COL 819 : Assignment - 3
Bitcoin

Shubham Gupta
csz198470@cse.iitd.ac.in

August 14, 2020

1 Introduction

Presently major transactions whether physical or digital involve some third party financial institutes to prevent the double spending problem. Nakamoto (2009) introduced the system to solve the problem of double spending without use of third party trusted institutes. He named it Bitcoin. It is a peer to peer distributed network which allows for transactions between any two party without involvement of any particular financial institutes. This allows for complete decentralization of payment system with no party actually knowing the actual users behind transactions.
The major benefits of this systems are - Easy mobile payments, security of your bitcoins, no down time, identity protection and no reversal of transactions.
The Bitcoin concepts can be understood in following simple steps -

1. Lets say Alice want to send Bob money, then Alice can simply scan the Bobs' public key using Bob's QR or ask him to send the public key.
2. Now Alice can create a transaction which will reference the transaction of which Alice was a recipient. And, it will put the public key of Bob as output in transaction and further the amount. Further, Alice will sign the transaction with its private key and put the sign in this transaction. And Alice will broadcast this transaction across the network.
3. Now miners will collect this transaction and verify the transaction using Alices' public key and signature in this transaction. Further it will see the public key of Alice matches with output public key present in the transaction Alice is referring to. These miner nodes will also further collect other transactions.
4. Now after certain amount of time, miner will combine all the collected transaction in a block and compute the proof of work.

5. **Proof of work** is to keep computing hash of a value concatenated with given string until the hash begins with fixed number of 0th bits. This fixed number of 0th bits is called nounce. Computation required is exponential in terms of nounce.

6. So, Miner will hash the block and compute the proof of work for it. Once done, it will attach the proof of work with the block and broadcast it to the network. Also, miners are given some fixed bitcoin as **incentive** for creating the block.

7. Now, other miners in network will verify the proof of work which has O(1) time complexity. They will accept this block by computing the next block on this block. Every block in the chain references has of previous block thus creating the chains of block which is called the blockchain. If some dishonest nodes add a block which contains a double spending transaction, other honest nodes will keep working on original block and keep extending it. Thus, preventing double spending

8. Once Bob sees this transaction in verified block, it assumes that transaction is completed and verified. Bob also can wait for further blocks before providing service to Alice in order to ensure that Alice doesn't double spend.

9. Now Bob can claim the transaction by putting its public key in next transaction with reference to the transaction created by Alice, then miner will only verify this transaction if its public key matches with public key in output of Alice's transaction and it can verify the content of Bobs' transaction by its Bobs' public key.

This system works as expected if majority of CPU power lies with honest miners. Further, nodes can keep changing their public and private key in order to avoid the personal identification. One of major disadvantage of bitcoin is long time of payment verification. Generally a block is created within the span of 10 minutes. So 10 minutes is minimum time to verify the transaction. Further if you want to prevent the double spending with high degree of confidence, then you will have to wait for 6 blocks which is approx. 1hr.

2 Implementation Details

I have created a bitcoin implementation for it to be close to true simulation of bitcoin. This implementation is part of course taken by Prof. Sarangi (2020). It

is implemented in Python3.7. Multiprocessing library is used to create multiple nodes which act both as a transaction entities and mining entity. We assume the instant noise less communication between nodes, so we have used Queue library for message passing between nodes. Further, I have conducted all experiments over my personal laptop which has following specification - 1.8GHz Intel core i5 , 8GB RAM and 64 bit Mac OS Mojave OS. Although, project should run without any issues on other Operating system but no guarantees.
We first define the transaction entity. Please note that this transaction class support Multi Input and Multi Output transactions i.e. a node can reference multiple coins that it has and send it to multiple nodes at same time.

```python
class Transaction():
    def __init__(self,sender_address,inputs,outputs,private_key,t_type,
    hash_type):
        self.t_type = t_type
        self.tx_in_ct = len(inputs)
        self.tx_in = inputs
        self.tx_out_ct = len(outputs)
        self.tx_out = outputs
        self.sender_address = sender_address.hex()
        self.time = str(int(time.time()))
        self.hash_type = hash_type
        self.sign = self.sign_transaction(key_in_RSA_object(private_key)
    )
        self.txid = generate_hash(self.string_of_transaction(),type=
    hash_type)
class Input():
    def __init__(self,prev_output_tid,index):
        self.prev_output_tid = prev_output_tid
        self.prev_output_index = index
class Output():
    def __init__(self,to_address,amount,hash_type):
        self.to_address = generate_hash(to_address,type = hash_type)
        self.amount = amount
```

Nodes digitally signs the transaction by using RSA signature protocol with specification wrt PKCS#1 v1.5 .Its library is provided in python as Crypto.

Now Lets further see the design of Block.

```python
class Block():
    def __init__(self,proof_of_work_zeros,index,narry,transactions,
    previous_block_hash,hash_type,block_type="Regular"):
        self.block_type = block_type
        self.previous_block_hash = previous_block_hash
        self.narry = narry
        self.index = index
        self.pow_number_of_zeros = proof_of_work_zeros
        self.hash_type = hash_type
        self.transactions_count = len(self.transactions)
        self.merkle_tree = create_merkle_tree([item.txid for item in
    self.transactions],self.narry,hash_type) #### given the list of
    transactions, create their hashes and get the merkle tree
        self.merkle_tree_root = self.merkle_tree[-1][0]
        self.pow_number_of_zeros)
        self.compute_proof_of_work()
```

In order to save the disk space, transactions are hashed, merkle tree is computed and root hash is stored in the block header. Further, Block chain also maintains the set of unspent transaction in order to quickly verify the upcoming transactions. The block architecture is inspired from `https://developer.bitcoin.org/devguide/block_chain.html` and transaction architecture is taken from `https://developer.bitcoin.org/devguide/transactions.html`

In this implementation, block creation time is set at 30 or 60 seconds(depending on the experiment) in order to do quick simulation. And the simulation is scalable up to 100 or 1000s of nodes.

3 Experiments

We first see the size of blockchain which uses SHA-256 hashing after 10 transactions. It can be seen from figure 1 that size of blockchain decreases after increasing the arity of merkle tree. Since, higher the arity of merkle tree, less intermediate nodes there will be in merkle hash tree. Thus, lesser size. Hash function is SHA-256 and proof of work nounce size is 4. Unless specified, network size is 10
We also see from figure 2 that impact of block-header size with different hash functions. Block headers in bitcoin design only store the root hash of merkle tree of transactions. Arity is 2 and proof of work nounce size is 4

We further see the plot of nounce sizes in proof of work vs Block creation time. We fix the arity of merkle tree at 2 and hash function at SHA-256. Figures 3 and 4 shows that the trend is exponential which is whole reason of doing proof of work. Harder the proof of work, less nodes will create block, thus less scope of double spending attack. 4 is a box plot of block creation time of 20 transaction, where sim-

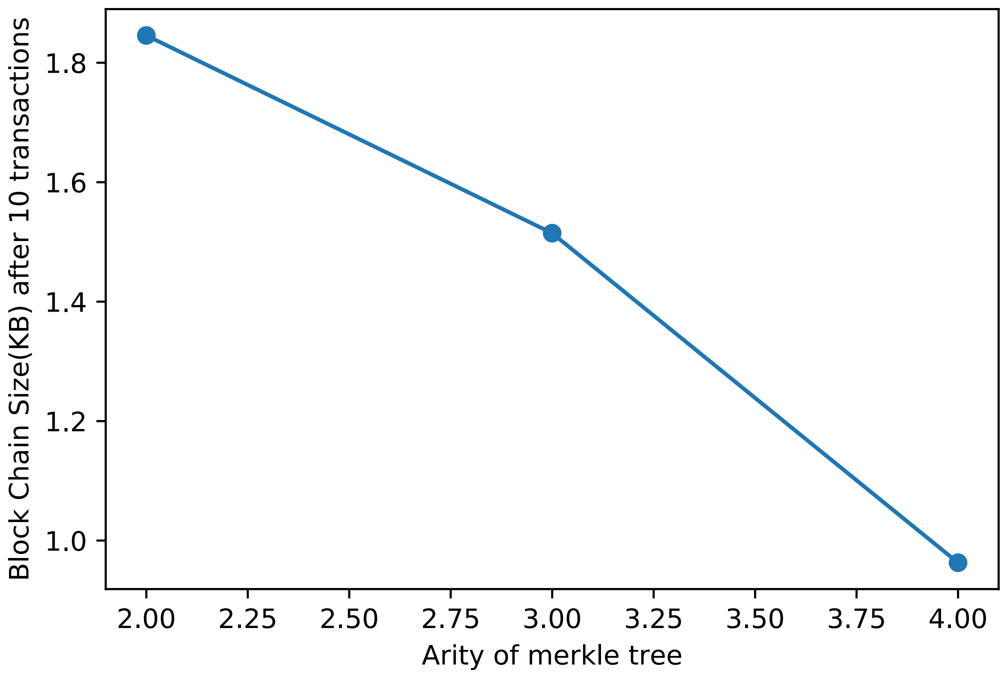

Figure 1: Blockchain size vs. Arity of merkle tree

ulation is repeated 100 times to get clear distribution for each nounce. Network size is 20.
We further see the plot of Genesis block creation time wrt various hash functions in Figure 5. We fix the proof of work nounce at 2 and arity of merkle tree at 2.

Now, we create a box plot figure 6 of the block creation time with 20 transactions vs the hash function time. We fix the arity at 2 and proof of work nounce size at 3. Network size is 20. We run this simulation 100 times to get a better approximation of true distribution. We further see that SHA384/SHA512 is performing faster than SHA256 or below version. On searching about this, I found that SHA-512/384 are faster than SHA256 on 64 bit platform and vice versa in 32 bit platform.

Next, we plot the box plots of time taken by a each transaction to get verified on different size node networks. We fix the number of transaction at 10 for each network size. Please note that block creation time is kept at 30 seconds, so typical verification time is around 60. further each node can only do transaction in interval of 15 seconds. It seems that as network size is increasing, median time is increasing as well which might be due to increased communication overheads.

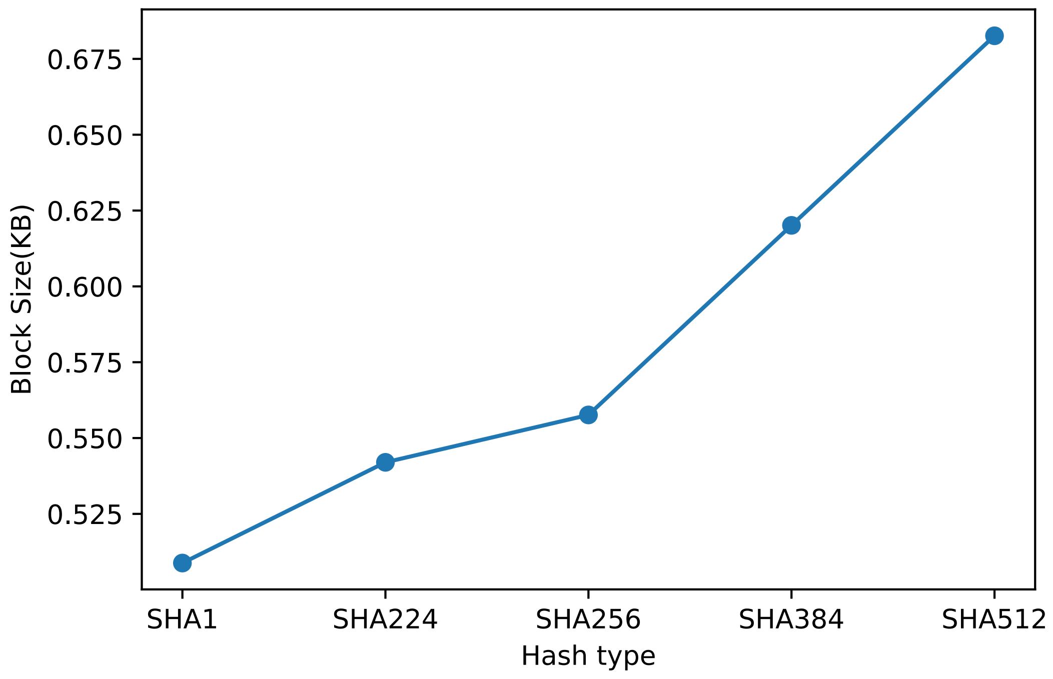

Figure 2: Block size vs hash function

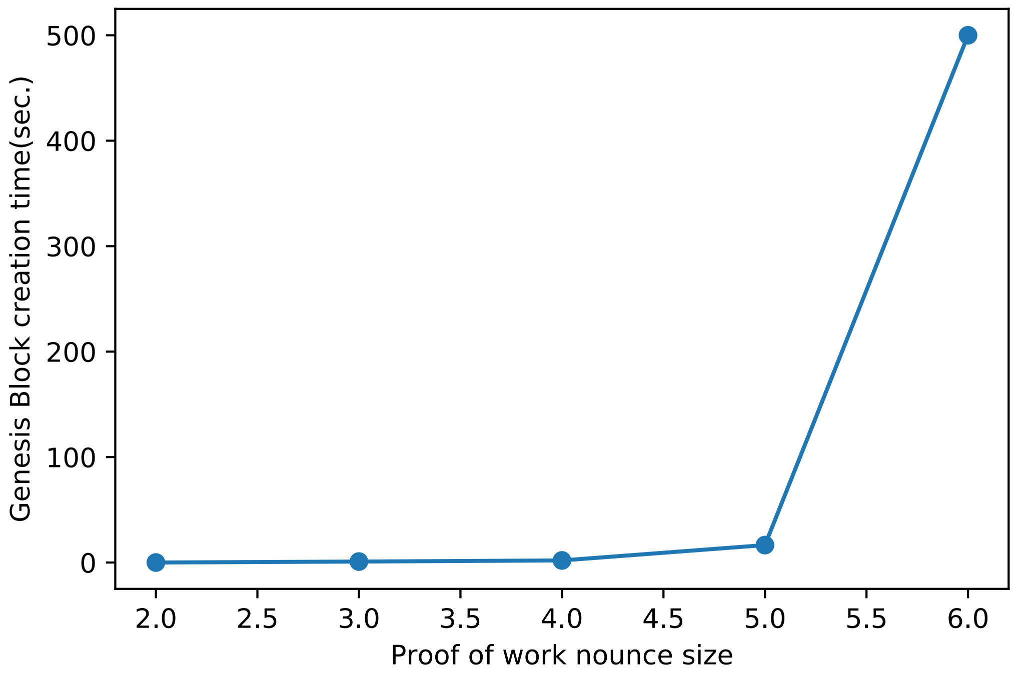

Figure 3: Genesis Block creation time vs proof of work nounce size

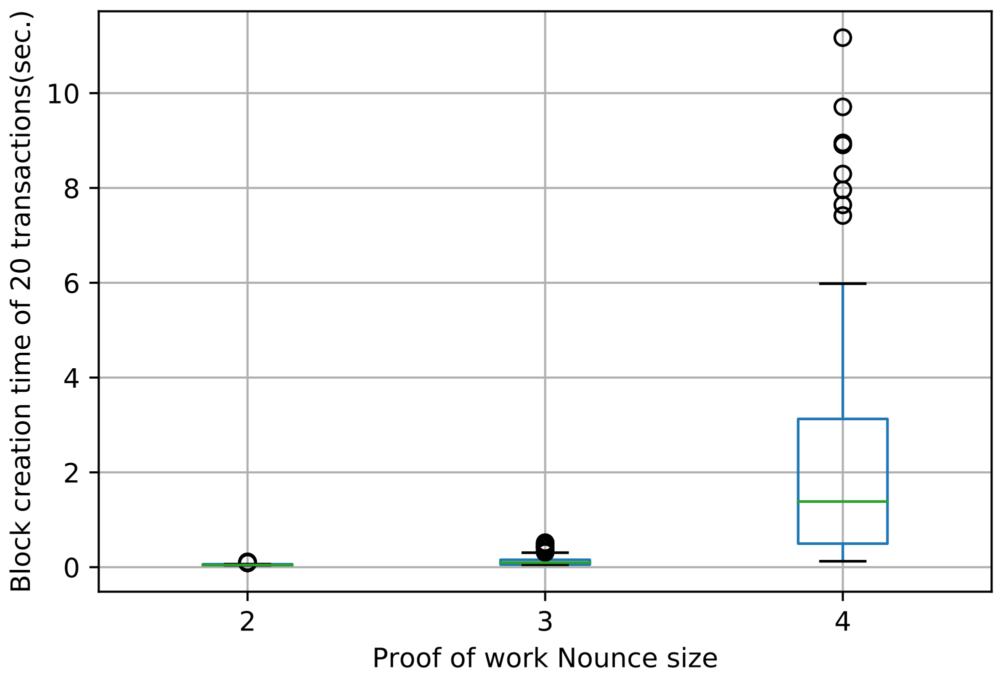

Figure 4: Block creation time of 20 transactions vs proof of work nounce size

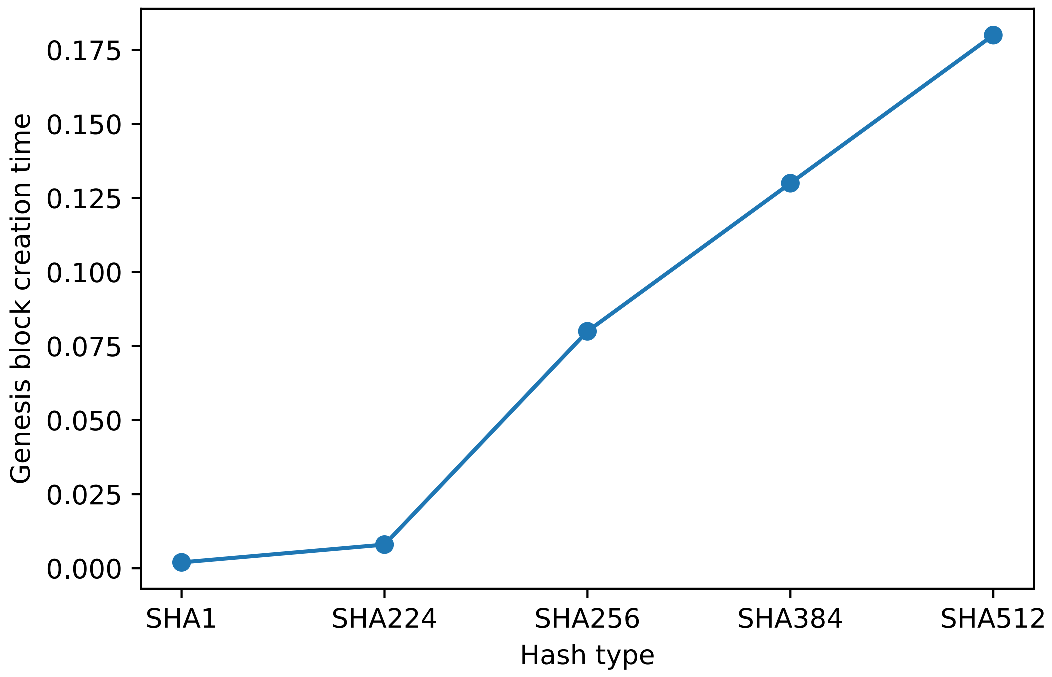

Figure 5: Genesis Block creation time(sec.) vs Hash function(proof of work nounce size at 2)

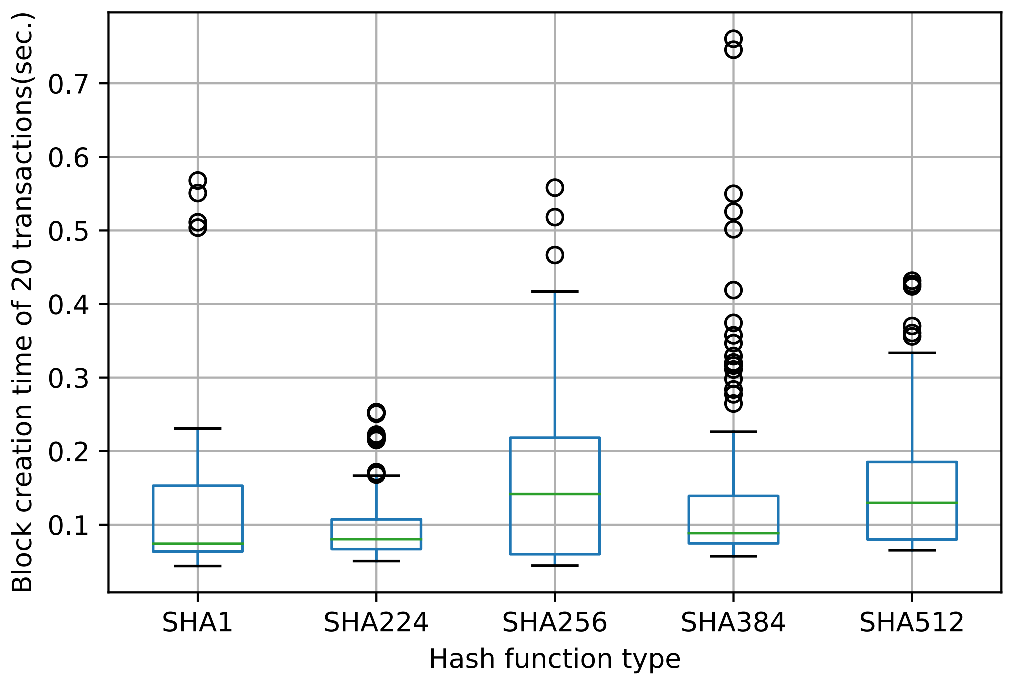

Figure 6: Block creation time vs Hash function(proof of work nounce size at 3, arity at 2)

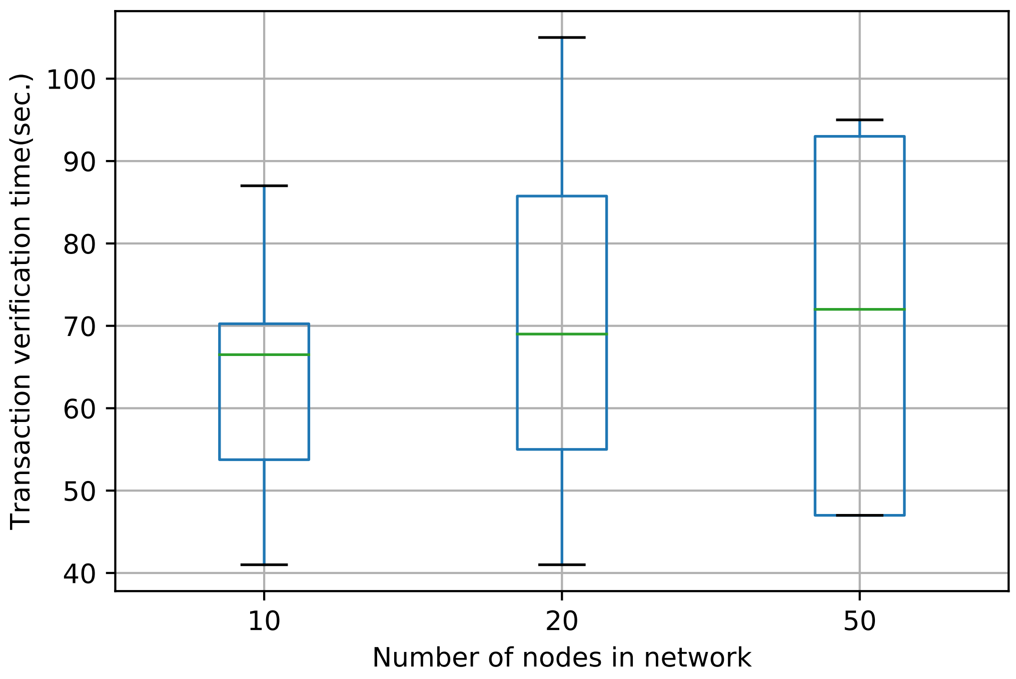

Figure 7: Transaction verification time(sec.) vs Network Size(proof of work nounce size at 4, arity at 2 and hash function is SHA-256)

4 Transaction Simulation

Now we further show some transaction on 10 Node network and show the initial states of nodes. Please note that each miner node will charge fixed 1BTC for every transaction it process. Further, block creation award is set at 10BTC. Proof of work is set at 4. A simulation is ran with limit of 4 transactions, each node can only do a transaction between interval of 20 seconds and block creation time is 30 seconds. Further, each transaction involve a transaction fee of 1BTC. Please also note that, in every simulation node 0 will create a genesis block and award 1000 btc to every node currently in network. Also it will award 10 BTC for block creation to itself. Further, it will broadcast this genesis block to all other nodes. They will accept this genesis block and start doing transactions. In current simulation, following transactions happened.

```
Node 3 sent 3 btc to node 7
Node 0 sent 1 btc to node 5
Node 4 sent 3 btc to node 8
Node 5 sent 4 btc to node 0
```

Table 1 shows the before and after status of each node after transactions.

Node id	BTC before transactions	BTC after 4 transactions
0	1010	1012
1	1000	1000
2	1000	1000
3	1000	996
4	1000	996
5	1000	1009
6	1000	1000
7	1000	1003
8	1000	1014
9	1000	1000

Table 1: Nodes Status Before and After Transactions

Following log also show the interactions as they happened during simulation. It can be seen that from logs that node 0 received 4 btc from node 5. Also, node 0 transferred 1 bitcoin to node 5 and paid 1 BTC as transaction fees. Also, It can be seen that node 8 created the block with 1 transaction. So, it got the BTC reward of 11 BTC and further received 3 BTC from 4. Thus, balance of 1014. Note that 2nd block of next 3 transactions in blockchain is created by node 5, so it receives 10 + 3 btc as rewards.

```
Initialized miner for node, 2
Initialized miner for node, 1
Initialized miner for node, 3
Initialized miner for node, 0
Initial money
0 1010
Initialized miner for node, 7
1 1000
2 1000
3 1000
4 1000
5 1000
6 1000
7 1000
8 1000
9 1000
Initialized miner for node, 5
Initialized miner for node, 8
Initialized miner for node, 6
Initialized miner for node, 9
Initialized miner for node, 4
Time taken in genesis block creation, 4.06429386138916
0 Received 10 BTC as block creation reward
Node 3 Sending 3 btc to node 7
3  Creating a block  1
8  Creating a block  1
9  Creating a block  1
2  Creating a block  1
8  No block has been recieved, so send it everyone
8  sending created block to  0
8  sending created block to  1
0 Block has been received from 8
8  sending created block to  2
8  sending created block to  3
8  sending created block to  4
8  sending created block to  5
8  sending created block to  6
8  sending created block to  7
8  sending created block to  9
0  Received block has been added to blockchain
4 Block has been received from 8
2 has balance of 1000 BTC
4  Received block has been added to blockchain
1 Block has been received from 8
8 Received 11 BTC as block creation reward
5 Block has been received from 8
1  Received block has been added to blockchain
7 Block has been received from 8
```

```
5  Received block has been added to blockchain
8 has balance of 1011 BTC
7  Received block has been added to blockchain
6 Block has been received from 8
3 verified last transaction
3 has balance of 996 BTC
6  Received block has been added to blockchain
9 has balance of 1000 BTC
Node 0 Sending 1 btc to node 5
Node 4 Sending 3 btc to node 8
Node 5 Sending 4 btc to node 0
5  Creating a block  3
4 has balance of 1000 BTC
1 has balance of 1000 BTC
6 has balance of 1000 BTC
0  Creating a block  3
7 Received 3 BTC from 3
7 has balance of 1003 BTC
5  No block has been recieved, so send it everyone
5  sending created block to  0
5  sending created block to  1
5  sending created block to  2
5  sending created block to  3
5  sending created block to  4
5  sending created block to  6
5  sending created block to  7
5  sending created block to  8
5  sending created block to  9
6 Block has been received from 5
7 Block has been received from 5
6  Received block has been added to blockchain
4 Block has been received from 5
9 Block has been received from 5
8 Block has been received from 5
7  Received block has been added to blockchain
5 has balance of 1000 BTC
4  Received block has been added to blockchain
9  Received block has been added to blockchain
1 Block has been received from 5
2 Block has been received from 5
8  Received block has been added to blockchain
3 Block has been received from 5
0 has balance of 1010 BTC
1  Received block has been added to blockchain
2  Received block has been added to blockchain
3  Received block has been added to blockchain
```

```
7 has balance of 1003 BTC
6 has balance of 1000 BTC
5 Received 13 BTC as block creation reward
5 Received 1 BTC from 0
5 verified last transaction
5 has balance of 1009 BTC
4 verified last transaction
4 has balance of 996 BTC
8 Received 3 BTC from 4
8 has balance of 1014 BTC
9 has balance of 1000 BTC
0 verified last transaction
0 Received 4 BTC from 5
0 has balance of 1012 BTC
3 has balance of 996 BTC
2 has balance of 1000 BTC
1 has balance of 1000 BTC
8  Creating a block  0
3  Creating a block  0
1  Creating a block  0
2  Creating a block  0
9  Creating a block  0
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
0 has balance of 1012 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
7 has balance of 1003 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
6 has balance of 1000 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
5 has balance of 1009 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
4 has balance of 996 BTC
9 has balance of 1000 BTC
8 has balance of 1014 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
3 has balance of 996 BTC
```

```
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
2 has balance of 1000 BTC
waited for transactions for more than  120   seconds, seems none is
    using bitcoin, lets die
1 has balance of 1000 BTC
```

5 Multi-transaction Simulation

We do a 1 multi transaction to show the ability of system to handle such transaction. We only do 1 to not complicate things in this reports. Node 6 creates the block with 1 transaction in it, so it gets a reward of 1 btc. Thus, balance of 1011. Further node 4 transfers 10 btc to node 3 and 0 and pays 1 btc for transaction processing. So, it has balance of 989.

```
Node 4 Sending 4 btc to node 3 and 6 btc to node 0
```

Table 2 shows the before and after status of each node after transactions.

Node id	BTC before multi transaction	BTC after 1 multi transaction
0	1010	1016
1	1000	1000
2	1000	1000
3	1000	1004
4	1000	989
5	1000	1000
6	1000	1011
7	1000	1000
8	1000	1000
9	1000	1000

Table 2: Nodes Status Before and After multi transaction

6 Smart Contracts

In this project, we have designed a smart contract on node 0 and node 1. Execute once smart Contract on node 0 is that if the balance of node 0 is more than or equal to 700, then it will transfer 100 BTC to node 8. Repeat Smart Contract on node 1 is that if it has a balance more than or equal to 700, it will keep transferring 100 to node 9. In this transaction total, 13 transactions were performed which can be seen from below formatted log. It can be seen that since we put ”execute once” smart contract on node 0, transfer of 100BTC to node 8 , happened only once. But since we put ”repeat” smart contract, at node 1, it is executed until bitcoins on node 1 are less than 700. Please note that blocks are created by 4,9,2,9,4,1,7,3. So they get 10 BTC block reward with more rewards as per transactions each block contains.

```
############
Node  0  : balance is  1010 btc  so as smart contract of execute once,
   sending  100 btc to  8
############
############
Node  1  : balance is  1000 btc so as smart contract of repeat, sending
    100 btc to  9
############
Node 0 Sending 4 btc to node 8
Node 4 Sending 3 btc to node 7
Node 3 Sending 4 btc to node 8
############
Node  1  : balance is  899 btc so as smart contract of repeat, sending
   100 btc to  9
############
Node 1 Sending 100 btc to node 9
Node 4 Sending 3 btc to node 3
Node 3 Sending 4 btc to node 9
Node 0 Sending 4 btc to node 8
############
Node  1  : balance is  798 btc so as smart contract of repeat, sending
   100 btc to  9
############
Node 3 Sending 3 btc to node 9
Node 4 Sending 4 btc to node 3
############
Node  1  : balance is  709 btc so as smart contract of repeat, sending
   100 btc to  9
############
```

Following table 3 shows the before and after status after execution of smart contracts.

Node id	BTC before smart contract execution	BTC after smart contract execution
0	1010	899
1	1000	608
2	1000	1013
3	1000	1004
4	1000	1009
5	1000	1000
6	1000	1000
7	1000	1014
8	1000	1112
9	1000	1431

Table 3: Nodes Status Before and After smart contract execution

7 Current Problems of bitcoins and possible Solutions

In this section, we further explore various problems with current bitcoin architecture and what are the possible solutions.

7.1 Private key theft

Essence of the issue is that, a way to access any bitcoin is to produce the private key corresponding to the public key it is referring to. If a thief is able to steal the private key, then S/he can easily create a new transaction and transfer that coin to its own public key. This basically means that private key need to be kept securely over digital wallet or in hardware wallets.
One solution is to keep k pairs of public and private key pairs. It can be called chained signing. Lets name these K public private key pairs as (PU1, PR1)....(PUK,PRK). And keep these private keys at different server or places. So, in order to receive bitcoin, send PU1, PU2... PUK to sender and ask him to maintain the order while putting these public keys in transaction. Further if you want to spent this bitcoin , then you first sign it using PRK, now send this signed msg to other server containing PRK-1, and sign it using this private key. Follow this until you sign message using PR1. Now send this sign across miner network. In order to verify this transaction, miner will open the transaction using PU1, PU2 ... PUK. If the content matches, then it will verify the transaction. K pairs of signs will reduce the scope of private key theft and all of them will be needed to spent this transaction.

7.2 Hash power accumulation

Earlier email was designed to be one on one decentralized messaging system. But now very few email service providers are in play like Gmail, Yahoo and Outlook. This issue is now present in bitcoin also. Look at the following figure 8 taken from `https://developer.bitcoin.org/devguide/transactions.html`. Difficulty started from 1 at start of bitcoin and now it has reached to 15 trillion. There are very few groups who will have this much compute power or specialized hardware to process this difficulty. And Now only those groups can mine further bitcoins or process transactions. The very idea of removing centralized third party systems from transactions is gone from bitcoin.

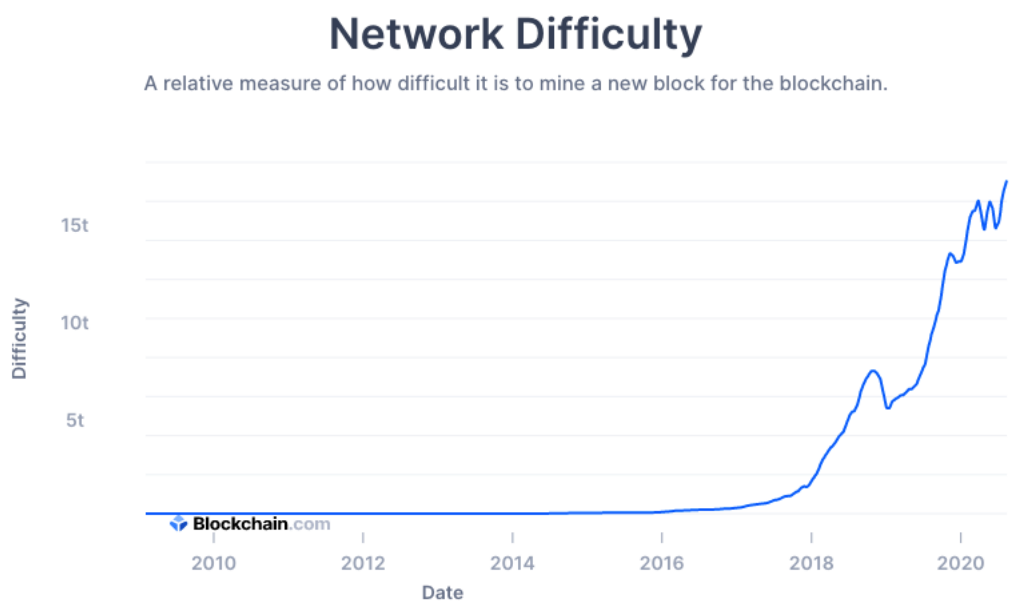

Figure 8: Mining difficulty over years

7.3 Privacy

Theoretically, an individual can create new public private key pair for every new transaction. This way, no outside party can figure out his identity. But, this also brings other horrific uses of bitcoin which can be detrimental to society. So, in order to avoid this, govt. can mandate to register every private key with certain govt. identity with govt. Thus, to demotivate the individuals to conduct illegal acts. But also keeping the identity secret with other individuals.

8 How to run the project

Program has been tested in python3.7.

```
python3 main.py <num_node> <arity_of_merkle_tree> <proof_of_work_nounce>
    <hash_function>
```

num_nodes is between 2 to 100, arity_of_merkle_tree can be between 2 and above, proof_of_work_nounce is an integer and hash_function can be from SHA1, SHA224, SHA256, SHA384 and SHA512. for example, following is a valid input to run the simulation.

```
python3 main.py 10 2 4 SHA256
```

References

Nakamoto, S. (2009). *Bitcoin: A peer-to-peer electronic cash system.* Retrieved from `https://bitcoin.org/bitcoin.pdf`

Sarangi, S. R. (2020). *Bitcoin and blockchains.* Retrieved from `http://www.cse.iitd.ac.in/~srsarangi/courses/2020/col_819_2020/docs/bitcoin.pptx`