of Work
7.Mining
7
Enter Sender name to check
ash
List is emptyList is emptyBlockchain B is longer hence , block will be added into List is empty
To continue press 0