Analysis of the Blockchain Protocol in Asynchronous Networks

Individual Project

Submitted By

Keshav Agarwal

Senior

B.S. / M.S. Computer Science

Gildart Haase School of Computer Sciences and Engineering

An **individual project paper** submitted in partial fulfillment of the requirements for the course - **CSCI 3420 Cryptography**

Under the supervision

of

Dr. Avimanyou Vatsa

Assistant Professor, Department of Computer Science

FAIRLEIGH DICKINSON UNIVERSITY

Fall 2019

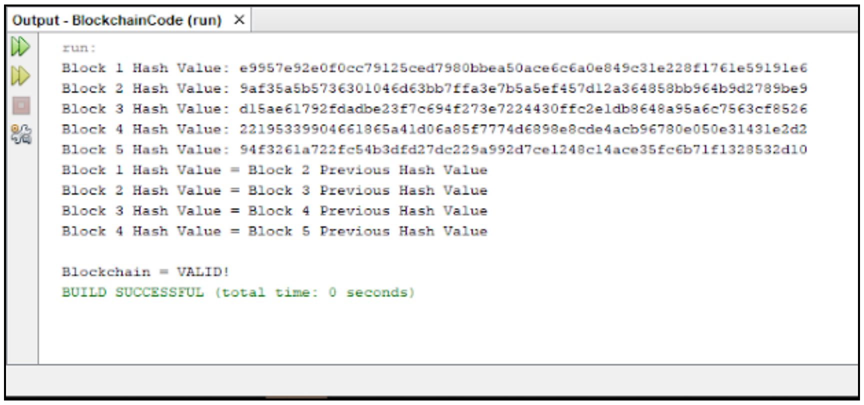
In the research paper, *Analysis of the Blockchain Protocol In Asynchronous Networks* written by Rafael Pass, Lior Seeman, and Abhi Shelat, the authors analyze Nakamoto's "blockchain protocols" and determine how effective blockchains really are. According to Google, a blockchain is a system in which a record of transactions made in bitcoin or another cryptocurrency are maintained across several computers that are linked in a peer-to-peer network. In this specific research paper, blockchain is defined as a chain of blocks that links to one another based on their previous hash values. Practically speaking, blockchain can be a ledger that records online transactions such as bitcoin. Blockchain ensures the integrity of a cryptocurrency by encrypting, validating, and recording transactions. For example, Tony Stark goes to Apple to buy a Macbook for $15000 but his credit limit is only $12000. His transaction will be sent to the credit card company but the company will decline it since the transaction is over the credit limit. On the other hand, if Stark's credit limit is $20000 and he spends $15000 at Apple, the transaction will go to the credit card company and they will approve it. The money will then be credited to the retailer or seller and there will now be a charge on the credit card. All of this is recorded in the credit card's log.

There are problems with the blockchain protocol. In the research paper, Satoshi Nakamoto has an idea of a blockchain protocol that he created in 2008 and this protocol has a "so-called permissionless setting .. -- anyone can join (or leave) the protocol execution, and the protocol instructions do not depend on the identities of the players." (Pass, Seeman, Shelat, 2) This permissionless setting can be dangerous because unauthorized users can join the system and cause faulty transactions or send invalid messages. For example, a class has a messaging system that includes the professor and twenty students. The twenty students can communicate with one another and the professor. The professor is sending out grades to the students and the students receive the grades. The attackers can change the grades and send it to the students as if the professor sent the new grade, meanwhile the professor thinks he sent the original grades to the students. Nowadays, people have to rely on third party applications to process transactions of any kind. With this advancement, other trust issues arise as well. Buyers can pretend to have never purchased a product or have never received a product and the seller can pretend to have sent the product to the buyer. Nakamoto also created his idea of the blockchain protocol in asynchronous networks. On this kind of networks, there are many delays when sending a message or transaction. There is no trust that the message or transaction will be sent to the intended user automatically. It can take a couple of seconds to minutes and possibly hours, depending on the network.

In the research paper, Satoshi Nakamoto proposed to use bitcoin instead of relying on a third party application. He created a protocol that is used on a network with message delays, asynchronous network. As I mentioned before, some users would receive a message sent by another user whereas some users would not receive the message for a very long time. Nakamoto's confirmed consistency suggests that all the users on the chain need to be honest or actual users that are not hiding their identity. In terms of bitcoin, that means the consistency verifies and ensures that the users are not spending their money multiple times on the same transaction.

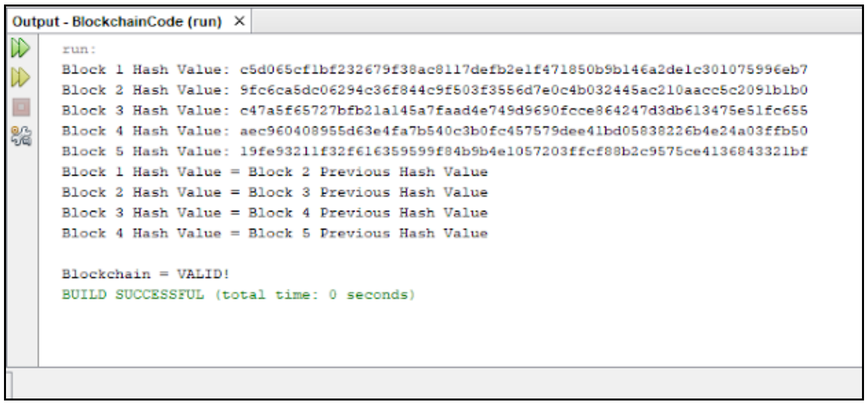
With Netbeans IDE 8.2 and a Macbook Pro with 32GB RAM and an Intel i9 processor, I was able to create a JAVA program that creates virtual blocks and an actual blockchain between the blocks. The blocks have a digital signature, a unique transaction and a timestamp. Each block has a current hash value and a previous hash value that matches the current hash value of the previous block. The first block is an empty non-real block because it has no previous hash value since there is no block before it. All of this goes through the SHA-256 algorithm. After the hash algorithm completes, the program outputs the block hash values. The program also authenticates whether the current hash value and previous hash value of the blocks are equal or not. If they are not equal, then the blockchain would not be valid. If they are equal, then the blockchain is valid and ready to go. The program also outputs and tells the user if the blockchain is valid or not. Even if a single block in the program is not valid, the entire blockchain would be invalid and then the hash values would say "null."

The program has successfully displayed the hash values of each block and has also checked current hash value of the current block is equal to the previous hash value of the next block. The program also checks if the blockchain itself is valid or not. If the blockchain is not valid, the program will not output any of the hash values and will just output "Blockchain = NOT VALID!" If a user runs the program again and again, the hash values will change and the blockchain will still be valid. It makes sure that the current hash value and the previous hash value are equal. Below will be five outputs of the same program. It demonstrates the different hash value every time.


Output - BlockchainCode (run)

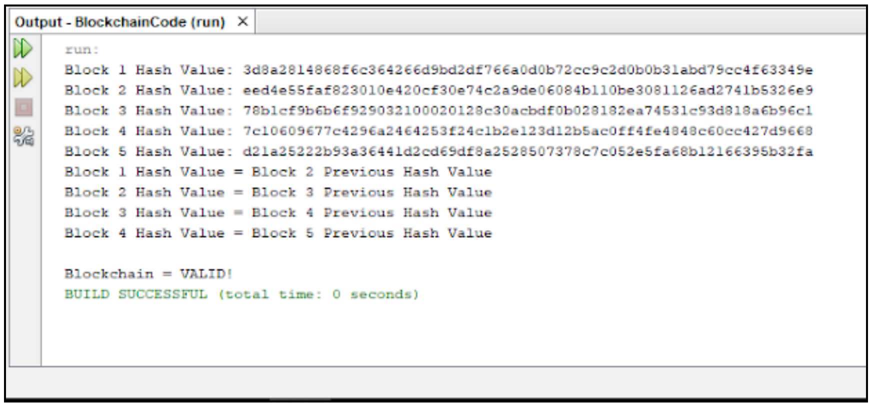
```
run:
Block 1 Hash Value: e9957e92e0f0cc79125ced7980bbea50ace6c6a0e849c31e228f1761e59191e6
Block 2 Hash Value: 9af35a5b5736301046d63bb7ffa3e7b5a5ef457d12a364858bb964b9d2789be9
Block 3 Hash Value: d15ae61792fdadbe23f7c694f273e7224430ffc2e1db8648a95a6c7563cf8526
Block 4 Hash Value: 2219533990466186 5a41d06a85f7774d6898e8cde4acb96780e050e31431e2d2
Block 5 Hash Value: 94f3261a722fc54b3dfd27dc229a992d7ce1248c14ace35fc6b71f1328532d10
Block 1 Hash Value = Block 2 Previous Hash Value
Block 2 Hash Value = Block 3 Previous Hash Value
Block 3 Hash Value = Block 4 Previous Hash Value
Block 4 Hash Value = Block 5 Previous Hash Value

Blockchain = VALID!
BUILD SUCCESSFUL (total time: 0 seconds)
```


Output - BlockchainCode (run)

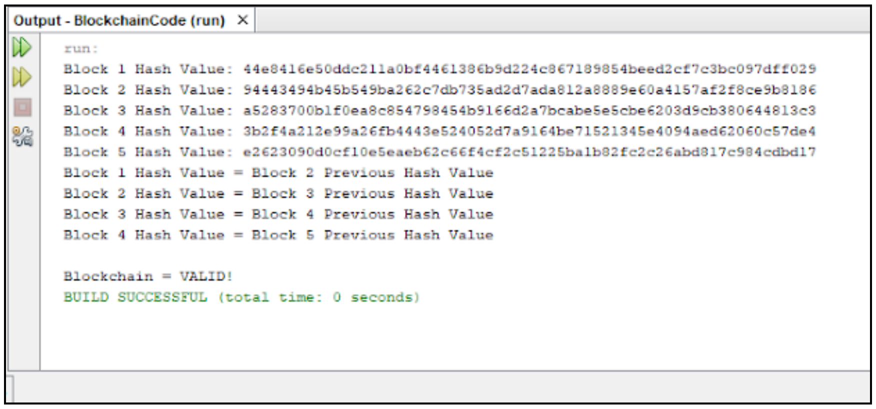
```
run:
Block 1 Hash Value: c5d065cf1bf232679f38ac8117defb2e1f471850b9b146a2de1c301075996eb7
Block 2 Hash Value: 9fc6ca5dc06294c36f844c9f503f3556d7e0c4b032445ac210aacc5c2091b1b0
Block 3 Hash Value: c47a5f65727bfb21a145a7faad4e749d9690fcce864247d3db613475e51fc655
Block 4 Hash Value: aec960408955d63e4fa7b540c3b0fc457579dee41bd05838226b4e24a03ffb50
Block 5 Hash Value: 19fe93211f32f616359599f84b9b4e1057203ffcf88b2c9575ce413684332 1bf
Block 1 Hash Value = Block 2 Previous Hash Value
Block 2 Hash Value = Block 3 Previous Hash Value
Block 3 Hash Value = Block 4 Previous Hash Value
Block 4 Hash Value = Block 5 Previous Hash Value

Blockchain = VALID!
BUILD SUCCESSFUL (total time: 0 seconds)
```


Output - BlockchainCode (run)

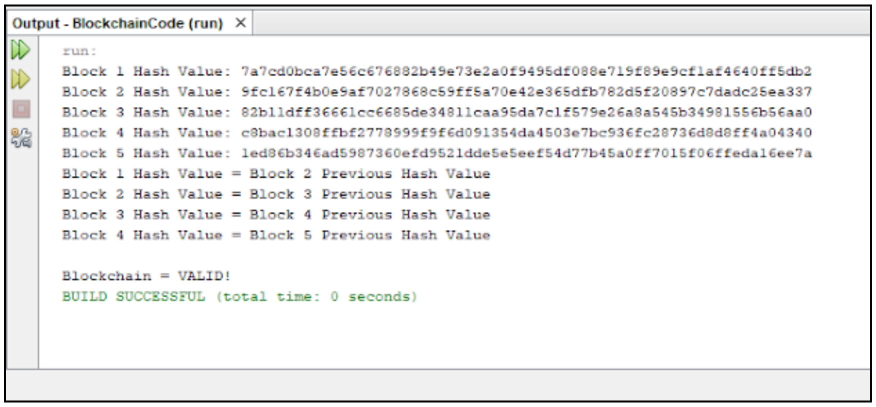
```
run:
Block 1 Hash Value: 3d8a2814868f6c364266d9bd2df766a0d0b72cc9c2d0b0b31abd79cc4f63349e
Block 2 Hash Value: eed4e55faf823010e420cf30e74c2a9de06084b110be3081126ad2741b5326e9
Block 3 Hash Value: 78b1cf9b6b6f9290321000 20128c30acbdf0b028182ea74531c93d818a6b96c1
Block 4 Hash Value: 7c10609677c4296a2464253f24c1b2e123d12b5ac0ff4fe4848c60cc427d9668
Block 5 Hash Value: d21a25222b93a36441d2cd69df8a2528507378c7c052e5fa68b12166395b32fa
Block 1 Hash Value = Block 2 Previous Hash Value
Block 2 Hash Value = Block 3 Previous Hash Value
Block 3 Hash Value = Block 4 Previous Hash Value
Block 4 Hash Value = Block 5 Previous Hash Value

Blockchain = VALID!
BUILD SUCCESSFUL (total time: 0 seconds)
```


Output - BlockchainCode (run)

```
run:
Block 1 Hash Value: 44e8416e50ddc211a0bf4461386b9d224c867189854beed2cf7c3bc097dff029
Block 2 Hash Value: 94443494b45b549ba262c7db735ad2d7ada812a8889e60a4157af2f8ce9b8186
Block 3 Hash Value: a5283700b1f0ea8c854798454b9166d2a7bcabe5e5cbe6203d9cb380644813c3
Block 4 Hash Value: 3b2f4a212e99a26fb4443e524052d7a9164be71521345e4094aed62060c57de4
Block 5 Hash Value: e2623090d0cf10e5eaeb62c66f4cf2c51225ba1b82fc2c26abd817c984cdbd17
Block 1 Hash Value = Block 2 Previous Hash Value
Block 2 Hash Value = Block 3 Previous Hash Value
Block 3 Hash Value = Block 4 Previous Hash Value
Block 4 Hash Value = Block 5 Previous Hash Value

Blockchain = VALID!
BUILD SUCCESSFUL (total time: 0 seconds)
```


Output - BlockchainCode (run)

```
run:
Block 1 Hash Value: 7a7cd0bca7e56c676882b49e73e2a0f9495df088e719f89e9cf1af4640ff5db2
Block 2 Hash Value: 9fc167f4b0e9af7027868c59ff5a70e42e365dfb782d5f20897c7dadc25ea337
Block 3 Hash Value: 82b11dff36661cc6685de34811caa95da7c1f579e26a8a545b34981556b56aa0
Block 4 Hash Value: c8bac1308ffbf2778999f9f6d091354da4503e7bc936fc28736d8d8ff4a04340
Block 5 Hash Value: 1ed86b346ad5987360efd9521dde5e5eef54d77b45a0ff7015f06ffeda16ee7a
Block 1 Hash Value = Block 2 Previous Hash Value
Block 2 Hash Value = Block 3 Previous Hash Value
Block 3 Hash Value = Block 4 Previous Hash Value
Block 4 Hash Value = Block 5 Previous Hash Value

Blockchain = VALID!
BUILD SUCCESSFUL (total time: 0 seconds)
```

In conclusion, Nakamoto invented the idea of the blockchain in 2008 to perform as a registry or log for all transactions in the Bitcoin or cryptocurrency world. Bitcoin came around a long time before the idea of blockchain, around the 1980s. There can be many hackers out there in systems that may try to fool the systems to make people pay multiple times for a single transaction. This digital currency helped bring a new level of security in our world of technology trying to find more and more ways to encrypt and secure our personal transactions and information. Hopefully, with the idea of blockchain, it will create more opportunities to do just that.

References

"Blockchain Definition." *Bankrate*, https://www.bankrate.com/glossary/b/blockchain/.

Pass, Rafael, et al. *Analysis of the Blockchain Protocol in Asynchronous Networks*.

Appendix

```
package block;

/**
 *
 * @author kagarwal
 */
public class BlockchainCode {

    public static void main(String[] args) {
      // Testing random transactions for the blockchain
      String Block1TXN = "Transaction 0";
      String Block2TXN = "Transaction 1";
      String Block3TXN = "Transaction 2";
      String Block4TXN = "Transaction 3";
      String Block5TXN = "Transaction 4";

      //A empty block for storing blocks in arraylist
      Block blockchain = new Block();

      /* There will be 5 blocks in this block chain.
      Block 1 Hash Value is 0 because the previous hash value of the block
      does not exist. */
      Block BLK1 = new Block(Block1TXN,"0");
      Block BLK2 = new Block(Block2TXN, BLK1.makeTHEHASH());
      Block BLK3 = new Block(Block3TXN, BLK2.makeTHEHASH());
      Block BLK4 = new Block(Block4TXN, BLK3.makeTHEHASH());
      Block BLK5 = new Block(Block5TXN, BLK4.makeTHEHASH());

      //Using the get hash method to get the hash value of previous blocks in each block
      System.out.println("Block 1 Hash Value: " + BLK1.makeTHEHASH());
      System.out.println("Block 2 Hash Value: " + BLK2.makeTHEHASH());
      System.out.println("Block 3 Hash Value: " + BLK3.makeTHEHASH());
      System.out.println("Block 4 Hash Value: " + BLK4.makeTHEHASH());
      System.out.println("Block 5 Hash Value: " + BLK5.makeTHEHASH());

      //Adding the 4 blocks to blockchain arraylist
      blockchain.addBlocktoChain(BLK1);
      blockchain.addBlocktoChain(BLK2);
      blockchain.addBlocktoChain(BLK3);
      blockchain.addBlocktoChain(BLK4);
      blockchain.addBlocktoChain(BLK5);

      /*Checks if all the hash values and previous hash values match or not
```

```
        In other words, it is checking if the blockchain itself is valid or not */
        blockchain.Authenticate();

    }
}
```

```
package block;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Date;

/* In a blockchain, a block has a digital signature, a hash value for the current and
previous block, the actual transaction, and a date a time of the transaction to know
when the transaction occurred. All of these will be strings except for the Date/Time;
The digital signature and hash value will be public but the date/time and the actual TXN
will be private for the user's privacy */

public class Block {
   public ArrayList<Block> blockchain = new ArrayList<Block>();
   public String DigitalSignature;              // Digital signature
   public String HashValuePrev;                 // Hash value of the previous block
   private long DateTime;                      // Date + time of the transaction
   private String TXN;                        // TXN = Transaction
   private int nonce;                        // Arbitrary number to randomize hash values

   //Added an empty constructor to store all the information for the blockchain
   Block(){
   }

   /* We need a previous hash value for the current or first block
   For that, we need to create an empty non-real block */
   Block(String Transaction, String PrevHashValue){
      this.HashValuePrev = PrevHashValue;   //Creates the Previous hash value of the
block
      this.TXN = Transaction;              //Creates the transaction
      this.DateTime = createDateTime();     //Creates Date time in milliseconds
      this.DigitalSignature = makeTHEHASH();
   }

   // Get the current hash value for the current block in question
   public String getTHEHASH(){
      return this.DigitalSignature;
```

```
  }

  // Get the previous hash value for the current / first block
  public String getPreviousHashValue(){
     return this.HashValuePrev;
  }

  /* The transaction needs to have a date and time so the user knows
  when the transaction occurred */
  public long createDateTime(){
     Date BLKDate = new Date();
     long time = BLKDate.getTime();   // This will return the time in milliseconds
     return time;
  }

  /* Now we have to make the HASH, we have to put together the
   previous hash value, the digital signature, date-time, etc.
   We have to convert the long time in milliseconds to a string
   To make the create the Hash, we have to make the current signature
   We do that adding the time value, the actual transaction and the
   previous hash value of the blocks */

  public String makeTHEHASH(){
     String DateTimetoString = Long.toString(DateTime);  // Convert time to string value
     String makeTHEHASH = HashValuePrev + DateTimetoString + TXN; // Compile to
make digital signature for the current block
     String hash = encryptTXN(makeTHEHASH);       // Previous hash values to make
the new hash
     return hash;            // Return hash value
  }

  // Hash algorithm
  public String encryptTXN(String TXN){
     try {
        MessageDigest PROG = MessageDigest.getInstance("SHA-256");   // SHA-256
ALGORITHM
        byte[] cipheredHASH = PROG.digest(TXN.getBytes(StandardCharsets.UTF_8));
        StringBuffer K = new StringBuffer();         // Put all ciphered hash into one string
        for (int i = 0; i < cipheredHASH.length; i++) {
        // Taking the hash just ciphered from the SHA-256 algorithm and converting to
hex
           String hex = Integer.toHexString(0xff & cipheredHASH[i]);
           K.append(hex);             //Adding hex values to String buffer K
        if(hex.length() == 1){             //Add 0s when the hex length is only 1
             K.append('0');
```

```
                }
              }
           String encrypted = K.toString(); // Change string buffer K to string datatype
           return encrypted;                // Return value
        }
        // Catch clause just in case netbeans doesn't support SHA-256 algorithm
        catch (NoSuchAlgorithmException ex) {

java.util.logging.Logger.getLogger(Block.class.getName()).log(java.util.logging.Level.SE
VERE, null, ex);
        }
        return null;
    }

    public void addBlocktoChain(Block block){
     blockchain.add(block);                  // Add all blocks to the actual blockchain
}
    // Checks if the blockchain itself is valid or not
    public void Authenticate(){
        Block beforeBLK;        //Before and present hash values have to be equal
        Block presentBLK;
        int count = 0;          // Count the blocks
        for(int i = 1; i < blockchain.size(); i++){
           presentBLK = blockchain.get(i);
           beforeBLK = blockchain.get(i-1);
           if(presentBLK.getPreviousHashValue().equals(beforeBLK.makeTHEHASH())){
              System.out.println("Block " + i+ " Hash Value" + " = Block " + (i+1) + "
Previous Hash Value");
              count++;
           }
           else{
             System.out.println("Block " + (i-1)+ " Hash Value" + " = Block " + i + " Previous
Hash Value");
           }
        }

        // if all the blocks in the blockchain are valid, then the blockchain itself is valid */
        if(count == blockchain.size() - 1){
          System.out.println("\nBlockchain = VALID!");
        }
        else{
           System.out.println("Blockchain = NOT VALID!");
        }
    }
}
```