



UNIVERSITY OF  
**WATERLOO**

## **PROJECT 2: PROOF-OF-WORK (PW) IN BITCOIN    BLOCKCHAIN** **ECE 628 – WINTER 2019**

Ruta Patel (ID: 20755706)  
University of Waterloo  
rv8patel@uwaterloo.ca

Ramneet Kaur (ID: 20778961)  
University of Waterloo  
rkramnee@uwaterloo.ca

**ABSTRACT:** In this digital era, it is said that technology is changing every minute. Over the years, technology has completely revolutionized our world and has created amazing tools and resources such as smart phones and smart watches, putting useful information at our fingertips.[1] When it comes to communication, modern technology has made a remarkable influence in our lives. Especially, the emergence of digital platform has made our lives much simpler, faster and convenient. Such an innovation, an idea, was conceived 10 years from now by the genius mind of **Satoshi Nakamoto** (original identity not revealed) that might be the strongest foundation ever laid in the field of cryptocurrency. Over the short span of time, Bitcoin has had a similar disruptive and innovative effect on the financial world and the technology supporting crypto currency has gone on to influence several sectors in the global economy and that is Blockchain. Blockchain and Bitcoin are the 2 words that are used together and for all the right reasons. This report is all about Bitcoin and Blockchain and all other parameters around it. We will be covering all the important aspects of blockchain Bitcoin, discussing about its emergence and the technology it uses. Our focus is on proof of work which involves solving a computational puzzle.

**KEY WORDS:** Bitcoin, Blockchain, Proof of Work (POW), double spending, 51% attack, Proof of Stake(POS), Proof of Burn(POB).

### **INTRODUCTION**

Since the introduction of Bitcoin [2] as a prototype for a decentralized cryptocurrency, the field of cryptocurrency technologies has experienced a rapid growth in popularity. To date, over 700 different cryptocurrencies have been created [2]. Some of those currencies only had a very short lifespan or were merely conceived for fraudulent purposes, while others brought additional innovations and still have vital and vibrant communities today. Bitcoin was designed to be a decentralized cryptographic currency that does not rely on trusted third parties. “It achieves this by combining clever incentive engineering and the right cryptographic primitives with a novel probabilistic distributed consensus approach. This combination and the practical demonstration

of its feasibility are proving to be a significant contribution that has the potential to profoundly impact other domains beyond cryptocurrencies.” [2]

The manufacturing and exchange of currencies have always been taken place through a third party. Whether, it is a payment for a small product bought online or transferring large sums of money through international wire transfer, there is always an involvement of a centralized entity such as banks. It is only through banks and the trust we hold into them that we can exchange money on a large and wide scale. If one wants to make a transaction, the bank is the institution which sets the rules on how this transaction will be executed, how long will it take, how much they will charge you and so on. Blockchain however eliminates this very important part or the ‘middle man’ from the picture. Blockchain basically helps us operate this transaction one-to-one without the bank as an intermediary. It is rightly said that, “Given the scale and complexity of the global economy as well as our knowledge about human nature, it would be extremely naïve to rely simply on spontaneous and voluntary ethical behavior by individuals and corporations to ensure fairness”[1]. Therefore, emergence of Blockchain and Bitcoin is considered a boon for us.

Blockchain is a platform that reduces the dependency on a single centralized authority such as banks to support secure and “trustless” transactions directly between interacting entities. In simpler terms, blockchain is a long chain of blocks of data added by the miners into the database network. Miners are the people that verify the digital signatures and private keys of each posted transactions in exchange of rewards i.e. bit coins. After, the transaction is verified and authenticated as a true piece of information it is added into the block with a hash function of its and that of the block before it.

## **PROOF OF WORK**

Proof-of-Work, or POW, is the original consensus algorithm in a Blockchain network. In Blockchain, this algorithm is used to confirm transactions and produce new blocks to the chain. POW is a mining process, i.e. a miner computes a nonce in the block such that its hash value has the required number of zeros. When the first miner finds the pre-image, he broadcast it and the other miners verify its validity i.e. the authenticity of the hash chain. If it is true, a block with signed transaction and POW is added, and this miner will be rewarded for some Bitcoins. In this way, transaction data is permanently recorded in Blocks, which is irreversible.

## **OUR WORK**

In this project, our objective is to implement a DSS module and a hash chain-based authentication of length 3 for Proof of Work i.e., blockchain with length 3. The project includes 5 tasks to be done and each step is explained specifically in the following section. Also, the procedure carried out, what methodology we used in executing the proof of work and what challenges we faced in doing so are mentioned in detail. All the computations are done in **Python** with some basic calculations done in **Sage**. The reason to select Python as a tool for this

project was due to its simple programming language and our background experience working with it. All the programs written are attached with this report. The basic flow of the project is as follows:

- i) Generating Private and Public keys for 3 blocks of blockchain. One pair of keys was already given. We generated the other 2 pairs of the key and verified them.
- ii) After generating the keys, the next step is to generate the signatures for the user transactions using a DSS module. SHA3-224 is used as hash function for the module.
- iii) The third and the most important step is to demonstrate the proof of work. Computations for T1 and T2 are carried out using the SHA3-224 function and a pre image of fixed hash format is generated.
- iv) The fourth step includes literature study on Double spending and how Proof of work prevents it. Also, the 3 methods used in cryptocurrency today to prevent the same are explained.
- v) Two possible attacks on Proof of Work and their possible counter measures are listed in the last section.

## **I. SYSTEM PARAMETERS VALIDATION**

The first step of our project was to generate the private and public keys for blockchains of length 3. Asymmetric cryptography or public cryptography is an essential component of Bitcoin. It ensures that the source of transaction is legitimate and that hackers cannot steal users' funds. The public key is used with a hash function to create the public address that Bitcoin users use to send and receive funds. The private key is kept secret and is used to sign a digital transaction to make sure that the origin of the transaction is legitimate.

For generating a digital signature, firstly public key and private keys are generated. 3 criteria are tested beforehand which are:

- a prime number  $p$ , whose length is a multiple of 64 bits lying between 512 and 1024 bits
- a 160-bit prime factor  $q$  of  $p - 1$
- an element  $g \in F_p$  with order  $q$

The first set of keys ( $sk_1$ ,  $pk_1$ ) was given to us. We generated the other 2 pair of the keys and verified them.

To check whether the  $p$  is prime number or not, the `is_prime()` function of sage is used because it was much simpler and consumed less time.

For verifying  $q$ , we used the formula  $\text{mod}((p-1),q)$  and the output came 0 which proved that indeed  $q$  is a prime factor of  $p-1$ .

Furthermore, we verified  $g$  by using the formula  $g = u^{((p-1)/q)} \bmod(p)$

Since we had value of  $g$ ,  $p$  and  $q$ , we computed value of  $u$  in C language and ran program in online IDE and then computed  $\text{inverse\_mod}(u,p)$ . The value of  $u$  was checked for condition

$0 < u < p$  and hence we verified  $g$ .

```
SageMath version 8.2, Release Date: 2018-05-05
Type "notebook()" for the browser-based notebook interface.
Type "help()" for help.

sage: p=168199388701209853920129085113302407023173962717160229197318545484823101018386724351964316301278642143567435810448472465887143222934545154943005714265124445244247988777471773193847131514083030740407543233616696550197643519458134465700691569680905568000063025830089599260400096259430726498683087138415465107499
sage: p
168199388701209853920129085113302407023173962717160229197318545484823101018386724351964316301278642143567435810448472465887143222934545154943005714265124445244247988777471773193847131514083030740407543233616696550197643519458134465700691569680905568000063025830089599260400096259430726498683087138415465107499
sage: q=959452661475451209325433595634941112150003865821
sage: is_prime(p)
True
sage: is_prime(q)
True
sage: mod(p-1,q)
0
sage: |
```

```
main.c
1- /******
2-
3-               Online C Compiler.
4-               Code, Compile, Run and Debug C program online.
5-               Write your code in this editor and press "Run" button to compile and execute it.
6-
7-               *****/
8-
9- #include <stdio.h>
10-
11- int main()
12- {
13-     int long g=9438919277632739858984532698034981452643386909341278234543094605920656880400518160085582590614296727187254837587773894987581254043315249550199954988698482800240809770832611439716298370837596936822863053814776793141937952361960552261492301676451260936250088824754901721955772380106289054700741139017131625506259085579415604926609378316109847120779729102035374024558376047279827439257236713822888684722501987212544295577973364701237120951383;
14-     int long p=168199388701209853920129085113302407023173962717160229197318545484823101018386724351964316301278642143567435810448472465887143222934545154943005714265124445244247988777471773193847131514083030740407543233616696550197643519458134465700691569680905568000063025830089599260400096259430726498683087138415465107499;
15-     int long q=959452661475451209325433595634941112150003865821;
16-     int long u = g^(q/(p-1));
17-     printf("g^(q/(p-1) is %ld \n", u);
18-
19- }
20-
21-
input
g^(q/(p-1) is 7232535151378612740
...Program finished with exit code 0
Press ENTER to exit console.

sage: u=7232535151378612740
sage: inverse_mod(u,p)
15249550199954988698482800240809770832611439716298370837596936822863053814776793141937952361960552261492301676451260936250088824754901721955772380106289054700741139017131625506259085579415604926609378316109847120779729102035374024558376047279827439257236713822888684722501987212544295577973364701237120951383
sage: |
```

We then verified  $sk_1, pk_1$  by using following conditions

**Private key:**  $sk_B = x, 0 < x < q$ ; **Public key:**  $pk_B = y = g^x$ .

Since  $sk_1$  must be less than  $q$  therefore it answers our question that why  $sk_1$  has to be less than 160 bits.

After verifying them, we assumed  $sk_2$  and  $sk_3$  and computed  $pk_2, pk_3$  in sage

```

SageMath 8.2 Console
SageMath version 8.2, Release Date: 2018-05-05
Type "notebook()" for the browser-based notebook interface.
Type "help()" for help.

sage: p=168199388701209853920129085113302407023173962717160229197318545484823101
..... 0183867243519643163012706421435674358104484724658714222293454515494300571
..... 4265124445244247988774717731938471315140830307404075432336166965501976435
..... 1945813446570069156968090556800063025830089599260400096259430726498683087
..... 138415465107499
sage: p
16819938870120985392012908511330240702317396271716022919731854548482310101838672
4351964316301278642143567435810448472465871432229345451549430057142651244452442
47988777471773193847131514083030740407543233616696550197643519458134465700691569
68090556800063025830089599260400096259430726498683087138415465107499
sage: g=94389192776327398589453269803498145264338690934127823454309460592065688
..... 0400518160085582590614296727187254837587738949875812540433223444968461350
..... 789461385043775029963900638123183435135372621529733549843299536450513891
..... 25697538596236498663751353531799626707987717707116474306269548642698689883
..... 71113567502852
sage: g
9438919277632739858945326980349814526433869093412782345430946059206568804005181
6008558259061429672718725483758773894987581254043322344496846135078946138504377
502996390063812318343513537262152973354984329953645051389125697558596236498663
7513535317936267079877177071164743062695486426988898837113567502852
sage: sk=432398415306986194693973996870836079581453988813
sage: pk=power_mod(g,sk,p)
sage: pk
49336018324808093534733548840411752485726058527829630668967480568854756416567496216294919051910148686186622706869702321664465094703247368465068210152903024809904501302806169292269172462551470
323292301724297680683401258636182185599124131170077548450754294083728885075516985144944984920010138492897272069257160
sage: sk2=36489612142620205291538422172980979878978995745
sage: sk2=36489612142620205291538422172980979878978995745
sage: pk2=power_mod(g,sk2,p)
sage: pk2
974051128216400353287962874772888322743022284517722678347627207290401366793417427715624808734388198549012510106985774428094389196947337492500497411061496940277359886260926697925352046816652717
62659664614542674529824442558625688259705406912617194643155467760576705623618141065873127384281600876726625048900
sage: sk3=859678451869238574123625789510261485758749163528
sage: pk3=power_mod(g,sk3,p)
sage: pk3
1280169915740075275463861503071077827418510488416907250134743185395237661378058102112742531192652667940626573465203802979294467769646
80840753792271484649603844540374503251217107197349531253543136676535714181314196638094634788652360665349380808183893389246792041278361
50035302905793904974409743545853476389370
sage: pk3
12801699157400752754638615030710778274185104884169072501347431853952376613780581021127425311926526679406265734652038029792944677696468084075379227148464960384454037450325121710719734953125354
313667653571418131419663809463478865236066534938080818389338924679204127836150035302905793904974409743545853476389370
sage:

```

## II. GENERATING A DIGITAL SIGNATURE

The second step was to generate a digital signature. In Bitcoin, it is done so that the user can sign the transaction and verify it. Digital signatures are quite like actual signatures on a document. They help ensure that the author of a transaction is in fact the individual who holds the private key. Every transaction has a different digital signature that depends on the private key of the user. In our project, we have generated 2 digital signatures using the SHA3-224 hash function, for  $k = 8$ . The algorithm used for generating the key was taken from our Professor Guang Gong book [3]. We generated 2 digital signatures and verified them using the following procedure:

### DSS Signature Generation

- 1 Generate  $k, 0 < k < q$  by PRNG (per message), and  $r = (g^k \pmod{p}) \pmod{q}$ .
- 2 Solve for  $s$  in the equation:

$$h(m) \equiv xr + ks \pmod{q} \text{ (referred to as a signing equation)}$$

The pair  $(r, s)$  is a digital signature of the message  $m$ ,  $Sig_B(m) = (r, s)$ . (both  $r$  and  $s$  are  $\text{len}(q)$ -bit numbers).

### DSS Verification Process

- 1 Reject the signature if either  $0 < r < q$  or  $0 < s < q$  is not satisfied.
- 2 Set  $u = h(m)s^{-1} \pmod{q}$ , and  $v = \dagger rs^{-1} \pmod{q}$ .
- 3 Compute  $w = g^u y^v$ .
- 4 Check whether  $w = r$ . If it is true, accept it as a valid signature; otherwise, reject it.

The generation of the signatures was quick. The program provided with the report runs correctly and holds much more detailed information. While generating and verifying the signature, the formula required computations which were complicated in python. Therefore, sage was used to calculate the inverse mod and the resultant value was directly used in the python program.

The first screenshot shows the PyCharm IDE with the file 'generation of s1.py' open. The code defines a SHA3\_224 hash object, updates it with a large hexadecimal string, and calculates a signature 's' using modular arithmetic. The console output shows the value of 's' as '19221926859279175194455936770324680369106344154'. The second screenshot shows the file 'generation of s2.py' open. It follows a similar process but with different input data. The console output shows the value of 's' as '6093231810265762170128684904056886958938539705'. Both screenshots show the project structure on the left and the run console at the bottom.

Initially, the verification time however took quite a long time since the keys were very large. The computations of  $g*u$  was very time consuming. So to check our logic used in program, we ran the program taking values small.



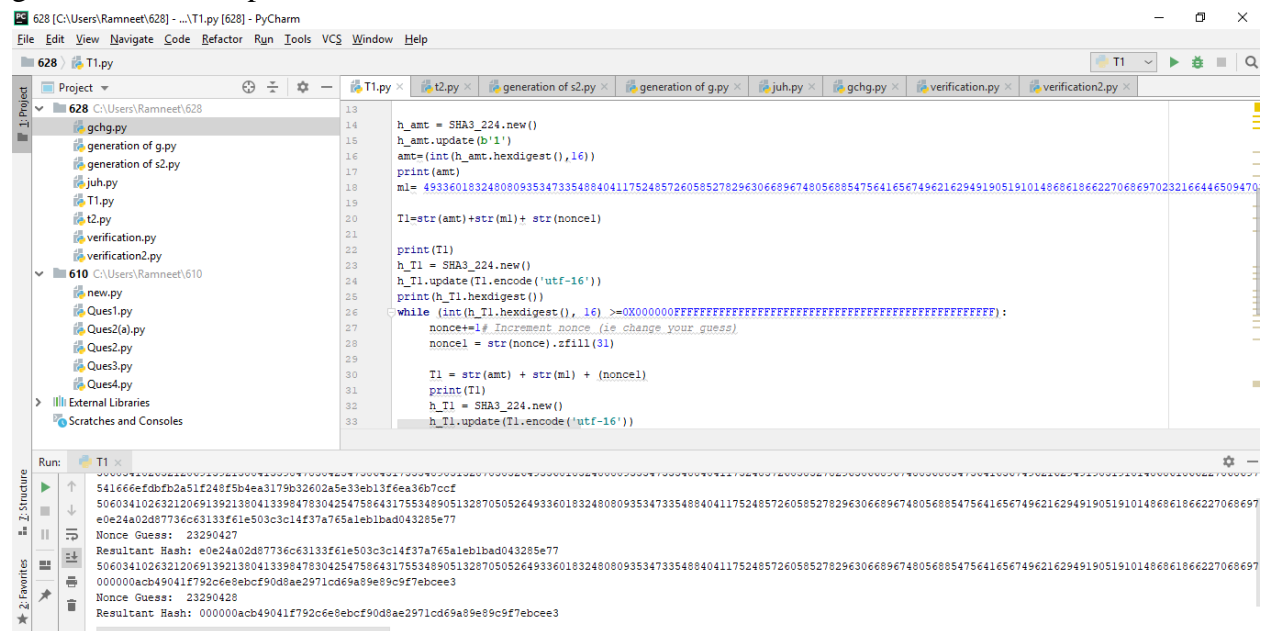
zeros in the beginning. This condition was tested repetitively. If the condition is not met than the value of nonce is increased by 1 till we get 6 zeros in the beginning.

Initially, when we tested the program for lower difficulty level i.e. 2 or 3 zeros in the beginning the results were generated very fast and easily. We got the resultant T at a much lower value of nonce than what we got for T1(6). We were able to generate T1 for 6 zero values in 2 to 3 hours.

Furthermore, the same procedure was followed for generating T2 where,

$$T2 = h(h(m1)||m2||nonce2)$$

Surprisingly, the resultant T2 was computed a lot quicker than T1. It took only an hour for us to get the desired output.



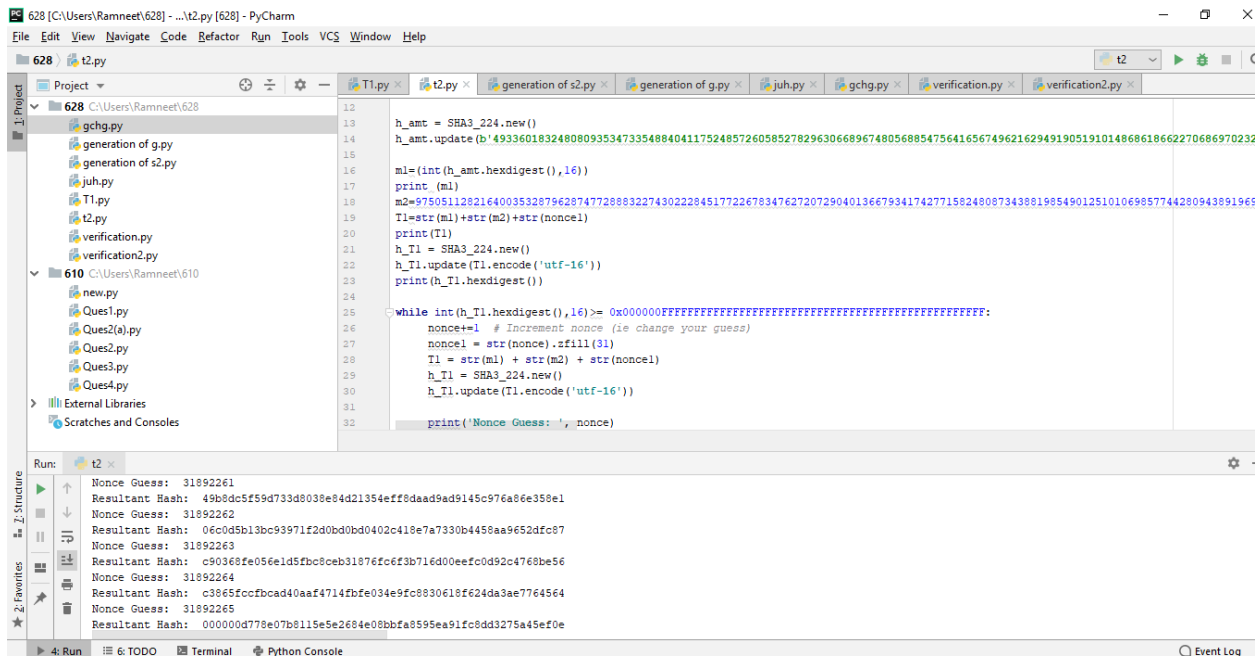
```
628 [C:\Users\Ramneet\628] - ...T1.py [628] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help

628 T1.py
Project 628 C:\Users\Ramneet\628
  gchg.py
  generation of g.py
  generation of s2.py
  juh.py
  T1.py
  T2.py
  verification.py
  verification2.py
610 C:\Users\Ramneet\610
  new.py
  Ques1.py
  Ques2(a).py
  Ques2.py
  Ques3.py
  Ques4.py
External Libraries
Scratches and Consoles

13 h_amt = SHA3_224.new()
14 h_amt.update(b'1')
15 amt=(int(h_amt.hexdigest(),16))
16 print(amt)
17 m1= 4933601832480809353473354884041175248572605852782963066896748056885475641656749621629491905191014868618662270686970232166446509470
18
19 T1=str(amt)+str(m1)+ str(nonce1)
20
21 print(T1)
22 h_T1 = SHA3_224.new()
23 h_T1.update(T1.encode('utf-16'))
24 print(h_T1.hexdigest())
25
26 while (int(h_T1.hexdigest(), 16) >= 0X000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF):
27     nonce+=1 # increment nonce (ie change your guess)
28     nonce1 = str(nonce).zfill(31)
29
30 T1 = str(amt) + str(m1) + (nonce1)
31 print(T1)
32 h_T1 = SHA3_224.new()
33 h_T1.update(T1.encode('utf-16'))

Run: T1 x
541666efdbf2a51f248f5b4ea3179b32602a5e33eb13f6ea36b7ccf
5060341026321206913921380413398478304254758643175534890513287050526493360183248080935347335488404117524857260585278296306689674805688547564165674962162949190519101486861866227068697
e0e24a02d87736c63133f61e503c3c14f37a765a1eb1bad043285e77
Nonce Guess: 23290427
Resultant Hash: e0e24a02d87736c63133f61e503c3c14f37a765a1eb1bad043285e77
5060341026321206913921380413398478304254758643175534890513287050526493360183248080935347335488404117524857260585278296306689674805688547564165674962162949190519101486861866227068697
000000acb49041f792c6e8ebcf90d8ae2971cd69a89e89c9f7ebceec3
Nonce Guess: 23290428
Resultant Hash: 000000acb49041f792c6e8ebcf90d8ae2971cd69a89e89c9f7ebceec3
```





## RESULTS:

All the Python programs are provided with the report. However, the results we generated are given below:

$S_k = x(\text{private key})$

$P_k = y(\text{public key})$

## KEYS PROVIDED

sk1 = 432398415306986194693973996870836079581453988813 (159bits)

pk1 =

493360183248080935347335488404117524857260585278296306689674805688547564165674  
962162949190519101486861866227068697023216644650947032473686465068210152903024  
809904501302806169292269172462551470632923017242976806834012586361821855991241  
31170077548450754294083728885075516985144944984920010138492897272069257160  
(1023 bits)

## KEYS GENERATED

sk2= 364896121426220205291538422172980979878978995745

Pk2=

975051128216400353287962874772888322743022284517722678347627207290401366793417  
427715824808734388198549012510106985774428094389196947337492500497411061496940  
277359886260926697925352046816652717626596646145426745298244425586256888259770  
5406912617194643155467760576705623618141065873127384281600876726625048900

Sk3=859678451869238574123625789510261485758749163528

pk3=

128016991574007527546386150307107782741851048842169072501347431853952376613780  
581021127425311926526679406265734652038029792944677696468084075379227148464960  
384454037450325121710719734953125354313667653571418131419663809463478865236066  
534938080818389338924679204127836150035302905793904974409743545853476389370

### **DSS Signatures**

**Sig=(r,s)**

Sig1= (34236423949667020016059495006613030875542929591,  
192219268592791751844559386770324683569106344154)

Sig2= (34236423949667020016059495006613030875542929591,  
609323181026576217012868490406568869588938539705)

### **FOR T1**

Nonce Guess: 23290428

Resultant Hash:

000000acb49041f792c6e8ebcf90d8ae2971cd69a89e89c9f7ebcee3

### **FOR T2**

Nonce Guess: 31892265

Resultant Hash:

000000d778e07b8115e5e2684e08bbfa8595ea91fc8dd3275a45ef0e

## **IV. DOUBLE SPENDING**

In the domain of distributed cryptographic currencies, the mitigation of double spending attacks is a core problem. Double spending is when the same bit coin is spent twice, or two transactions are posted for the same coin. To explain this issue more clearly, let us take an example of the physical currency. Once the physical currency is spent there is no way one can spend it again or duplicate unless someone steals it. Once you spent it, you spent it. There is no way one can use it again. However, that's not the case with the digital currency. Bitcoin is digital money, not physical cash. Hence, Bitcoin transactions have a possibility of being copied and rebroadcasted. As digital currency is not tangible and there is no physical proof or copy of it, it is possible to use it twice for 2 or more transactions. To manage the double spending problem, Bitcoin relies on a on Blockchain POW. There is no mechanism to prevent someone from broadcasting two transactions. A malicious miner can use a double-spend attack to pay for something very expensive and then erase that transaction from the blockchain but there are various ways double spending can be identified and prevented in Bitcoin.

## **1) PROOF OF WORK**

One of the best ways to solve the problem of double spending is the Proof of Work of Block chain. We know that blockchain is a pool of transactions where the blocks are added to the chain by the miners and all the miners would like to join the longest chain that they feel to be the true and honest chain. So, even if an attacker manages to release 2 transactions for the same coin, one will be verified by other Bitcoin miner, confirming the transaction and the other transaction becomes invalid automatically. As Blockchain is a distributed network, where there are thousands of miners wanting to add their block, there is very little chance that both the false posted transactions would be verified. Also, in Bitcoin there is a rule to wait till for at least 5 confirmations. It is common to not accept the data in the newest block but to wait until 5 or more blocks are appended to it to confirm the data in the older block. The transaction becomes more irreversible as the number of confirmations increases and even if an attacker tries to reverse the 6 transactions the entire network will be aware about it which can result in unnecessary attention for the attacker. With Bitcoin, all transactions are publicly announced to all nodes. The miners can then agree on a single history of the order in which they were received. Additionally, an attacker should never have a significant amount of the miner's share, the honest miners should be able to establish a longer chain in the network that does not include the attacker's data and the attacker's chain with the duplicate/false information will be left as a fork and completely irrelevant.

Another way, POW mitigates the double spending attack it by using the time stamps. Satoshi Nakamoto white paper [4] proposed the use of a timestamp server as a solution to the double-spending problem. In the Bitcoin all the transactions are recorded using time stamps. Every Bitcoin shows the chain of ownership. The server takes a hash of a block of transactions and then broadcasts this hash to all the nodes in the Bitcoin network. The timestamp proves that all the data in the hash couldn't have been created after the hash was published. As each timestamp includes the previous timestamp in its hash, this creates an unchangeable record of the order in which transactions took place which cannot be simply changed.

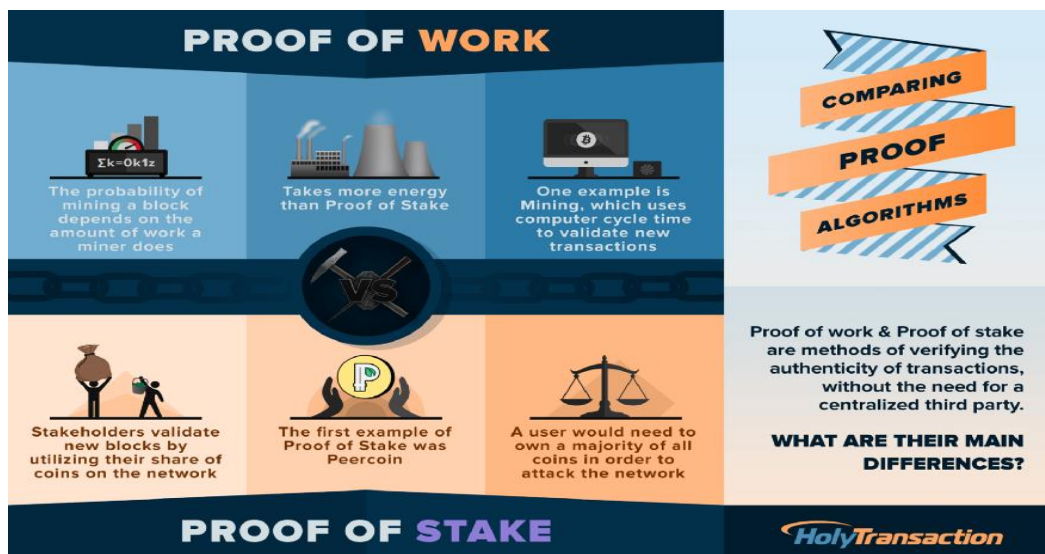
The only way an attacker can successfully execute a Double spending attack is through the 51% attack which is discussed in the last section.

## **2) PROOF OF STAKE**

It is an alternative to proof of work and attempts to provide consensus and prevention against double spending. This algorithm is being used by Cardano, Stellar, and eventually Ethereum which are other forms of cryptocurrencies. In proof of stake, the double spending attempts are identified by an auditor/leader. In POW, the trust is that most of the hash power is controlled by the honest auditors whereas in POS it is assumed that the majority of staked funds belong to honest auditors. The auditor with the highest Bitcoins or values (stakes) is chosen to be the leader, who will keep a record of all the transactions. Which means higher a person shows the stakes or wealth, higher will be the probability of him being chosen as a leader. The leader than

signs all messages with keys that it controls while updating the log book. In both, POW and POS, the auditors are required to provide collateral i.e. hash power and energy in POW whereas POS requires collateral in the form of stakes.

If the auditor attempts to pass 2 similar transactions, his mistake will be quickly identified by other auditors in the network who can publish a proof. Hence, double spending attack can be mitigated since auditors identify themselves by signing all changes made to the log. Therefore, the auditor can be easily identified consequently destroying his stakes.



Source [6]

### 3) PROOF OF BURN:

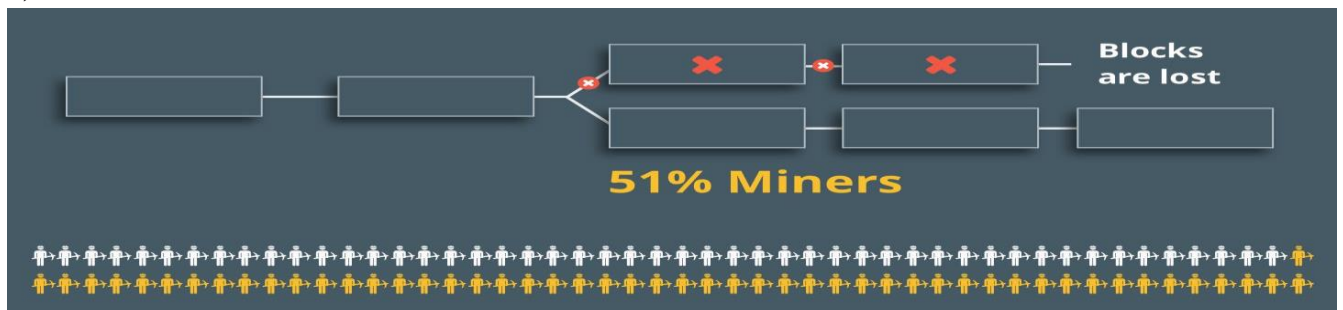
Proof of Burn is an alternative consensus algorithm. In the proof-of-burn mechanism users are required to “burn” or permanently make some of their mined POW cryptocurrency unavailable. In Bitcoin, if an attacker wishes to execute double spending, he/she needs to have lots of computing power. More the attacker invests in terms of money to buy the equipment to achieve the power to mine several blocks in a row, more will be the chances of a successful attack. POB has the same effect in more direct way.

The cost is mimicked by letting the nodes destroy or burn their coins in order to write their blocks. That is the principle behind POB, where no real is done. Burnt coins are mining rig.[5] That means, the act of burning coins can be compared to the act of buying a mining rig. In Proof of Burn, every time someone burn coins, they buy a virtual mining rig that gives them the power to mine blocks. The more coins one burns, the bigger the virtual mining rig they own. A mining rig is something more durable and it increases the chance to get blocks for a long time. The block rewards are high enough to allow the participants to make a financial profit from minting. In POB, the participants eventually get more coins than they burnt. One advantage that POB has over POW is that it consumes very less energy.

## V. ATTACKS ON PROOF OF WORK.

Any computer network can be attacked and blockchains running on distributed networks are no different. However, blockchains are vulnerable to somewhat different kinds of attacks. In Blockchain, attacker's aim would not be to steal someone's private information or compromise a password, but the main task would be to manipulate the POW consensus process in order to hack or alter any information on a blockchain. Integrity of data is established across a distributed network using the consensus algorithm. Many individuals participate in running the distributed network but only one individual gets a chance to add any one new piece of data to the blockchain at a time. Therefore, in order to change the consensus, the attacker must add more blocks to the network than the other participants. This can be done through the 51% attack.

### 1) 51% ATTACK:



source [7]

In the 51% attack, the attacker controls the 51% of the information added to the blockchain and can use this power to ensure that the consensus path only includes their chosen data. This means,

an individual or a group, in control of 51% or more of the total “mining power” in the system will win the right to add the next block to the blockchain more often than the rest of the network. The attacker in the 51% attack uses that position to then add blocks to the forks that he prefers, thus changing the consensus view of the blockchain. With this ability to add information much of the time and creating more than half of the blocks, any fork he chooses will eventually form the longest chain. The consensus is that the longest chain is the honest chain or is the chain that contains only the true data. Therefore, if the attacker manages to create the longest chain with his false data will be successful in convincing the people that it is indeed the honest chain and hence people will start adding their blocks into it.

The counter measure would be that if no individual or group controls more than 50% of the mining power, the network is safe from this type of attack. Also, the 51% attacks are actually very difficult to execute in practice. The Bitcoin network consumes as much electricity as small countries do. For a miner to have a chance at successfully attacking the network, they would need to purchase a huge amount of specialized mining equipment and consume around half the

electricity that it takes to power a small country. This is a very big risk to potential attackers and makes it very difficult for them to profit in a double-spend attack.

The proof of stake avoids this 'tragedy' by making it disadvantageous for a miner with a 51% stake in a cryptocurrency to attack the network. Although it would be difficult and expensive to accumulate 51% of a reputable digital coin, a miner with 51% stake in the coin would not have it in his best interest to attack a network which he holds a majority share. If the value of the cryptocurrency falls, this means that the value of his holdings would also fall, and so the majority stake owner would be more incentivized to maintain a secure network.

## **2) RACE ATTACKS**

The second possible attack on POW is the race attack. This attack is quite like double spending attack. A race attack is executed when an attacker creates two conflicting transactions. The first transaction is sent to a merchant, who accepts the payment and sends the product without waiting for confirmation of the transaction. At the same time, a conflicting transaction returning the same amount of cryptocurrency to the attacker is broadcasted to the network, eventually making the first transaction invalid resulting in loss to the merchant. Another way this attack can take place is when an attacker creates two transactions from the same send it to different shops/services. The shops without waiting for the confirmation accept the payment and sends their product can be in a loss. As in the case of Double spending attack, only one of the shops will receive the payment whose transaction will first appear in the blockchain and gets confirmed.

To avoid such attack and fraud it is always recommended to wait at least for 5 blocks being added to the chain after one's transaction. As all the blocks are interconnected, more the blocks added means more are the chances of the transaction being true and less are the chances for an attacker to reverse that transaction.

## CONCLUSION

This project gave us a deep understanding about Bit coin and Blockchain and how it works. We understood each aspect of blockchain very clearly. We implemented the DSS module to generate the digital signature and successfully computed the pre-images of the T1 and T2. However, talking about the time complexities, the time taken to compute pre images were a lot. It took us approximately 3 to 4 hours combined for T1 and T2. But we do realize that it may be because of the limited specifications of our systems.

Overall talking about the possible attacks, it is very unlikely in Bitcoin blockchain to successfully execute an attack without making equal investments in terms of money. Also, there is no guarantee that the profit made from the attack would be worthy of that investment. Moreover, every participant wants to make a long-term profit from Bitcoin. Therefore, the stakes are very high for someone to make an unsure attack and for other participants to not notice an error in a transaction and not report it as everyone wants to follow the true chain. Should a miner add a new block to an invalid chain, their block reward i.e. the whole reason they are participating in processing information onto the database in the first place will not be acknowledged as legitimate by the system. So, the self-interest of the blockchain's miners helps keep system working, despite the lack of any personal trust in the individuals operating the system.

## REFERENCES

- 1) <https://www.weforum.org/agenda/2018/06/law-too-slow-for-new-tech-how-keep-up/>
- 2) Introduction to Bitcoin, Cryptocurrencies, and their Consensus Mechanisms, Aljosha Judmayer, Nicholas Stifter, Katharina Krombhol, Edgar Weippl.
- 3) Communication system security, Lidong Chen, Guang Gong
- 4) <https://cointelegraph.com/news/bitcoin-whitepaper-10-years-since-satoshis-vision-was-brought-to-life>
- 5) <https://www.coinbureau.com/education/proof-of-burn-explained/>
- 6) <https://medium.com/@robertgreenfieldiv/explaining-proof-of-stake-f1eae6feb26f>
- 7) <https://cointelegraph.com/explained/proof-of-work-explained>
- 8) <https://komodoplatform.com/proof-of-work/>
- 9) [https://www.researchgate.net/publication/312022712\\_The\\_Balance\\_Attack\\_Against\\_Proof-Of-Work\\_Blockchains\\_The\\_R3\\_Testbed\\_as\\_an\\_Example](https://www.researchgate.net/publication/312022712_The_Balance_Attack_Against_Proof-Of-Work_Blockchains_The_R3_Testbed_as_an_Example)
- 10) <https://www.coindesk.com/a-solution-to-cryptos-51-attack-fine-miners-before-it-happens>