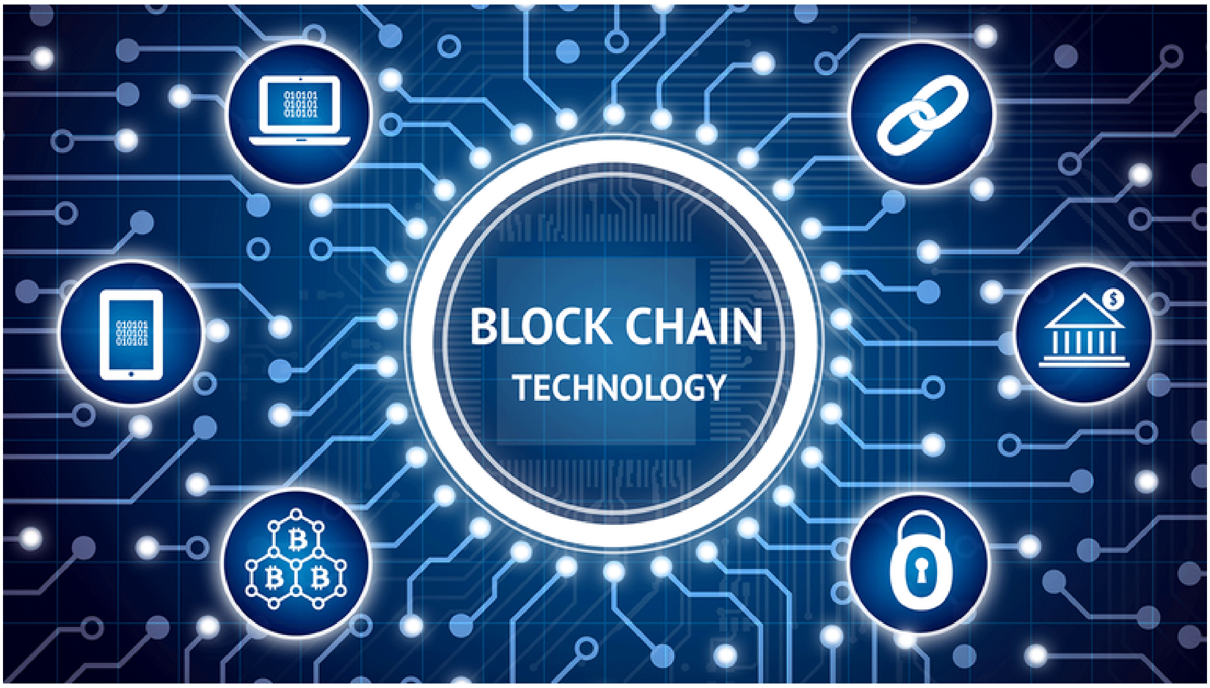
27016005 - CS3BC20

BlockChain Computing

Implementation of an offline CryptoCurrency



Date of Completion: 19/03/2021

Actual hours spent on assignment: 40

Assignment Evaluation:

1. Assignment was interesting, fun, informative, engaging. Probably the best assignment throughout this degree. Lecturer put in an amazing amount of effort to make sure the students were understanding content and requirements.
2. Resources supplied were detailed, thorough, useful and applicable. Lecture content was related, helpful, interesting and applicable to the coursework. Detailed mark scheme was extremely useful and helpful. Lecturer was thorough and
3. Assignment was well defined, related to the course theory content, lecture content, practical content and did brilliantly bridging it all together in a way which effectively demonstrated the importance of the concepts.

Cannot fault this module. Taught amazingly, interesting, engaging, and the most effort from any lecturer in all the time at university. Thoroughly appreciated.

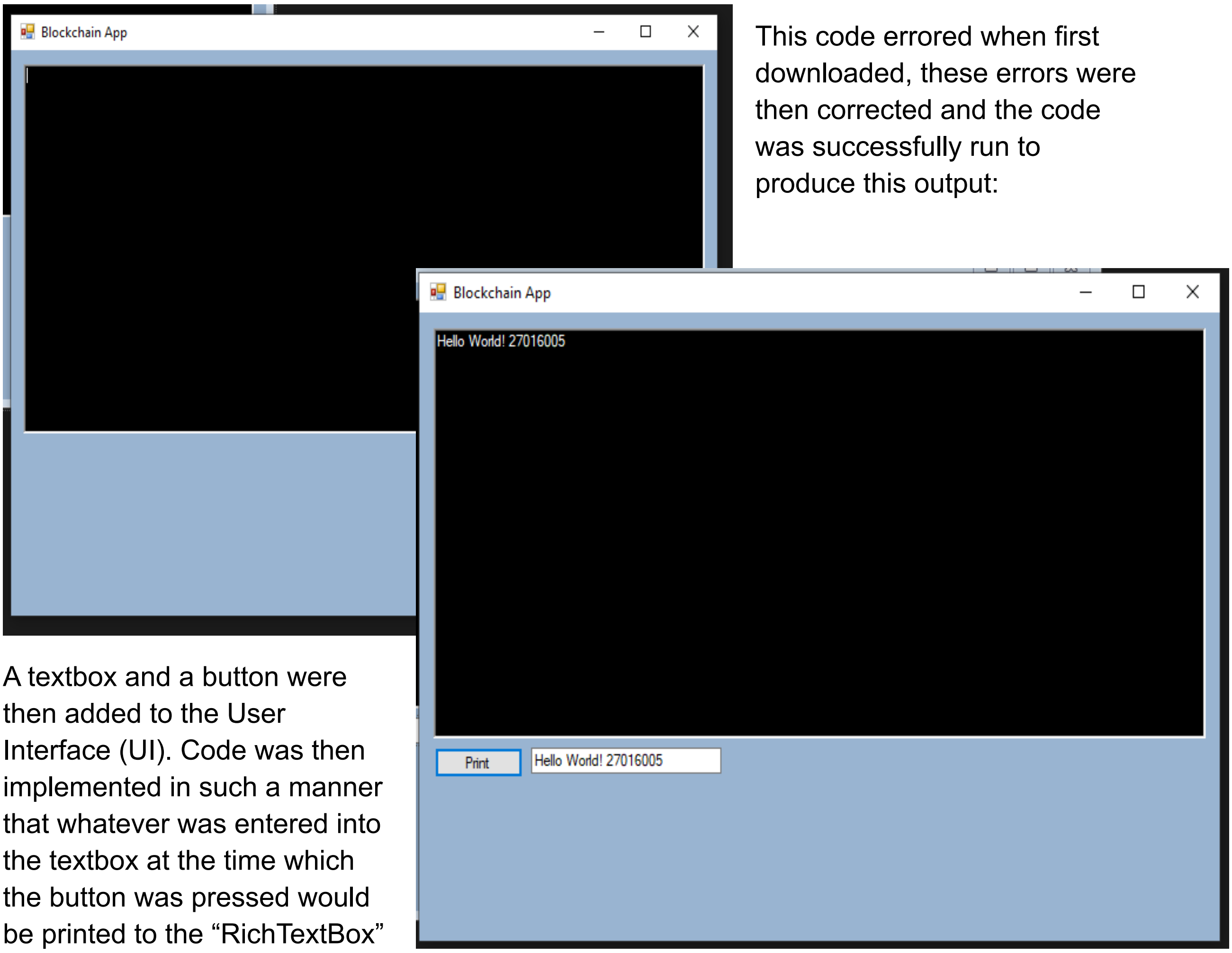
- Blockchain made of cryptographically connected Blocks
- Transaction generation – including digital signature via asymmetric encryption
- A Proof-of-Work consensus algorithm – including hashing and threading
- Validation methods to ensure the Blockchain is valid
- A basic UI that can verify the implementation of the above features

Part 1 - Project Setup & Preamble:

This Project was created in C#
Microsoft Visual Studio Community 2019 Version 16.2.5
Using the .NET Framework Version 4.8.04084 - 2019
This code was maintained in a GitHub Repository:
https://github.com/Dannybrush/BlockChainProject
Then ported to a CSGitLab Repository for submission.

A Foundation and Skeleton for the code was provided by the University of Reading Computer Science Department. This is available on the GitHub Repository:
https://github.com/Dannybrush/BlockChainProject/tree/main/Uni-Resources/CODE/Extracted/Practical_Exercises_Implementation_Environment

This code errored when first downloaded, these errors were then corrected and the code was successfully run to produce this output:



A textbox and a button were then added to the User Interface (UI). Code was then implemented in such a manner that whatever was entered into the textbox at the time which the button was pressed would be printed to the “RichTextBox” (Large black output area).

This was achieved using event handlers:

```
private void Button1_Click(object sender, EventArgs e)
{
    richTextBox1.Text = textBox1.Text;
}
```

Resulting in a Basic Hello World Program.

Part 2 - Blocks & the Blockchain

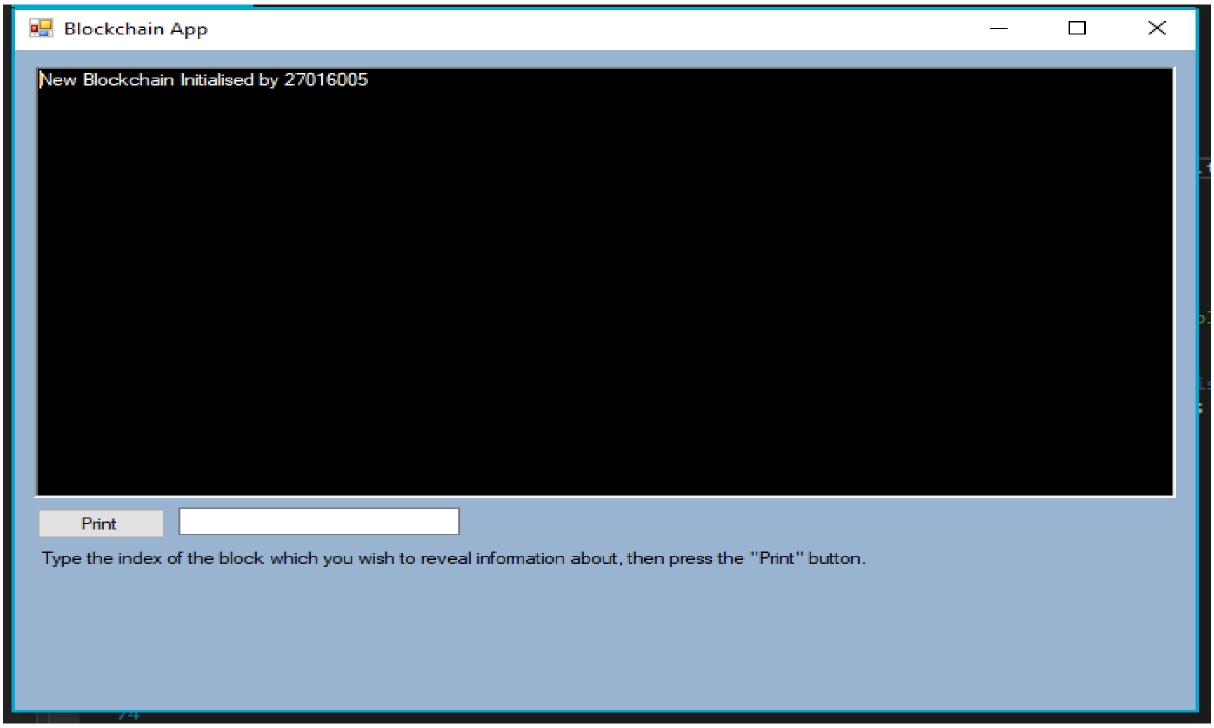
The code used for part 1 contains the classes:

- HashTools.cs
- Wallet.cs
- BlockchainApp.cs (+BlockchainApp.Designer.cs)
- program.cs

More Classes were added for Part 2:

- Blockchain.cs
- Block.cs

A list of type Block was added to the Blockchain class to create a way to store the "Chain" of Blocks. The Blockchain was then initialised.



Fields were then added to the Block.cs class. In most programming languages this would take form of a variety of Public, Private, Protected, and possibly even global variables, C# utilises the ability of creating properties. This has been implemented for the following properties in the Block Class:

- TimeStamp -> an indicator of when the block was initialised
  - Used for storing time, and uses DateTime.Now function which returns a DateTime
- Index -> the position of the block in the blockchain
  - Position therefore always integer → type int
- Hash -> the hash of the block
- PrevHash -> the hash of the previous block
  - 64 character Hexadecimal string produced by hashing function

```
public DateTime timeStamp{ get; set; }
public int index { get; set; }
public string hash { get; set; }
public string prevHash { get; set; }
```

Three Constructors have then been created for the Block class. One Constructor which takes no input parameters, one which is overloaded with the input of the previous block, and one which just takes the index and hash of the previous block. Implementation of both of these latter two constructors is possibly redundant. The "DateTime.Now" function retrieves the current time.

```
// Constructor which is passed the previous block
public Block(Block lastBlock) {
    this.timeStamp = DateTime.Now;
    this.index = lastBlock.index;
    this.prevHash = lastBlock.hash;
}
```

```
// Constructor which is passed the index & hash of previous block
public Block(int lastIndex, string lastHash) {
    this.timeStamp = DateTime.Now;                  // new time
    this.index = lastIndex + 1;                     // increment on last block
    this.prevHash = lastHash;                           //needs fixing
}
```

Hashing & Creation of Genesis Block:

A genesis block is the initial block in a blockchain, often quoted as "Block 0"

```
// Constructor which is not passed anything
public Block() {
    // This generates the Genesis Block
    this.timeStamp = DateTime.Now;                       // new time
    this.index = 0;                                     //  First Block = 0
    this.prevHash = string.Empty;                      //   No Previous Hash
    this.hash = this.Create256Hash();                 //    Create hash from index, prevhash and time
}
```

With the Genesis block being the first block in the blockchain, it is not possible for the last block to be passed into the constructor, therefore the index must be set to 0 and the hash must be created.

Hashing:

The hashing algorithm used, used SHA256 hashing, and is obtainable from https://emn178.github.io/online-tools/sha256.html

```
SHA256 hasher;
hasher = SHA256Managed.Create();
String input = index.ToString() + timestamp.ToString() + prevHash;
Byte[] hashByte = hasher.ComputeHash(Encoding.UTF8.GetBytes((input)));

String hash = string.Empty;

foreach (byte x in hashByte)
{
        hash += String.Format("{0:x2}", x);
}
return hash;
```

When implemented it looks like this:

```
private string Create256Hash() { // i think this can be simplified // Simplified Heavily is "this" needed?
    SHA256 hasher;
    hasher = SHA256Managed.Create();
    String input = this.index.ToString() + this.timeStamp.ToString() + this.prevHash;
    Byte[] hashByte = hasher.ComputeHash(Encoding.UTF8.GetBytes((input)));

    String hash = string.Empty;

    foreach (byte x in hashByte)
    {
        hash += String.Format("{0:x2}", x);
    }
    return hash;
```

A default constructor was then added to the "Blockchain" class, this constructor calls the Genesis Block Constructor from the Block Class, and then the genesis block to the blockchain list created.

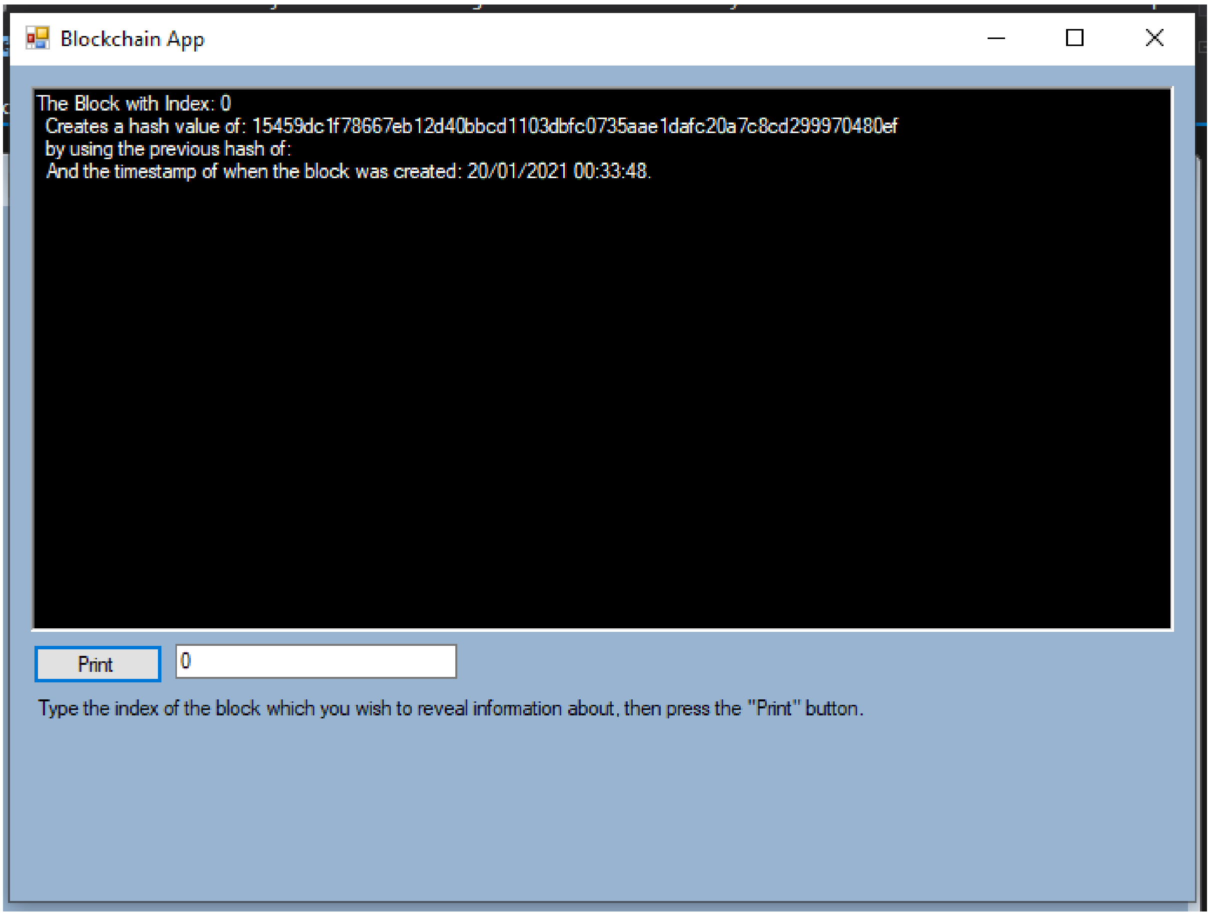
```
public Blockchain()
{
    Block genesis = new Block();
    Blocks.Add(genesis);
    // Blocks.Add(new Block());            // Another way of writing the same thing

}
```

Printing block to UI :

A function was then added to Block.cs which returns all the details about a given block in a string format. A further function was added to Blockchain.cs which prints the string to the output region given the strings index in the blockchain. The previously created buttons were repurposed for this effect.

```
public string ReturnString() {
    return (" The Block with Index: " + this.index +
            "\n Creates a hash value of: " + this.hash +
            "\n By using the previous hash of: " + this.prevHash +
            "\n And the timestamp of when the block was created: " + this.timeStamp + ". "
            );
}

public string BlockString(int index) {
    return (Blocks.ElementAt(index).ReturnString());
}
```

This prints the details of the block with the index inputted into the UI Textbox.


Blockchain App
The Block with Index: 0
Creates a hash value of: 15459dc1f78667eb12d40bbcd1103dbfc0735aae1dafc20a7c8cd299970480ef
by using the previous hash of:
And the timestamp of when the block was created: 20/01/2021 00:33:48.
Print
0
Type the index of the block which you wish to reveal information about, then press the "Print" button.

Part 3 - Transactions & Digital Signatures

Wallet Creation:

A new button is added to the UI, with the functionality of generating a new wallet. Two Textboxes are added which show the Public and Private keys on the UI, these also have labels to go with them. A "Validate keys" button was also added which confirms if the public and private keys in the respective textboxes are a matching pair. If the pair matches, the button turns green in colour. The methods implemented behind this functionality were provided in the original code skeleton as part of the wallet.cs class. The public key is 64 characters long, and the private key is 32 characters long.

Transactions:

From here we add transactions to the blockchain. This is done by first creating a transaction.cs class, with all of the required properties:

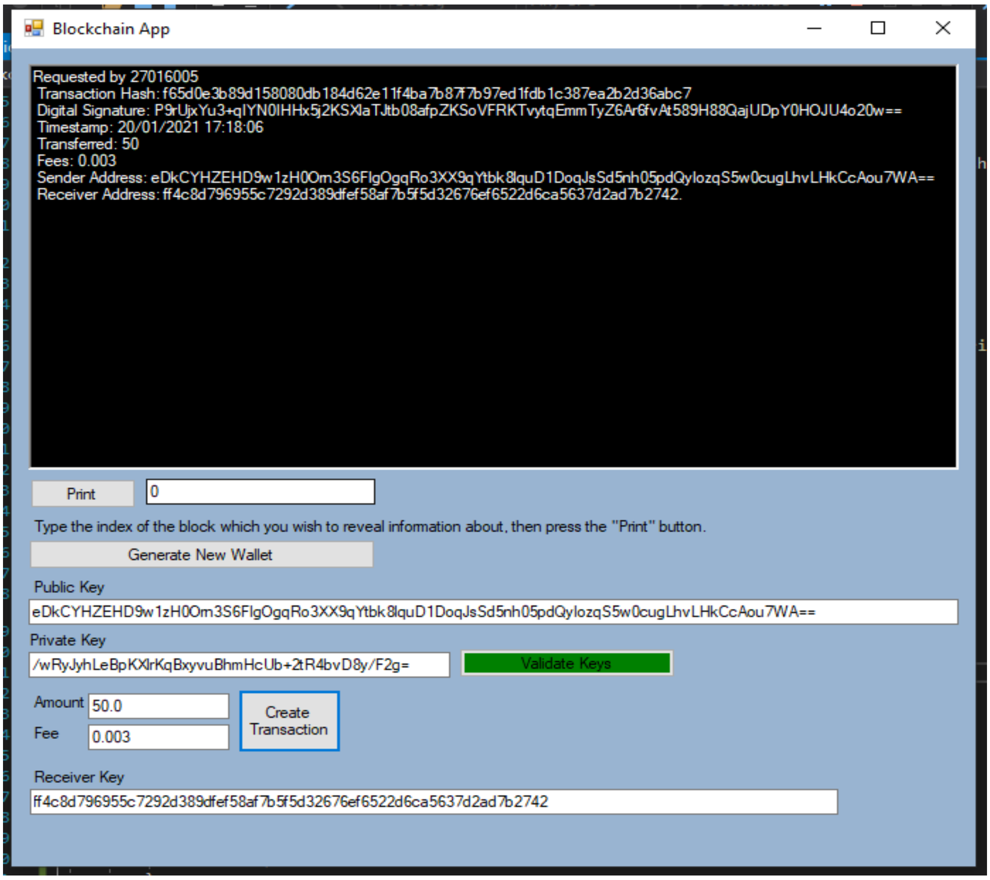
```
class Transaction
{
    public string Hash{get; set;} // The Hash
    public string Signature{get; set;} // Hash signed with private key of sender
    public string SenderAddress{get; set;} // Public key of sender
    public string RecipientAddress{get; set;} // public key of reciever
    public DateTime TimeStamp{get; set;} // Time of transaction
    public float Amount{get; set;} // amount sent  // decimal Amount = 2.1M (suffix M needed )
    public float Fee{get; set;} // the fee added to transaction
```

From here a constructor is created for the transaction class, accepting both public keys, the amount, the fee and the sender's key as input parameters. These are assigned to their relative properties, the timestamp is then generated, the transaction hash is then calculated and finally the hash is signed using a signature which can be created from the static method "CreateSignature()" in the wallet.cs class. Note: The senderPrivate key is not stored.

```
public Transaction(string senderPublic, string senderPrivate, string recipientPublic, float amount, float fee)
{
    this.TimeStamp = DateTime.Now;

    this.SenderAddress = senderPublic;

    this.RecipientAddress = recipientPublic;

    this.Amount = amount;

    this.Fee = fee;

    this.Hash = Create256Hash();

    this.Signature = Wallet.Wallet.CreateSignature(SenderAddress, senderPrivate, Hash);

}
```

Two more textboxes and labels were then added to the UI: Amount and Fee respectively. Another Textbox and label was added for the receiver address. A "Create Transaction" button was also added.

When the “Create transaction” button is clicked: Amount, Fee, Public Key, Private Key, and receiver key are passed through to create a new instance of transaction. The Transaction is generated, and then printed. The transaction is also added to a transaction pool, which is a list of pending transactions stored in the blockchain.




```
class Blockchain
{
    1 reference
    public int maxBlock { get => this.Blocks.Count; }                        // Maximum number of transactions per block
    public List<Block> Blocks = new List<Block>();                          // List of block objects forming the blockchain
    public List<Transaction> TransactionPool = new List<Transaction>();    // List of pending transactions to be mined
```

```
public void add2TPool(Transaction Trans)
{
    TransactionPool.Add(Trans);
}
```

```
private void CreateTransBtn_Click(object sender, EventArgs e)
{
    Transaction transaction = new Transaction(pubKeyTBox.Text, privKeyTBox.Text, recieverKeyTBox.Text, Convert.ToSingle(amountTBox.Text), Convert.ToSingle(feeTBox.Text));
    blockchain.add2TPool(transaction);
    outputToRichTextBox1(transaction.ReturnString());
}
```

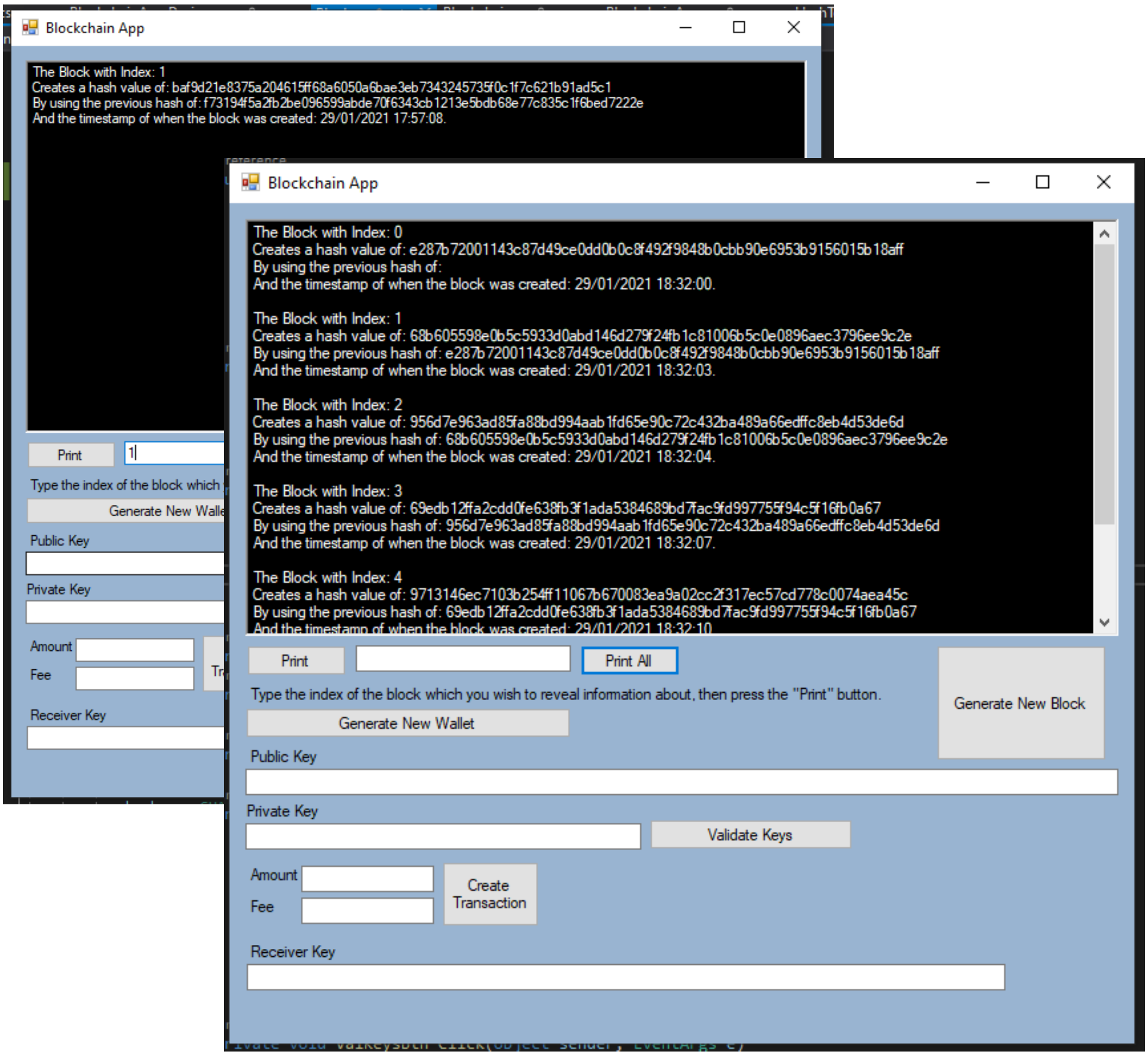
Part 4 - Consensus algorithms

Generating new blocks:

A button was added to the UI with the text "Generate New Block", this button adds a new block to the blockchain, passing the previous block as the argument for the constructor.

```
private void BlockGenBtn_Click(object sender, EventArgs e)
{
    Block block = new Block(blockchain.GetLastBlock());
    blockchain.add2Block(block);
}
```

Methods were implemented to print all the blocks in a block chain in its entirety. This was in addition to the functionality of printing a chosen specified block.



Adding Transactions to blocks:

A List for holding transactions was adding to the block.cs class.

```
12 references
public List<Transaction> transactionList { get; set; }            // List of transactions in this block
```

A function was then implemented which A new button was then again added to the UI to Generate Blocks which take transactions from the transaction pool. The blocks cannot take all the blocks so a finite limit was added - currently this is set to an arbitrary value of 5.

The Block class constructor was then adapted to accept a list of transactions as well as the previous block. The pending transaction pool is passed into the constructor, where the block retrieves a maximum of 5 transactions from the pool. These transactions are then purged from the pending transaction pool.

```
private void Button1_Click(object sender, EventArgs e)
{
    Block block = new Block(blockchain.GetLastBlock(), blockchain.retTPool());
    blockchain.purgeTPool(block.transactionList);
    blockchain.add2Block(block);
}
```

```
public Block(Block lastBlock, List<Transaction> TPool)
{
    this.transactionList = new List<Transaction>();
    this.nonce = 0;
    this.timeStamp = DateTime.Now;
    this.index = lastBlock.index + 1;
    this.prevHash = lastBlock.hash;
    this.addFromPool(TPool);
    this.hash = this.Create256Hash();    //    Create hash from index, prevhash and time
}
```

```
public void addFromPool(List<Transaction> TP )
{
    int LIMIT = 5;
    for (int i = 0; ((i < TP.Count) && (i < LIMIT)); i++   )
    {
        this.transactionList.Add(TP.ElementAt(i));
    }
}
```

Proof-of-Work:

Proof-of-work (PoW) is the standardised consensus algorithm for BitCoin and many other existing Cryptocurrencies. PoW dictates that for a block to be added to the blockchain it must have a hash which satisfies a given difficulty threshold. The chosen default difficulty level for this implementation was 4, this is because it could be calculated in a reasonable time while maintaining consistent results.

For the proof of work algorithm to work, the block must have a "nonce" field else every time the hash is created, it would be identical, by including a nonce (Number Only used oNCE) and changing this every time the hashing algorithm does not produce a hash of sufficient difficulty different hashes can be created.

Due to the hexadecimal nature of the hashing algorithm, each increase in difficulty level is an increase, factor 16 of workload. The Difficulty level is determined by a certain criteria in which the hash must meet. In this instance, leading “0” values have been chosen, thus a difficulty level of 4 means a hash must start with “0000” and difficulty level of 2 means it must start with “00”. Difficulty has been included in the hash composition.

Proof of work, Nonce Gen, Difficulty levels, Rewards and Fees.

```
private string Create256Mine()
{
    string hash =  "";          // could also do Createhash here, then no need to nonce--
    string diffString = new string('0', this.difficulty);
    while (hash.StartsWith(diffString)== false)
    {
        hash = this.Create256Hash();
        this.nonce++;
    }
    this.nonce--;
    return hash;
}
```

Rewards & Fees:

After a block is successfully mined, the miner receives a reward. This is a flat reward given in the form of a transaction from a “Reward” wallet. Rewards are the most obvious motivation and incentive for blockmining. The implemented reward system is a hardcoded fixed value, however a similar option as bitcoin could be implemented where the reward changes with every x blocks mined. Alternatively rewards could be based on number of calculations to make a block, or the time taken per block.

```
public Transaction createRewardTransaction(List<Transaction> transactions)
{
    double fees = transactions.Aggregate(0.0, (acc, t) => acc + t.Fee); // Sum all transaction fees
    return new Transaction("Mine Rewards", "", minerAddress, (this.reward + fees), 0); // Issue reward as a transaction in the new block
}
```

Evidence:

```
                    [BLOCK START]
Index: 1   Timestamp: 16/03/2021 21:31:51
Previous Hash: 00005d1c35ef5721f350fede214a636d6ed93c7f37afd72f83eaef78b4300605
                    -- PoW --
Difficulty Level: 4
Nonce: 47286
Hash: 0000c06f606f99f0bfede02fdac1e2ae397ec0cfb6eb897d9b747b48e26ca83f
                    -- Rewards --
Reward: 1
Miners Address: vU6OnB6OF3cDE+pZfsd7YrsmGvHRaSy7z/FzPgfHhzHR3QNIUcCaXJ8axchzlp2P12aZhk8phdJYQWEHtqPJ8g==
```

For a Series of screenshots showing the addition of blocks, creation of transactions and more etc see appendix.

Part 5 - Validation

The concept of a blockchain network revolves around the idea of a “trustless” network where no node in the network can be trusted, and only the system can be trusted.

Blockchain Continuity Validation:

As the blockchain is a linear structure of connected blocks, it is possible to check the integrity by checking that for the entire blockchain, the previous hash is equal to the hash of the previous block. This iterates through the blockchain making sure that for the entire chain, the “PreviousHash” of the current block is equal to the “CurrentHash” of the previous block.

Block & MerkleRoot Validation

Block Validation :

Rehashes the block using all of the properties, checks to see if the hash matches the original hash.

```
// Check validity of a blocks hash by recomputing the hash and comparing with the mined value
3 references
public static bool ValidateHash(Block b){
    string rehash =string.Empty ;
    /* if (b.THREADING == true){
        rehash = b.ThreadedMine();
     }
     else { rehash = b.Create256Hash(); }*/
    rehash = b.Create256Hash();
    Console.WriteLine("Rehash: " + rehash + " --> Hash: " + b.hash);
    return rehash.Equals(b.hash);
}
```

```
Hash for block 1
Rehash: 000a8d3c0e54416778c865772ca94cde881a28e54b6be0e2f57f765f4a84e2de --> Hash: 000a8d3c0e54416778c865772ca94cde881a28e54b6be0e2f57f765f4a84e2de
Hash for block 2
Rehash: 001a6fe50f93992e6518b308a3f971b44a874d91ff51023eff5977d53428db40 --> Hash: 001a6fe50f93992e6518b308a3f971b44a874d91ff51023eff5977d53428db40
```

Merkle Root Validation:

The merkle root algorithm is used to iteratively combine the transaction hashes for a block until only one hash remains.

- A Block has five transactions within: T_1, T_2, T_3, T_4, and T_5.
- The hash of T_1 and T_2 are combined to make $H_{1,2}$
- The hash of T_3 and T_4 are combined to make $H_{3,4}$
- The hash of T_5 is left untouched for now to make H_5
- Now the hashes $H_{1,2}$ and $H_{3,4}$ are combined to make $H_{1,2,3,4}$
- Finally, the hashes $H_{1,2,3,4}$ and H_5 are combined to make $H_{1,2,3,4,5}$
- $H_{1,2,3,4,5}$ is the final hash, and therefore it is the Merkle root.

```
2 references
public static string MerkleRoot(List<Transaction> transactionList) {

    // X => f(X) means given X return f(X)
    List<String> hashlist = transactionList.Select(t => t.Hash).ToList(); // Get a list of transaction hashlist for "combining"
    // Handle Blocks with...
    if (hashlist.Count == 0) // No transactions
    {
        return String.Empty;
    }
    else if (hashlist.Count == 1) // One transaction - hash with "self"
    {
        return HashCode.HashTools.combineHash(hashlist[0], hashlist[0]);
    }
    while (hashlist.Count != 1) // Multiple transactions - Repeat until tree has been traversed
    {
        List<String> merkleLeaves = new List<String>(); // Keep track of current "level" of the tree

        for (int i = 0; i < hashlist.Count; i += 2) // Step over neighbouring pair combining each
        {
            if (i == hashlist.Count - 1)
            {
                merkleLeaves.Add(HashCode.HashTools.combineHash(hashlist[i], hashlist[i])); // Handle an odd number of leaves
            }
            else
            {
                merkleLeaves.Add(HashCode.HashTools.combineHash(hashlist[i], hashlist[i + 1])); // Hash neighbours leaves
            }
        }
        hashlist = merkleLeaves; // Update the working "layer"
    }
    return hashlist[0]; // Return the root node
}
```

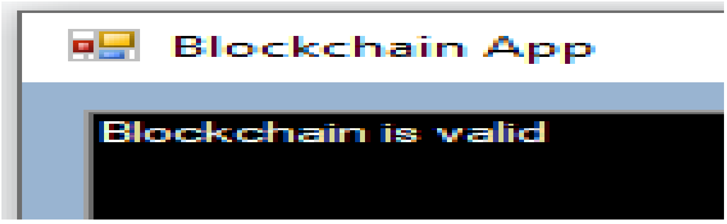
This is then validated upon clicking the “Validate” button. Where it recalculates the merkle root and checks the two match.

```
// Check validity of the merkle root by recalculating the root and comparing with the mined value
2 references
public static bool ValidateMerkleRoot(Block b){
    String reMerkle = Block.MerkleRoot(b.transactionList);
    return reMerkle.Equals(b.merkleRoot);
}
```

Transaction Verification:

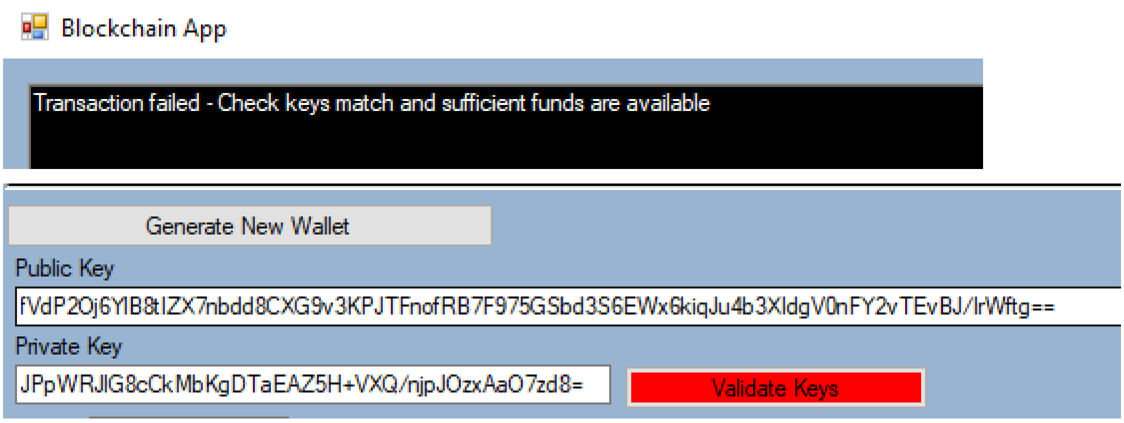
Validation has also been applied to the Transactions, so that when transactions are created, the keys must match, and the wallet must have sufficient funds to support the transaction.

```
if ((Wallet.Wallet.ValidatePrivateKey(privKeyTBox.Text, pubKeyTBox.Text)) && (blockchain.GetBalance(pubKeyTBox.Text) > Convert.ToSingle(amountTBox.Text))){
    Transaction transaction = new Transaction(pubKeyTBox.Text, privKeyTBox.Text, recieverKeyTBox.Text, Convert.ToSingle(amountTBox.Text), Convert.ToSingle(feeTBox.Text));
    blockchain.add2TPool(transaction);
    outputToRichTextBox1(transaction.ReturnString());
}
else
{
    outputToRichTextBox1("Transaction failed - Check keys match and sufficient funds are available");
}
```




```csharp
1 reference
private void Validate_Click(object sender, EventArgs e)
{
    // CASE: Genesis Block - Check only hash as no transactions are currently present
    if (blockchain.Blocks.Count == 1)
    {
        if (!Blockchain.ValidateHash(blockchain.Blocks[0])) // Recompute Hash to check validity
            outputToRichTextBox1("Blockchain is invalid - Hash ");
        else
            outputToRichTextBox1("Blockchain is valid");
        return;
    }

    Console.WriteLine(" NewBlock: " + (blockchain.Blocks.Count - 1));
    for (int i = 1; i < blockchain.Blocks.Count - 1; i++)
    {
        Console.WriteLine("Hash for block " + i);
        if (
            blockchain.Blocks[i].prevHash != blockchain.Blocks[i - 1].hash || // Check hash "chain"
            !Blockchain.ValidateHash(blockchain.Blocks[i]) ||  // Check each Block hash
            !Blockchain.ValidateMerkleRoot(blockchain.Blocks[i]) // Check transaction integrity using Merkle Root
        )
        {
            outputToRichTextBox1("Blockchain is invalid " + (blockchain.Blocks[i].prevHash != blockchain.Blocks[i - 1].hash).ToString() + "  " +
            !Blockchain.ValidateHash(blockchain.Blocks[i]) + "  " + // Check each Block hash
            !Blockchain.ValidateMerkleRoot(blockchain.Blocks[i]) + " " + blockchain.Blocks[i].nonce);// C);
            return;
        }
    }
    outputToRichTextBox1("Blockchain is valid");
}
```

Also note the Validate keys is green confirming that the keys match.



Part 6 - Assignment Tasks

Threading During Proof-of-Work

By adding a property to the block to say if threaded mining was activated or not.

```
class Block
{
    2 references
    public Boolean THREADING { get; set; } = false;                    // Boolean to tell whether to use threading or not
```

Adapting the mining function to accept an overload value containing the nonce:

```
//Version of above method to take nonce as a parameter
2 references
public String Create256Hash(int inNonce)
{
    SHA256 hasher;
    hasher = SHA256Managed.Create();
    String input = this.index.ToString() + this.timeStamp.ToString() + this.prevHash + inNonce + this.merkleRoot + this.reward.ToString();
    Byte[] hashByte = hasher.ComputeHash(Encoding.UTF8.GetBytes((input)));
    String hash = string.Empty;

    foreach (byte x in hashByte)
    {
        hash += String.Format("{0:x2}", x);
    }
    return hash;
}
```

Creating two threads, which split the workload in half, one working with odd number Nonce values and one working with even:

```
/*FOR MULTI THREADING */

2 references
public void ThreadedMine(){
    Thread th1 = new Thread(this.Mine0);
    Thread th2 = new Thread(this.Mine1);

    th1.Start();
    th2.Start();

    while (th1.IsAlive == true || th2.IsAlive == true){Thread.Sleep(1);}

    //if (this.nonce0  < this.nonce1){
    if (this.finalHash1 is null) {
        this.nonce = this.nonce0;
        this.finalHash = this.finalHash0;
    }
    else{
        this.nonce = this.nonce1;
        this.finalHash = this.finalHash1;
    }
    if (this.finalHash is null)
    {
        Console.WriteLine(this.ReturnString());
        throw new Exception("NULL finalhash" +
            " Nonce0: " + this.nonce0 +
            " Nonce1: "+ this.nonce1 +
            " Nonce: " + this.nonce +
            " finalhash0 " + this.finalHash0 +
            " finalhash1: " + this.finalHash1 +
            " NewHash: " + this.Create256Hash());

    }
    //Console.WriteLine("FINALHASH = " + finalHash);
    //return finalHash;
}
```

Two separate Nonce values were established, one for each thread. One thread handled all even value nonce, while the other handles all odd valued nonces, this means both threads could increment by two each time, halving the workload and preventing any duplication of work. When one thread found a solution, both threads were killed.

```
New Difficulty:
4
Here
added new block to chain - with 1 transactions
Time since last mine
00:00:01.1147589
New Difficulty:
5
Here
added new block to chain - with 1 transactions
Time since last mine
00:00:05.7485792
```

For difficulty 4 and 5 the non-threaded version produced these values:
4: 01.1147589 Seconds
5: 05.7485792 Seconds

```
Block index: 14
Thread 2 closed: Thread 2 found: 00
73625
Th2 mine:
00:00:00.5114923
Thread 1 closed: Thread 2 found: 00
Here
added new block to chain - with 1 t
Time since last mine
00:00:00.5516533
New Difficulty:
5
Block index: 15
Thread 2 closed: Thread 2 found: 00
142701
Th2 mine:
00:00:00.9403647
Thread 1 closed: Thread 2 found: 00
```

The threaded versions produce these values:
4: 00.5114923
5: 00.9403647

This shows that threading has a major effect on the result and time taken for the hashing process.

Adjusting the Difficulty in Proof-of-Work

Variable difficulty was implemented by creating a desired block time; in this case 5 seconds was chosen. If the previous block took more than 5 seconds to be generated, the difficulty decreases, if the previous block took less than 5 seconds to be generated the difficulty increases. For testing purposes, this was limited to difficulty values of 0-6 due to the hexadecimal exponential nature of the increase in time taken with each increase in difficulty level.

A further option could be to implement a hexadecimal level of difficulty where instead of just adding another "0" to increase the difficulty it could add a threshold based on the hexadecimal value of the string. This would allow for more extreme fine tuning and [1]consistency between block generation periods.
For example, target = 0000 -> decrease difficulty → target <= 0001 etc. this gradually adjusts the difficulty with a 16x higher accuracy. This is similar to how BitCoin & Ethereum handle difficulty. (See links below)

```
//Function to adjust the difficulty
1 reference
public void adjustdiff(DateTime lastTime)
{
    //Gets the elapsed time between now and the last block mined
    DateTime startTime = DateTime.UtcNow;
    TimeSpan timeDiff = startTime - lastTime;

    //If the difference is less than 5 seconds, the difficulty is increased to attempt to increase the time
    if (timeDiff < TimeSpan.FromSeconds(5))
    {
        this.difficulty++;
        Console.WriteLine("Time since last mine");
        Console.WriteLine(timeDiff);
        Console.WriteLine("New Difficulty:");
        Console.WriteLine(this.difficulty);
    }
    //If the difference is more than 5 seconds, the difficulty is increased to attempt to decrease the time
    else if (timeDiff > TimeSpan.FromSeconds(5))
    {
        difficulty--;
        Console.WriteLine("Time since last mine");
        Console.WriteLine(timeDiff);
        Console.WriteLine("New Difficulty:");
        Console.WriteLine(this.difficulty);
    }

    //Difficulty can never be higher than 5 or lower than 0
    if (this.difficulty <= 0)
    {
        this.difficulty = 0;
        Console.WriteLine("Difficulty too low, new difficulty:");
        Console.WriteLine(this.difficulty);
    }
    else if (this.difficulty >= 6)
    {
        this.difficulty = 4;
        Console.WriteLine("Difficulty too high, new difficulty:");
        Console.WriteLine(this.difficulty);
    }
}
```

```
New Difficulty:
4
Endless loop
Block index: 25
Thread 1 closed: Thread 1
86590
Th1 mine:
00:00:00.5901127
Thread 2 closed: Thread 1
Here
added new block to chain
Time since last mine
00:00:00.6548138
New Difficulty:
5
```

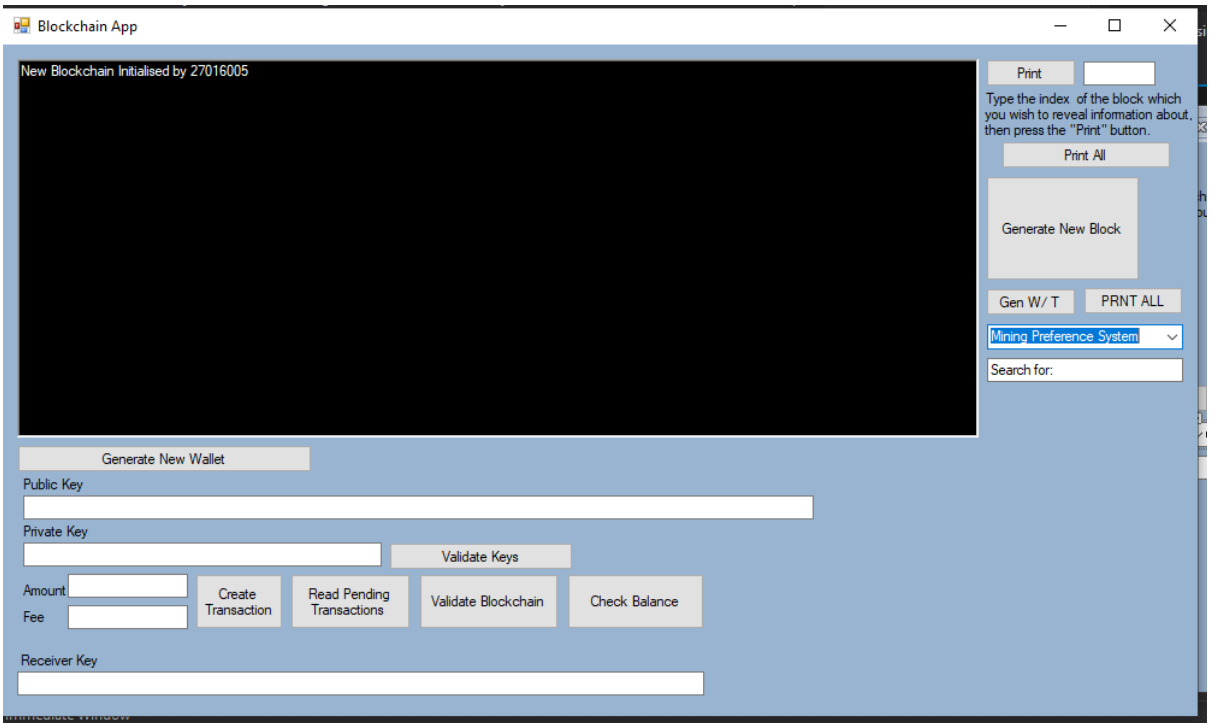
This shows the program successfully finding a result in quicker than the expected time, and thus increasing the difficulty: this is debugging output and not available to the user.

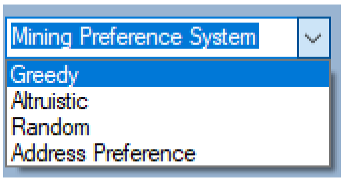
[1] https://en.bitcoin.it/wiki/Difficulty https://en.bitcoinwiki.org/wiki/Difficulty_in_Mining
https://medium.com/coinmonks/part-5-implementing-blockchain-and-cryptocurrency-with-pow-consensus-algorithm-a7f8853d23dc

Mining settings

A combo-box was implemented with 4 options of how to prioritise the order of transactions from the pending pool. These were: Greedy; Altruistic; Random; Addressed based.
Greedy method prioritises the transactions with the highest fees.
Altruistic method operates on a first in- first out style methodology.
Random selected a random order.
Addressed based takes any transactions to or from a given address.

```
while (transactionList.Count < LIMIT && TP.Count > 0 ) {
    if (mode == 0 ) {// greedy
        //int idx = 0;
        for (int i = 0; ((i < TP.Count)); i++)
        {
            if (TP.ElementAt(i).Fee > TP.ElementAt(idx).Fee)
            {
                idx = i;
            }
        }
        this.transactionList.Add(TP.ElementAt(idx));
    }
    else if (mode == 1) {// altruistic
        for (int i = 0; ((i < TP.Count) && (i < LIMIT)); i++)
        {
            this.transactionList.Add(TP.ElementAt(i));
        }
    }
    else if (mode == 2 ) {  //random
        Random random = new Random();
        idx = random.Next(0, TP.Count);
        this.transactionList.Add(TP.ElementAt(idx));
    }
    else if (mode == 3) {

        //this.transactionList.Add(TP.ElementAt(0));
        for (int i = 0; i < TP.Count && (transactionList.Count < LIMIT); i++)
        {
            if (TP.ElementAt(i).SenderAddress == address)
            {
                this.transactionList.Add(TP.ElementAt(i));
            }
            else if (TP.ElementAt(i).RecipientAddress == address)
            {
                this.transactionList.Add(TP.ElementAt(i));
            }
            else
            {
                // ONLY TAKE FROM THIS ADDRESS:
                // If take address as priority and then add up to get to 5 --> add mode = 2 here
            }
            //TP = TP.Except(this.transactionList).ToList();

        }
        Console.WriteLine("Endless loop");
    }
    else
    { // No Valid input, choose default --> Altruistic
        mode = 1;
    }
    TP = TP.Except(this.transactionList).ToList();

}
```

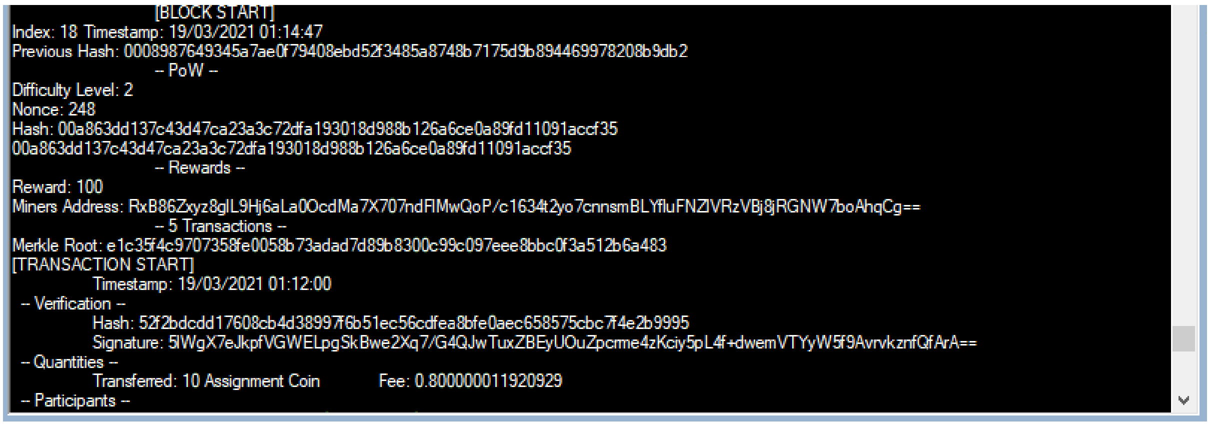




By Default, altruistic is chosen:

Here is proof of greedy:

For Transactions of 10 coins for the from the same wallet, to the same wallet, with only the fee changing -> between 1 and 0 at 0.1 increments randomly ordered.



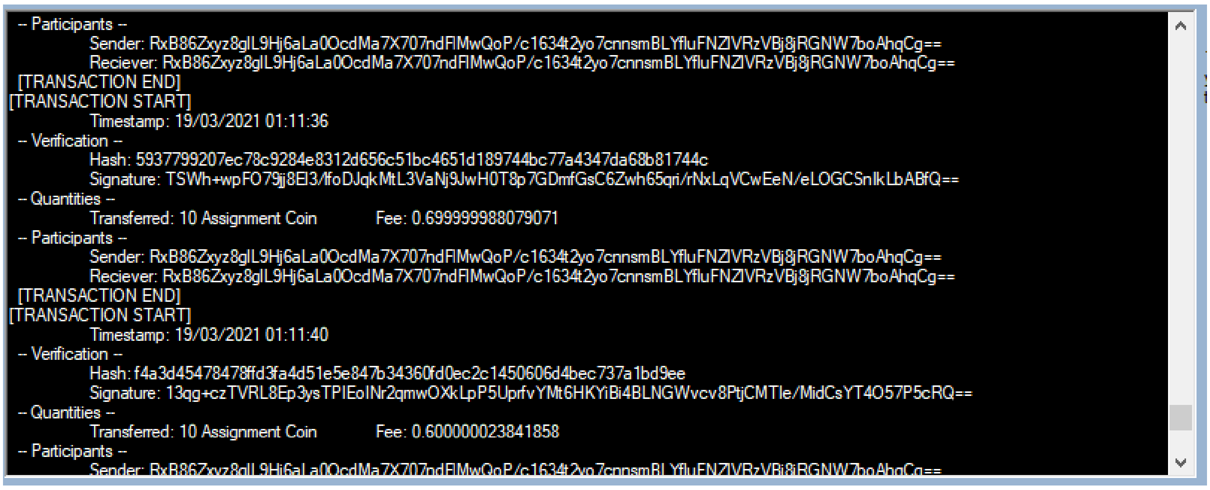
Use Cases:

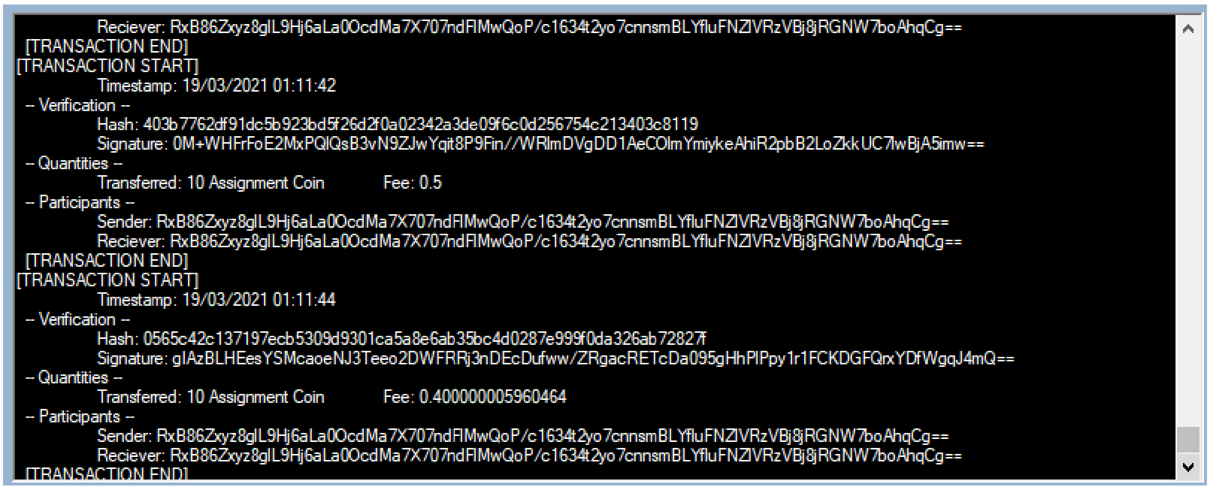
Greedy: Prioritise highest fees = most profit → business model

Altruistic: Fi-Fo → customer satisfaction / fair model. No Competition between miners resulting in more stability. Predictable.

Random: Fair model.

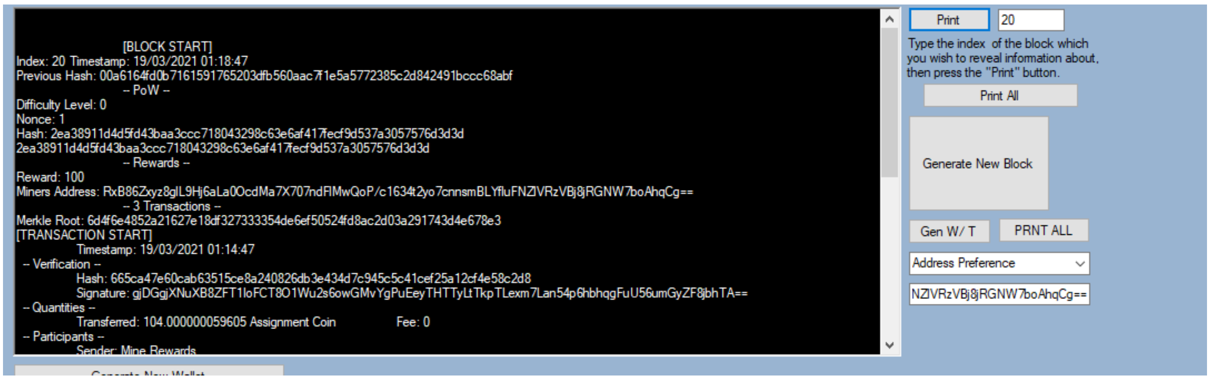
Address Preference: Analytics / Statistics, personal tracking





This shows that no matter which order the transactions are created in, they can be processed with whichever method is selected.

Here is proof with the Address preference functionality:

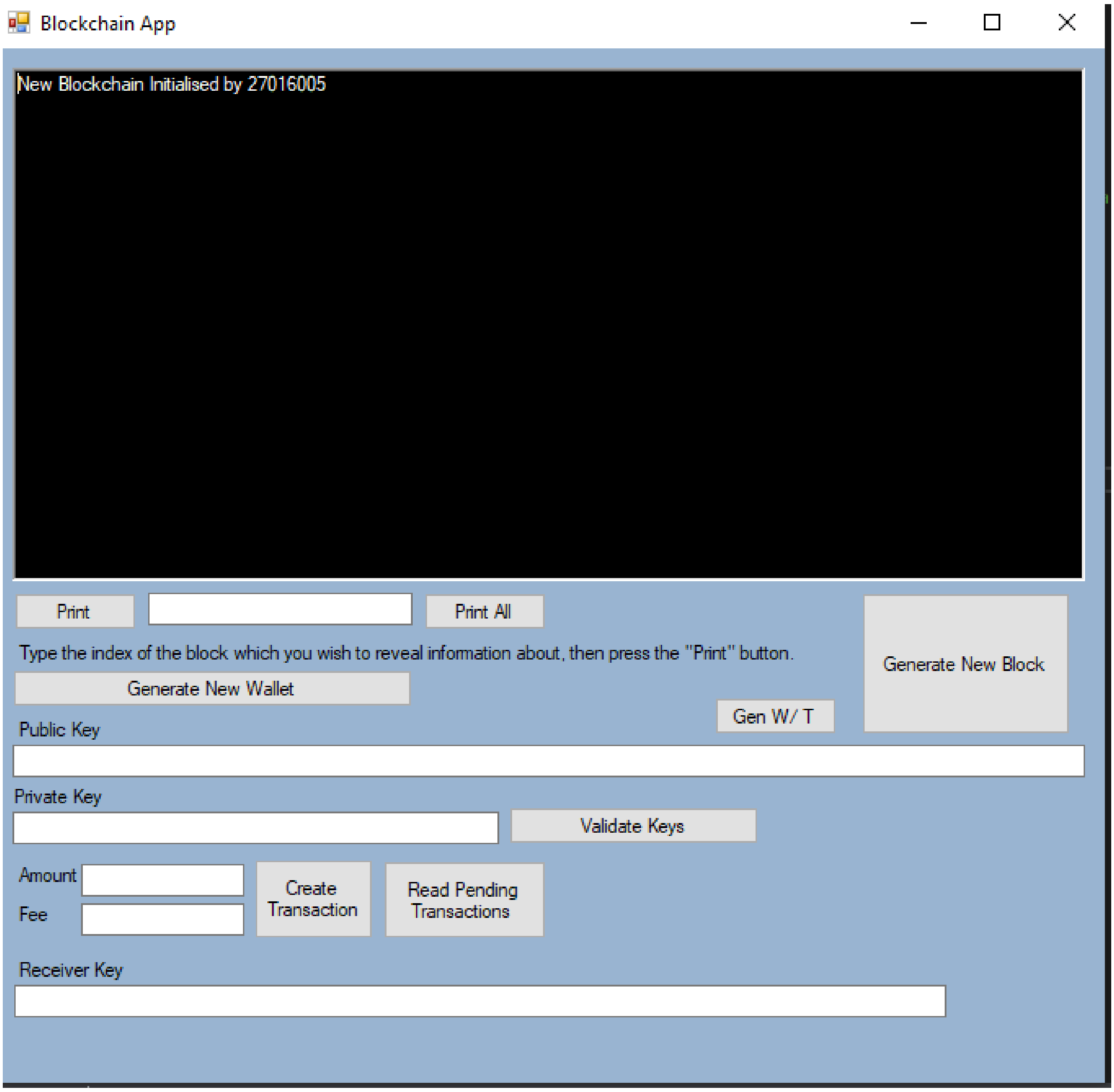


References:

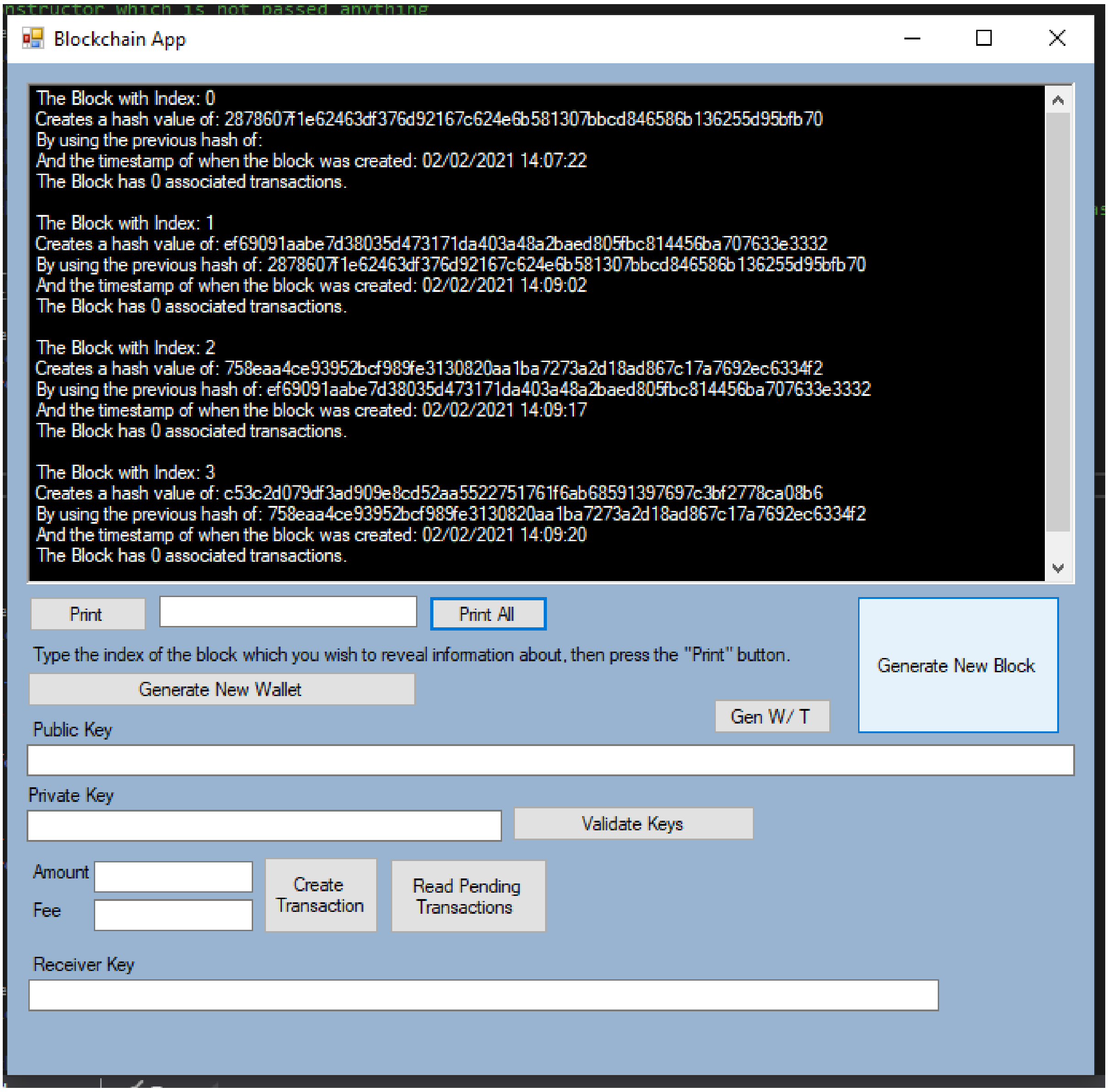
https://en.bitcoin.it/wiki/Difficulty
https://en.bitcoinwiki.org/wiki/Difficulty_in_Mining
https://medium.com/coinmonks/part-5-implementing-blockchain-and-cryptocurrency-with-pow-consensus-algorithm-a7f8853d23dc

APPENDIX:

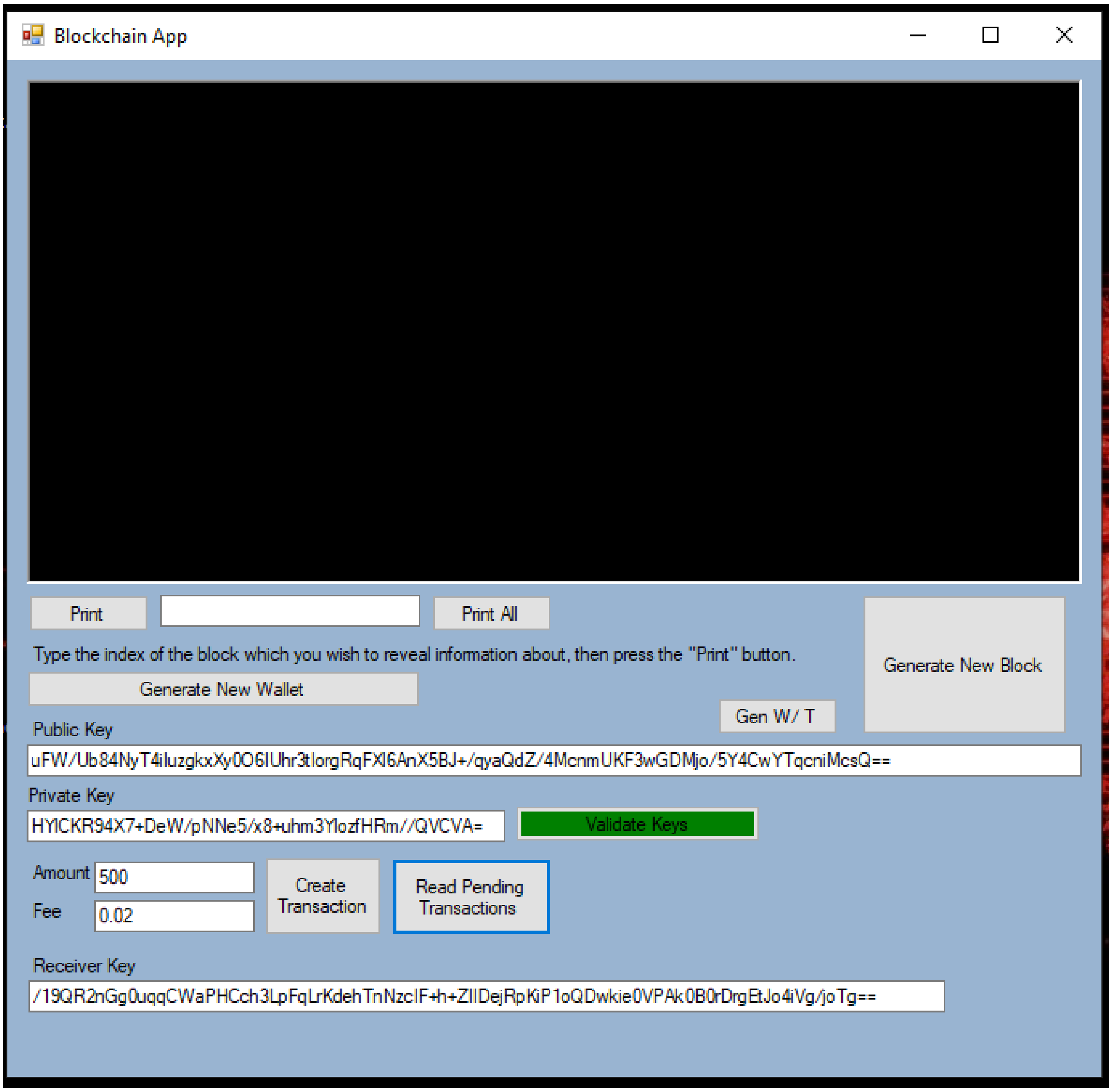
Showing Pending pool → Pre-extension tasks.
Initialise:



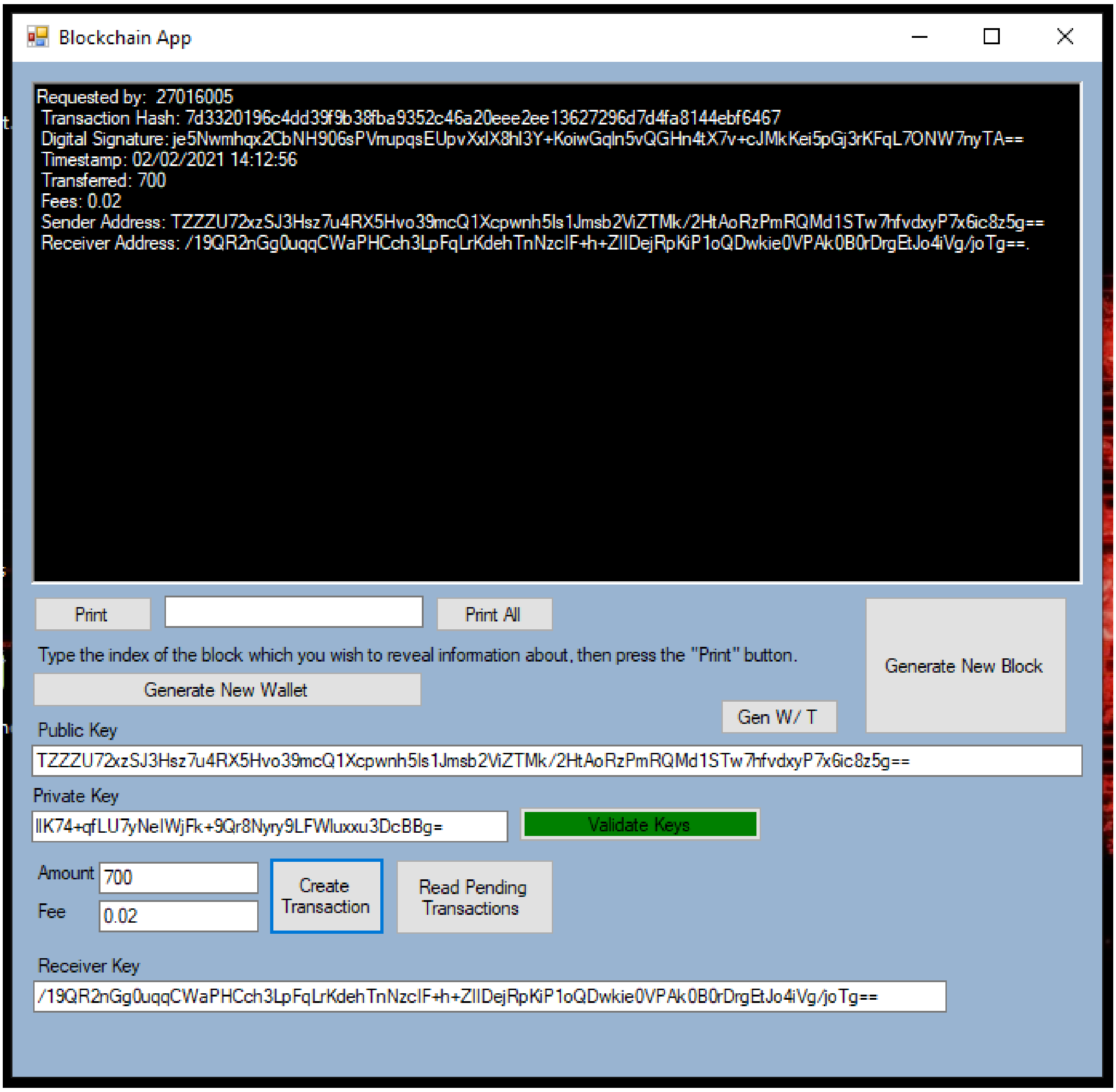
Create 3 (plus genesis) blocks (Printall)

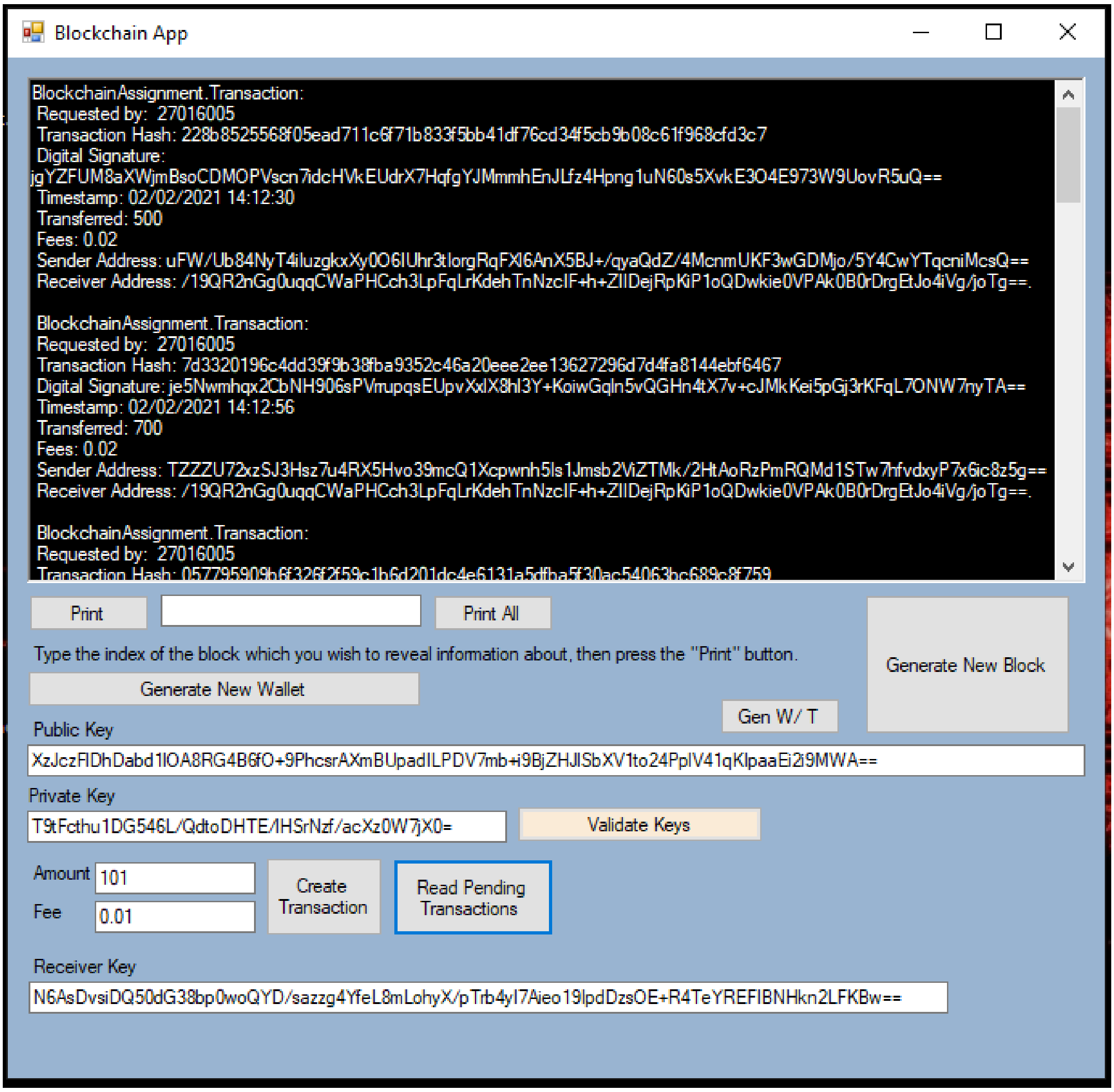

nstructor which is not passed anything
Blockchain App
The Block with Index: 0
Creates a hash value of: 2878607f1e62463df376d92167c624e6b581307bbcd846586b136255d95bfb70
By using the previous hash of:
And the timestamp of when the block was created: 02/02/2021 14:07:22
The Block has 0 associated transactions.
The Block with Index: 1
Creates a hash value of: ef69091aabe7d38035d473171da403a48a2baed805fbc814456ba707633e3332
By using the previous hash of: 2878607f1e62463df376d92167c624e6b581307bbcd846586b136255d95bfb70
And the timestamp of when the block was created: 02/02/2021 14:09:02
The Block has 0 associated transactions.
The Block with Index: 2
Creates a hash value of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
By using the previous hash of: ef69091aabe7d38035d473171da403a48a2baed805fbc814456ba707633e3332
And the timestamp of when the block was created: 02/02/2021 14:09:17
The Block has 0 associated transactions.
The Block with Index: 3
Creates a hash value of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
By using the previous hash of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
And the timestamp of when the block was created: 02/02/2021 14:09:20
The Block has 0 associated transactions.
Print
Print All
Type the index of the block which you wish to reveal infomation about, then press the "Print" button.
Generate New Block
Generate New Wallet
Gen W/ T
Public Key
Private Key
Validate Keys
Amount
Fee
Create Transaction
Read Pending Transactions
Receiver Key

View transactions:

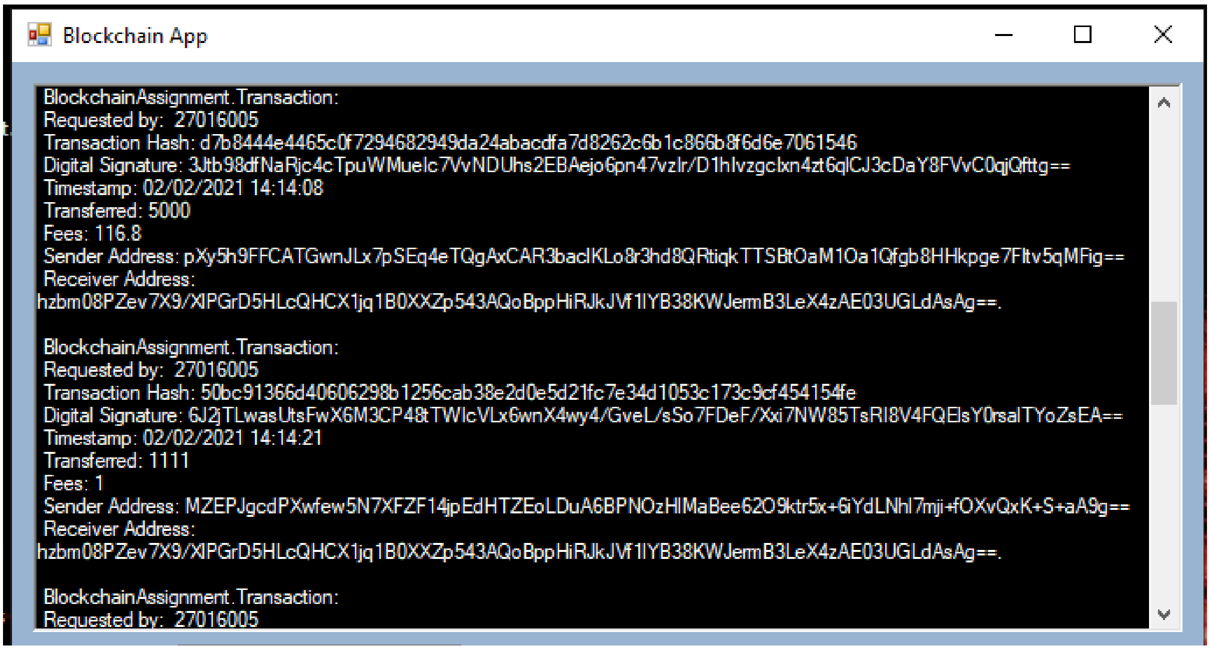


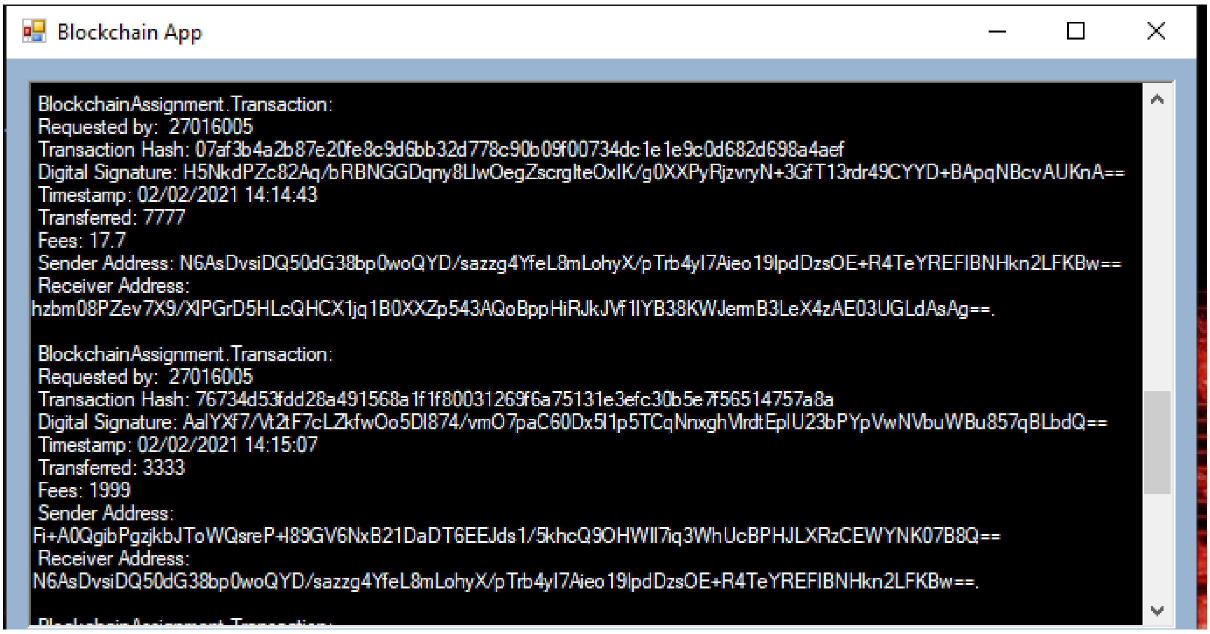
Val keys + Add 7 Transactions




Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 228b8525568f05ead711c6f71b833f5bb41df76cd34f5cb9b08c61f968cfd3c7
Digital Signature:
jgYZFUM8aXWjmBsoCDMOPVscn7idcHVkEUdrX7HqfgYJMmmhEnJLfz4Hpng1uN60s5XvkE3O4E973W9UovR5uQ==
Timestamp: 02/02/2021 14:12:30
Transferred: 500
Fees: 0.02
Sender Address: uFW/Ub84NyT4iluzgkxXy0O6IUhr3tlorgRqFXl6AnX5BJ+/qyaQdZ/4McnmUKF3wGDMjo/5Y4CwYTqcniMcsQ==
Receiver Address: /19QR2nGg0uqqCWaPHCch3LpFqLrKdehTnNzclF+h+ZIIDejRpKiP1oQDwkie0VPAk0B0rDrgEtJo4iVg/joTg==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 7d3320196c4dd39f9b38fba9352c46a20eee2ee13627296d7d4fa8144ebf6467
Digital Signature: je5Nwmhqx2CbNH906sPVmupqsEUpvXxlX8hl3Y+KoiwGqln5vQGHn4tX7v+cJMkKei5pGj3rKFqL7ONW7nyTA==
Timestamp: 02/02/2021 14:12:56
Transferred: 700
Fees: 0.02
Sender Address: TZZZU72xzSJ3Hsz7u4RX5Hvo39mcQ1Xcpwnh5ls1Jmsb2ViZTMk/2HtAoRzPmRQMd1STw7hfvdxyP7x6ic8z5g==
Receiver Address: /19QR2nGg0uqqCWaPHCch3LpFqLrKdehTnNzclF+h+ZIIDejRpKiP1oQDwkie0VPAk0B0rDrgEtJo4iVg/joTg==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 057795909b6f326f2f59c1b6d201dc4e6131a5dfba5f30ac54063bc689c8f759
Print
Print All
Type the index of the block which you wish to reveal information about, then press the "Print" button.
Generate New Block
Generate New Wallet
Gen W/ T
Public Key
XzJczFlDhDabd1lOA8RG4B6fO+9PhcsrAXmBUpadILPDV7mb+i9BjZHJlSbXV1to24PplV41qKlpaaEi2i9MWA==
Private Key
T9tFcthu1DG546L/QdtoDHTE/lHSrNzf/acXz0W7jX0=
Validate Keys
Amount
101
Fee
0.01
Create Transaction
Read Pending Transactions
Receiver Key
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==


Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 057795909b6f326f2f59c1b6d201dc4e6131a5dfba5f30ac54063bc689c8f759
Digital Signature: F8hstxYAdK8EmZZfvyAnpFm2heka/WKa+960CbMfiZ4Bfw6gAkwHqwEs4ObLLQkrUY5cPpnwsKPJerfC1JnpTg==
Timestamp: 02/02/2021 14:13:28
Transferred: 1500
Fees: 20
Sender Address: hzbm08PZev7X9/XlPGrD5HLcQHCX1jq1B0XXZp543AQoBppHiRJkJVf1lYB38KWJemB3LeX4zAE03UGLdAsAg==
Receiver Address: /19QR2nGg0uqqCWaPHCch3LpFqLrKdehTnNzclF+h+ZIIDejRpKiP1oQDwkie0VPAk0B0rDrgEtJo4iVg/joTg==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 5f7200692cb8168fc4b67ef16fc8995d93d25bb40dda9b125227b6301799b123
Digital Signature: cxyx6s1yjRbrvm23tAaPSCLkrbIRPXnC8ufkYadqcfPq2Tj7Q5x0fF3/GMdJW0DnkPQJ9LCj5yDu0vpXU9EBcg==
Timestamp: 02/02/2021 14:13:51
Transferred: 250
Fees: 1.5
Sender Address:
1j2nb2cRaiO5/EH5XU11g3Hv2wJ+WjMMv+Ga64uDePgvu5va8wRkEGHConWzVg1NGGdxUAtpgu7gLC4EtpKV+w==
Receiver Address:
hzbm08PZev7X9/XlPGrD5HLcQHCX1jq1B0XXZp543AQoBppHiRJkJVf1lYB38KWJemB3LeX4zAE03UGLdAsAg==.
BlockchainAssignment.Transaction:

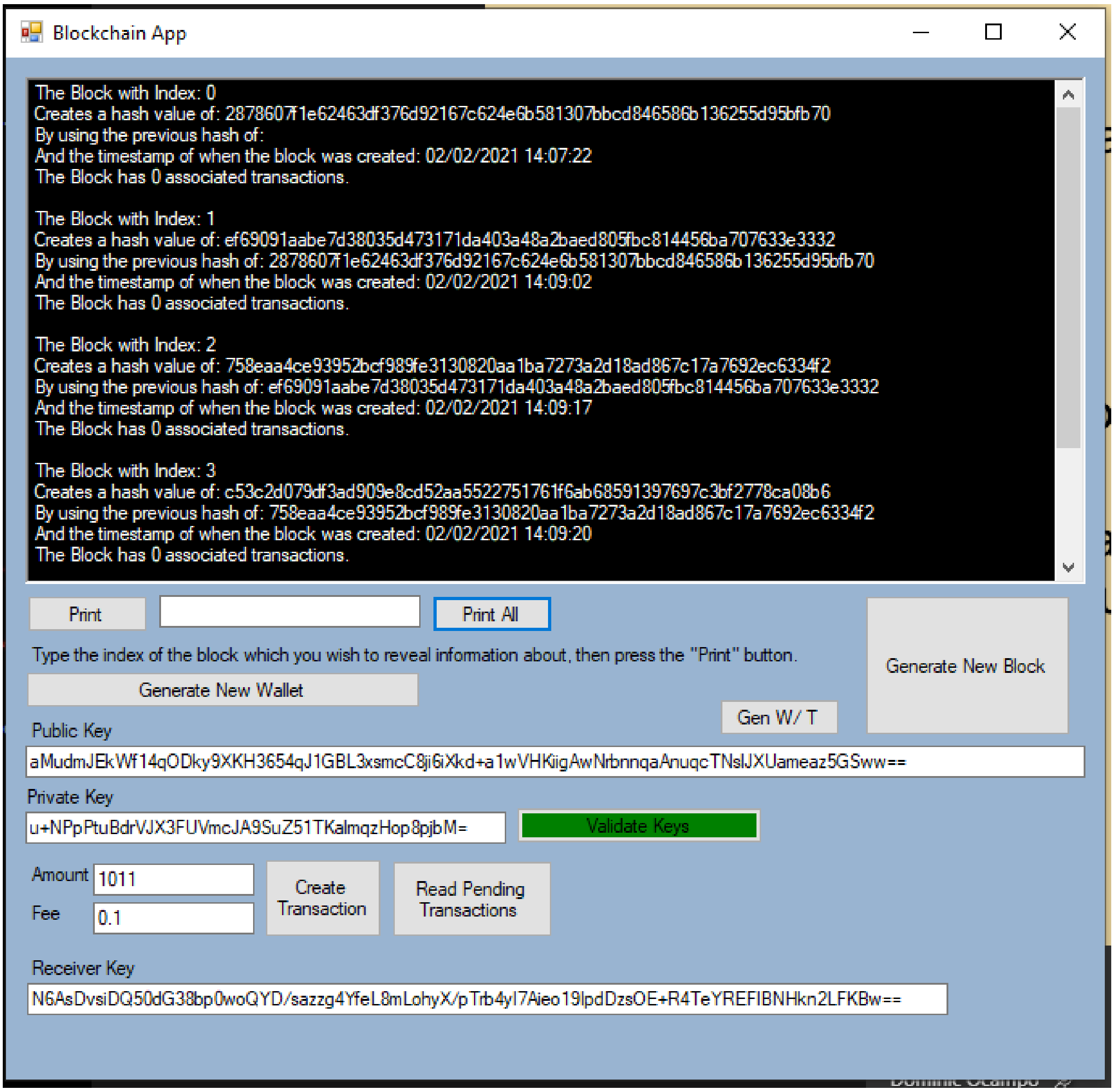

Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: d7b8444e4465c0f7294682949da24abacdfa7d8262c6b1c866b8f6d6e7061546
Digital Signature: 3Jtb98dfNaRjc4cTpuWMuelc7VvNDUhs2EBAejo6pn47vzIr/D1hIvzgclxn4zt6qlCJ3cDaY8FVvC0qjQfttg==
Timestamp: 02/02/2021 14:14:08
Transferred: 5000
Fees: 116.8
Sender Address: pXy5h9FFCATGwnJLx7pSEq4eTQgAxCAR3baclKLo8r3hd8QRtiqkTTSBtOaM1Oa1Qfgb8HHkpge7Fltv5qMFig==
Receiver Address:
hzbm08PZev7X9/XIPGrD5HLcQHCX1jq1B0XXZp543AQoBppHiRJkJVf1lYB38KWJermB3LeX4zAE03UGLdAsAg==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 50bc91366d40606298b1256cab38e2d0e5d21fc7e34d1053c173c9cf454154fe
Digital Signature: 6J2jTLwasUtsFwX6M3CP48tTWlcVLx6wnX4wy4/GveL/sSo7FDeF/Xxi7NW85TsRI8V4FQElsY0rsalTYoZsEA==
Timestamp: 02/02/2021 14:14:21
Transferred: 1111
Fees: 1
Sender Address: MZEPJgcdPXwfew5N7XFZF14jpEdHTZEoLDuA6BPNOzHIMaBee62O9ktr5x+6iYdLNhl7mji+fOXvQxK+S+aA9g==
Receiver Address:
hzbm08PZev7X9/XIPGrD5HLcQHCX1jq1B0XXZp543AQoBppHiRJkJVf1lYB38KWJermB3LeX4zAE03UGLdAsAg==.
BlockchainAssignment.Transaction:
Requested by: 27016005

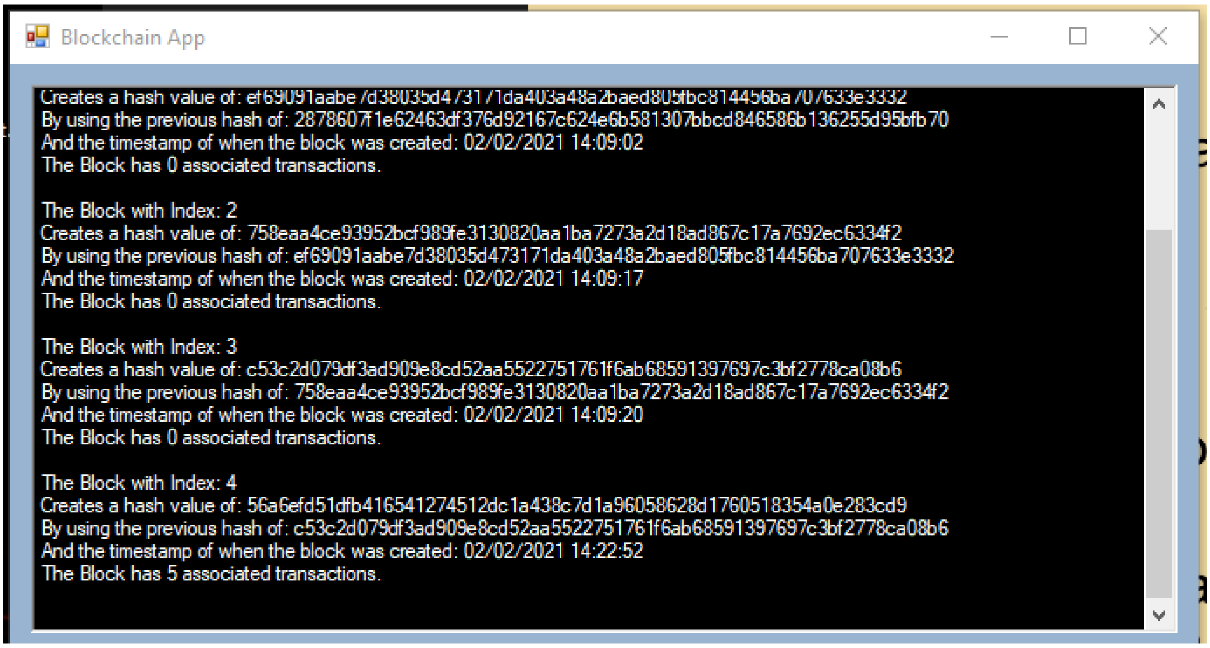

Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 07af3b4a2b87e20fe8c9d6bb32d778c90b09f00734dc1e1e9c0d682d698a4aef
Digital Signature: H5NkdPZc82Aq/bRBNGGDqny8LlwOegZscrglteOxIK/g0XXPyRjzvryN+3GfT13rdr49CYYD+BApqNBcvAUKnA==
Timestamp: 02/02/2021 14:14:43
Transferred: 7777
Fees: 17.7
Sender Address: N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==
Receiver Address:
hzbm08PZev7X9/XIPGrD5HLcQHCX1jq1B0XXZp543AQoBppHiRJkJVf1lYB38KWJermB3LeX4zAE03UGLdAsAg==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 76734d53fdd28a491568a1f1f80031269f6a75131e3efc30b5e7f56514757a8a
Digital Signature: AalYXf7/Vt2tF7cLZkfwOo5DI874/vmO7paC60Dx5I1p5TCqNnxghVIrdtEplU23bPYpVwNVbuWBu857qBLbdQ==
Timestamp: 02/02/2021 14:15:07
Transferred: 3333
Fees: 1999
Sender Address:
Fi+A0QgibPgzjkbJToWQsreP+I89GV6NxB21DaDT6EEJds1/5khcQ9OHWlI7iq3WhUcBPHJLXRzCEWYNK07B8Q==
Receiver Address:
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==.

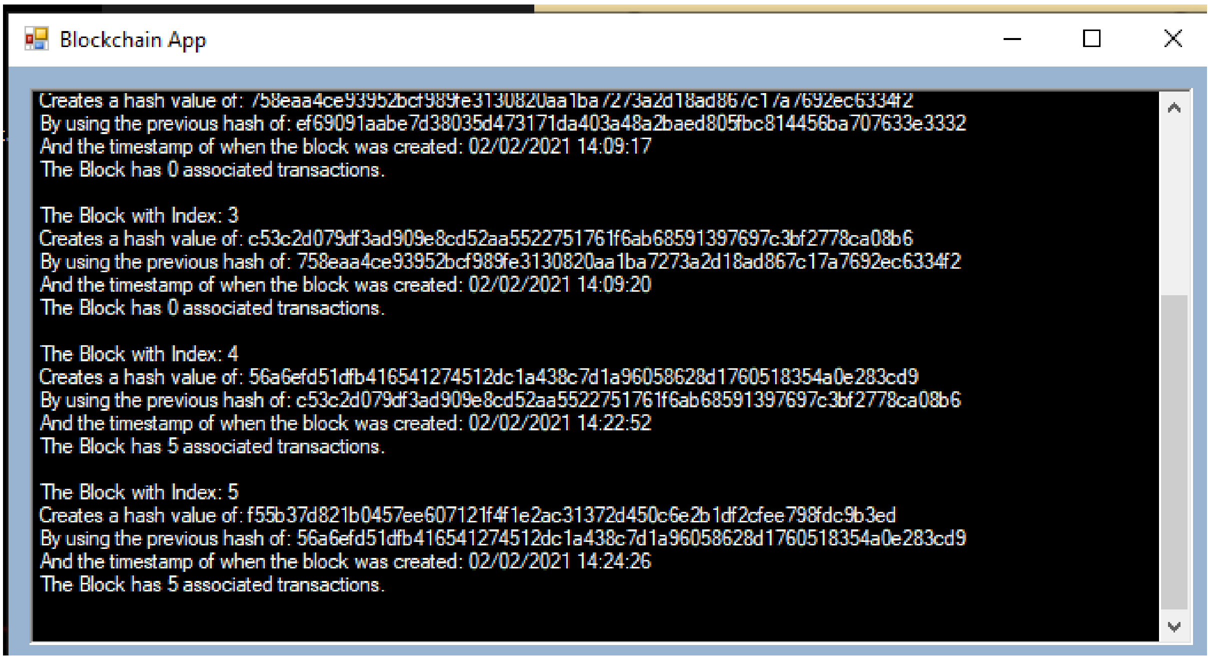

Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 9b464c2161cee3d997a1e92459633f898ce2736f488a92da705d005e08530719
Digital Signature: jp2A9AU7r1lsj7Pzel6y+bGJXvG+Ya8giVij0OxrRbRV6ECxh/qSXo0rW9nU7cV7rWro+8L87T/grWfdkEY7rw==
Timestamp: 02/02/2021 14:15:24
Transferred: 10
Fees: 0.01
Sender Address: C3q3h9uMx5DQ5OC+
0mqnla5MeJbnWbqZZGMHVyauPEyClOVqB5eZo60Z7LbIR3UbfuDPSjhowGkB0NvCMaPszA==
Receiver Address:
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 1469585aea73bb1313d9ff8c38eaeb39e4ef17f6d00ad865d691d699e40ee29e
Digital Signature: 2VF7VaAcxpykgiBS1pA5iJql8n4W3AMbxACvj4UeF/fHJBdTPzt1r/7NE622X6DJbGA2XEpeZ1iZl842E2XUZQ==
Timestamp: 02/02/2021 14:15:38
Transferred: 101
Fees: 0.01
Sender Address: XzJczFlDhDabd1lOA8RG4B6fO+9PhcsrAXmBUpadILPDV7mb+i9BjZHJISbXV1to24PplV41qKlpaaEi2i9MWA==
Receiver Address:
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==.

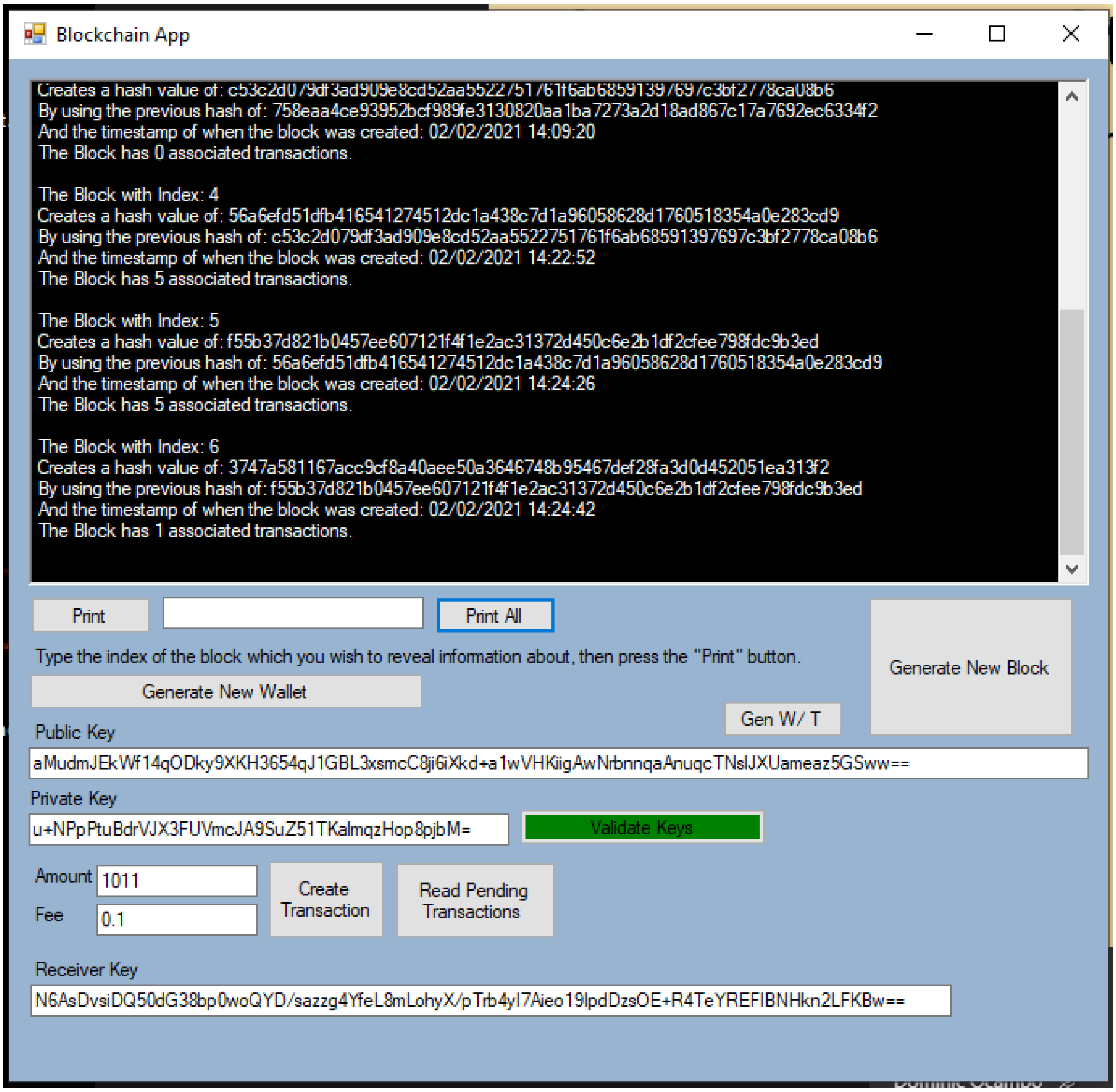

Blockchain App
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: 1469585aea73bb1313d9ff8c38eaeb39e4ef17f6d00ad865d691d699e40ee29e
Digital Signature: 2VF7VaAcxpykgiBS1pA5iJql8n4W3AMbxACvj4UeF/fHJBdTPzt1r/7NE622X6DJbGA2XEpeZ1iZl842E2XUZQ==
Timestamp: 02/02/2021 14:15:38
Transferred: 101
Fees: 0.01
Sender Address: XzJczFlDhDabd1lOA8RG4B6fO+9PhcsrAXmBUpadILPDV7mb+i9BjZHJISbXV1to24PplV41qKlpaaEi2i9MWA==
Receiver Address:
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==.
BlockchainAssignment.Transaction:
Requested by: 27016005
Transaction Hash: d4676902cb8b302043c88505ad7fd08f810426a14b9d9c486d20f8bdc544fcc1
Digital Signature:
3lClwf8qyKBnDiUnh/RWMe1DBVaUvC88OaxRnWV09qDKr00XAELujfNujWPaBSBthfOpEuAyvdRyzRLswyJwTw==
Timestamp: 02/02/2021 14:18:51
Transferred: 1011
Fees: 0.1
Sender Address: aMudmJEkWf14qODky9XKH3654qJ1GBL3xsmcC8ji6iXkd+a1wVHKigAwNrbnnqaAnuqcTNslJXUameaz5GSww==
Receiver Address:
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==.

Generate a new block (with transactions)



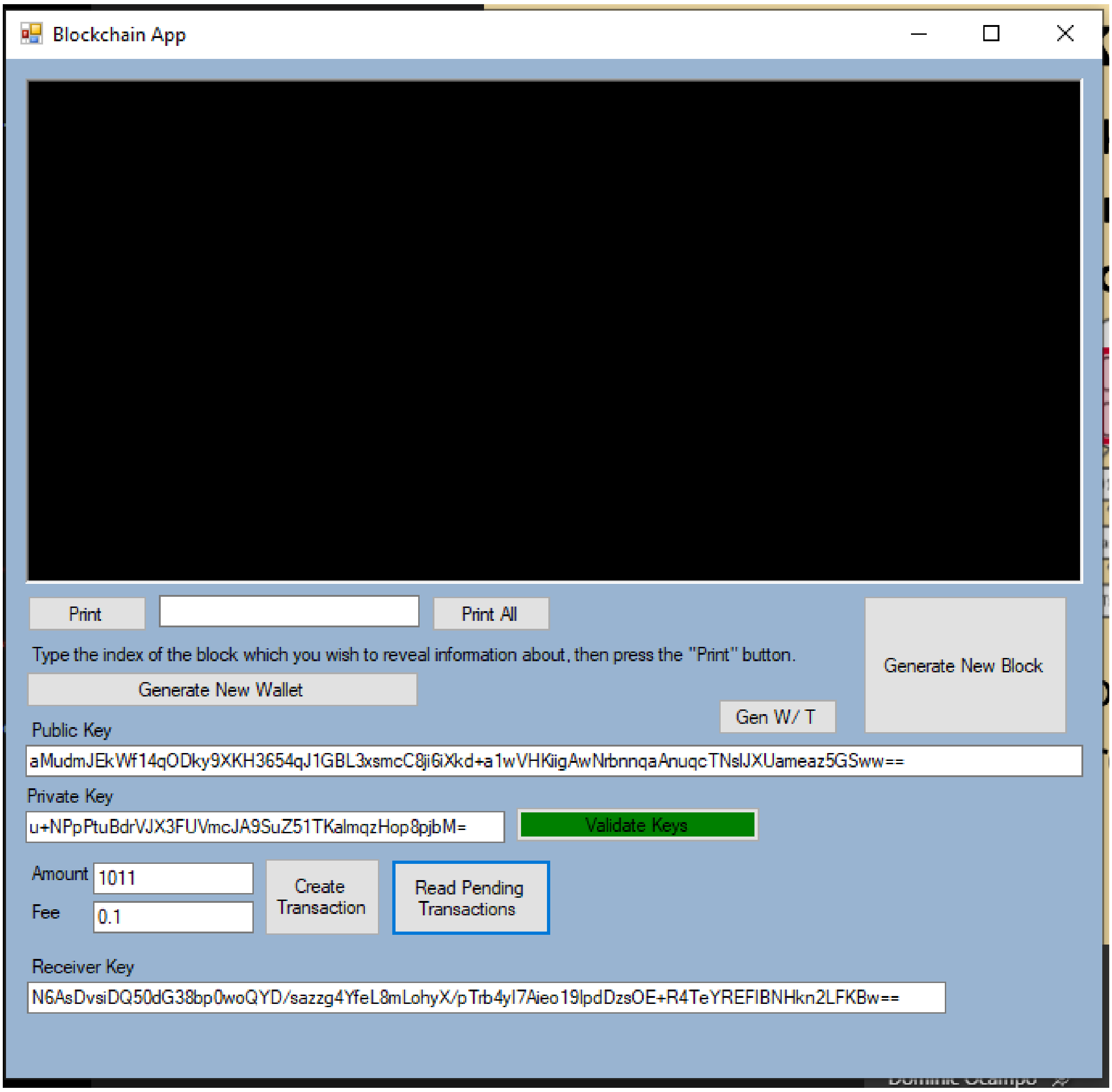

Blockchain App
Creates a hash value of: ef69091aabe7d38035d473171da403a48a2baed805fbc814456ba707633e3332
By using the previous hash of: 2878607f1e62463df376d92167c624e6b581307bbcd846586b136255d95bfb70
And the timestamp of when the block was created: 02/02/2021 14:09:02
The Block has 0 associated transactions.
The Block with Index: 2
Creates a hash value of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
By using the previous hash of: ef69091aabe7d38035d473171da403a48a2baed805fbc814456ba707633e3332
And the timestamp of when the block was created: 02/02/2021 14:09:17
The Block has 0 associated transactions.
The Block with Index: 3
Creates a hash value of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
By using the previous hash of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
And the timestamp of when the block was created: 02/02/2021 14:09:20
The Block has 0 associated transactions.
The Block with Index: 4
Creates a hash value of: 56a6efd51dfb416541274512dc1a438c7d1a96058628d1760518354a0e283cd9
By using the previous hash of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
And the timestamp of when the block was created: 02/02/2021 14:22:52
The Block has 5 associated transactions.


Blockchain App
Creates a hash value of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
By using the previous hash of: ef69091aabe7d38035d473171da403a48a2baed805fbc814456ba707633e3332
And the timestamp of when the block was created: 02/02/2021 14:09:17
The Block has 0 associated transactions.
The Block with Index: 3
Creates a hash value of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
By using the previous hash of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
And the timestamp of when the block was created: 02/02/2021 14:09:20
The Block has 0 associated transactions.
The Block with Index: 4
Creates a hash value of: 56a6efd51dfb416541274512dc1a438c7d1a96058628d1760518354a0e283cd9
By using the previous hash of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
And the timestamp of when the block was created: 02/02/2021 14:22:52
The Block has 5 associated transactions.
The Block with Index: 5
Creates a hash value of: f55b37d821b0457ee607121f4f1e2ac31372d450c6e2b1df2cfee798fdc9b3ed
By using the previous hash of: 56a6efd51dfb416541274512dc1a438c7d1a96058628d1760518354a0e283cd9
And the timestamp of when the block was created: 02/02/2021 14:24:26
The Block has 5 associated transactions.


Blockchain App
Creates a hash value of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
By using the previous hash of: 758eaa4ce93952bcf989fe3130820aa1ba7273a2d18ad867c17a7692ec6334f2
And the timestamp of when the block was created: 02/02/2021 14:09:20
The Block has 0 associated transactions.
The Block with Index: 4
Creates a hash value of: 56a6efd51dfb416541274512dc1a438c7d1a96058628d1760518354a0e283cd9
By using the previous hash of: c53c2d079df3ad909e8cd52aa5522751761f6ab68591397697c3bf2778ca08b6
And the timestamp of when the block was created: 02/02/2021 14:22:52
The Block has 5 associated transactions.
The Block with Index: 5
Creates a hash value of: f55b37d821b0457ee607121f4f1e2ac31372d450c6e2b1df2cfee798fdc9b3ed
By using the previous hash of: 56a6efd51dfb416541274512dc1a438c7d1a96058628d1760518354a0e283cd9
And the timestamp of when the block was created: 02/02/2021 14:24:26
The Block has 5 associated transactions.
The Block with Index: 6
Creates a hash value of: 3747a581167acc9cf8a40aee50a3646748b95467def28fa3d0d452051ea313f2
By using the previous hash of: f55b37d821b0457ee607121f4f1e2ac31372d450c6e2b1df2cfee798fdc9b3ed
And the timestamp of when the block was created: 02/02/2021 14:24:42
The Block has 1 associated transactions.
Print
Print All
Generate New Block
Type the index of the block which you wish to reveal information about, then press the "Print" button.
Generate New Wallet
Gen W/ T
Public Key
aMudmJEkWf14qODky9XKH3654qJ1GBL3xsmcC8ji6iXkd+a1wVHKiigAwNrbnnqaAnuqcTNslJXUameaz5GSww==
Private Key
u+NPpPtuBdrVJX3FUVmcJA9SuZ51TKalmqzHop8pjbM=
Validate Keys
Amount
1011
Fee
0.1
Create Transaction
Read Pending Transactions
Receiver Key
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yl7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==

shows +5 +5 + 1

Pending:


Blockchain App
Print
Print All
Generate New Block
Type the index of the block which you wish to reveal information about, then press the "Print" button.
Generate New Wallet
Gen W/ T
Public Key
aMudmJEkWf14qODky9XKH3654qJ1GBL3xsmcC8ji6iXkd+a1wVHKiigAwNrbnnqaAnuqcTNslJXUameaz5GSww==
Private Key
u+NPpPtuBdrVJX3FUVmcJA9SuZ51TKalmqzHop8pjbM=
Validate Keys
Amount
1011
Fee
0.1
Create Transaction
Read Pending Transactions
Receiver Key
N6AsDvsiDQ50dG38bp0woQYD/sazzg4YfeL8mLohyX/pTrb4yI7Aieo19lpdDzsOE+R4TeYREFIBNHkn2LFKBw==