הכפר הירוק
HAKFAR HAYAROK

GreenCoin

A decentralized digital currency.

A Computer Science Final Project

By Michael Kuperfish Steinberg
I.D. 214288912
HaKfar HaYarok 2021
Advisor: Yooda Or

Student Contact Information:
Name: Michael Kuperfish Steinberg (מיכאל קופרפיש שטיינברג)
I.D.: 214288912
Date of Birth: 01/06/2003
Address: Oppenheimer 6, Tel Aviv
Tel: +972-58-676-2020
Email: m.kuper.steinberg@gmail.com

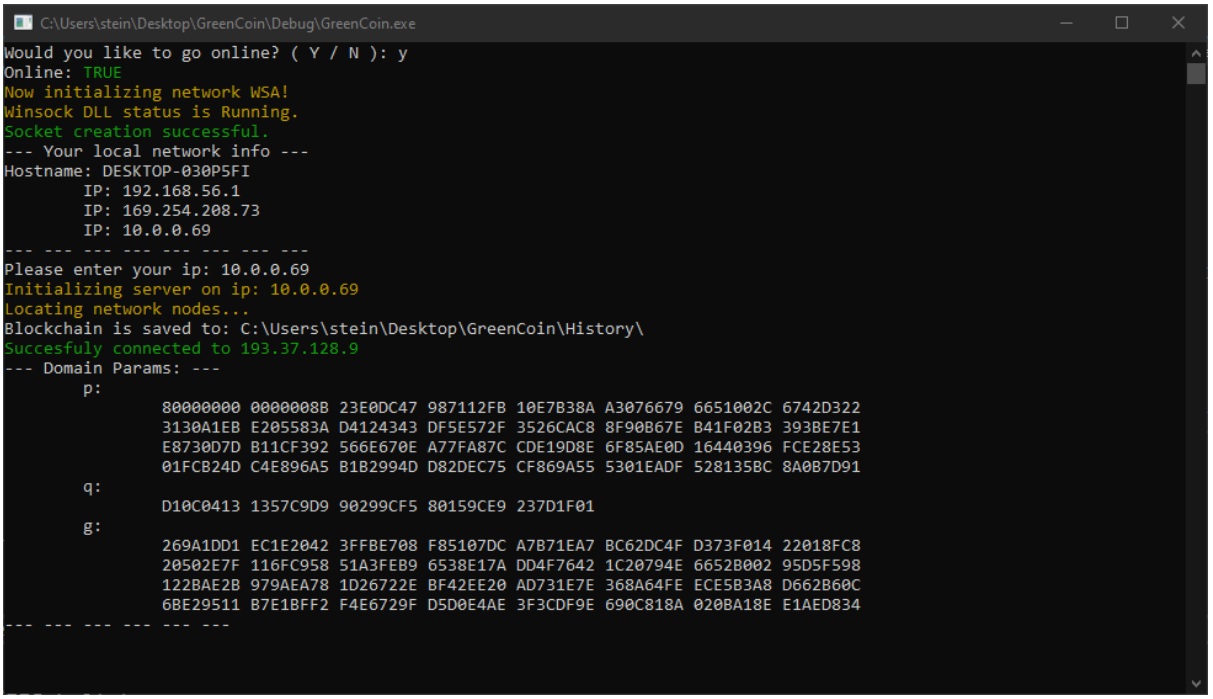
General Information:
School: הכפר הירוק ע"ש לוי אשכול
School Tel: 03-645-5666
Field: Computer Science
Study Units: 5 Units

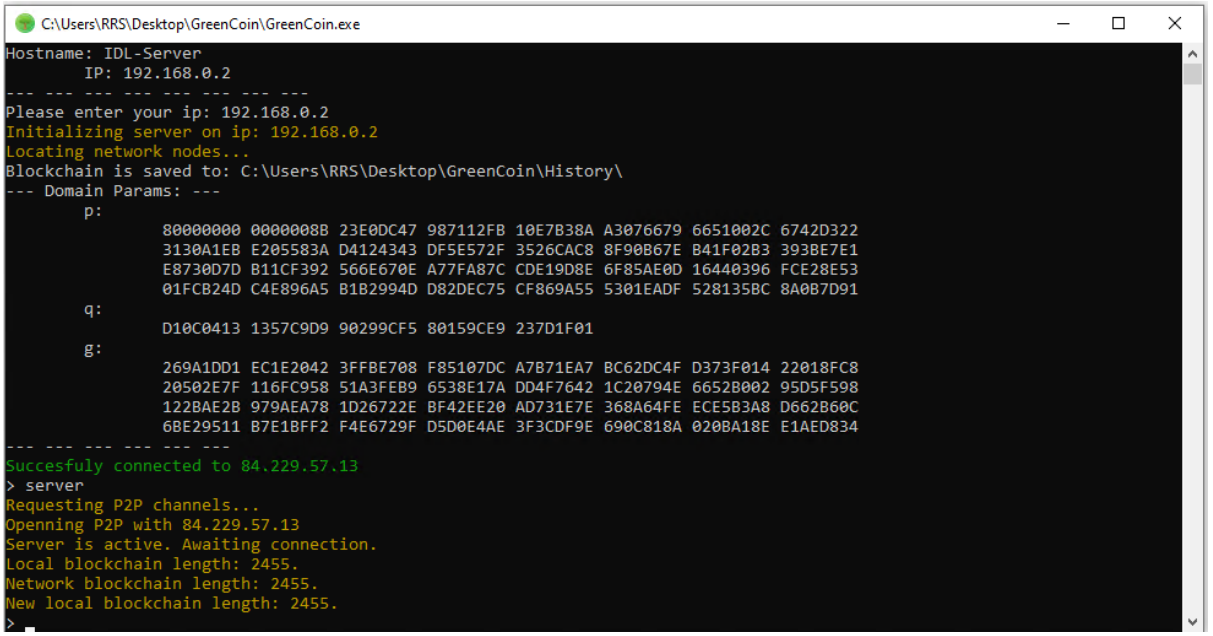
Advisor Contact Information:
Advisor: Yooda Or (יהודה אור)
I.D.: 023098007
Tel: +972-50-734-4457
Email: yoooda@gmail.com
Address: HaKerem 3, Tel Aviv
Academic Degree: MA Engineer, Technion Certified Engineer, Microsoft Certified, Academic transfer to Computer Science on behalf of the country since the year 2000.
Workplaces: Weizmann Institute of Technology, John Bryce College, HaKfar HaYarok, Youth Engineering College of Computer Science.

Table of contents:

All code can be found on GitHub: https://github.com/Michael-K-Stein/GreenCoin

Final Result

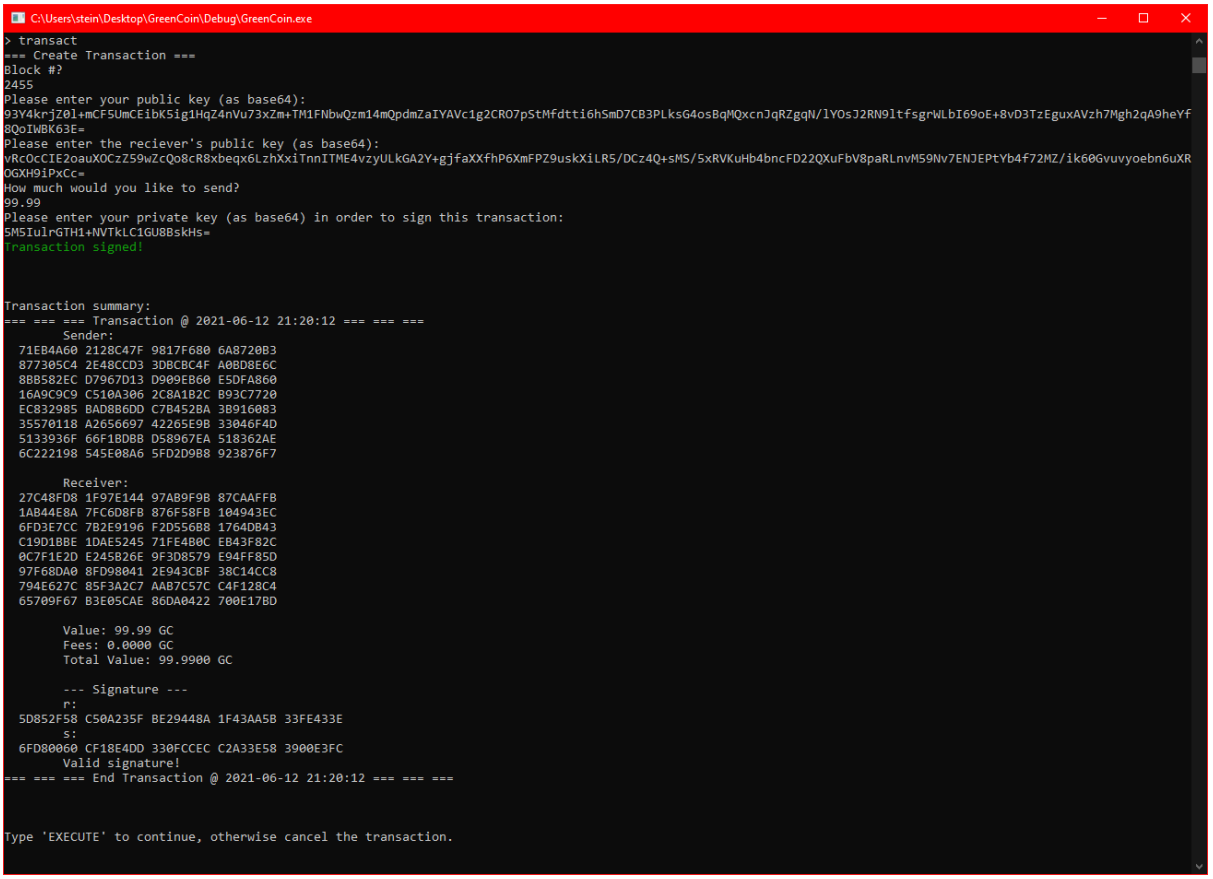


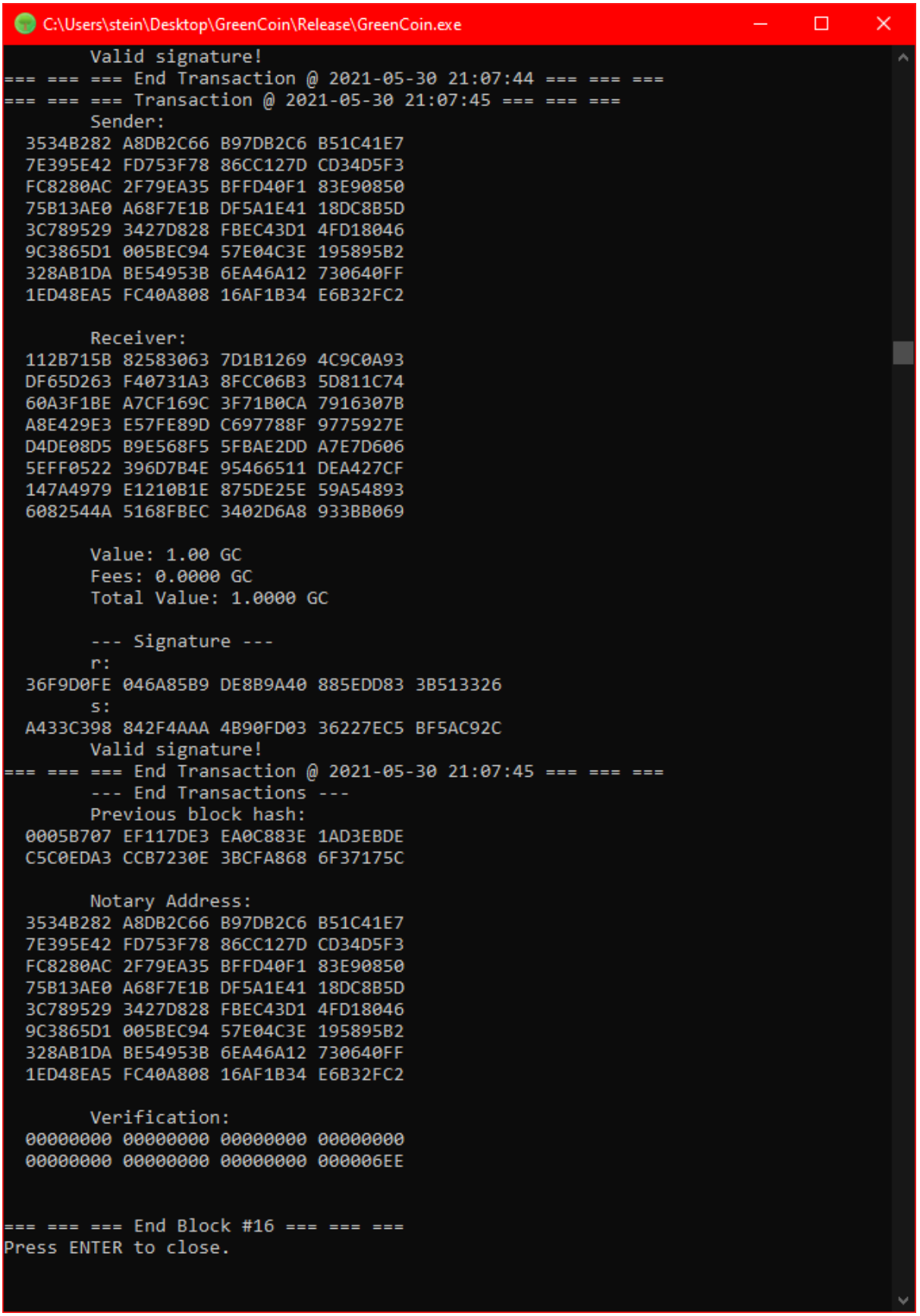


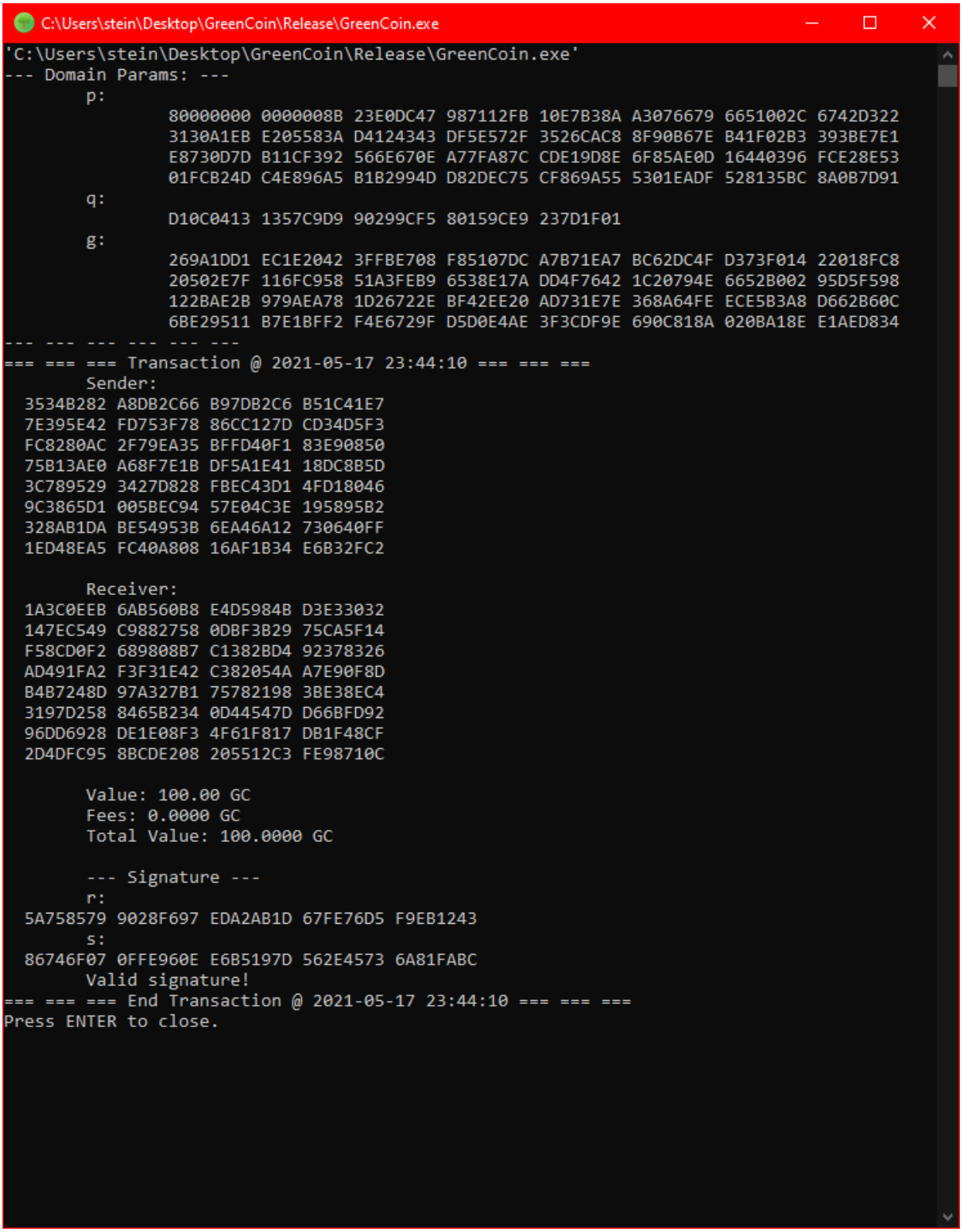
```
C:\Users\stein\Desktop\GreenCoin\Debug\GreenCoin.exe
wallet
Now in wallet command line.
Type 'exit' to return to general command line.
> help
Wallet help menu:
        help            | Displays help menu.
        generate        | Generates new wallet credentials.
        value           | Calculates the value of a wallet.
        values          | Prints all wallet values.
        exit            | Exits the wallet command-line.
> values
There are currently 736228.96084006398450583220 GreenCoins in circulation.
R3JlZW5Db2luIGJsb2NrIQpFdmVyeXRoaW5nIGJlZ2lucyBoZXJlISA6KQoAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=     : 0.00%          : Value: 0.00000000000000000000
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=     : 0.00%          : Value: 0.00000000000000000000
abA7k6jWAjTs+2hRSlSCYJNIpVle4l2HHgsh4XlJehTPJ6TeEWVGlU57bTkiBf9eBtbnp93iul/1aOW51Qje1H6SdZePeJfGneh/5eMp5Kh7MBZ5yrBxP5wW
z6e+8aNgdByBXbMGzI+jMQf0Y9Jl35MKnExpEht9YzBYgltxKxE=     : 21.54%         : Value: 158603.99676000227918848395
wi+z5jQbrxYIqED8pY7UHv9ABnMSaqRuO5VUvtqxijKylVgZPkzgV5TsWwDRZTicRoDRT9FD7Pso2Cc0KZV4PF2L3BhBHlrfG36PpuA6sXVQCOmD8UD9vzXq
eS+sgIL889U0zX0SzIZ4P3X9Ql45fudBHLXGsn25ZizbqIKyNDU=     : 78.46%         : Value: 577624.96408002823591232300
>
```

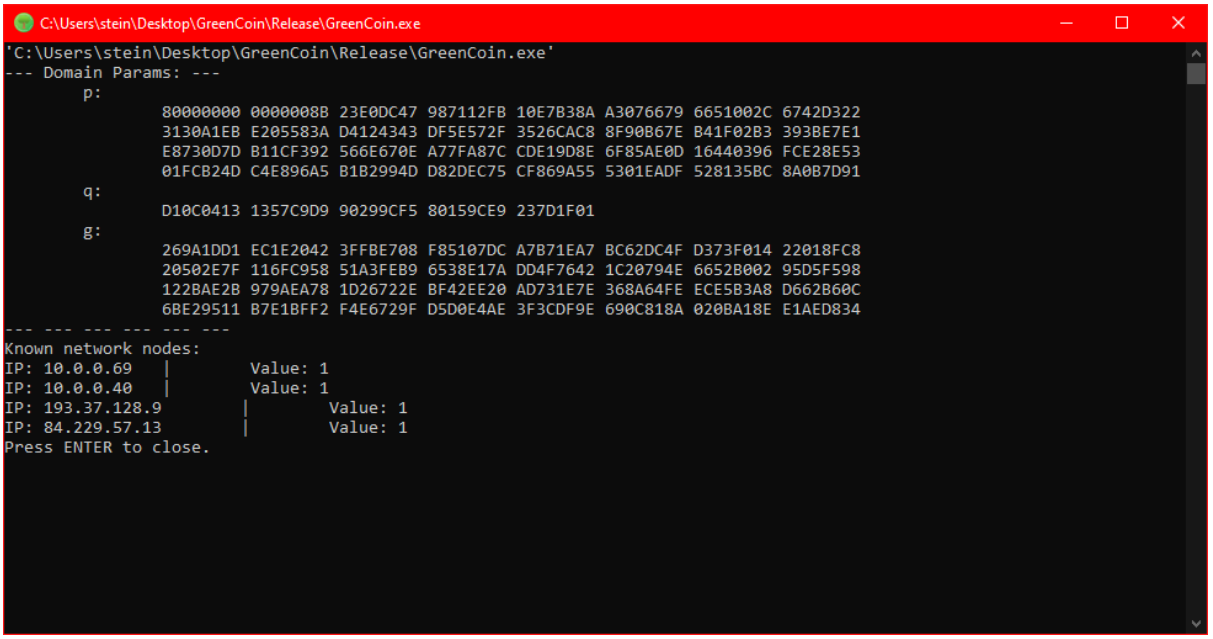
```
C:\Users\stein\Desktop\GreenCoin\Debug\GreenCoin.exe
> generate
  x:
    CC5A66DE 4B62596E B9D5A403 BD7EDE5D ED2FB9F7
X B64: '97kv7V3efr0DpNW5blliS95mWsw='
  y:
    27C48FD8 1F97E144 97AB9F9B 87CAAFFB 1AB44E8A 7FC6D8FB 876F58FB 104943EC
    6FD3E7CC 7B2E9196 F2D556B8 1764DB43 C19D1BBE 1DAE5245 71FE4B0C EB43F82C
    0C7F1E2D E245B26E 9F3D8579 E94FF85D 97F68DA0 8FD98041 2E943CBF 38C14CC8
    794E627C 85F3A2C7 AAB7C57C C4F128C4 65709F67 B3E05CAE 86DA0422 700E17BD
Y B64: 'vRcOcCIE2oauXOCzZ59wZcQo8cR8xbeqx6LzhXxiTnnITME4vzyULkGA2Y+gjfaXXfhP6XmFPZ9uskXiLR5/DCz4Q+sMS/5xRVKuHb4bncFD22QX
uFbV8paRLnvM59Nv7ENJEPtYb4f72MZ/ik60Gvuvyoebn6uXROGXH9iPxCc='
> generate
  x:
    7B906CC0 53462D2C E454357E 7D4CC65A BA48CEE4
X B64: '5M5IulrGTH1+NVTkLC1GU8BskHs='
  y:
    71EB4A60 2128C47F 9817F680 6A8720B3 877305C4 2E48CCD3 3DBCBC4F A0BD8E6C
    8BB582EC D7967D13 D909EB60 E5DFA860 16A9C9C9 C510A306 2C8A1B2C B93C7720
    EC832985 BAD8B6DD C7B452BA 3B916083 35570118 A2656697 42265E9B 33046F4D
    5133936F 66F1BDBB D58967EA 518362AE 6C222198 545E08A6 5FD2D9B8 923876F7
Y B64: '93Y4krjZ0l+mCF5UmCEibK5ig1HqZ4nVu73xZm+TM1FNbwQzm14mQpdmZaIYAVc1g2CRO7pStMfdtti6hSmD7CB3PLksG4osBqMQxcnJqRZgqN/l
YOsJ2RN9ltfsgrWLbI69oE+8vD3TzEguxAVzh7Mgh2qA9heYf8QoIWBK63E='
> generate
  x:
    6DC3F5E4 AD2CF28C 98C77A1E 195DC4BA 11192A0E
X B64: 'DioZEbrEXRkeeseYjPIsreT1w20='
  y:
    12870195 3967E854 5330D8A1 B147A770 CF9A3982 9D827206 FB7913C6 22D165F0
    816C8E95 469F02BF 83336390 8E593C90 368C088E F38E6D3E B9847224 AFEB72B8
    9D070D18 5E1AD11D B45D987D C7B4823C 9F80FDD2 13DAFB4E 139C172C F6FB14B1
    DBA636E8 09B0F7E0 819EFB50 340F5CF9 53013B71 4BFE48B4 96208AEB 7FAC41E4
Y B64: '5EGsf+uKIJa0SP5LcTsBU/lcDzRQ+56B4PewCeg2ptuxFPv2LBecE0772hPS/YCfPIK0x32YXbQd0RpeGA0Hnbhy668kcoS5Pm2O844IjDaQPFmO
kGMzg78Cn0aVjmyB8GXRIsYTefsGcoKdgjmaz3CnR7Gh2DBTVOhnOZUBhxI='
>
```

```
C:\Users\stein\Desktop\GreenCoin\Debug\GreenCoin.exe
> help
General help menu:
        help            | Displays help menu.
        transact        | Opens transaction command-line where transactions are made and executed in order to transfer funds between wallets.
        server          | Starts local server which accepts P2P connections from other nodes. Starting a server is a prerequisite to block mining.
        gcbh            | GreenCoin Block History. Requests blockchain history from all known nodes. Use this to keep up to date with the blockchain.
        gcbl            | GreenCoin BlockChain Length. Requests the blockchain length from all known nodes.
        wallet          | Opens wallet command-line (avaliable offline). Use this to monitor wallet funds and create wallets.
        notary          | Set the local notary signing address. This address will be used when mining blocks to award you your fee.
>
```

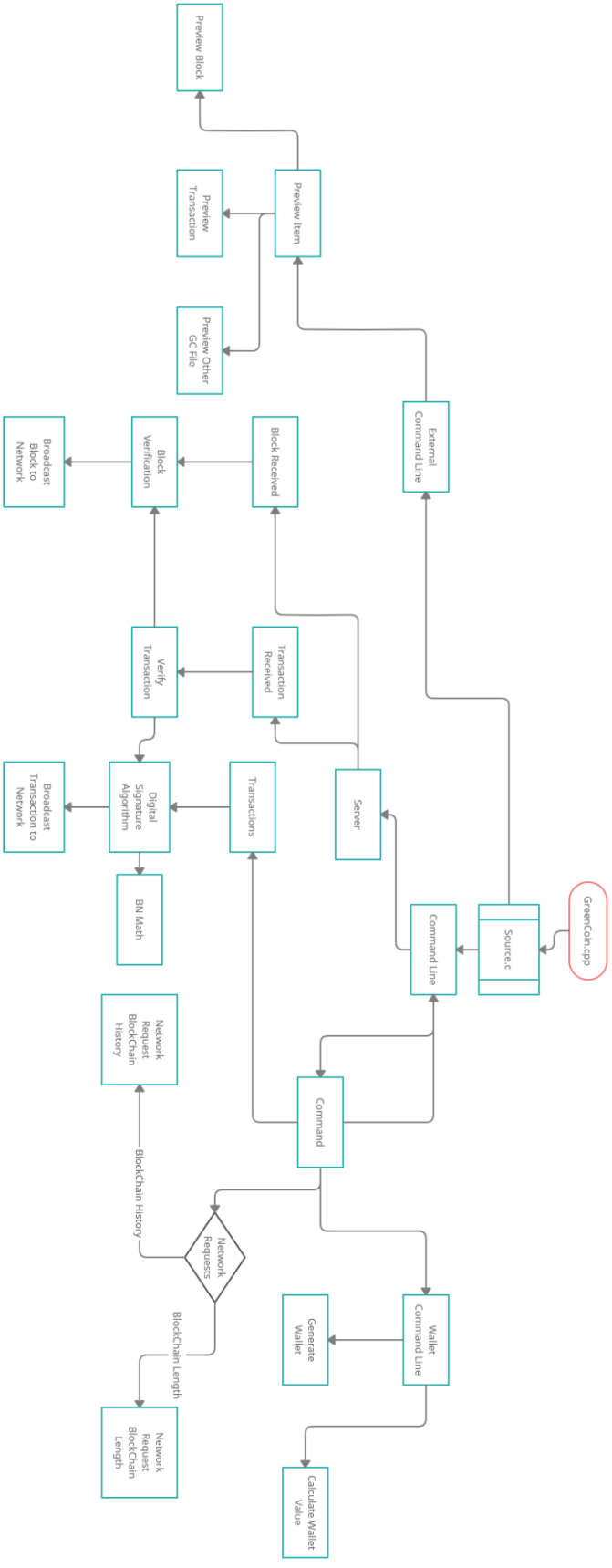

C:\Users\stein\Desktop\GreenCoin\Debug\GreenCoin.exe
> transact
=== Create Transaction ===
Block #?
2455
Please enter your public key (as base64):
93Y4krjZ0l+mCF5UmCEibK5ig1HqZ4nVu73xZm+TM1FNbwQzm14mQpdmZaIYAVc1g2CRO7pStMfdtti6hSmD7CB3PLksG4osBqMQxcnJqRZgqN/lYOsJ2RN9ltfsgrWLbI69oE+8vD3TzEguxAVzh7Mgh2qA9heYf
8QoIWBK63E=
Please enter the reciever's public key (as base64):
vRcOcCIE2oauXOCzZ59wZcQo8cR8xbeqx6LzhXxiTnnITME4vzyULkGA2Y+gjfaXXfhP6XmFPZ9uskXiLR5/DCz4Q+sMS/5xRVKuHb4bncFD22QXuFbV8paRLnvM59Nv7ENJEPtYb4f72MZ/ik60Gvuvyoebn6uXR
OGXH9iPxCc=
How much would you like to send?
99.99
Please enter your private key (as base64) in order to sign this transaction:
5M5IulrGTH1+NVTkLC1GU8BskHs=
Transaction signed!
Transaction summary:
=== === === Transaction @ 2021-06-12 21:20:12 === === ===
Sender:
71EB4A60 2128C47F 9817F680 6A8720B3
877305C4 2E48CCD3 3DBCBC4F A0BD8E6C
8BB582EC D7967D13 D909EB60 E5DFA860
16A9C9C9 C510A306 2C8A1B2C B93C7720
EC832985 BAD8B6DD C7B452BA 3B916083
35570118 A2656697 42265E9B 33046F4D
5133936F 66F18DBB D58967EA 518362AE
6C222198 545E08A6 5FD2D9B8 923876F7
Receiver:
27C48FD8 1F97E144 97AB9F9B 87CAAFFB
1AB44E8A 7FC6D8FB 876F58FB 104943EC
6FD3E7CC 7B2E9196 F2D556B8 1764DB43
C19D1BBE 1DAE5245 71FE4B0C EB43F82C
0C7F1E2D E245B26E 9F3D8579 E94FF85D
97F68DA0 8FD98041 2E943CBF 38C14CC8
794E627C 85F3A2C7 AAB7C57C C4F128C4
65709F67 B3E05CAE 86DA0422 700E17BD
Value: 99.99 GC
Fees: 0.0000 GC
Total Value: 99.9900 GC
--- Signature ---
r:
5D852F58 C50A235F BE29448A 1F43AA5B 33FE433E
s:
6FD80060 CF18E4DD 330FCCEC C2A33E58 3900E3FC
Valid signature!
=== === === End Transaction @ 2021-06-12 21:20:12 === === ===
Type 'EXECUTE' to continue, otherwise cancel the transaction.


C:\Users\stein\Desktop\GreenCoin\Release\GreenCoin.exe
Valid signature!
=== === === End Transaction @ 2021-05-30 21:07:44 === === ===
=== === === Transaction @ 2021-05-30 21:07:45 === === ===
Sender:
3534B282 A8DB2C66 B97DB2C6 B51C41E7
7E395E42 FD753F78 86CC127D CD34D5F3
FC8280AC 2F79EA35 BFFD40F1 83E90850
75B13AE0 A68F7E1B DF5A1E41 18DC8B5D
3C789529 3427D828 FBEC43D1 4FD18046
9C3865D1 005BEC94 57E04C3E 195895B2
328AB1DA BE54953B 6EA46A12 730640FF
1ED48EA5 FC40A808 16AF1B34 E6B32FC2
Receiver:
112B715B 82583063 7D1B1269 4C9C0A93
DF65D263 F40731A3 8FCC06B3 5D811C74
60A3F1BE A7CF169C 3F71B0CA 7916307B
A8E429E3 E57FE89D C697788F 9775927E
D4DE08D5 B9E568F5 5FBAE2DD A7E7D606
5EFF0522 396D7B4E 95466511 DEA427CF
147A4979 E1210B1E 875DE25E 59A54893
6082544A 5168FBEC 3402D6A8 933BB069
Value: 1.00 GC
Fees: 0.0000 GC
Total Value: 1.0000 GC
--- Signature ---
r:
36F9D0FE 046A85B9 DE8B9A40 885EDD83 3B513326
s:
A433C398 842F4AAA 4B90FD03 36227EC5 BF5AC92C
Valid signature!
=== === === End Transaction @ 2021-05-30 21:07:45 === === ===
--- End Transactions ---
Previous block hash:
0005B707 EF117DE3 EA0C883E 1AD3EBDE
C5C0EDA3 CCB7230E 3BCFA868 6F37175C
Notary Address:
3534B282 A8DB2C66 B97DB2C6 B51C41E7
7E395E42 FD753F78 86CC127D CD34D5F3
FC8280AC 2F79EA35 BFFD40F1 83E90850
75B13AE0 A68F7E1B DF5A1E41 18DC8B5D
3C789529 3427D828 FBEC43D1 4FD18046
9C3865D1 005BEC94 57E04C3E 195895B2
328AB1DA BE54953B 6EA46A12 730640FF
1ED48EA5 FC40A808 16AF1B34 E6B32FC2
Verification:
00000000 00000000 00000000 00000000
00000000 00000000 00000000 000006EE
=== === === End Block #16 === === ===
Press ENTER to close.


C:\Users\stein\Desktop\GreenCoin\Release\GreenCoin.exe
'C:\Users\stein\Desktop\GreenCoin\Release\GreenCoin.exe'
--- Domain Params: ---
p:
80000000 0000008B 23E0DC47 987112FB 10E7B38A A3076679 6651002C 6742D322
3130A1EB E205583A D4124343 DF5E572F 3526CAC8 8F90B67E B41F02B3 393BE7E1
E8730D7D B11CF392 566E670E A77FA87C CDE19D8E 6F85AE0D 16440396 FCE28E53
01FCB24D C4E896A5 B1B2994D D82DEC75 CF869A55 5301EADF 528135BC 8A0B7D91
q:
D10C0413 1357C9D9 90299CF5 80159CE9 237D1F01
g:
269A1DD1 EC1E2042 3FFBE708 F85107DC A7B71EA7 BC62DC4F D373F014 22018FC8
20502E7F 116FC958 51A3FEB9 6538E17A DD4F7642 1C20794E 6652B002 95D5F598
122BAE2B 979AEA78 1D26722E BF42EE20 AD731E7E 368A64FE ECE5B3A8 D662B60C
6BE29511 B7E1BFF2 F4E6729F D5D0E4AE 3F3CDF9E 690C818A 020BA18E E1AED834
--- --- --- --- --- ---
=== === === Transaction @ 2021-05-17 23:44:10 === === ===
Sender:
3534B282 A8DB2C66 B97DB2C6 B51C41E7
7E395E42 FD753F78 86CC127D CD34D5F3
FC8280AC 2F79EA35 BFFD40F1 83E90850
75B13AE0 A68F7E1B DF5A1E41 18DC8B5D
3C789529 3427D828 FBEC43D1 4FD18046
9C3865D1 005BEC94 57E04C3E 195895B2
328AB1DA BE54953B 6EA46A12 730640FF
1ED48EA5 FC40A808 16AF1B34 E6B32FC2
Receiver:
1A3C0EEB 6AB560B8 E4D5984B D3E33032
147EC549 C9882758 0DBF3B29 75CA5F14
F58CD0F2 689808B7 C1382BD4 92378326
AD491FA2 F3F31E42 C382054A A7E90F8D
B4B7248D 97A327B1 75782198 3BE38EC4
3197D258 8465B234 0D44547D D66BFD92
96DD6928 DE1E08F3 4F61F817 DB1F48CF
2D4DFC95 8BCDE208 205512C3 FE98710C
Value: 100.00 GC
Fees: 0.0000 GC
Total Value: 100.0000 GC
--- Signature ---
r:
5A758579 9028F697 EDA2AB1D 67FE76D5 F9EB1243
s:
86746F07 0FFE960E E6B5197D 562E4573 6A81FABC
Valid signature!
=== === === End Transaction @ 2021-05-17 23:44:10 === === ===
Press ENTER to close.


C:\Users\stein\Desktop\GreenCoin\Release\GreenCoin.exe
'C:\Users\stein\Desktop\GreenCoin\Release\GreenCoin.exe'
--- Domain Params: ---
p:
80000000 0000008B 23E0DC47 987112FB 10E7B38A A3076679 6651002C 6742D322
3130A1EB E205583A D4124343 DF5E572F 3526CAC8 8F90B67E B41F02B3 393BE7E1
E8730D7D B11CF392 566E670E A77FA87C CDE19D8E 6F85AE0D 16440396 FCE28E53
01FCB24D C4E896A5 B1B2994D D82DEC75 CF869A55 5301EADF 528135BC 8A0B7D91
q:
D10C0413 1357C9D9 90299CF5 80159CE9 237D1F01
g:
269A1DD1 EC1E2042 3FFBE708 F85107DC A7B71EA7 BC62DC4F D373F014 22018FC8
20502E7F 116FC958 51A3FEB9 6538E17A DD4F7642 1C20794E 6652B002 95D5F598
122BAE2B 979AEA78 1D26722E BF42EE20 AD731E7E 368A64FE ECE5B3A8 D662B60C
6BE29511 B7E1BFF2 F4E6729F D5D0E4AE 3F3CDF9E 690C818A 020BA18E E1AED834
--- --- --- --- --- ---
Known network nodes:
IP: 10.0.0.69 | Value: 1
IP: 10.0.0.40 | Value: 1
IP: 193.37.128.9 | Value: 1
IP: 84.229.57.13 | Value: 1
Press ENTER to close.

UML Diagram


Preview Block
Preview Item
Preview Transaction
Preview Other GC File
External Command Line
Block Received
Block Verification
Broadcast Block to Network
Transaction Received
Verify Transaction
Server
Transactions
Digital Signature Algorithm
Broadcast Transaction to Network
BN Math
GreenCoin.cpp
Source.c
Command Line
Network Request BlockChain History
Command
BlockChain History
Network Requests
BlockChain Length
Generate Wallet
Wallet Command Line
Network Request BlockChain Length
Calculate Wallet Value

Introduction

Digital Currencies are a complete parallel to fiat currencies (government issued money). Each digital currency creates its own economy which hosts transactions using that currency, similarly to how John could use his salary, which was paid in United States Dollars (USD), and dine at a local restaurant, paying for the meal in USD, John could have been paid in GreenCoins and have done the exact same (assuming the restaurant accepts GreenCoins as legal tender).

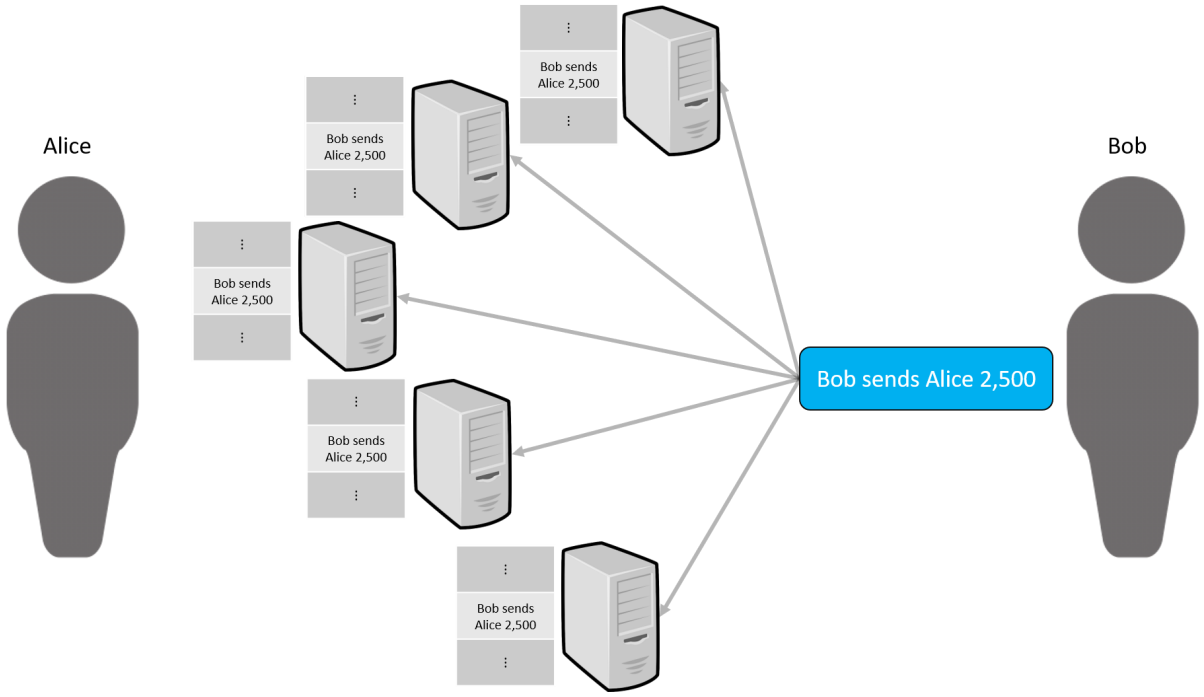
However, unlike traditional currency and banking systems, digital currencies are generally minted and controlled very differently. In traditional economies, a globally trusted financial institution such as a bank would record and store transactions between accounts, and thus conclude how much money each account has and can spend.

Assume Bob sends Alice $2,500. Bob would send the transaction to the bank who would verify that Bob has the balance / credit to pay the transaction. Then the bank would deduce the 2,500 from Bob's account and credit Alice's account the same amount. This transaction would then be recorded on a ledger which the bank keeps. When either Alice or Bob go to the bank to withdraw their balance the bank can use the ledger to calculate how much money they each have.

⋮	⋮
Bob	-2,500
Alice	2,500
⋮	⋮

The same transaction using a decentralized digital currency such as GreenCoin would not go through a central bank. Instead, Bob would broadcast that he wishes to send Alice 2,500GC. Mining nodes on the network would look at the ledger history to verify that Bob has enough coins and that the transaction is not fraudulent, and if they deem the transaction

valid, add it to the ledger history.



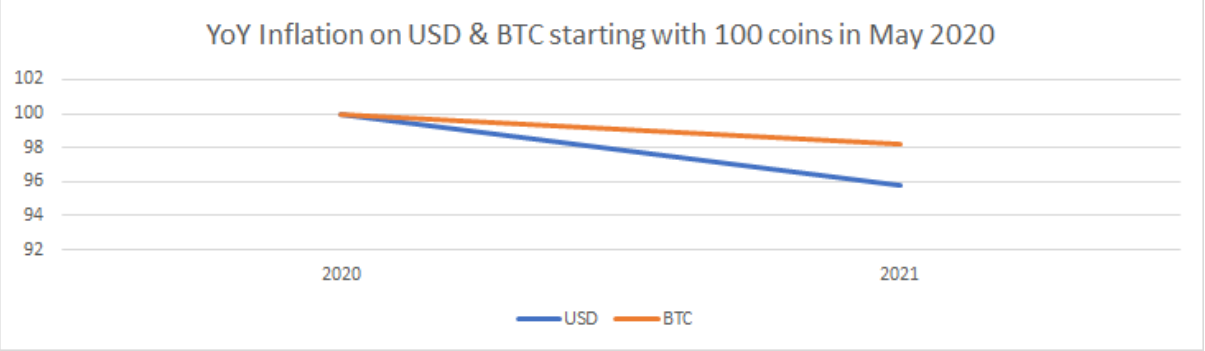
Bob and Alice are now not dependent on the bank, since assuming there are many nodes connected to the network, plenty of servers can confirm that the transaction was made, and Alice can now use the 2,500GC freely. One of the main reasons Bitcoin, the first digital currency, was made was due to the 2008 financial crisis when many banks and other financial institutions had to be bailed out by governments, on the taxpayers dime. When a bank becomes insolvent, meaning it has more liabilities than assets (net negative), it will not pay back its depositors (at least not in full).

Assume it is 2008 and Alice's bank, which processed Bob's transaction (see above), declares bankruptcy. Alice will rush to the bank to try to withdraw her money, but will unfortunately only receive pennies on the dollar. Alice had absolutely no control over the bank and the financial sector's reckless actions which led to the crisis, but she will nevertheless lose the $2,500 which she had in the bank.

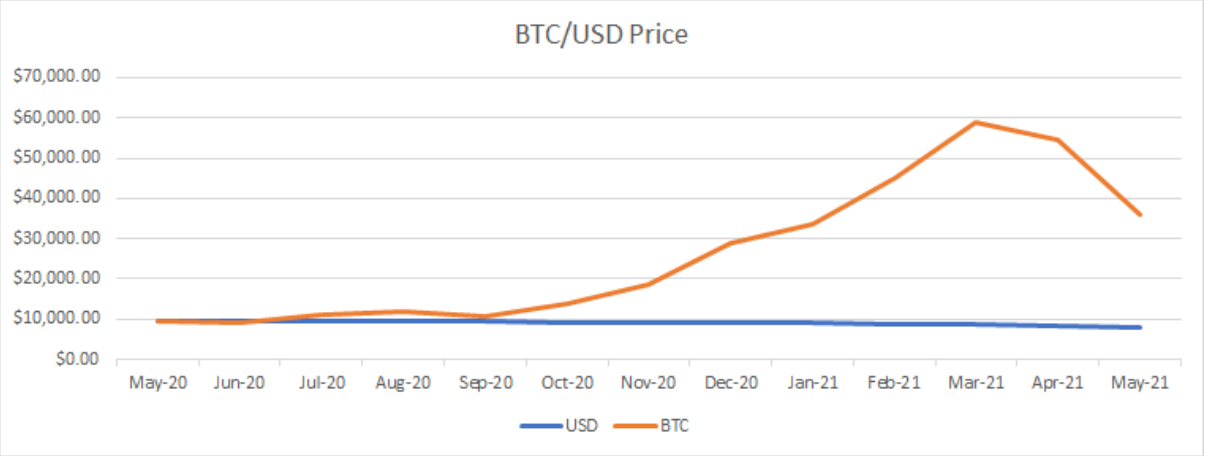
On the contrary, had Alice and Bob done the transaction through a digital currency, no single entity could make the system "crash". No matter how deep in debt a single validator (miner) is, as long as over 50% of validators hold a correct ledger history, no money will ever be lost. Moreover, in most digital currencies, an account cannot hold a negative value, and therefore, there is never any debt.

Furthermore, while the USD is solely controlled by the United States Government, no-one controls a decentralized digital currency. Therefore, hyperinflation is not a hazard. There is no government to wake up and decide to pass a trillion dollar military contracts bill, which will increase inflation. Digital currencies, either have no inflation, meaning all coins are minted at genesis and then distributed, or have a completely predictable inflation rate, which is generally capped (such as Bitcoin's 21,000,000 coin capacity). In situations where there is little trust in the government, digital currencies can be a clear solution. For a real world example, this is a graph assuming you held 100USD and 100BTC, how much would

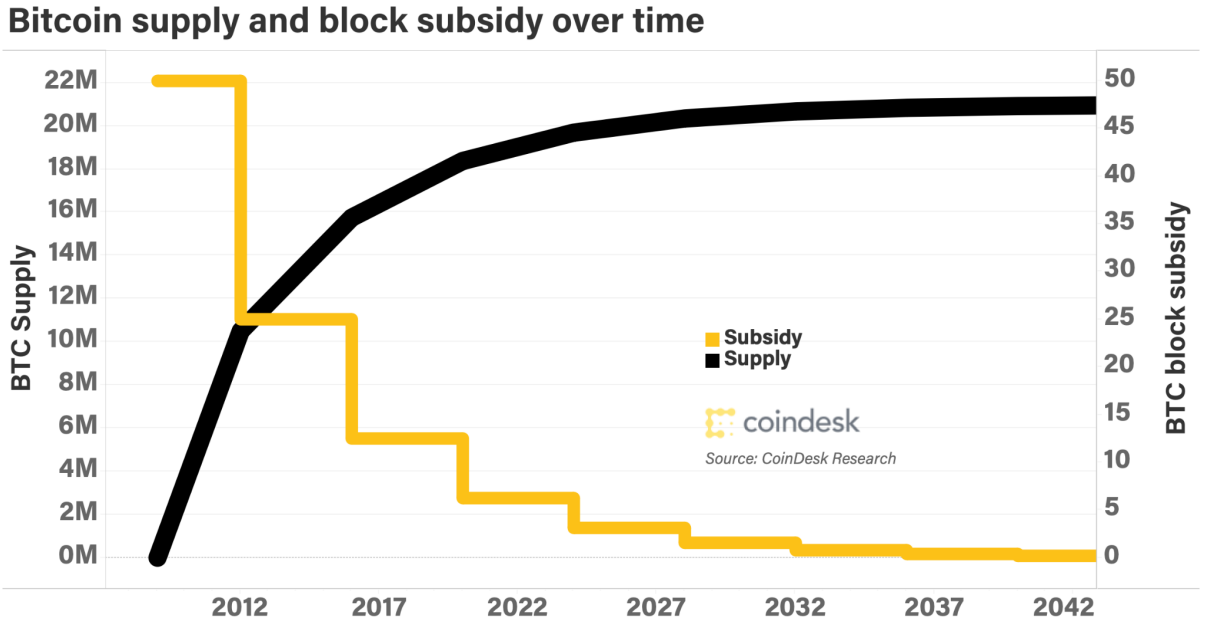
you have after inflation (from May 2020 until May 2021):



It is clear that holding BTC would be advantageous in this time frame, as less value would be lost over the year. Specifically with Bitcoin, assuming you had 9,688USD, enough for 1BTC, this would be the dollar value equivalent over the year from May 2020:



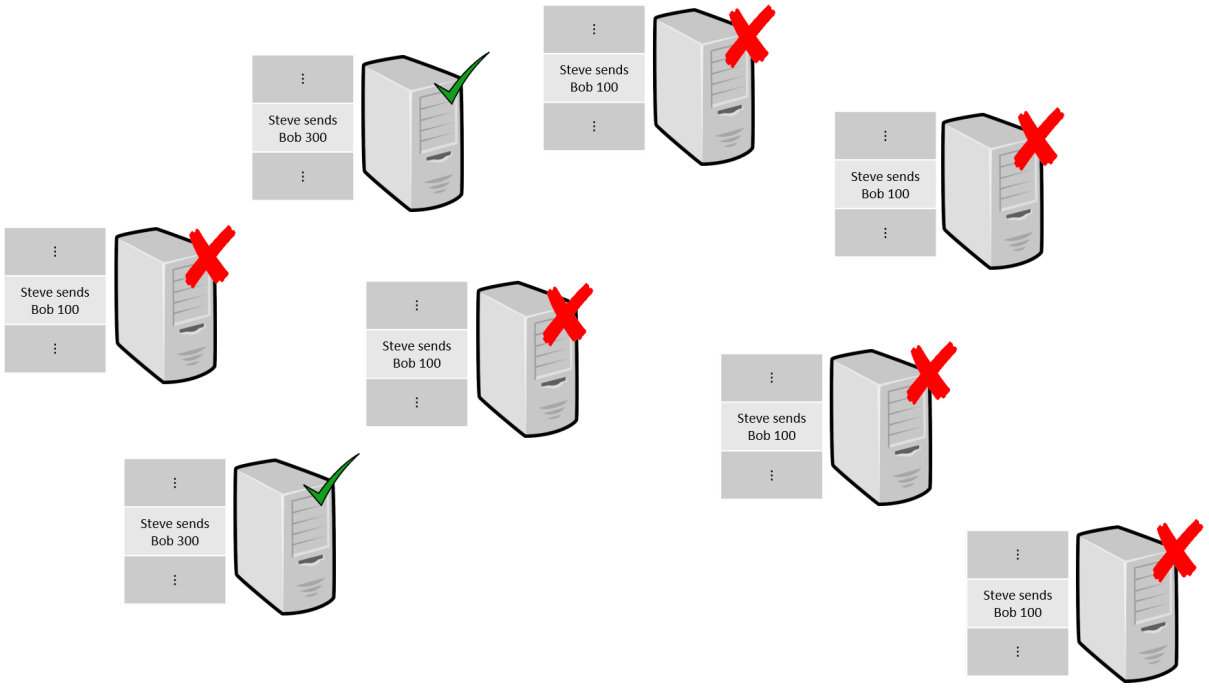
From this, it is clear that while the USD depreciates in value, BTC increases significantly in value. This is partially due to the fact that over time more and more USD is minted and printed by the government, while BTC follows a steady rule which creates a geometric limit of the total supply:



The concept of not having a central governing authority seems very appealing. However, there is still one critical issue: How do you host an economy without any central trusted authority? How can Alice be sure that Bob will pay her? How can society be sure that Bob did not send himself fraudulent transactions from fake accounts crediting himself millions? How does Alice protect her account funds from being stolen? Who decides what is "real"?

Bob can claim that Steve sent him 300GC, and show the transaction on his copy of the ledger. If there is no central authority which everyone trusts to say "We never saw this", how can society know if Steve sent Bob 300GC? Assume Steve also has a copy of a ledger, and his does not have the transaction. Now there is a dispute, Bob claims Steve paid him 300GC, while Steve claims this never happened. What is the "truth"? Did the transaction happen? Who is qualified to decide? There are no bank records or courts to settle the dispute. Therefore, digital currencies had to create consensus in a trustless environment. No single entity can be trusted, yet a common understanding of the "truth" must be had. Hence, blockchain came into existence. Blockchain provides consensus in a trustless environment. Each transaction is recorded on a block, and each block is verified and signed by someone on the network. The verifier of a block has to do some intense computation as a kind of "insurance" that they verified the block's transactions. Using this system, if Bob fraudulently claims that Steve paid him 300GC, Bob would also have to sign any future block wherein he uses those 300GC, since, hopefully, the other verifiers on the network would have rejected Bob's claimed transaction and would not credit his account. Due to this, Bob would have to beat all other verifiers to sign the next block, which would mean Bob theoretically needs over 50% of the computing power of the entire network - virtually unfeasible in a large enough network. The "truth" in the blockchain would be the chain of blocks with the highest consensus - the chain which appears in the most nodes' history. Simply, the "truth" is what the majority agree upon.

See below a situation where Steve paid Bob 100GC, and Bob fraudulently claims it was 300. Then Bob tries to use those 300GC to pay Alice.



Bob's payment to Alice will only be valid if one of the nodes with a green checkmark verifies the block, as they believe that Steve did send Bob 300GC (they are probably Bob's servers). However, since most nodes have the transaction saved as being only worth 100 coins, they will reject Bob;s payment to Alice. Therefore, the payment will only go through if the two servers which believe Bob produce the next blocks faster than all other nodes combined

The incentive of a node to verify transactions correctly and verify blocks is economic. A block verifier receives two rewards: a mining fee and transaction fees. The mining fee is actually how new coins are minted; as a reward for a node verifying transactions the system credits them with some fresh coins. The transaction fees are simply the cumulative fees tacked on each transaction on the block - each transaction has a fee which is awarded to whoever verifies the transaction - an incentive for block verifiers to include the transaction on their block. If a verifier does a sloppy job and does not verify the transactions on their block, other nodes will reject the block and not credit the verifier their reward. This would be economically unworthwhile since most of the computational power of verifying a block is not in verifying the transactions. Therefore, it would be a waste of time, power, and money to validate a block without verifying its transactions.

All this allows the network to reach consensus in a trustless environment, where all the rules are predefined, and no one-entity controls the economy.

Theoretical Background

BlockChain:

BlockChain technology is a concept by which data, specifically digital transactions and contracts, are stored on "blocks" and each block is dependent on the previous block. This dependency secures the "chain" of blocks from a nefarious block being inserted in retrospect or previous blocks being tampered with. BlockChain forces each block to be created with consideration for the previous block, and by induction, every block before it. Thus, each block is entangled with all the previous blocks, which creates an unbreakable chain.

Digital Signature Algorithm:

The Digital Signature Algorithm (DSA) is a public-key cryptography algorithm which can verify the authenticity of a sender as the actual source of a message, and tamper-check the data, using publicly available knowledge.

Hash Functions:

Hash functions transform data of an arbitrary size into a fixed length digest. This digest cannot be reversed in order to retrieve the original data, yet calculating the hash of the data is fairly simple. Hash functions are one-way functions which can quickly check whether two chunks of data match exactly.

Packets, Sockets and Ports:

Computers communicate with a network by sending Packets through Sockets and Ports.

A **packet** is the actual content of the data which is sent over the network. A packet is made up of the protocol metadata, usually IPs and ports, and actual data the applications want to send.

A **socket** is the actual application endpoint. An application can create a socket and through it communicate with other computers. The communications of a socket pass through a port.

Computers have many **ports**, 65,535 ($2^{16} - 1$) to be exact, which allow them to communicate with remote endpoints. A socket will use a port to send or receive packets from other computers.

Practical situation: An application opens a socket and binds it to a port, then creates a packet with information it wants to send and tells the socket to send it. The socket then sends the packet through the binded port to a remote endpoint - a remote socket.

Internet Protocols - TCP:

Communications over the internet are implemented using protocols. Each protocol has a set of rules which make it ideal for certain situations, and award it advantages as well as disadvantages. One of the most widely used protocols is Transmission Control Protocol (TCP), which is favored thanks to its reliability.

TCP requires a connection between two endpoints, which begins with with a three-way-handshake where the initiator sends a “Synchronize” (SYN) packet to remote peer, the remote peer returns an “Acknowledgement” (ACK) and another “Synchronize” (SYN), and finally the initiator returns an “Acknowledgement” (ACK) to the peer. Due to TCP’s reliable connection and various packet verification methods, two endpoints can know if a packet was received or lost and whether the packet was received correctly. While all this validation creates a reliable protocol, the redundancy heavily weighs on the performance time. Therefore, TCP is used when packet sending frequency is low, and accuracy must be high.

Decentralized Networks - P2P:

A decentralized network is a set of connections between endpoints which communicate with each other, without the communication passing through a central server. In order to achieve a server-less communication net, each endpoint acts as both a server and a client. This will be referred to as a peer-to-peer connection (**P2P**), as network peers are connected to one another directly rather than through a server. Ideally, each peer on the network would host a server, and simultaneously connect to all other peers’ servers.

Cryptography

Cryptography is the mathematical field and study of obscuring, hiding, and securing communication. The classical definition of cryptography is encrypting communication data in order to prevent third-parties from intercepting messages. This technique was used throughout history, mostly by militaries, and was infamously used by the Nazi regieme with the “Enigma” machine. Moreover, cryptography in the form of ciphers have been used for around 4,000 years, the earliest use being on a stone carving in Egypt. Ancient Greece initiated the dependency of militaries on cryptography.

However, since the advent of computer technology, classical methods of encryption have been rendered utterly useless. Arguably, the first ever computer, Alan Turing’s Enigma cryptanalysis machine, was actually specifically designed to crack a cryptographic cipher. What took a brilliant team of mathematicians several months to decrypt would now take anyone with an internet connection minutes. Therefore, much more advanced encryption methods have been developed and implemented, both in military and civilian use.

A commonly used type of cryptography is Cryptographic Hash Functions. These functions are used to convert data into a digest / hash which cannot be reversed. This is further explained below in Hash Functions.

Another type of cryptography is Public-Key Cryptography, which has two major use cases: Encrypting communication between two parties, and verifying the authenticity of a message. The former, which is widely used on the web, recognizable by the ‘S’ in ‘HTTPS’, is a method of protecting communication between two parties without them needing to share a common “key”. Bob could use Alice’s public key to encrypt some data, and only Alice, who has the corresponding private key, would be able to understand the message. The first practical implementation of Public-Key Cryptography, is the RSA algorithm. The latter of the use cases is what transactions in Crypto-currencies use. An algorithm such as DSA can confirm the authenticity of a message using the signer’s public key, which matches the private key they used to sign the message. This form of public-key encryption does not encrypt a message in any way, rather proves that the message received is accurate to what the sender meant to send, and proves that said sender is really the originator of the message.

Hash Functions

A hash function is a function which maps data of an arbitrary length to a hash (data) of a fixed and predefined length.

The main properties of hash functions used in this project are the defined length and determinism. All hashing will be done using SHA256 (see below) which produces a fixed length output hash of 256 bits (32 bytes). This property of defined length allows the placement of a hash inside of a block in our blockchain (later explained in BlockChain) which is arguably the core of blockchain technology. The second property, determinism, creates certainty that data hashed today will produce the exact same hash tomorrow, and the steps by which this hash is produced are all mathematical, and therefore, deterministic. This means that a chunk of data hashed by Bob on Sunday in Tel Aviv, will produce the exact same hash for Alice on Tuesday in California. Since this is an absolute certainty, if Alice's hash is different from Bob's, either the data or the hash we tampered with (possibly by signal interference), or either Bob or Alice are lying.

A key feature and use of hashes is in data protection, specifically passwords. Hashes are one-way functions, meaning that a value for an input is easily computed, however, inverting a value to an input is practically impossible. Therefore, a password, say "qwerty", can be saved to a publicly accessible file as a hash, say a SHA256 hash "65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5", and it would be impossible to compute from this hash that "qwerty" is the password. This can be used in a function such as:

```
int SuperSecureLogin(char * password_input){

   char * correct_password_hash =
"65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5";

   char * input_as_hash = Hash(password_input);

   if (strcpy(correct_password_hash, input_as_hash) == 0) {
      // correct_password_hash == input_as_hash
      printf("Login successful!\n");
      printf("Welcome!\n");
   } else {
      // correct_password_hash != input_as_hash
      printf("Incorrect password!\n");
   }

}
```

A hacker could now access the source code for the login, say it was client-side javascript on a website, and even with the code, and the password being hardcoded, not be able to guess the password (Note: "qwerty" is a very weak and common password. Therefore,

the hash of qwerty can be found in hash directories and crackers such as JohnTheRipper or hashtoolkit.com).

Due to hash functions' one-way functionality, deriving data from a hash is not possible. This feature will be exploited in BlockChain Mining.

SHA-256 - Secure Hash Algorithm - 256

SHA, the most common hashing algorithm family, was first published by the National Institute of Standards and Technology (NIST). The original, SHA-0, was developed by the National Security Agency for DSA use, though SHA-0 was withdrawn due to a significant flaw.

SHA-256 is part of the SHA-2 family, published in 2001, and is 256 bits in size. SHA-256 features 64 rounds of And, XOr, Rotation, Add, Or, Shift Right operations. As of June 2021, no collisions have been found for SHA-256, meaning it is secure and safe to use.

SHA-256, being a strong hash algorithm, produces completely different hashes for very similar inputs. For example:

Input	Hash
Hello World	a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e
Hello, World	03675ac53ff9cd1535ccc7dfcdfa2c458c5218371f418dc136f2d19ac1fbe8a5
Hello world	64ec88ca00b268e5ba1a35678a1b5316d212f4f366b2477232534a8aeca37f3c

This will be used in BlockChain Mining.

Code

The code, in C, to hash data of arbitrary length:

SHA_OPs.h + SHA_OPs.c

```
#pragma once

#include <stdint.h>
#include <string.h>

uint32_t SHA256_op_ch(uint32_t x, uint32_t y, uint32_t z);

uint32_t SHA256_op_maj(uint32_t x, uint32_t y, uint32_t z);

uint32_t SHA256_op_a(uint32_t x);

uint32_t SHA256_op_b(uint32_t x);
```

```
uint32_t SHA256_op_c(uint32_t x);

uint32_t SHA256_op_d(uint32_t x);
```

```
#include "SHA_OPs.h"

uint32_t SHA256_op_ch(uint32_t x, uint32_t y, uint32_t z)
{
	return ((x & y) ^ (~x & z));
}

uint32_t SHA256_op_maj(uint32_t x, uint32_t y, uint32_t z)
{
	return ((x & y) ^ (x & z) ^ (y & z));
}

uint32_t SHA256_op_a(uint32_t x)
{
	return (bit_rotate_right(x, 2) ^ bit_rotate_right(x, 13) ^ bit_rotate_right(x, 22));
}

uint32_t SHA256_op_b(uint32_t x)
{
	return (bit_rotate_right(x, 6) ^ bit_rotate_right(x, 11) ^ bit_rotate_right(x, 25));
}

uint32_t SHA256_op_c(uint32_t x)
{
	return (bit_rotate_right(x, 7) ^ bit_rotate_right(x, 18) ^ (x >> 3));
}

uint32_t SHA256_op_d(uint32_t x)
{
	return (bit_rotate_right(x, 17) ^ bit_rotate_right(x, 19) ^ (x >> 10));
}
```

SHA_Rotate.h + SHA_Rotate.c

```
#pragma once
```

```c
#include <stdint.h>
#include <string.h>

uint32_t bit_rotate_left(uint32_t x, char rotations);
uint32_t bit_rotate_right(uint32_t x, char rotations);
uint64_t bit_rotate_right_64(uint64_t x, char rotations);
```

```c
#include "SHA_Rotate.h"

uint32_t	bit_rotate_left(uint32_t x, char rotations)
{
	return ((x << rotations) | (x >> (32 - rotations)));
}

uint32_t	bit_rotate_right(uint32_t x, char rotations)
{
	return ((x >> rotations) | (x << (32 - rotations)));
}

uint64_t	bit_rotate_right_64(uint64_t x, char rotations)
{
	return ((x >> rotations) | (x << (64 - rotations)));
}
```

SHA256.h + SHA256.c

```c
#pragma once

#ifndef __SHA256_H
#define __SHA256_H

#include <stdint.h>
#include <string.h>

#include "SHA_Rotate/SHA_Rotate.h"
#include "SHA_OPs/SHA_OPs.h"

typedef uint32_t	t_4_uint32[4];
typedef uint32_t	t_8_uint32[8];
typedef uint64_t	t_8_uint64[8];
```

```
typedef uint32_t        t_16_uint32[16];
typedef uint32_t        t_64_uint32[64];
typedef uint64_t        t_80_uint32[80];
typedef uint64_t        t_80_uint64[80];

#define DEC(x) (x-1)

// Chunks of 512 bits
#define SHA256_CHUNK_SIZE 64

#define SHA256_CHUNKS_SIZE(len) ((len + 1 + 8 + DEC(SHA256_CHUNK_SIZE)) &
-DEC(SHA256_CHUNK_SIZE))
#define SHA256_CHUNK_COUNT(len) (SHA256_CHUNKS_SIZE(len) /
SHA256_CHUNK_SIZE)

const uint32_t g_SHA256_k[64];

const uint32_t g_SHA256_default_buffers[8];

// Format a message into a standard 512bit block
size_t format_message(char * message, size_t message_len, unsigned char **
output_message_ptr);

uint32_t bit_swap_32(uint32_t x);

uint64_t bit_swap_64(uint64_t x);

char * SHA256_strncpy(char *dst, const char *src, size_t n);

void uint32_arr_assign_add(uint32_t *dst, const uint32_t *src, size_t len);
void uint32_arr_cpy(uint32_t *dst, const uint32_t *src, size_t len);
void uint64_arr_assign_add(uint64_t *dst, const uint64_t *src, size_t len);
void uint64_arr_cpy(uint64_t *dst, const uint64_t *src, size_t len);



char * uitoa_base(uintmax_t nb, intmax_t base, char letter);
char * uitoa_base_len(uintmax_t nb, intmax_t base, char letter, size_t len);

void SHA256_init_w_array(t_64_uint32 w_array, unsigned char *formatted_msg);

void SHA256_shuffle_buffers(t_8_uint32 buffers, t_64_uint32 w_array);
void SHA256_run_ops(t_8_uint32 buffers, unsigned char *formatted_msg, size_t
```

```
msg_len);

char * build_hash(uint32_t *buffers, size_t buffer_count);
// The SHA-256 hash algorithm
char * Hash_SHA256(char * message, size_t message_len);

#endif // !__SHA256_H
```

```
#include "SHA256.h"

const uint32_t g_SHA256_k[64] = {
        0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1,
        0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786,
        0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
        0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147,
        0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
        0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b,
        0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
        0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a,
        0x5b9cca4f, 0x682e6ff3, 0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
        0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
};

const uint32_t g_SHA256_default_buffers[8] = {
        0x6a09e667,
        0xbb67ae85,
        0x3c6ef372,
        0xa54ff53a,
        0x510e527f,
        0x9b05688c,
        0x1f83d9ab,
        0x5be0cd19
};

// Format a message into a standard 512bit block
size_t format_message(char * message, size_t message_len, unsigned char **
output_message_ptr) {

        size_t output_message_len = message_len + (512 - (message_len % 512));
```

```
	size_t pos;

	*output_message_ptr = (unsigned char *)malloc(output_message_len);

	unsigned char * output_message = *output_message_ptr;

	// Copy the message into the beginning of the formatted message
	memcpy(output_message, message, message_len);

	// Set bit after message to 1, and the rest to 0.
	output_message[message_len] = 0b10000000;

	pos = message_len + 1;

	while (pos < output_message_len) {
	output_message[pos++] = 0;
	}

	output_message[output_message_len - 9] = (uint64_t)message_len;

	return output_message_len;
}


uint32_t bit_swap_32(uint32_t x)
{
	x = ((x << 8) & 0xFF00FF00) | ((x >> 8) & 0xFF00FF);
	return (x << 16) | (x >> 16);
}

uint64_t bit_swap_64(uint64_t x)
{
	x = ((x << 8) & 0xFF00FF00FF00FF00ULL)
	| ((x >> 8) & 0x00FF00FF00FF00FFULL);
	x = ((x << 16) & 0xFFFF0000FFFF0000ULL)
	| ((x >> 16) & 0x0000FFFF0000FFFFULL);
	return (x << 32) | (x >> 32);
}

char   *SHA256_strncpy(char *dst, const char *src, size_t n)
{
	size_t i;
```

```
	i = 0;
	while (src[i] != '\0' && i < n)
	{
	dst[i] = src[i];
	i++;
	}
	while (i < n)
	{
	dst[i] = '\0';
	i++;
	}
	return (dst);
}

void	uint32_arr_assign_add(uint32_t *dst, const uint32_t *src, size_t len)
{
	size_t i;

	i = 0;
	while (i < len)
	{
	dst[i] += src[i];
	i++;
	}
}

void	uint32_arr_cpy(uint32_t *dst, const uint32_t *src, size_t len)
{
	size_t i;

	i = 0;
	while (i < len)
	{
	dst[i] = src[i];
	i++;
	}
}

void	uint64_arr_assign_add(uint64_t *dst, const uint64_t *src, size_t len)
{
	size_t i;

	i = 0;
```

```
        while (i < len)
        {
        dst[i] += src[i];
        i++;
        }
}

void   uint64_arr_cpy(uint64_t *dst, const uint64_t *src, size_t len)
{
        size_t i;

        i = 0;
        while (i < len)
        {
        dst[i] = src[i];
        i++;
        }
}




char   *uitoa_base(uintmax_t nb, intmax_t base, char letter)
{
        uintmax_t       temp;
        int             power;
        char            *str;

        temp = nb;
        power = 1;
        while (temp /= base)
        power++;
        if (!(str = (char *)calloc(power + 1, sizeof(char)))) {
        return (NULL);
        }
        while (power--)
        {
        if (nb % base >= 10)
                str[power] = nb % base - 10 + letter;
        else
                str[power] = nb % base + '0';
        nb /= base;
```

```
        }
        return (str);
}
char   *uitoa_base_len(uintmax_t nb, intmax_t base, char letter, size_t len)
{
        int       i;
        int       diff;
        char   *str;
        char   *new_str;

        i = 0;
        str = uitoa_base(nb, base, letter);
        diff = len - strlen(str);
        if (diff > 0)
        {
        if (!(new_str = (char*)calloc(len + 1, sizeof(char))))
                return (NULL);
        while (i < diff)
                new_str[i++] = '0';
        SHA256_strncpy(new_str + i, str, len - diff);
        free(str);
        return (new_str);
        }
        return (str);
}

void SHA256_init_w_array(t_64_uint32 w_array, unsigned char *formatted_msg)
{
        int i;

        i = 0;
        while (i < 64)
        {
        if (i < 16) {
                w_array[i] = bit_swap_32(((uint32_t *)formatted_msg)[i]);
        }
        else {
                w_array[i] = SHA256_op_d(w_array[i - 2]) + w_array[i - 7] +
SHA256_op_c(w_array[i - 15]) + w_array[i - 16];
        }
        i++;
        }
}
```

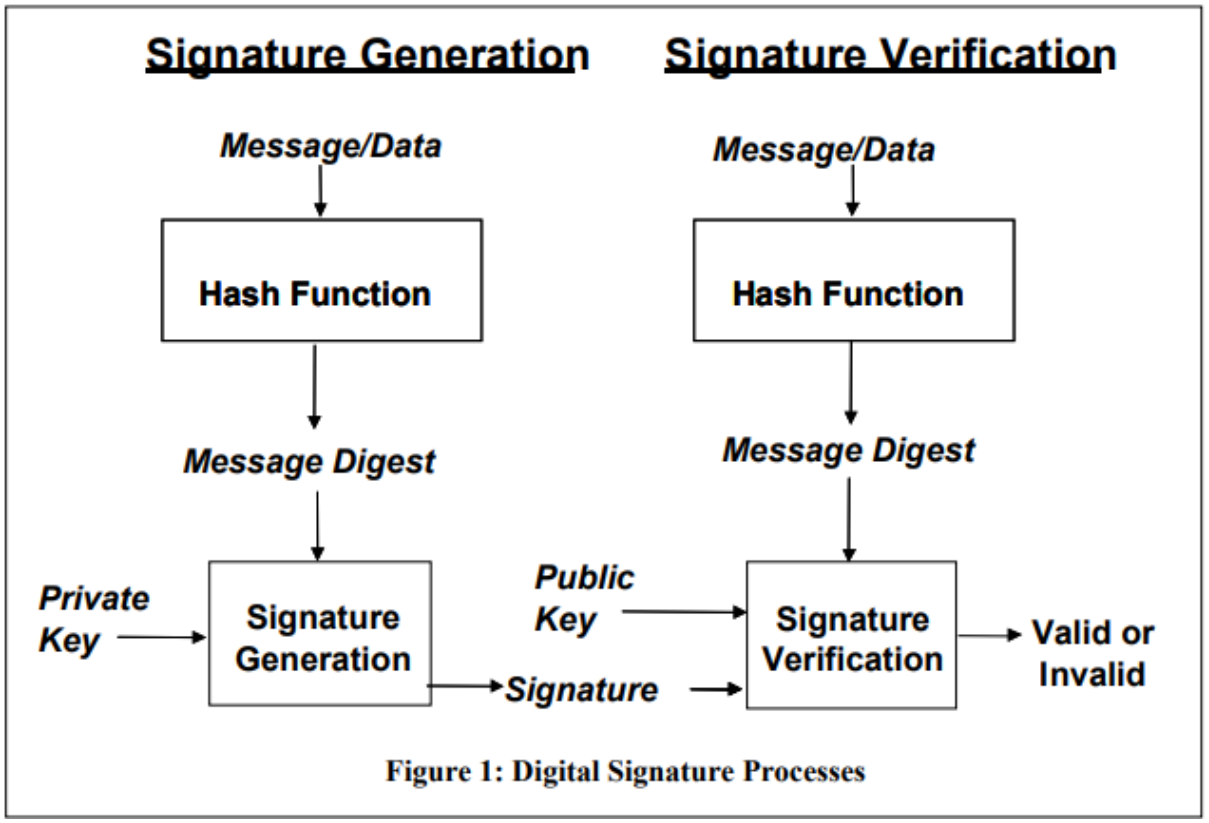
```
void SHA256_shuffle_buffers(t_8_uint32 buffers, t_64_uint32 w_array)
{
	int	i = 0;
	uint32_t temp_a;
	uint32_t temp_b;

	while (i < 64)
	{
	temp_a = buffers[7] + SHA256_op_b(buffers[4]) + SHA256_op_ch(buffers[4],
buffers[5], buffers[6]) + g_SHA256_k[i] + w_array[i];
	temp_b = SHA256_op_a(buffers[0]) + SHA256_op_maj(buffers[0], buffers[1],
buffers[2]);
	buffers[7] = buffers[6];
	buffers[6] = buffers[5];
	buffers[5] = buffers[4];
	buffers[4] = buffers[3] + temp_a;
	buffers[3] = buffers[2];
	buffers[2] = buffers[1];
	buffers[1] = buffers[0];
	buffers[0] = temp_a + temp_b;
	i++;
	}
}

void	SHA256_run_ops(t_8_uint32 buffers,
	unsigned char *formatted_msg, size_t msg_len)
{
	size_t		chunk_i;
	t_64_uint32		w_array;
	t_8_uint32		internal_buffers;

	chunk_i = 0;
	while (chunk_i < SHA256_CHUNK_COUNT(msg_len))
	{
	SHA256_init_w_array(w_array, formatted_msg + chunk_i *
SHA256_CHUNK_SIZE);
	uint32_arr_cpy(internal_buffers, buffers, 8);
	SHA256_shuffle_buffers(internal_buffers, w_array);
	uint32_arr_assign_add(buffers, internal_buffers, 8);
	chunk_i++;
	}
}
```

```
char GLOBAL_HASH_BUFFER;
char * build_hash(uint32_t *buffers, size_t buffer_count)
{
	char * hash;
	char * hash_tmp;
	size_t buffer_i;
	uint32_t buffer;

	buffer_i = 0;
	if (!(hash = (char*)calloc((buffer_count * 8) + 1, sizeof(char)))) {
	return NULL;
	}

	while (buffer_i < buffer_count)
	{
	buffer = 0 ? bit_swap_32(buffers[buffer_i]) : buffers[buffer_i];
	if (!(hash_tmp = uitoa_base_len(buffer, 16, 'a', 8))) {
		return (NULL);
	}
	SHA256_strncpy(hash + (buffer_i * 8), hash_tmp, 8);
	free(hash_tmp);
	buffer_i++;
	}
	return hash;
}

// The SHA-256 hash algorithm
char * Hash_SHA256(char * message, size_t message_len) {
	unsigned char *formatted_msg;
	t_8_uint32		buffers;

	size_t formatted_size = format_message(message, message_len, &formatted_msg);
	if (formatted_size != SHA256_CHUNK_COUNT(message_len) *
SHA256_CHUNK_SIZE) {
	//printf("Size missmatch!\n");
	}

	uint32_arr_cpy(buffers, g_SHA256_default_buffers, 8);
	SHA256_run_ops(buffers, formatted_msg, message_len);
	free(formatted_msg);
	return build_hash(buffers, 8);
}
```

DSA - Digital Signature Algorithm

The Digital Signature Algorithm, developed by the National Institute of Standards and Technology (NIST), is a method of verifying that a certain chunk of data was approved and signed by the relevant parties. DSA bases verification on the discrete logarithm problem, meaning that DSA can be trusted under the assumption that calculating the discrete logarithm of a very large number is obstinate.

Similarly to RSA, DSA uses a private and public key pair, the private key is used to sign the message, and the public key is used to verify the signature.



(Figure taken from FIPS 186-3, Digital Signature Standard)

Assuming the private key is kept secret, it can be confirmed with absolute certainty that a message with a valid signature was signed in its current form by the proper signatory. Should the message have been tampered with, say a malicious party altered the transaction value from 100 to 500, the hash function will return a different digest (as explained in Hash Functions) and the signature will no longer match. Reversing the private key from the public key is also intractable due to the discrete logarithm problem, and fixing the signature to match the altered data is just as difficult. Therefore, a DSA signature verifies that:

1. The message / data have not been tampered with.
2. The message was approved by the signatory (who's public key was used in verification).

Assume that Alice is a spy in a foreign country, and she wants to send Bob, another spy, secret information about the location of a warehouse he must visit. However, Bob is very

sceptical, and believes that the message is not really from Alice, rather that the local authorities are trying to set him up and send him into a trap. Alice can assure Bob with absolute certainty that she indeed sent the message by attaching some extra bits of information to this message. Alice will add the following:

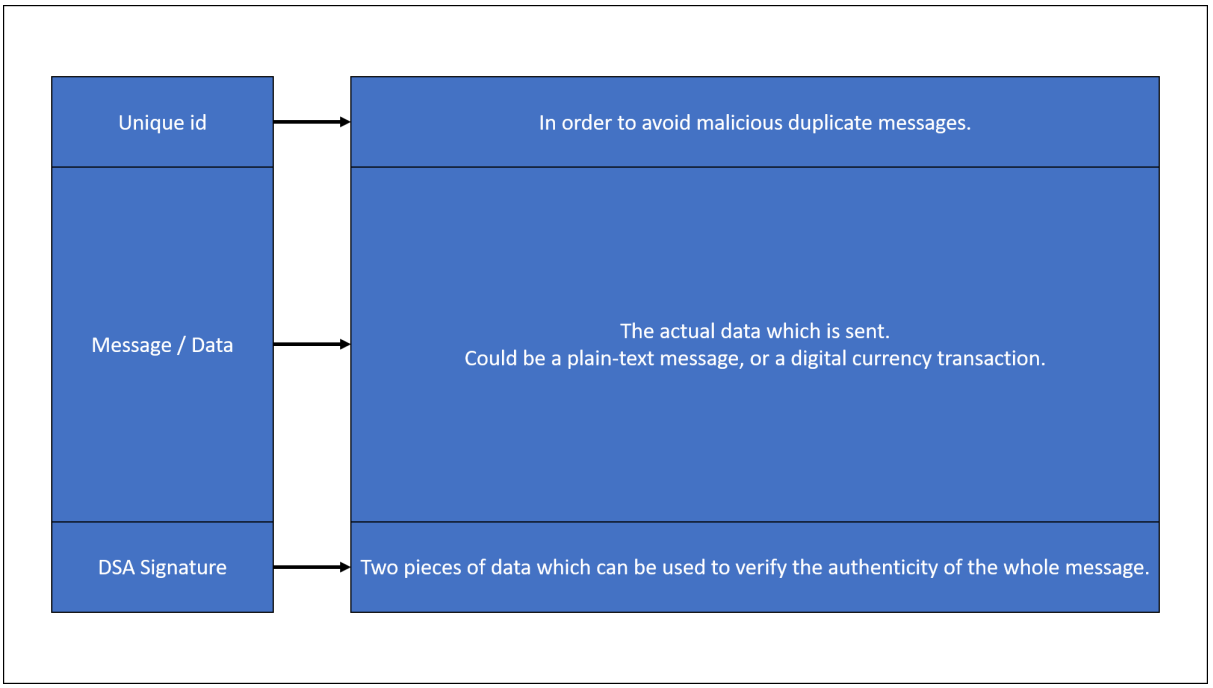
1. A unique identifier to the message which will never be reused. Like the current time.
2. A DSA signature.

Assuming that before being deployed, Bob was given Alice's public key, Bob can now use this key and the signature attached to the message to verify that Alice was indeed the source. The unique identifier is used in order to avoid a situation such as the following:

A few days after Alice sent Bob the message and he verified it and went to the warehouse, the local authorities, which managed to sniff the packet as Alice was sending it to Bob, set up a trap at the same warehouse, and without needing to do any hacking, resend the message as-is to Bob. Bob verifies the message, and sees absolutely no problem - the signature is valid. Bob goes to the warehouse again and is caught.

This could be avoided by having Alice attach a timestamp, the current time, to the original message. Now if Bob receives it again he can see that it is old news and ignore it, and for the local authorities to make it seem new, they would have to re-sign the whole message which would mean cracking the hash function - virtually impossible - or finding Alice's private key - which should also be impossible if Alice is responsible.

Alice's message structure would look like this:



Domain Parameters

Before signing and verifying messages, however, some public and general information should be agreed upon:

- p - A prime number which defines the Galois Field over which all operations are made, GF(p).
- q - A prime factor of p - 1.

- g - The generator, of order q, of the cyclic group of GF(p)*. That is, g is of order q, and is in the multiplicative group GF(p).

Generating g:

Generate_g(p, q) {

1. $e = \frac{p-1}{q}$
2. $h \in \mathbb{N}. 1 < h < (p - 1)$
3. $g = h^e \bmod p$
4. if ($g = 1$), choose a different h, and go to step 2.
5. Return g.

}

Signature

The actual signature part of a message is quite short and simple. The algorithm produces two numbers, called 'r' and 's', or '(r, s)' as a whole. The pair (r, s) can then be used to verify the authenticity of the signature and the message. 'r' and 's' both satisfy $0 < r < q$, $0 < s < q$ respectively. Thus, a DSA signature is twice the bit length of q.

Generation

The algorithm to produce a signature corresponding to a message is as follows:

I. z represents the hash of the message. z = the min(N, outlen) leftmost bits of the hash of the message (where N is the bit length of q, and outlen is the bit length of the hash digest).
II. Choose a random k such that $0 < k < q$, and then calculate k^{-1} such that $1 = (k \times k^{-1}) \bmod q$.
   - In order to keep the private key a secret, a new and different k should be used for each signature.
III. $r = \left(g^k \bmod p \right) \bmod q$
IV. $s = \left(k^{-1} (z + x \cdot r) \right) \bmod q$
V. If $(r = 0) \vee (s = 0)$ choose a different k and repeat from step II.

Let's demonstrate this signature generation using small values: Let:

- p = 11.
- q = 5 (Notice that 5 is a prime factor of 11-1).
- Generate g:
  - $e = \frac{p-1}{q} = \frac{11-1}{5} = \frac{10}{5} = 2.$
  - Choose random h = 3.
  - $g = h^e \bmod p = 3^2 mod\ 11 = 9\ mod\ 11 = 9.$

    - g = 9.
- And let's give ourselves the private key x = 4.
- Assume our message was "Hello World":
    - The SHA-256 hash of "Hello World" is: "a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e" = "1010010110010001101001101101010000000101111110100001000000010000000100101000000000100010111001100111100111110110111101100011001000011010110001011000110010110111111000010111100110110100011001010110101011110110010011101111110110011010110110011111100010100011011110" in binary.
    - The bit length of q (N) is 3, since 5 = 0b101.
    - min(3, 256) = 3.
    - z = 0b101 = 5.
- Choose random k = 4.
    - Calculating the inverse: $k^{-1} = 4$, since $(4 \times 4) \, mod \, 5 = 16 \, mod \, 5 = 1$.
- $r = (g^k \, mod \, p) \, mod \, q = (9^4 \, mod \, 11) \, mod \, 5$

  $= (6561 \, mod \, 11) \, mod \, 5 = 5 \, mod \, 5 = 0$

    - This is an invalid r, and therefore we will choose a different k and repeat.
- Choose a new random k = 3.
    - Calculating the inverse: $k^{-1} = 2$, since $(3 \times 2) \, mod \, 5 = 6 \, mod \, 5 = 1$.
- $r = (g^k \, mod \, p) \, mod \, q = (9^3 \, mod \, 11) \, mod \, 5$

  $= (729 \, mod \, 11) \, mod \, 5 = 3 \, mod \, 5 = 3$

- $s = \left(k^{-1}(z + xr)\right) mod \, q = (2(5 + 4 \cdot 3)) mod \, 5$ $= (2(5 + 12)) mod \, 5 = (2 \cdot 17) mod \, 5 = 4$
- Signature: $(r, s) = (3, 4)$.

Verification

In order to verify a signature, the following steps are done. However, let's first calculate our public key 'y':

$$y = g^x \bmod p = 9^4 \bmod 11 = 5.$$

I. z represents the hash of the message. z = the min(N, outlen) leftmost bits of the hash of the message (where N is the bit length of q, and outlen is the bit length of the hash digest).
II. Check that $0 < r < q \wedge 0 < s < q$.
III. $w = s^{-1} \bmod q$
IV. $u_1 = (z \cdot w) \bmod q$
V. $u_2 = (r \cdot w) \bmod q$
VI. $v = \left((g^{u_1} \cdot y^{u_2}) \bmod p\right) \bmod q$
VII. If (v = r), the signature is valid.

If either r or s are not in the range (0, q), the signature is invalid. If at the end of the calculation, v = r, the signature is valid. Any other outcome means either the message has been tampered with, or the signature is fraudulent.

Let's continue with the demonstration now that we have the signature (3, 4) on the message "Hello World":

- $w = s^{-1} \bmod q = 4^{-1} \bmod 5 = 4.$
- $u_1 = (z \cdot w) \bmod q = (5 \cdot 4) \bmod 5 = 0.$
- $u_2 = (r \cdot w) \bmod q = (3 \cdot 4) \bmod 5 = 2.$
- $v = ((g^{u_1} \cdot y^{u_2}) \bmod p) \bmod q = ((9^0 \cdot 5^2) \bmod 11) \bmod 5$
  $= (25 \bmod 11) \bmod 5 = 3$
- $v = r \equiv 3 = 3.$
  And voilà, v = r, meaning the signature is valid.

Bitwise Math

However, using small numbers like in the demonstration is extremely ill advised. A hacker could easily brute-force a signature by just guessing all the values as there are not many. Specifically, in our demonstration, the simplest attack with zero cryptographic skill would only have to guess only 16 numbers, something that would take any computer less than a second to calculate, and could even be done by hand in about a minute. To overcome this, using very large numbers is mandatory. Instead of defining p and q to be small 3-4 bit numbers, we choose extremely large values consisting of thousands of bits. The specific value GreenCoin uses would require $(2^{160} - 1)^2$ guesses, meaning the chance of guessing correctly is around 2^{-320}which is less than the chance of randomly selecting a star in the universe and then flying blindly until you land on a random star, and then landing on the desired star four times in a row. Or more accurately, the chance is one in this number:

2,135,987,035,920,910,082,395,021,706,169,552,114,602,704,522,353,729,766,672,379,801,985,812,356,115,207,983,983,650,221,850,625.

The catch is, in order to represent numbers this large, we need lots of bits - specifically, up to 2,048 bits for p. This is absurdly large considering we could not even fit the *bit length* of this number into some integer types!

Computers have many data types, the most common ones for storing numbers are:

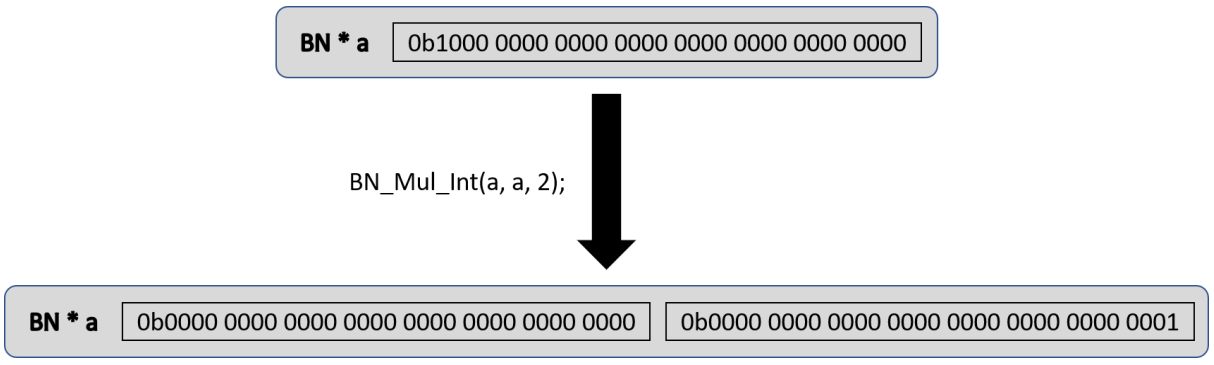
Byte Size	Type Name	Range	Name
1	Byte / Char	-128 -> 127	int8_t
		0 -> 255	uint8_t
2	Word	-32,768 -> 32,767	int16_t
		0 -> 65,535	uint16_t
4	Double Word	-2,147,483,648 -> 2,147,483,647	int32_t
		0 -> 4,294,967,295	uint32_t
		3.4E +/- 38 (7 digits)	float
8	Quad Word	-9,223,372,036,854,775,808 -> 9,223,372,036,854,775,807	int64_t
		0 -> 18,446,744,073,709,551,615	uint64_t
		1.7E +/- 308 (15 digits)	double

Technically, some computers will also support SSE instructions which use 128 bit registers. While theoretically we could use these registers to perform 128 bit operations, this is not trivial and simply not worth the effort as it would not even work on plenty of computers.

Therefore, a whole new data type must be created - one which can hold arbitrarily large values. Notice, though, that we only actually need to deal with whole integers, specifically positive integers: $x \in \mathbb{N}^+$.

For this purpose we create a new library: BNMath (Big/Bit Number Math), which will allow us to manipulate arbitrarily large integers in a way similar to how we use normal integers. These BNs are implemented by using dynamically allocated arrays of uint32_t, which store the actual value of the number. Metadata such as sign and size is stored separately.

A multiplication operation on a BN would look something like this:



Since the number in 'a' multiplied by 2 was too large for the 32 bit storage, the BN increased the storage size to 2 uint32_t, stored in a semi-little endian format - the lower 32

bits being first. This method essentially uses 32 bit values as if they were bytes, and thus uses an array of them to form numbers.

Code

BNMath.h + BNMath.c

```
#pragma once
#ifndef ___BNMath_H
#define ___BNMath_H

#include "../../General/error.h"
#include <string.h>
#include <math.h>
#include <stdint.h>
#include "stdio.h"

#define BN_INT_SIZE 4
#define NULL_DATA 0

typedef unsigned int uint_t;
typedef int int_t;

typedef enum  {
	BN_LITTLE_ENDIAN,
	BN_BIG_ENDIAN
} BN_ENDIAN_FORMAT;

typedef struct {
	int_t sign;	// Positive or negative
	uint_t size;  // Bit size of the number - as multiples of uint_t
	uint_t * data;		// The actual bit value of the number
} BN;


static const signed char HEX_REVERSE_SEQUENCE[256];

#define arraysize(a)    (sizeof(a) / sizeof(a[0]))
#define BN_Is_Even(a)	!BN_Get_Bit_Value(a, 0)
#define BN_Is_Odd(a)	BN_Get_Bit_Value(a, 0)

int_t MAX(int_t a, int_t b);
```

```
int_t MIN(int_t a, int_t b);

void BN_Init(BN ** r); // Initialize a bit number;
void BN_Init_Stack(BN * r); // Initialize a bit number onto the stack. No memory
allocation
void BN_Free(BN * r); // Free a bit number;

error_t BN_Resize_Decrease(BN * r, uint_t size);
error_t BN_Resize(BN * n, uint_t size); // Resize the memory capacity of an existing bit
number;

uint_t BN_Get_Length(const BN * r);
uint_t BN_Get_Byte_Length(const BN * r);
uint_t BN_Get_Bit_Length(const BN * r);

error_t BN_Set_Bit_Value(BN * r, uint_t index, uint_t value);
uint_t BN_Get_Bit_Value(const BN * r, uint_t index);

int_t BN_Compare(const BN * a, const BN * b);
int_t BN_Compare_Int(const BN * a, int_t b);
int_t BN_Compare_Abs(const BN * a, const BN * b);

error_t BN_Copy(BN * r, const BN * a);
error_t BN_Set_Value(BN * r, long long int a);

error_t BN_Randomize(BN * r, uint_t length);

error_t BN_Import(BN * r, const uint8_t * data, uint_t length, BN_ENDIAN_FORMAT
format);
error_t BN_Import_Hex_String(BN * r, char * data, uint_t length,
BN_ENDIAN_FORMAT format);

error_t BN_Add(BN * r, const BN * a, const BN * b);                 // Adds a and b into r. r
= a + b
error_t BN_Add_Int(BN * r, const BN * a, int_t b);                  // Addition of Bit
number and a normal number
error_t BN_Add_Absolute(BN * r, const BN * a, const BN * b);  // Addition assuming
both a and b are positive

error_t BN_Sub(BN * r, const BN * a, const BN * b);                 // Subtracts b from a
and returns into r. r = a - b
error_t BN_Sub_Int(BN * r, const BN * a, int_t b);
error_t BN_Sub_Absolute(BN * r, const BN * a, const BN * b);
```

```
error_t BN_Shift_Left(BN * r, uint_t l);
error_t BN_Shift_Right(BN * r, uint_t l);

error_t BN_Mul(BN * r, const BN * a, const BN * b);
error_t BN_Mul_Int(BN * r, const BN * a, int_t b);

error_t BN_Div(BN * q, BN * r, const BN * a, const BN * b);
error_t BN_Div_Int(BN * q, BN * r, const BN * a, int_t b);

error_t BN_Mod(BN * r, const BN * a, const BN * p);                    // r = a % p
error_t BN_Add_Mod(BN * r, const BN * a, const BN * b, const BN * p); // r = (a + b) %
p
error_t BN_Sub_Mod(BN * r, const BN * a, const BN * b, const BN * p); // r = (a - b) % p
error_t BN_Mul_Mod(BN * r, const BN * a, const BN * b, const BN * p); // r = (a * b) % p
error_t BN_Exp_Mod(BN * r, const BN * a, const BN * e, const BN * p); // r = pow(a,e)
% p
error_t BN_Inv_Mod(BN * r, const BN * a, const BN * p);                    // Returns (into
r) the multiplicative inverse of a over the field p

error_t BN_Montgomery_Mul(BN * r, const BN * a, const BN * b, uint_t k, const BN * p,
BN * t);
error_t BN_Montgomery_Red(BN * r, const BN * a, uint_t k, const BN * p, BN * t);

void BN_Mul_Core(uint_t *r, const uint_t *a, int_t m, const uint_t b);

void BN_Dump(FILE * stream, const char * prepend, const BN * a);

int BN_Is_Prime(BN * r);

#endif // !___BNMath_H
```

```
#include "BNMath.h"

static const signed char HEX_REVERSE_SEQUENCE[256] = {
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
0,  1,  2,  3,  4,  5,  6,  7,  8,  9, -1, -1, -1, -1, -1, -1,
-1, 10, 11, 12, 13, 14, 15, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

```
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, 10, 11, 12, 13, 14, 15, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1
};

int_t MAX(int_t a, int_t b) {
	return (a > b) ? a : b;
}
int_t MIN(int_t a, int_t b) {
	return (a < b) ? a : b;
}
int_t ceil_div(float a, float q) {
	float b = ceil(a/q);
	int c = b / 1;
	return c;
}

void BN_Init(BN ** r) {
	*r = (BN*)malloc(sizeof(BN));
	BN * rt = *r;
	rt->sign = 1;
	rt->size = 0;
	rt->data = NULL_DATA;
}

void BN_Init_Stack(BN * r) {
	r->sign = 1;
	r->size = 0;
	r->data = NULL_DATA;
}

void BN_Free(BN * r) {
	if (r->data != NULL_DATA) {
	memset(r->data, 0, r->size * BN_INT_SIZE);
	free(r->data);
```

```
        }

        r->size = 0;
        r->data = NULL_DATA;

        free(r);
}

error_t BN_Resize_Decrease(BN * r, uint_t size) {
        uint_t * data;

        size = (uint_t)MAX(size, 1); // Override if size < 1

        data = calloc(size * BN_INT_SIZE);
        if (data == 0) { /* Could not allocate memory! */ return ERROR_FAILED; }


        if (r->size > 0) {
        memcpy(data, r->data, size * BN_INT_SIZE);
        /*printf("[0] 0x%.8X 0x%.8X\n", r->data[0], data[0]);
        printf("[1] 0x%.8X 0x%.8X\n", r->data[1], data[1]);
        printf("[2] 0x%.8X\n", r->data[2]);*/
        free(r->data);
        }

        r->size = size;
        r->data = data;

        return ERROR_NONE;
}
error_t BN_Resize(BN * r, uint_t size) {
        uint_t * data;

        size = (uint_t)MAX(size, 1); // Override if size < 1

        if (r->size == size) { return ERROR_NONE; }
        else if (r->size > size) {
        //return BN_Resize_Decrease(r, size);
        return ERROR_NONE;
        }

        data = calloc(size * BN_INT_SIZE, sizeof(char));
        if (data == 0) { /* Could not allocate memory! */ return ERROR_FAILED; }
```

```
        if (r->size > 0) {
        memcpy(data, r->data, r->size * BN_INT_SIZE);
        free(r->data);
        }

        r->size = size;
        r->data = data;

        return ERROR_NONE;
}

uint_t BN_Get_Length(const BN * r) {
        if (r->size == 0) { return 0; }

        uint_t i = 0;
        for (i = r->size - 1; i >= 0; i--) {
        if (r->data[i] != 0) {
                break;
        }
        }

        return i + 1;
}
uint_t BN_Get_Byte_Length(const BN * r) {
        uint_t n = 0;
        uint32_t m;

        if (r->size == 0) { return 0; }

        for (n = r->size - 1; n > 0; n--) {
        if (r->data[n] != 0) {
                break;
        }
        }

        m = r->data[n];
        n *= BN_INT_SIZE;

        for (; m != 0; m >>= 8) { n++; }

        return n;
```

```
}
uint_t BN_Get_Bit_Length(const BN * r) {
        uint_t n;
        uint32_t m;

        if (r->size == 0) { return 0; }

        for (n = r->size - 1; n > 0; n--) {
        if (r->data[n] != 0) { break; }
        }

        m = r->data[n];
        n *= BN_INT_SIZE * 8;

        // Final bit count
        for (; m != 0; m >>= 1) {
        n++;
        }

        return n;
}


error_t BN_Set_Bit_Value(BN * r, uint_t index, uint_t value) {
        uint_t quot;
        uint_t rema;

        quot = index / (BN_INT_SIZE * 8);
        rema = index % (BN_INT_SIZE * 8);

        error_t  error;
        error = BN_Resize(r, quot + 1);
        if (error) { return error; }

        if (value) { // 1
        r->data[quot] |= (1 << rema); // Or
        }
        else { // 0
        r->data[quot] &= ~(1 << rema); // And
        }

        return ERROR_NONE;
}
```

```
uint_t BN_Get_Bit_Value(const BN * r, uint_t index) {
	uint_t quot = index / (BN_INT_SIZE * 8);
	uint_t rema = index % (BN_INT_SIZE * 8);

	if (quot >= r->size) { // Index out of range
	return 0;
	}

	return (r->data[quot] >> rema) & 0x01;
}

int_t BN_Compare(const BN * a, const BN * b) {
	uint_t m = BN_Get_Length(a);
	uint_t n = BN_Get_Length(b);

	if (!m && !n) {
	return 0;
	}
	else if (m > n) {
	return a->sign;
	} else if (m < n) {
	return -b->sign;
	}

	if (a->sign > 0 && b->sign < 0) {
	return 1;
	} else if (a->sign < 0 && b->sign > 0) {
	return -1;
	}

	while (n--)
	{
	if (a->data[n] > b->data[n]) {
		return a->sign;
	} else if (a->data[n] < b->data[n]) {
		return -a->sign;
	}
	}

	return 0;
}
int_t BN_Compare_Int(const BN * a, int_t b) {
	uint_t value = (b >= 0) ? b : -b;
```

```
        // Demo BN from b
        BN t;
        t.sign = (b >= 0) ? 1 : -1;
        t.size = 1;
        t.data = &value;

        return BN_Compare(a, &t);
}
int_t BN_Compare_Abs(const BN * a, const BN * b) {
        uint_t m = BN_Get_Length(a);
        uint_t n = BN_Get_Length(b);

        if (!m && !n) {
        return 0; // Numbers are both of size 0 -> empty
        } else if (m > n) {
        return 1;
        } else if (m < n) {
        return -1;
        }

        while (n--)
        {
        if (a->data[n] > b->data[n]) {
                return 1;
        } else if (a->data[n] < b->data[n]) {
                return -1;
        }
        }

        return 0;
}


error_t BN_Copy(BN * r, const BN * a) {
        if (r == a) { // are r and a the exact same
        return ERROR_NONE;
        }

        uint_t n = BN_Get_Length(a);

        //Adjust the size of the destination operand
        error_t error = BN_Resize(r, n);
```

```
        //Any error to report?
        if (error) {
        return error;
        }

        memset(r->data, 0, r->size * BN_INT_SIZE);

        memcpy(r->data, a->data, n * BN_INT_SIZE);

        r->sign = a->sign;

        return ERROR_NONE;
}
error_t BN_Set_Value(BN * r, long long int a) {
        error_t error = BN_Resize(r, sizeof(a) / BN_INT_SIZE);
        if (error) {
        return error;
        }

        memset(r->data, 0, r->size * BN_INT_SIZE);

        r->data[0] = (a >= 0) ? a : -a;
        r->data[1] = (a >= 0) ? (a>>8*BN_INT_SIZE) : -(a >> 8 * BN_INT_SIZE);

        r->sign = (a >= 0) ? 1 : -1;

        return ERROR_NONE;
}


error_t BN_Randomize(BN * r, uint_t length) {
        uint_t n = (length + (BN_INT_SIZE * 8) - 1) / (BN_INT_SIZE * 8);
        uint_t m = length % (BN_INT_SIZE * 8);

        error_t error = BN_Resize(r, n);

        if (error) {
        return error;
        }

        memset(r->data, 0, r->size * BN_INT_SIZE);
        r->sign = 1;
```

```
        //Generate a random pattern
        for (int i = 0; i < length; i++) {
        error |= BN_Set_Bit_Value(r, i, rand()%2);
        }
        if (error) {
        return error;
        }

        if (n > 0 && m > 0)
        {
        r->data[n - 1] &= (1 << m) - 1;
        }

        return ERROR_NONE;
}


error_t BN_Add(BN * r, const BN * a, const BN * b) {
        error_t error = 0;

        int_t sign = a->sign;

        if (a->sign == b->sign) {
        error = BN_Add_Absolute(r, a, b);
        r->sign = sign;
        } else {
        if (BN_Compare_Abs(a, b) >= 0) {
                //Perform subtraction
                error = BN_Sub_Absolute(r, a, b);
                r->sign = sign;
        } else {
                error = BN_Sub_Absolute(r, b, a);
                r->sign = -sign;
        }
        }

        return error;
}
error_t BN_Add_Int(BN * r, const BN * a, int_t b) {
        uint_t value;

        // Create demo BN from b
        BN t;
```

```
        value = (b >= 0) ? b : -b;
        t.sign = (b >= 0) ? 1 : -1;
        t.size = 1;
        t.data = &value;

        return BN_Add(r, a, &t);
}
error_t BN_Add_Absolute(BN * r, const BN * a, const BN * b) {
        error_t error = 0;
        uint_t i;
        uint_t n;
        uint_t c;
        uint_t d;

        // If b and r are the same, swap a and b
        if (r == b)
        {
        const BN * t = a;
        a = b;
        b = t;
        } else if (r != a)
        {
        BN_Copy(r, a); // Copy a to r
        }

        n = BN_Get_Length(b);
        BN_Resize(r, n);

        r->sign = 1;

        c = 0; // Carry bit

        // Add operands
        for (i = 0; i < n; i++)
        {
        // Add carry bit
        d = r->data[i] + c;
        // Update carry bit
        if (d != 0) c = 0;
        // Perform addition
        d += b->data[i];
        // Update carry bit
        if (d < b->data[i]) c = 1;
```

```
        // Save result
        r->data[i] = d;
        }


    // Loop as long as the carry bit is set
        for (i = n; c && i < r->size; i++)
        {
        // Add carry bit
        r->data[i] += c;
        // Update carry bit
        if (r->data[i] != 0) c = 0;
        }

        // Check the final carry bit
        if (c && n >= r->size)
        {
        // Extend the size of the destination register
        BN_Resize(r, n + 1);
        // Add carry bit
        r->data[n] = 1;
        }

        return error;
}

error_t BN_Sub(BN * r, const BN * a, const BN * b) {
        error_t error = 0;
        int_t sign = a->sign;

        if (a->sign == b->sign)
        {
        if (BN_Compare_Abs(a, b) >= 0)
        {
                error = BN_Sub_Absolute(r, a, b);
                r->sign = sign;
        }
        else
        {
                error = BN_Sub_Absolute(r, b, a);
                r->sign = -sign;
        }
        } else {
```

```
        error = BN_Add_Absolute(r, a, b);
        r->sign = sign;
        }
        return error;
}
error_t BN_Sub_Int(BN * r, const BN * a, int_t b) {
        uint_t value;

        // Create BN from b
        BN t;
        value = (b >= 0) ? b : -b;
        t.sign = (b >= 0) ? 1 : -1;
        t.size = 1;
        t.data = &value;

        return BN_Sub(r, a, &t);
}
error_t BN_Sub_Absolute(BN * r, const BN * a, const BN * b) {
        error_t error = 0;
        uint_t c;
        uint_t d;
        uint_t i;
        uint_t m;
        uint_t n;

        if (BN_Compare_Abs(a, b) < 0)
        {
        const BN * t = b;
        a = b;
        b = t;
        }

        m = BN_Get_Length(a);
        n = BN_Get_Length(b);

        BN_Resize(r, m);

        r->sign = 1;

        c = 0; // Carry bit

        for (i = 0; i < n; i++)
        {
```

```
d = a->data[i];

//Check the carry bit
if (c)
{
	//Update carry bit
	if (d != 0) c = 0;
	//Propagate carry bit
	d -= 1;
}

//Update carry bit
if (d < b->data[i]) c = 1;
//Perform subtraction
r->data[i] = d - b->data[i];
}

//Loop as long as the carry bit is set
for (i = n; c && i < m; i++)
{
//Update carry bit
if (a->data[i] != 0) c = 0;
//Propagate carry bit
r->data[i] = a->data[i] - 1;
}

//R and A are not the same instance?
if (r != a)
{
//Copy the remaining words
for (; i < m; i++)
{
	r->data[i] = a->data[i];
}

//Zero the upper part of R
for (; i < r->size; i++)
{
	r->data[i] = 0;
}
}

return error;
```

```
}


error_t BN_Shift_Left(BN * r, uint_t l) {
	error_t error = 0;
	uint_t i;

	//Number of 32-bit words to shift
	uint_t n1 = l / (BN_INT_SIZE * 8);
	//Number of bits to shift
	uint_t n2 = l % (BN_INT_SIZE * 8);

	if (!r->size || !l) {
	return ERROR_NONE;
	}

	error = BN_Resize(r, r->size + (l + 31) / 32);
	if (error) {
	return error;
	}

	//First, shift words
	if (n1 > 0)
	{
	//Process the most significant words
	for (i = r->size - 1; i >= n1; i--)
	{
		r->data[i] = r->data[i - n1];
	}

	//Fill the rest with zeroes
	for (i = 0; i < n1; i++)
	{
		r->data[i] = 0;
	}
	}

	//Then shift bits
	if (n2 > 0)
	{
	//Process the most significant words
	for (i = r->size - 1; i >= 1; i--)
	{
```

```
                r->data[i] = (r->data[i] << n2) | (r->data[i - 1] >> (32 - n2));
        }

        //The least significant word requires a special handling
        r->data[0] <<= n2;
        }

        return ERROR_NONE;
}
error_t BN_Shift_Right(BN * r, uint_t l) {
        uint_t i;
        uint_t m;

        //Number of 32-bit words to shift
        uint_t n1 = l / (BN_INT_SIZE * 8);
        //Number of bits to shift
        uint_t n2 = l % (BN_INT_SIZE * 8);

        //Check parameters
        if (n1 >= r->size)
        {// If we are moving more bits than there are, then reset all the data.
        memset(r->data, 0, r->size * BN_INT_SIZE);
        return ERROR_NONE;
        }

        //First, shift words
        if (n1 > 0)
        {
        //Process the least significant words
        for (m = r->size - n1, i = 0; i < m; i++)
        {
                r->data[i] = r->data[i + n1];
        }

        //Fill the rest with zeroes
        for (i = m; i < r->size; i++)
        {
                r->data[i] = 0;
        }
        }

        //Then shift bits
        if (n2 > 0)
```

```
        {
        //Process the least significant words
        for (m = r->size - n1 - 1, i = 0; i < m; i++)
        {
                r->data[i] = (r->data[i] >> n2) | (r->data[i + 1] << (32 - n2));
        }

        //The most significant word requires a special handling
        r->data[m] >>= n2;
        }

        // Check if we need to resize
        uint_t length_whole;
        uint_t length_bits;
        uint_t total_length;

        total_length = BN_Get_Bit_Length(r);
        length_whole = total_length / (BN_INT_SIZE * 8);
        length_bits = total_length % (BN_INT_SIZE * 8);

        uint_t actual_length;
        actual_length = ceil_div(((length_whole * BN_INT_SIZE * 8) + length_bits) ,
(BN_INT_SIZE * 8));

        if (actual_length != r->size && actual_length > 0) {
        // Resize
        BN_Resize(r, actual_length);
        }

        return ERROR_NONE;
}


error_t BN_Mul(BN * r, const BN * a, const BN * b) {
        error_t error = 0;
        int_t i;
        int_t m;
        int_t n;
        BN ta;
        BN tb;

        //Initialize multiple precision integers
        BN_Init_Stack(&ta);
```

```
BN_Init_Stack(&tb);

if (r == a)
{
//Copy A to TA
BN_Copy(&ta, a);
a = &ta;
}

if (r == b)
{
//Copy B to TB
BN_Copy(&tb, b);
b = &tb;
}

m = BN_Get_Length(a);
n = BN_Get_Length(b);

//Adjust the size of R
BN_Resize(r, m + n);

r->sign = (a->sign == b->sign) ? 1 : -1;

//Clear the contents of the destination integer
memset(r->data, 0, r->size * BN_INT_SIZE);

//Perform multiplication
if (m < n)
{
for (i = 0; i < m; i++)
{
        BN_Mul_Core(&r->data[i], b->data, n, a->data[i]);
}
}
else
{
for (i = 0; i < n; i++)
{
        BN_Mul_Core(&r->data[i], a->data, m, b->data[i]);
}
}
```

```
        free(ta.data);
        free(tb.data);

        return error;
}
error_t BN_Mul_Int(BN * r, const BN * a, int_t b) {
        uint_t value;

        // Create BN from b
        BN t;
        value = (b >= 0) ? b : -b;
        t.sign = (b >= 0) ? 1 : -1;
        t.size = 1;
        t.data = &value;

        return BN_Mul(r, a, &t);
}


error_t BN_Div(BN * q, BN * r, const BN * a, const BN * b) {
        error_t error = 0;
        uint_t m;
        uint_t n;
        BN c;
        BN d;
        BN e;

        //Check whether the divisor is zero
        if (!BN_Compare_Int(b, 0)) {
        return ERROR_FAILED;
        }

        BN_Init_Stack(&c);
        BN_Init_Stack(&d);
        BN_Init_Stack(&e);

        BN_Copy(&c, a);
        BN_Copy(&d, b);
        BN_Set_Value(&e, 0);

        m = BN_Get_Bit_Length(&c);
        n = BN_Get_Bit_Length(&d);
```

```
        if (m > n) {
        BN_Shift_Left(&d, m - n);
        }

        while (n++ <= m)
        {
        BN_Shift_Left(&e, 1);

        int_t cmp = BN_Compare(&c, &d);

        if (cmp >= 0)
        {
                BN_Set_Bit_Value(&e, 0, 1);
                BN_Sub(&c, &c, &d);
        }

        BN_Shift_Right(&d, 1);
        }

        if (q != NULL) {
        BN_Copy(q, &e);
        }

        if (r != NULL) {
        BN_Copy(r, &c);
        }

        free(c.data);
        free(d.data);
        free(e.data);

        return error;
}
error_t BN_Div_Int(BN * q, BN * r, const BN * a, int_t b) {
        uint_t value;

        // Create BN from b
        BN t;
        value = (b >= 0) ? b : -b;
        t.sign = (b >= 0) ? 1 : -1;
        t.size = 1;
        t.data = &value;
```

```
	return BN_Div(q, r, a, &t);
}

error_t BN_Mod(BN * r, const BN * a, const BN * p) {
	error_t error = 0;
	int_t sign;
	uint_t m;
	uint_t n;
	BN c;

	if (BN_Compare_Int(p, 0) <= 0) {
	return ERROR_FAILED;
	}

	BN_Init_Stack(&c);

	sign = a->sign;
	m = BN_Get_Bit_Length(a);
	n = BN_Get_Bit_Length(p);


	BN_Copy(r, a);

	if (m >= n)
	{
	BN_Copy(&c, p);
	BN_Shift_Left(&c, m - n);

	while (BN_Compare_Abs(r, p) >= 0)
	{
		if (BN_Compare_Abs(r, &c) >= 0)
		{
		BN_Sub_Absolute(r, r, &c);
		}

		BN_Shift_Right(&c, 1);
	}
	}

	if (sign < 0)
	{
	BN_Sub_Absolute(r, p, r);
	}
```

```
        free(c.data);

        return error;
}

error_t BN_Add_Mod(BN * r, const BN * a, const BN * b, const BN * p) {
        error_t error = 0;

        BN_Add(r, a, b);
        BN_Mod(r, r, p);

        return error;
}

error_t BN_Sub_Mod(BN * r, const BN * a, const BN * b, const BN * p) {
        error_t error = 0;

        BN_Sub(r, a, b);
        BN_Mod(r, r, p);

        return error;
}

error_t BN_Mul_Mod(BN * r, const BN * a, const BN * b, const BN * p) {
        error_t error = 0;

        BN_Mul(r, a, b);
        BN_Mod(r, r, p);

        return error;
}

error_t BN_Exp_Mod(BN * r, const BN * a, const BN * e, const BN * p) {
        error_t error = 0;
        int_t i;
        int_t j;
        int_t n;
        uint_t d;
        uint_t k;
        uint_t u;
        BN b;
        BN c2;
```

```
BN t;
BN s[8];

//Initialize
BN_Init_Stack(&b);
BN_Init_Stack(&c2);
BN_Init_Stack(&t);

//Initialize precomputed values
for (i = 0; i < arraysize(s); i++)
{
BN_Init_Stack(&s[i]);
}

//Very small exponents are often selected with low Hamming weight.
//The sliding window mechanism should be disabled in that case
d = (BN_Get_Bit_Length(e) <= 32) ? 1 : 4;

if (BN_Is_Even(p))
{
//Let B = A^2
BN_Mul_Mod(&b, a, a, p);
BN_Copy(&s[0], a);

//Precompute S[i] = A^(2 * i + 1)
for (i = 1; i < (1 << (d - 1)); i++)
{
        BN_Mul_Mod(&s[i], &s[i - 1], &b, p);
}

//Let R = 1
BN_Set_Value(r, 1);

//The exponent is processed in a left-to-right fashion
i = BN_Get_Bit_Length(e) - 1;

//Perform sliding window exponentiation
while (i >= 0)
{
        //The sliding window exponentiation algorithm decomposes E
        //into zero and nonzero windows
        if (!BN_Get_Bit_Value(e, i))
        {
```

```
            //Compute R = R^2
            BN_Mul_Mod(r, r, r, p);
            //Next bit to be processed
            i--;
            }
            else
            {
            //Find the longest window
            n = MAX(i - d + 1, 0);

            //The least significant bit of the window must be equal to 1
            while (!BN_Get_Bit_Value(e, n)) n++;

            //The algorithm processes more than one bit per iteration
            for (u = 0, j = i; j >= n; j--)
            {
                  //Compute R = R^2
                  BN_Mul_Mod(r, r, r, p);
                  //Compute the relevant index to be used in the precomputed table
                  u = (u << 1) | BN_Get_Bit_Value(e, j);
            }

            //Perform a single multiplication per iteration
            BN_Mul_Mod(r, r, &s[u >> 1], p);
            //Next bit to be processed
            i = n - 1;
            }
      }
      }
      else
      {
      //Compute the smaller C = (2^32)^k such as C > P
      k = BN_Get_Length(p);

      //Compute C^2 mod P
      BN_Set_Value(&c2, 1);
      BN_Shift_Left(&c2, 2 * k * (BN_INT_SIZE * 8));
      BN_Mod(&c2, &c2, p);

      //Let B = A * C mod P
      if (BN_Compare(a, p) >= 0)
      {
            BN_Mod(&b, a, p);
```

```
            BN_Montgomery_Mul(&b, &b, &c2, k, p, &t);
        }
        else
        {
            BN_Montgomery_Mul(&b, a, &c2, k, p, &t);
        }

        //Let R = B^2 * C^-1 mod P
        BN_Montgomery_Mul(r, &b, &b, k, p, &t);
        //Let S[0] = B
        BN_Copy(&s[0], &b);

        //Precompute S[i] = B^(2 * i + 1) * C^-1 mod P
        for (i = 1; i < (1 << (d - 1)); i++)
        {
            BN_Montgomery_Mul(&s[i], &s[i - 1], r, k, p, &t);
        }

        //Let R = C mod P
        BN_Copy(r, &c2);
        BN_Montgomery_Red(r, r, k, p, &t);

        //The exponent is processed in a left-to-right fashion
        i = BN_Get_Bit_Length(e) - 1;

        //Perform sliding window exponentiation
        while (i >= 0)
        {
            //The sliding window exponentiation algorithm decomposes E
            //into zero and nonzero windows
            if (!BN_Get_Bit_Value(e, i))
            {
            //Compute R = R^2 * C^-1 mod P
            BN_Montgomery_Mul(r, r, r, k, p, &t);
            //Next bit to be processed
            i--;
            }
            else
            {
            //Find the longest window
            n = MAX(i - d + 1, 0);

            //The least significant bit of the window must be equal to 1
```

```
                while (!BN_Get_Bit_Value(e, n)) n++;

                //The algorithm processes more than one bit per iteration
                for (u = 0, j = i; j >= n; j--)
                {
                        //Compute R = R^2 * C^-1 mod P
                        BN_Montgomery_Mul(r, r, r, k, p, &t);
                        //Compute the relevant index to be used in the precomputed table
                        u = (u << 1) | BN_Get_Bit_Value(e, j);
                }

                //Compute R = R * T[u/2] * C^-1 mod P
                BN_Montgomery_Mul(r, r, &s[u >> 1], k, p, &t);
                //Next bit to be processed
                i = n - 1;
                }
        }

        //Compute R = R * C^-1 mod P
        BN_Montgomery_Red(r, r, k, p, &t);
        }

        free(b.data);
        free(c2.data);
        free(t.data);

        for (i = 0; i < arraysize(s); i++)
        {
        free(s[i].data);
        }

        return error;
}

error_t BN_Inv_Mod(BN * r, const BN * a, const BN * p) {
        error_t error = 0;
        BN b;
        BN c;
        BN q0;
        BN r0;
        BN t;
        BN u;
        BN v;
```

```
//Initialize multiple precision integers
BN_Init_Stack(&b);
BN_Init_Stack(&c);
BN_Init_Stack(&q0);
BN_Init_Stack(&r0);
BN_Init_Stack(&t);
BN_Init_Stack(&u);
BN_Init_Stack(&v);

BN_Copy(&b, p);
BN_Copy(&c, a);
BN_Set_Value(&u, 0);
BN_Set_Value(&v, 1);

while (BN_Compare_Int(&c, 0) > 0)
{
BN_Div(&q0, &r0, &b, &c);

BN_Copy(&b, &c);
BN_Copy(&c, &r0);

BN_Copy(&t, &v);
BN_Mul(&q0, &q0, &v);
BN_Sub(&v, &u, &q0);
BN_Copy(&u, &t);
}

if (BN_Compare_Int(&b, 1))
{
//MPI_CHECK(ERROR_FAILURE);
}

if (BN_Compare_Int(&u, 0) > 0)
{
BN_Copy(r, &u);
}
else
{
BN_Add(r, &u, p);
}

free(b.data);
```

```
	free(c.data);
	free(q0.data);
	free(r0.data);
	free(t.data);
	free(u.data);
	free(v.data);

	return error;
}

error_t BN_Montgomery_Mul(BN * r, const BN * a, const BN * b, uint_t k, const BN * p,
BN * t)
{
	error_t error = 0;
	uint_t i;
	uint_t m;
	uint_t n;
	uint_t q;

	//Use Newton's method to compute the inverse of P[0] mod 2^32
	for (m = 2 - p->data[0], i = 0; i < 4; i++)
	{
	m = m * (2 - m * p->data[0]);
	}

	//Precompute -1/P[0] mod 2^32;
	m = ~m + 1;

	//We assume that B is always less than 2^k
	n = MIN(b->size, k);

	//Make sure T is large enough
	BN_Resize(t, 2 * k + 1);
	//Let T = 0
	BN_Set_Value(t, 0);

	//Perform Montgomery multiplication
	for (i = 0; i < k; i++)
	{
	//Check current index
	if (i < a->size)
	{
		//Compute q = ((T[i] + A[i] * B[0]) * m) mod 2^32
```

```
                q = (t->data[i] + a->data[i] * b->data[0]) * m;
                //Compute T = T + A[i] * B
                BN_Mul_Core(t->data + i, b->data, n, a->data[i]);
        }
        else
        {
                //Compute q = (T[i] * m) mod 2^32
                q = t->data[i] * m;
        }

        //Compute T = T + q * P
        BN_Mul_Core(t->data + i, p->data, k, q);
        }

        //Compute R = T / 2^(32 * k)
        BN_Shift_Right(t, k * (BN_INT_SIZE * 8));
        BN_Copy(r, t);

        //A final subtraction is required
        if (BN_Compare(r, p) >= 0)
        {
        BN_Sub(r, r, p);
        }

        return error;
}
error_t BN_Montgomery_Red(BN * r, const BN * a, uint_t k, const BN * p, BN * t)
{
        uint_t value;

        //Let B = 1
        BN b;
        value = 1;
        b.sign = 1;
        b.size = 1;
        b.data = &value;

        //Compute R = A / 2^k mod P
        return BN_Montgomery_Mul(r, a, &b, k, p, t);
}


void BN_Mul_Core(uint_t *r, const uint_t *a, int_t m, const uint_t b)
```

```
{
	int_t i;
	uint32_t c;
	uint32_t u;
	uint32_t v;
	uint64_t p;

	//Clear variables
	c = 0;
	u = 0;
	v = 0;

	//Perform multiplication
	for (i = 0; i < m; i++)
	{
	p = (uint64_t)a[i] * b;
	u = (uint32_t)p;
	v = (uint32_t)(p >> 32);

	u += c;
	if (u < c) v++;

	u += r[i];
	if (u < r[i]) v++;

	r[i] = u;
	c = v;
	}

	//Propagate carry
	for (; c != 0; i++)
	{
	r[i] += c;
	c = (r[i] < c);
	}
}



void BN_Dump(FILE * stream, const char * prepend, const BN * a)
{
	uint_t i;
```

```
		//Process each word
		for (i = 0; i < a->size; i++)
		{
		//Beginning of a new line?
		if (i == 0 || ((a->size - i - 1) % 8) == 7)
			fprintf(stream, "%s", prepend);

		//Display current data
		fprintf(stream, "%08X ", a->data[a->size - 1 - i]);

		//End of current line?
		if (((a->size - i - 1) % 8) == 0 || i == (a->size - 1))
			fprintf(stream, "\r\n");
		}
}

error_t BN_Import(BN * r, const uint8_t * data, uint_t length, BN_ENDIAN_FORMAT
format) {
		error_t error;
		uint_t i;

		//Check input format
		if (format == BN_LITTLE_ENDIAN)
		{
		//Skip trailing zeroes
		while (length > 0 && data[length - 1] == 0)
		{
			length--;
		}

		error = BN_Resize(r, (length + BN_INT_SIZE - 1) / BN_INT_SIZE);

		if (!error)
		{
			memset(r->data, 0, r->size * BN_INT_SIZE);
			r->sign = 1;

			//Import data
			for (i = 0; i < length; i++, data++)
			{
			r->data[i / BN_INT_SIZE] |= *data << ((i % BN_INT_SIZE) * 8);
			}
		}
```

```
        }
        else if (format == BN_BIG_ENDIAN)
        {
        //Skip leading zeroes
        while (length > 1 && *data == 0)
        {
                data++;
                length--;
        }

        error = BN_Resize(r, (length + BN_INT_SIZE - 1) / BN_INT_SIZE);

        if (!error)
        {
                memset(r->data, 0, r->size * BN_INT_SIZE);
                r->sign = 1;

                //Start from the least significant byte
                data += length - 1;

                //Import data
                for (i = 0; i < length; i++, data--)
                {
                r->data[i / BN_INT_SIZE] |= *data << ((i % BN_INT_SIZE) * 8);
                }
        }
        }
        else
        {
        error = ERROR_FAILED;
        }

        return error;
}
error_t BN_Import_Hex_String(BN * r, char * data, uint_t length,
BN_ENDIAN_FORMAT format) {
        error_t error;
        uint_t i;

        //Skip leading zeroes
        /*while (length > 1 && *data == 0)
        {
        data++;
```

```
        length--;
        }*/

        error = BN_Resize(r, (length + (2*BN_INT_SIZE) - 1) / (2*BN_INT_SIZE));

        if (!error)
        {
        memset(r->data, 0, r->size * BN_INT_SIZE);
        r->sign = 1;

        //Start from the least significant byte
        data += length - 1;

        //Import data
        for (i = 0; i < length/(BN_INT_SIZE*2); i++)
        {
                for (int a = 0; a < (2 * BN_INT_SIZE); a++) {
                r->data[i] |= HEX_REVERSE_SEQUENCE [*data] << (a*4);
                data--;
                }
        }
        }

        return error;
}

int BN_Is_Prime(BN * r) {

        uint_t order = BN_Get_Bit_Length(r);
        uint_t sqrt_order = ceil_div(order, 2);

        BN sqr;
        BN_Init_Stack(&sqr);

        for (int i = 0; i < sqrt_order; i++) {
        BN_Set_Bit_Value(&sqr, i, 1);
        }

        BN c;
        BN_Init_Stack(&c);
        BN_Set_Value(&c, 2);

        BN q;
```

```
        BN rem;
        BN_Init_Stack(&q);
        BN_Init_Stack(&rem);

        while (BN_Compare(&c, &sqr) <= 0) {
        BN_Div(&q, &rem, r, &c);

        if (BN_Compare_Int(&rem, 0) == 0) {
                free(sqr.data);
                free(c.data);
                free(q.data);
                free(rem.data);
                return 0;
        }

        BN_Add_Int(&c, &c, 1);
        }

        free(sqr.data);
        free(c.data);
        free(q.data);
        free(rem.data);

        return 1;
}
```

Code

DSA.h + DSA.c

```
#pragma once

#ifndef ___DSA_H

#define ___DSA_H


#include "../BNMath/BNMath.h"
```

```
#include "../../General/debug.h"


typedef BN* uint;


typedef BN DSA_Private_Key;

typedef BN DSA_Public_Key;


typedef struct {

        uint p; // Prime modulus

        uint q; // Order of the group

        uint G; // Group generator

} DSA_Domain_Parameters;


DSA_Domain_Parameters * params;


/*typedef struct {

        uint p;

        uint q;

        uint G;

        uint x; // Secret part of private key

} DSA_Private_Key;


typedef struct {

        uint p;
```

```
        uint q;
        uint G;
        uint y;
} DSA_Public_Key;*/

typedef struct {
        uint r;
        uint s;
} DSA_Signature;

typedef enum {
        SIGNATURE_VALID,
        SIGNATURE_INVALID
} SIGNATURE_VALID_STATE;

void DSA_Init_Public_Key(DSA_Public_Key * key);
void DSA_Free_Public_Key(DSA_Public_Key * key);
//void DSA_Load_Public_Key(DSA_Public_Key * key, uint p, uint q, uint G);
//void DSA_Load_Public_Key_From_Params(DSA_Public_Key ** key,
DSA_Domain_Parameters * params);
//void DSA_Load_Public_Key_Domain_Params(DSA_Public_Key * key,
DSA_Domain_Parameters * params);

void DSA_Init_Private_Key(DSA_Private_Key * key);
void DSA_Free_Private_Key(DSA_Private_Key * key);
```

```
//void DSA_Load_Private_Key(DSA_Private_Key * key, uint p, uint q, uint G);


error_t DSA_Create_Keys(DSA_Private_Key * priv_key, DSA_Public_Key * pub_key);


void DSA_Init_Signature(DSA_Signature ** signature);

void DSA_Free_Signature(DSA_Signature * signature);


void DSA_Generate_P(uint p, uint q, uint_t L);

void DSA_Generate_G(uint G, uint p, uint q, uint h);


error_t DSA_Generate_Signature(const DSA_Private_Key * key, const uint8_t *
message_digest, size_t message_len, DSA_Signature * signature);

SIGNATURE_VALID_STATE DSA_Verify_Signature(const DSA_Public_Key * key,
const uint8_t * message_digest, size_t message_len, DSA_Signature * signature);


#endif // !___DSA_H
```

```
#include "DSA.h"


void DSA_Init_Public_Key(DSA_Public_Key ** key) {

        BN_Init(key);

        /**key = (DSA_Public_Key*)malloc(sizeof(DSA_Public_Key));

        DSA_Public_Key * key_addr = *key;
```

```
        /*BN_Init(&(key_addr->p));

        BN_Init(&(key_addr->q));

        BN_Init(&(key_addr->G));

        BN_Init(&(key_addr->y));*/

}

/*void DSA_Load_Public_Key_From_Params(DSA_Public_Key ** key,
DSA_Domain_Parameters * params) {

        *key = (DSA_Public_Key*)malloc(sizeof(DSA_Public_Key));

        DSA_Public_Key * key_addr = *key;

        key_addr->p = params->p;

        key_addr->q = params->q;

        key_addr->G = params->G;

        BN_Init(&(key_addr->y));

}

void DSA_Load_Public_Key_Domain_Params(DSA_Public_Key * key,
DSA_Domain_Parameters * params) {

        if (key->p != NULL) { memcpy(key->p, params->p, params->p->size + 2); }

        else {

        key->p = params->p;

        }

        if (key->q != NULL) { memcpy(key->q, params->q, params->q->size + 2); }

        else {

        key->q = params->q;

        }

        if (key->G != NULL) { memcpy(key->G, params->G, params->G->size + 2); }
```

```
        else {
        key->G = params->G;
        }
}*/
void DSA_Free_Public_Key(DSA_Public_Key * key) {
        /*BN_Free(&(key->p));
        BN_Free(&(key->q));
        BN_Free(&(key->G));
        BN_Free(&(key->y));*/
        BN_Free(key);
}
/*void DSA_Load_Public_Key(DSA_Public_Key * key, uint p, uint q, uint G) {
        key->p = p;
        key->q = q;
        key->G = G;
}*/

void DSA_Init_Private_Key(DSA_Private_Key ** key) {
        BN_Init(key);
        /*
        *key = (DSA_Private_Key*)malloc(sizeof(DSA_Private_Key));
        DSA_Private_Key * key_addr = *key;
        BN_Init(&(key_addr->p));
        BN_Init(&(key_addr->q));
```

```
        BN_Init(&(key_addr->G));
        BN_Init(&(key_addr->x));*/
}
void DSA_Free_Private_Key(DSA_Private_Key * key) {
        /*BN_Free(&(key->p));
        BN_Free(&(key->q));
        BN_Free(&(key->G));
        BN_Free(&(key->x));*/
        BN_Free(key);
}
/*void DSA_Load_Private_Key(DSA_Private_Key * key, uint p, uint q, uint G) {
        key->p = p;
        key->q = q;
        key->G = G;
}*/

error_t DSA_Create_Keys(DSA_Private_Key * priv_key, DSA_Public_Key * pub_key) {
        BN * p;
        BN * q;
        BN * G;
        /*if (BN_Compare(priv_key->p, pub_key->p) != 0) { return ERROR_FAILED; }
        if (BN_Compare(priv_key->q, pub_key->q) != 0) { return ERROR_FAILED; }
        if (BN_Compare(priv_key->G, pub_key->G) != 0) { return ERROR_FAILED; }*/
```

```
p = params->p;

q = params->q;

G = params->G;


uint_t N;

uint_t L;


N = BN_Get_Bit_Length(q);

L = BN_Get_Bit_Length(p);


BN c1;

BN c;

BN_Init_Stack(&c1);

BN_Init_Stack(&c);


BN_Randomize(&c1, N + 64);


BN qt;

BN_Init_Stack(&qt);

BN_Sub_Int(&qt, q, 1);


BN_Mod(&c, &c1, &qt);

BN_Add_Int(priv_key, &c, 1);
```

```
        BN_Exp_Mod(pub_key, G, priv_key, p);


        free(c1.data);
        free(c.data);


        return ERROR_NONE;
}


void DSA_Init_Signature(DSA_Signature ** signature) {
        *signature = (DSA_Signature*)malloc(sizeof(signature));
        DSA_Signature * sig_addr = *signature;
        BN_Init(&(sig_addr->r));
        BN_Init(&(sig_addr->s));
}
void DSA_Free_Signature(DSA_Signature * signature) {
        BN_Free((signature->r));
        BN_Free((signature->s));
        //free(signature);
}


void DSA_Generate_P(uint p, uint q, uint_t L) {
        BN_Add_Int(p, q, 1);


        int_t m = 2;
```

```
        while (!BN_Is_Prime(p)) {
        BN_Mul_Int(p, q, m);
        BN_Add_Int(p, p, 1);
        m++;
        }
}

void DSA_Generate_G(uint G, uint p, uint q, uint h) {
        BN * e;
        BN_Init(&e);
        BN * r;
        BN_Init(&r);

        BN * t; BN_Init(&t);
        BN_Sub_Int(t, p, 1);

        BN_Div(e, r, t, q);

        do {
        BN_Exp_Mod(G, h, e, p);
        BN_Add_Int(h, h, 1);
        } while (!BN_Compare_Int(G, 1));
```

```
        BN_Free(e);

        BN_Free(t);

        BN_Free(r);

}


error_t DSA_Generate_Signature(const DSA_Private_Key * key, const uint8_t *
message_digest, size_t message_len, DSA_Signature * signature) {

        error_t error = 0;

        uint_t n;

        BN k;

        BN z;


        //Check parameters

        if (key == NULL || message_digest == NULL || signature == NULL) {

        return ERROR_FAILED;

        }


        //Debug message

        TRACE_DEBUG("DSA signature generation...\r\n");

        TRACE_DEBUG("  p:\r\n");

        TRACE_DEBUG_MPI("      ", (key->p));

        TRACE_DEBUG("  q:\r\n");

        TRACE_DEBUG_MPI("      ", (key->q));

        TRACE_DEBUG("  g:\r\n");

        TRACE_DEBUG_MPI("      ", (key->G));
```

```
TRACE_DEBUG("  x:\r\n");
TRACE_DEBUG_MPI("       ", (key->x));
TRACE_DEBUG("  digest:\r\n");
TRACE_DEBUG_ARRAY("", message_digest, message_len);

//Initialize multiple precision integers
BN_Init_Stack(&k);
BN_Init_Stack(&z);

//Let N be the bit length of q
n = BN_Get_Bit_Length((params->q));

//Compute N = MIN(N, outlen)
n = min(n, message_len * 4);

//Convert the digest to a multiple precision integer
BN_Import_Hex_String(&z, message_digest, (n + 3) / 4, BN_BIG_ENDIAN);

//Keep the leftmost N bits of the hash value
if ((n % 8) != 0)
{
BN_Shift_Right(&z, 8 - (n % 8));
}
```

```
//Debug message
TRACE_DEBUG("  z:\r\n");
TRACE_DEBUG_MPI("       ", &z);



do {


//Generated a pseudorandom number
BN_Randomize(&k, n);


//Make sure that 0 < k < q
if (BN_Compare(&k, (params->q)) >= 0)
        BN_Shift_Right(&k, 1);


//Compute r = (g ^ k mod p) mod q
BN_Exp_Mod((signature->r), (params->G), &k, (params->p));
BN_Mod((signature->r), (signature->r), (params->q));


//Compute k ^ -1 mod q
BN_Inv_Mod(&k, &k, params->q);


//Compute s = k ^ -1 * (z + x * r) mod q
BN_Mul((signature->s), (key), (signature->r));
BN_Add((signature->s), (signature->s), &z);
```

```
        //BN_Mod((signature->s), (signature->s), (key->q));
        BN_Mul_Mod((signature->s), (signature->s), &k, (params->q));
        } while ((BN_Compare_Int(signature->r, 0) == 0 || BN_Compare_Int(signature->s,
0) == 0));


        //Dump DSA signature
        TRACE_DEBUG("  r:\r\n");
        TRACE_DEBUG_MPI("      ", (signature->r));
        TRACE_DEBUG("  s:\r\n");
        TRACE_DEBUG_MPI("      ", (signature->s));


        free(k.data);
        free(z.data);


        //Clean up side effects if necessary
        if (error)
        {
        //Release (R, S) integer pair
        BN_Free((signature->r));
        BN_Free((signature->s));
        }


        return error;
}
error_t DSA_Verify_Signature(const DSA_Public_Key * key, const uint8_t *
```

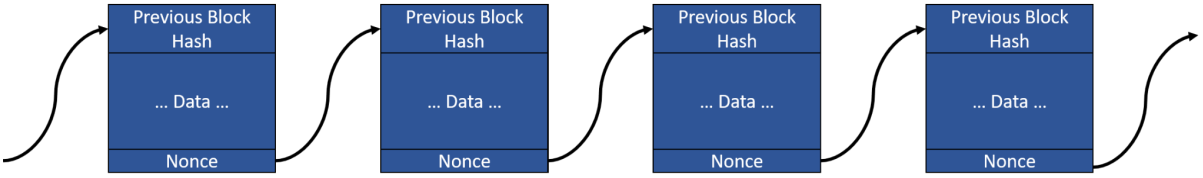
```
message_digest, size_t message_len, DSA_Signature * signature) {


        BN * r = signature->r;

        BN * s = signature->s;


        BN * p = params->p;

        BN * q = params->q;

        BN * G = params->G;

        BN * y = key;


        if (!(BN_Compare_Int(r, 0) > 0 && BN_Compare(r, q) < 0)) { return
SIGNATURE_INVALID; }

        if (!(BN_Compare_Int(s, 0) > 0 && BN_Compare(s, q) < 0)) { return
SIGNATURE_INVALID; }


        BN w;

        BN z;

        BN u1;

        BN u2;

        BN v;

        BN_Init_Stack(&w);

        BN_Init_Stack(&z);

        BN_Init_Stack(&u1);

        BN_Init_Stack(&u2);

        BN_Init_Stack(&v);
```

```
BN_Inv_Mod(&w, s, q);



uint_t n = BN_Get_Bit_Length(q);
n = min(n, message_len * 4);
BN_Import_Hex_String(&z, message_digest, (n + 3) / 4, BN_BIG_ENDIAN);
if ((n % 8) != 0)
{
BN_Shift_Right(&z, 8 - (n % 8));
}



BN_Mul_Mod(&u1, &z, &w, q);
BN_Mul_Mod(&u2, r, &w, q);

BN tmp1;
BN tmp2;
BN_Init_Stack(&tmp1);
BN_Init_Stack(&tmp2);

BN_Exp_Mod(&tmp1, G, &u1, p);
BN_Exp_Mod(&tmp2, y, &u2, p);
```

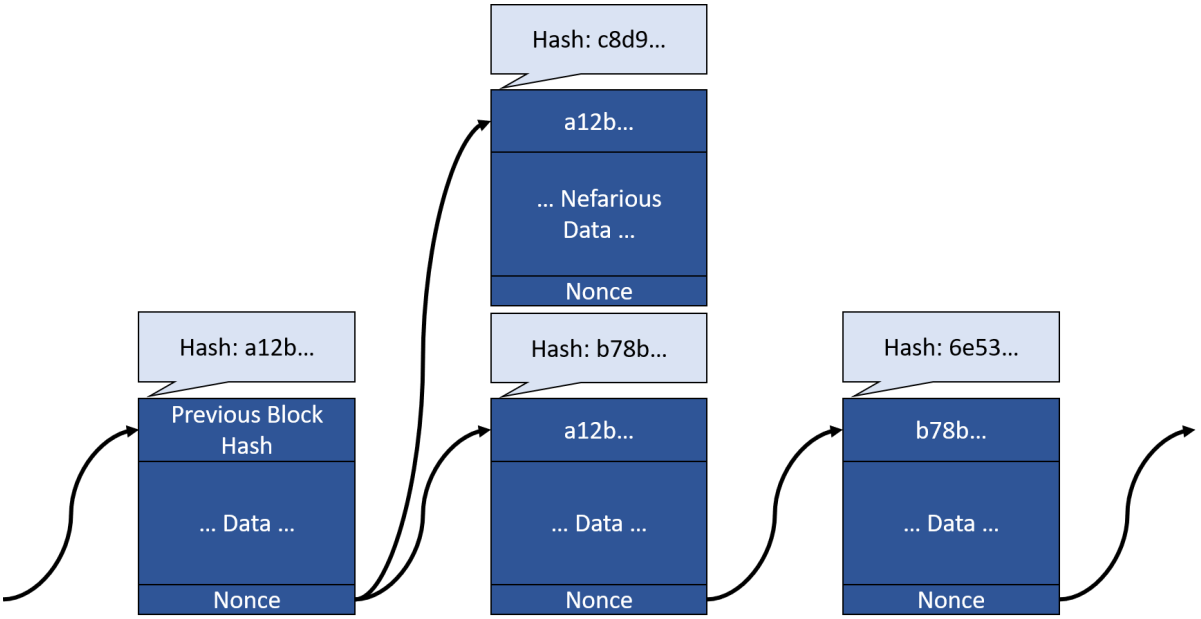
```
        BN_Mul_Mod(&v, &tmp1, &tmp2, p);
        BN_Mod(&v, &v, q);

        free(tmp1.data);
        free(tmp2.data);

        TRACE_DEBUG("  v:\r\n");
        TRACE_DEBUG_MPI("       ", &v);

        int valid = (BN_Compare(&v, r) == 0);

        free(w.data);
        free(z.data);
        free(u1.data);
        free(u2.data);
        free(v.data);

        if (valid) {
        return SIGNATURE_VALID;
        }

        return SIGNATURE_INVALID;
}
```

BlockChain

Blockchain technology is the backbone of all cryptocurrencies. A blockchain is a growing set of blocks linked together using hash algorithms. The purpose of this linkage is to avoid the situation where a malicious party injects a fraudulent block into the history of blocks.



Now assume a hacker did try and fake a block. Adding the block at the head of the chain would be practically impossible as all connected nodes will reject the block and not add it. However, if the hacker can insert the block into the middle of the chain, some nodes might skip checking the transactions and just accept the block if the hash chain matches.



Here it is clear that while a hacker can use the previous block hash, it would be virtually impossible to match the hash of the nefarious block's hash to the real block's hash. Therefore, this new block will create an entire new chain, which will necessarily be shorter, or simply have less computational value, and thus be less valuable than the real blockchain. Hence, the community will easily be able to differentiate between the real chain and the fake one, and uphold the authenticity of the blockchain.

BlockChain is useful since it essentially splits a very complicated problem into smaller easier problems; instead of the problem of verifying that every transaction ever made is valid, just verify this set of transactions, and then have a notary sign on the set as a whole and call it a block. Now instead of verifying each transaction, an entire set can be verified at once as a block.

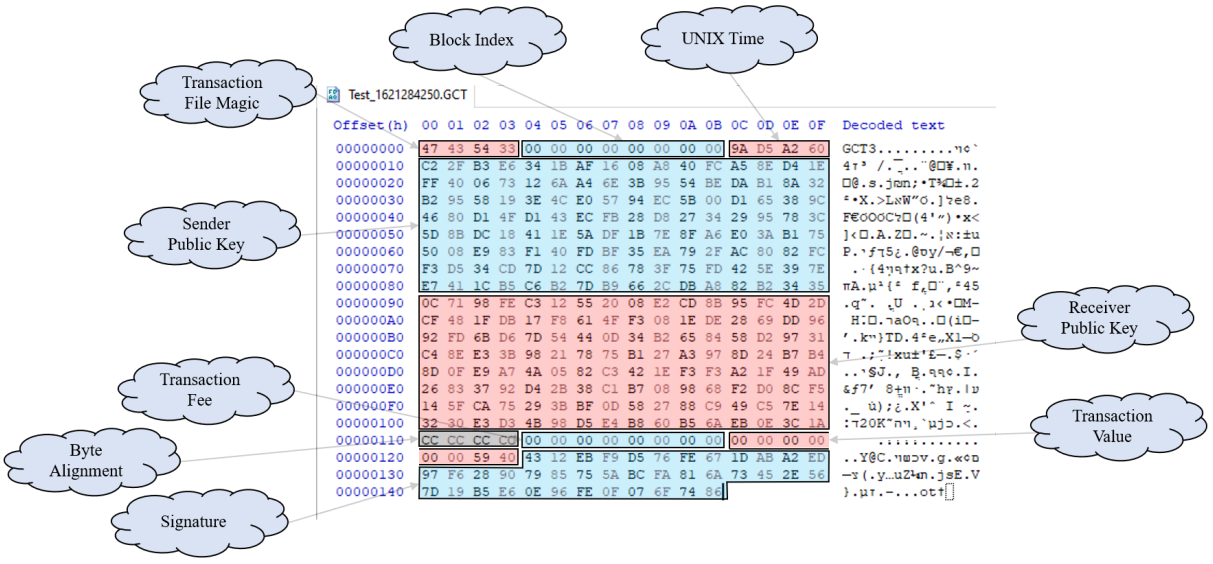
Transactions

Transactions are the actual register of the exchange of value on the blockchain. Each transaction will hold the information about the sender, receiver, the value of the transaction, some metadata, and a signature - similar to a common receipt. A group of transactions, in the case of GreenCoin - 64, will be written onto a block at which point a notary will verify the whole set and sign them off. Each transaction is completely unique as the transaction must include the time it was signed. This facilitates easy counterfeiting handling, as the second occurrence of the exact same transaction will necessarily be fraudulent. To make this verification even faster, the transaction metadata also includes the block it is on, and therefore a transaction can only be duplicated on the same block. Thus, a notary will verify that there are no duplicate transactions on a block, and if there are, will take only one.

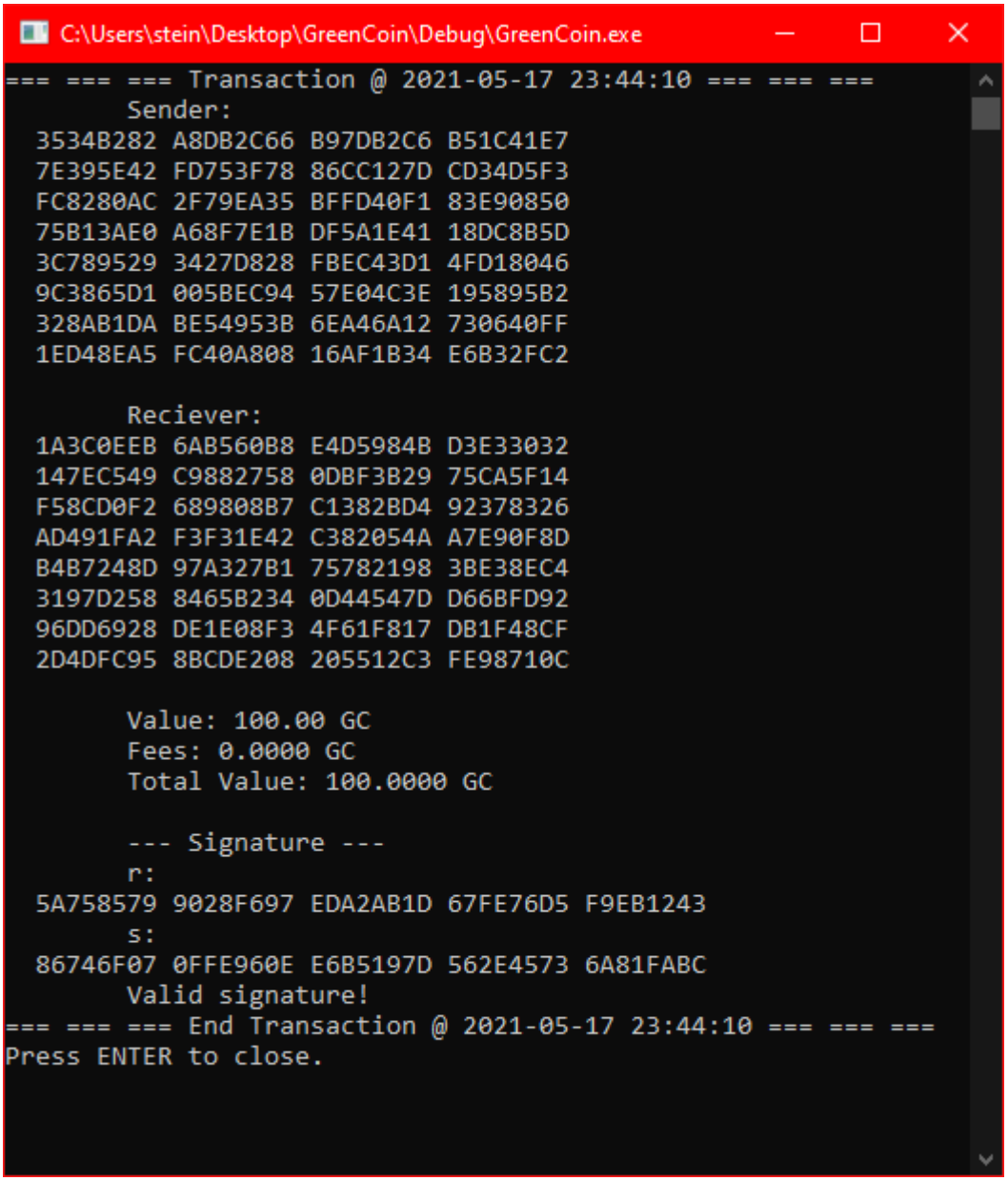
A transaction, including its metadata, has the following structure:

Byte Size	Data Name	Description
8	Block Index	The index of the block the transaction is destined for.
4	Time (UNIX)	The time the transaction was signed.
128	Sender	Public key of the sender.
128	Receiver	Public key of the receiver.
8	Fee	The fee to be collected by the notary.
8	Value	The value to be passed to the receiver. Note: Receiver += Value; Sender -= (Value + Fee);
40	Signature	The DSA signature (r, s).

An actual exported transaction will look like this in hexadecimal representation:



And the actual GreenCoin program will show a preview of the transaction in the form of a receipt like this:



A transaction will take up exactly 328 bytes of data (4 of which are alignment bytes), and thus the cumulative size of all transactions on a block should be $328 \cdot 64 = 20,992$ bytes. An exported transaction, meaning a transaction saved to a file or broadcasted on the network, has 4 added bytes which are the magic. The magic allows the program to differentiate between data type stored in files, or on the network, by having the first two bytes be the ASCII values of “GC” (GreenCoin), then one byte about the type itself - a transaction having the code “T” (ASCII 84 \ 0x54) - and a final reserved byte which can be used in the future as a version indicator, though it is currently the char “3” (ASCII 51 \ 0x33).

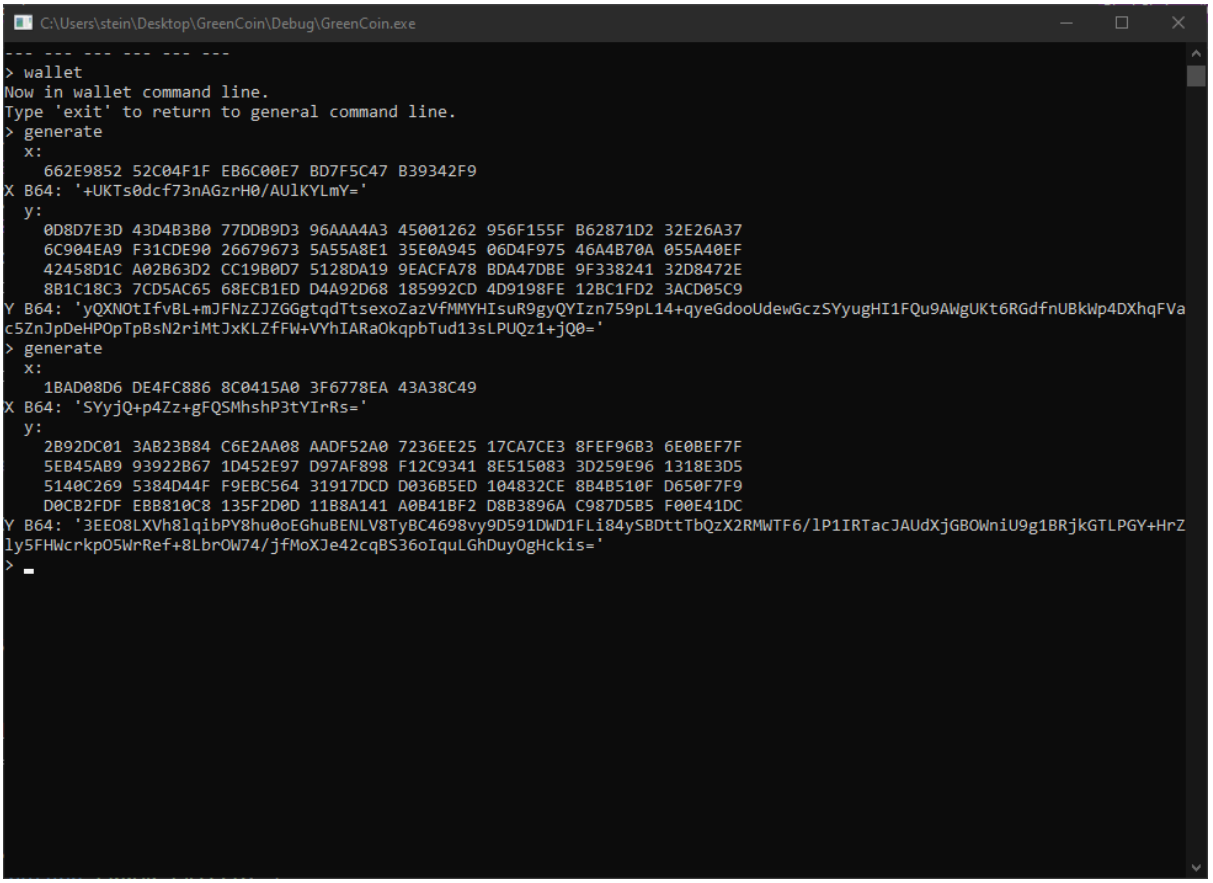
Signing & Verification

The core piece of each transaction is its signature. The signature allows anyone who stumbles upon the transaction to verify its authenticity. The signature is of course a DSA signature as described here. The signature is quickly checked each time a transaction is previewed, and a corresponding message will appear after the (r, s) signature pair at the bottom of the transaction.

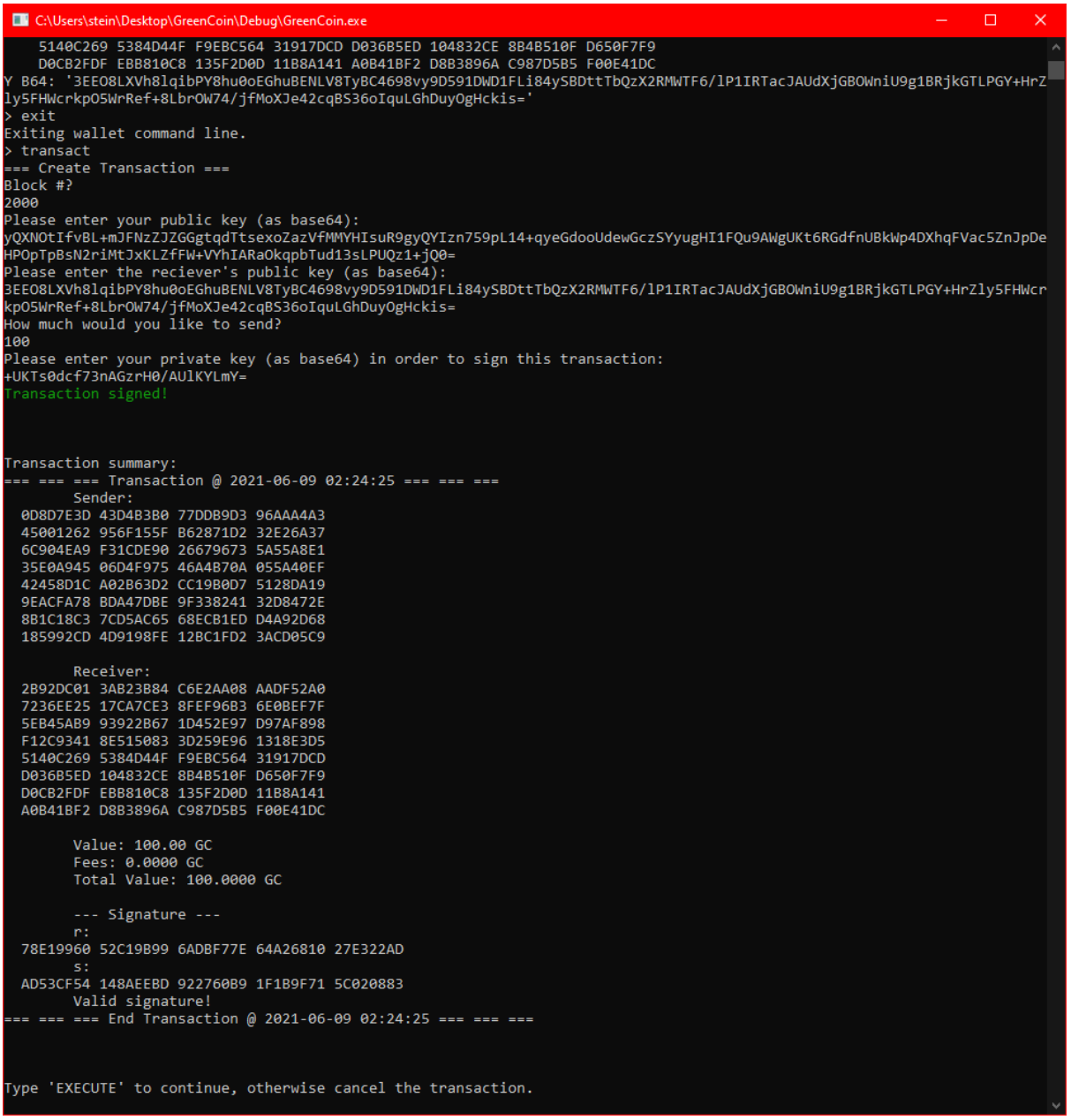
In order to match and verify a signature, the public key of the sender must match the private key used to sign the transaction. This means that the person who's coins are being transferred from them has to also be the one to sign the transaction. The purpose of this is obviously to avoid coin theft.

From the user's side, signing is simply entering the private key which matches the public key.

Here is an example of a transaction being created, which will send 100GC from Alice's wallet (yQXNOtIfvBL+mJFNzZJZGGgtqdTtsexoZazVfMMYHIsuR9gyQYIzn759pL14+qyeGdooUdewGczSYyugHI1FQu9AWgUKt6RGdfnUBkWp4DXhqFVac5ZnJpDeHPOpTpBsN2riMtJxKLZfFW+VYhIARaOkqpbTud13sLPUQz1+jQ0=) to Bob's wallet (3EEO8LXVh8lqibPY8hu0oEGhuBENLV8TyBC4698vy9D591DWD1FLi84ySBDttTbQzX2RMWTF6/lP1IRTacJAUdXjGBOWniU9g1BRjkGTLPGY+HrZly5FHWcrkpO5WrRef+8LbrOW74/jfMoXJe42cqBS36oIquLGhDuyOgHckis=). Generating these two wallets right now to demonstrate this, as one should never publish or share their private key:



So Alice's private key is "+UKTs0dcf73nAGzrH0/AUlKYLmY=".



Signing

Signing the transaction simply means taking the data of the transaction, not including the signature:

```
char * transaction_info = (char*)malloc(sizeof(_Transaction) - sizeof(_Signature));

memcpy(transaction_info, transaction, sizeof(_Transaction) - sizeof(_Signature));
```

And producing a hash digest of this data. Doing so will entangle the signature to the transaction's actual data, which protects the signature from being tacked onto other transactions which might be fraudulent.

```
char * message_digest = Hash_SHA256(transaction_info, sizeof(_Transaction) - sizeof(_Signature));
```

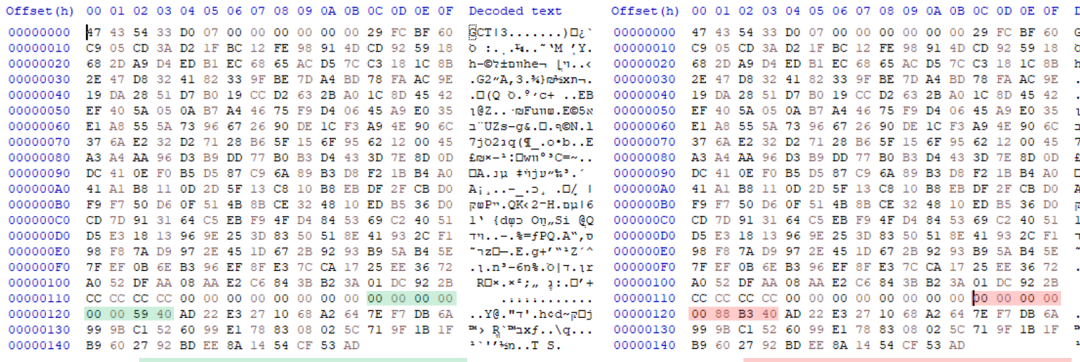
Now we simply use our DSA signing function to produce the (r, s) signature pair and tack it onto the transaction.

Verification

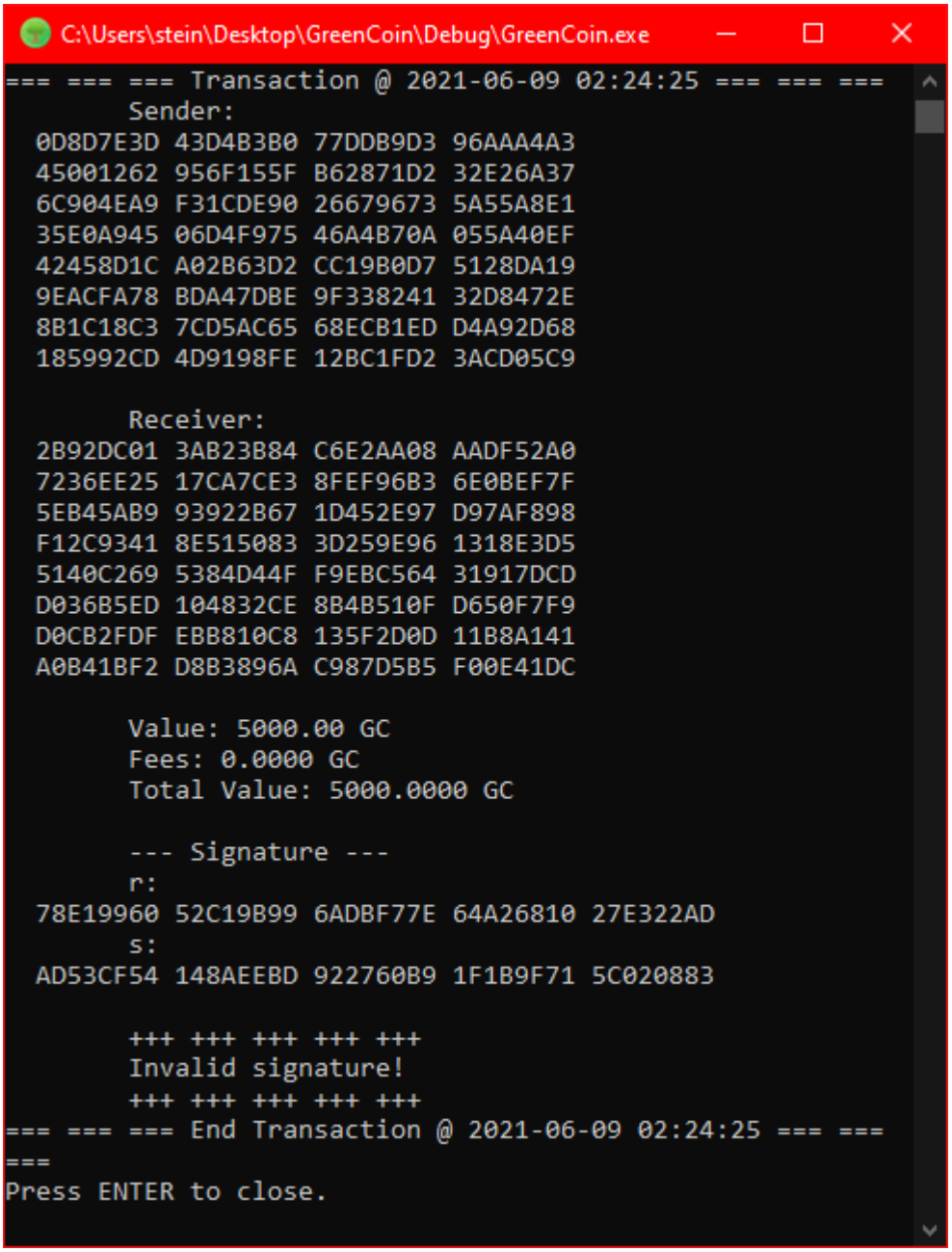
Verifying a transaction is even easier. All we need is the transaction data, including the signature. Then we perform the same operations, calculate the hash digest of the transaction data (not including the signature). This time, since we are verifying, we also deduce the public key of the signer from the transaction - this is simply the wallet address of the sender. We also extract the DSA signature pair (r, s) from the transaction and send everything off to the DSA signature verification function, which will return a SIGNATUE_VALID_STATE of either valid or invalid.

Notice that verifying a transaction only checks its authenticity, meaning it only checks that the sender signed the transaction. Verification regarding funds is done by notaries and is part of block verification - the blockchain.

If a transaction was tampered with, the verification will catch the tamper. Assume after Alice sends the transaction, Bob tries to mess with it and changes the value of 100GC to 5,000GC.



The signature itself was really generated by Alice and matches her private-public key pair. However, since the signature is dependant on the hash of the transaction, the following occurs:



The GreenCoin program recognizes that the transaction is fraudulent and rejects the signature.

This is due to the fact that even such a small change, only two bytes, changed the hash from
“3575ce08858deb9c469e3b34cb2fa5e95247827aae650d0fca770e8642cc7584” to
“79dae68885019d7fbf26784c91d1fe130bd9a7da0138b07db9e96f0474cb6f99”.

Code

Transaction.h + Transaction.c

```
#pragma once

#ifndef __TRANSACTION_H
#define __TRANSACTION_H

#include "../Wallet/Wallet.h"

typedef struct _Signature {
	uint_t r[5];
	uint_t s[5];
} _Signature;

typedef struct _Transaction {
	// Index of the block in the chain
	uint64_t Block_Index;

	/* DEPRECATED
	// Index of the transaction on the ledger
	uint32_t Index;
	*/

	// The time this transaction was signed.
	uint32_t Time;
```

```
        /*
        The combination of the block_index and the index are completely unique in the chain,
         and therefore protect the signature from being compied either again on the same block
         or at the same position on a different block.
        */

        // Address of the sender
        _Wallet_Address Sender;

        // Address of the receiver
        _Wallet_Address Receiver;

        // Fees - This can currently be disregarded
        double Fee;

        // Value of the transaction in GreenCoins
        double Value;

        // Signature in order to validate sender
        _Signature Signature;
} _Transaction;
void Print_Transaction_Signature_Part(FILE * fstream, uint_t * q);
void Print_Transaction_Wallet_Address(FILE * fstream, uint_t * q);
```

```
void Print_Transaction(FILE * fstream, _Transaction * transaction);


void Transaction_Export(FILE * fstream, _Transaction * transaction);

void Transaction_Export_To_File(char * file_path, _Transaction * transaction);


void Sign_Transaction(_Transaction * transaction, BN * priv_key);


// Returns the value of the transaction with regards to a wallet.

// If the wallet was the sender, deduce the TOTAL value of the transaction.

// If the wallet was the receiver, add the value of the transaction.

double Calculate_Transaction_Change_To_Wallet(_Transaction * transaction,
_Wallet_Address pk);


SIGNATURE_VALID_STATE Verify_Transaction(_Transaction * transaction);


void Transaction_Demo(void * wsadata, void * socket);


#endif // !__TRANSACTION_H
```

```
#include "Transaction.h"


#include "../General/Print/PrettyPrint.h"
```

```
void Print_Transaction_Signature_Part(FILE * fstream, uint_t * q) {
	for (int i = 4; i >= 0; i--) {
	fprintf(fstream, "%.8X ", q[i]);
	}
	fprintf(fstream, "\n");
}
void Print_Transaction_Wallet_Address(FILE * fstream, uint_t * q) {
	fprintf(fstream, "  ");
	for (int i = 31; i >= 0; i--) {
	fprintf(fstream, "%.8X ", q[i]);
	if (i % 4 == 0) { fprintf(fstream, "\n  "); }
	}
	fprintf(fstream, "\n");
}
void Print_Transaction(FILE * fstream, _Transaction * transaction) {
	fprintf(fstream, "=== === === Transaction @ %s === === ===\n",
HumanFormatDateTimeInt(transaction->Time));
	fprintf(fstream, "\tSender: \n");
	Print_Transaction_Wallet_Address(fstream, transaction->Sender);
	fprintf(fstream, "\tReciever: \n");
	Print_Transaction_Wallet_Address(fstream, transaction->Receiver);
	fprintf(fstream, "\tValue: %.2f GC\n", transaction->Value);
	fprintf(fstream, "\tFees: %.4f GC\n", transaction->Fee);
	fprintf(fstream, "\tTotal Value: %.4f GC\n", transaction->Value + transaction->Fee);
	fprintf(fstream, "\n\t--- Signature ---\n");
```

```
        fprintf(fstream, "\tr:\n  ");
        Print_Transaction_Signature_Part(fstream, transaction->Signature.r);
        fprintf(fstream, "\ts:\n  ");
        Print_Transaction_Signature_Part(fstream, transaction->Signature.s);


        DSA_Public_Key * pub_key=0;
        DSA_Init_Public_Key(&pub_key);
        //DSA_Load_Public_Key(pub_key, params->p, params->q, params->G);
        BN_Resize(pub_key, 32);
        memcpy(pub_key->data, transaction->Sender, 32 * sizeof(uint_t));


        DSA_Signature * sign;
        DSA_Init_Signature(&sign);
        BN_Resize(sign->r, 5);
        BN_Resize(sign->s, 5);
        memcpy(sign->r->data, transaction->Signature.r, 5 * sizeof(uint_t));
        memcpy(sign->s->data, transaction->Signature.s, 5 * sizeof(uint_t));


        char * digest = Hash_SHA256(transaction, sizeof(_Transaction) -
sizeof(_Signature));


        SIGNATURE_VALID_STATE valid = DSA_Verify_Signature(pub_key, digest, 64,
sign);
        if (valid == SIGNATURE_VALID) {
        fprintf(fstream, "\tValid signature!\n");
```

```
	}

	else {

	fprintf(fstream, "\n\t+++ +++ +++ +++ +++\n\tInvalid signature!\n\t+++ +++ +++ +++ +++\n");

	}



	fprintf(fstream, "=== === === End Transaction @ %s === === ===\n", HumanFormatDateTimeInt(transaction->Time));

}


void Transaction_Export(FILE * fstream, _Transaction * transaction) {

	fwrite(GCT_MAGIC, 1, 4, fstream);

	fwrite(transaction, sizeof(_Transaction), 1, fstream);

}


void Transaction_Export_To_File(char * file_path, _Transaction * transaction) {

	FILE * ft;

	fopen_s(&ft, file_path, "wb");


	// Write magic

	fwrite(GCT_MAGIC, 1, 4, ft);


	// Write transaction data

	fwrite(transaction, sizeof(_Transaction), 1, ft);
```

```
	fclose(ft);
}

void Sign_Transaction(_Transaction * transaction, BN * pk) {
	char * transaction_info = (char*)malloc(sizeof(_Transaction) - sizeof(_Signature));
	memcpy(transaction_info, transaction, sizeof(_Transaction) - sizeof(_Signature));

	//char * message = "abc";
	char * message_digest = Hash_SHA256(transaction_info, sizeof(_Transaction) - sizeof(_Signature));//"ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad";

	DSA_Signature * signature = 0;
	DSA_Init_Signature(&signature);

	DSA_Private_Key * priv_key;
	priv_key = pk;

	DSA_Generate_Signature(priv_key, message_digest, 64, signature);

	_Signature sig;
	memcpy((sig.r), signature->r->data, 5 * sizeof(uint_t));
	memcpy((sig.s), signature->s->data, 5 * sizeof(uint_t));
```

```
        memcpy(&(transaction->Signature), &sig, sizeof(_Signature));


        free(transaction_info);

        //DSA_Free_Private_Key(priv_key);
}


SIGNATURE_VALID_STATE Verify_Transaction(_Transaction * transaction) {
        DSA_Public_Key * pub_key = 0;
        DSA_Init_Public_Key(&pub_key);
        BN_Resize(pub_key, 32);
        memcpy(pub_key->data, transaction->Sender, 32 * sizeof(uint_t));


        DSA_Signature * sign;
        DSA_Init_Signature(&sign);
        BN_Resize(sign->r, 5);
        BN_Resize(sign->s, 5);
        memcpy(sign->r->data, transaction->Signature.r, 5 * sizeof(uint_t));
        memcpy(sign->s->data, transaction->Signature.s, 5 * sizeof(uint_t));


        char * digest = Hash_SHA256(transaction, sizeof(_Transaction) -
sizeof(_Signature));


        SIGNATURE_VALID_STATE valid = DSA_Verify_Signature(pub_key, digest, 64,
sign);
```

```
	DSA_Free_Public_Key(pub_key);

	DSA_Free_Signature(sign);


	return valid;

}


double Calculate_Transaction_Change_To_Wallet(_Transaction * transaction,
_Wallet_Address pk) {

	if (memcmp(transaction->Sender, pk, 32 * sizeof(uint_t)) == 0) {

	return -(transaction->Value + transaction->Fee);

	}

	else if (memcmp(transaction->Receiver, pk, 32 * sizeof(uint_t)) == 0) {

	return transaction->Value;

	}

	else {

	return 0;

	}

	return 0;

}


void Transaction_Demo(void * wsadata, void * socket) {

	printf("=== Create Transaction ===\n");


	_Transaction transaction;
```

```
printf("Block #?\n");
char buffer[64]; fgets(buffer, 64, stdin);
transaction.Block_Index = strtol(buffer, NULL, 10);

/*printf("Transaction #?\n");
fgets(buffer, 64, stdin);
transaction.Index = strtol(buffer, NULL, 10);*/

printf("Please enter your public key (as base64): \n");
char sender_64[180]; fgets(sender_64, 180, stdin);

printf("Please enter the receiver's public key (as base64): \n");
char reciever_64[180]; fgets(reciever_64, 180, stdin);

printf("How much would you like to send?\n");
fgets(buffer, 64, stdin);
double value = strtod(buffer, NULL);
transaction.Value = value;
transaction.Fee = 0;

byte * sender;
byte * receiver;
B64_Decode(sender_64, &sender);
B64_Decode(reciever_64, &receiver);
```

```
        memcpy(transaction.Sender, sender, sizeof(_Wallet_Address));
        memcpy(transaction.Receiver, receiver, sizeof(_Wallet_Address));

        printf("Please enter your private key (as base64) in order to sign this transaction:
\n");
        char priv_key_64[180]; fgets(priv_key_64, 180, stdin);
        byte * priv_k; size_t priv_key_byte_length = B64_Decode(priv_key_64, &priv_k);
        BN * pk; BN_Init(&pk);
        BN_Resize(pk, 16);
        pk->sign = 1;
        memcpy(pk->data, priv_k, priv_key_byte_length);

        transaction.Time = time(NULL);

        Sign_Transaction(&transaction, pk);

        printf_Success("Transaction signed!\n");

        printf("\n\n\n");
        printf("Transaction summary: \n");

        Print_Transaction(stderr, &transaction);
        printf("\n\n\n");
```

```
        printf("Type 'EXECUTE' to continue, otherwise cancel the transaction.\n");


        char * EXECUTE = "EXECUTE";

        fgets(buffer, 64, stdin); buffer[strlen(EXECUTE)] = 0x0;


        if (strcmp(buffer, EXECUTE) == 0) {

        printf_Success("Transaction executed!\n");


        char export_path[256];

        sprintf_s(export_path, 256, "Demo\\Transaction_%u.GCT", transaction.Time);


        printf_Info("Now exporting to: '%s'\n", export_path);


        Transaction_Export_To_File(export_path, &transaction);


        Network_Broadcast_Transaction(wsadata, socket, &transaction,
sizeof(_Transaction));

        }

        else {

        printf_Info("Transaction has been cancelled!\n");

        }

}
```

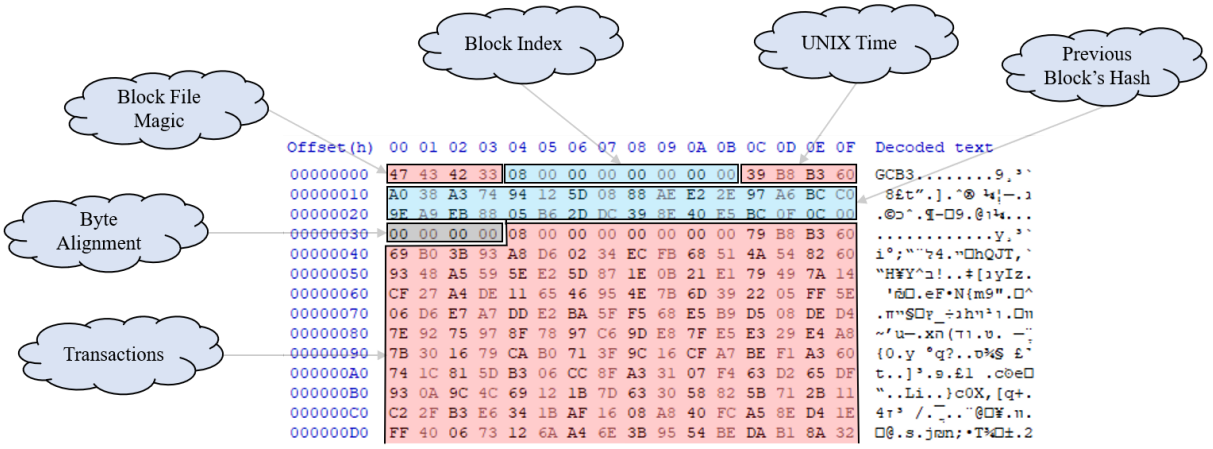
Blocks

Blocks are what makes up the blockchain. These blocks each hold a set of transactions and a verification that the transactions are valid. The blocks themselves do not contain any complicated functions, they are simply the external packaging of the transactions. Each block will of course contain 64 transactions, some metadata, and verification information - the nonce. This nonce is a 256 bit number which is chosen in order to produce a hash digest for the block which matches some criteria. Blocks are of course indexed starting at 0 and can go as high as a uint64_t ($2^{64} - 1 = 18,446,744,073,709,551,615$). The physical layout of a single block is as follows:

Byte Size	Data Name	Description
8	Block Index	The index of the block.
4	Time (UNIX)	The time the block was created.
32	Previous Block's Hash	The SHA-256 hash of the previous block.
328 * 64	Transactions	An array of 64 transactions.
128	Notary Address	The address of the block's verifier's wallet.
32	Nonce	A 256 bit value which is changed in order to produce the desired hash digest for the block.

Which produces a total of 21,196 bytes + 4 extra alignment bytes. Meaning that a block's file, which also has the magic "GCB3" (GreenCoin Block v3), is 21,204 bytes in size.

In hexadecimal view, the header of a block looks like this:



After the array of transactions, the block contains a notary address, which is the public key of the user who verified the transactions on the block and signed it - hence the name notary, and a nonce. The nonce is the actual "signing" of the block - see more in block mining below. The address which is written on the block as the notary will collect the sum of the transaction fees on the block and a bonus newly minted set of coins. This is the compensation given to a block verifier for their work in verifying the transactions, and is the economic incentive for people to mine new blocks.

The protection of a block comes from a combination of factors:

1. The block index.
    a. The unique index of a block allows for quick checking of whether a block with the same index already exists. In the case this occurs, there is a split in the blockchain, assuming both blocks are valid, and eventually the one which is longer will be chosen, and the other will be forgotten and all transactions on it invalid.
    b. This unique index also allows for checking of transactions and making sure that only transactions destined for the block are on it. This eliminates the possibility of duplicating a transaction from an old block to the new one, as the block indices will necessarily not match.
    c. The index also allows us to verify that the order of the blockchain is valid, and that the chain of hashes matches the indices. This is crucial, as it makes the challenge of injecting a fraudulent block into the chain exponentially more difficult.
2. The block creation time. The block creation time, other than being useful information, is used in a few ways:
    a. Creation time will aid in the verification of the order of the chain, as a block must have been made after the block before it, otherwise someone could pre-make future blocks with fake transactions.
    b. Creation time can also be used to verify that all transactions on the block were signed within reasonable time of the block being created. As there might be a backlog of transactions, they will not necessarily have been signed after the block's creation. However, if a transaction was signed a significant enough time ago, several blocks, and newer transactions (transactions with a more up-to-date time) have been verified, the transaction is nullified and invalid.
    c. The creation time also adds some variability to blocks which will make the hash digests produced more random.
3. The previous block's hash.
    a. Adding the previous block's hash eliminates the possibility of someone signing a block in the future, as the hash will change according to the previous block' hash. This improves security in the way that a single entity with minimal computing power could not spend much time pre-making a future block, since they would necessarily need the hash of the preceding block.
    b. Having the previous block's hash in the next block links the two blocks together directly, which protects the chain from a block being inserted in between them.
    c. The previous block's hash can also be used to recursively verify the entire chain quickly by hashing all the blocks and seeing that the blockchain matches all the hashes in the exact order.

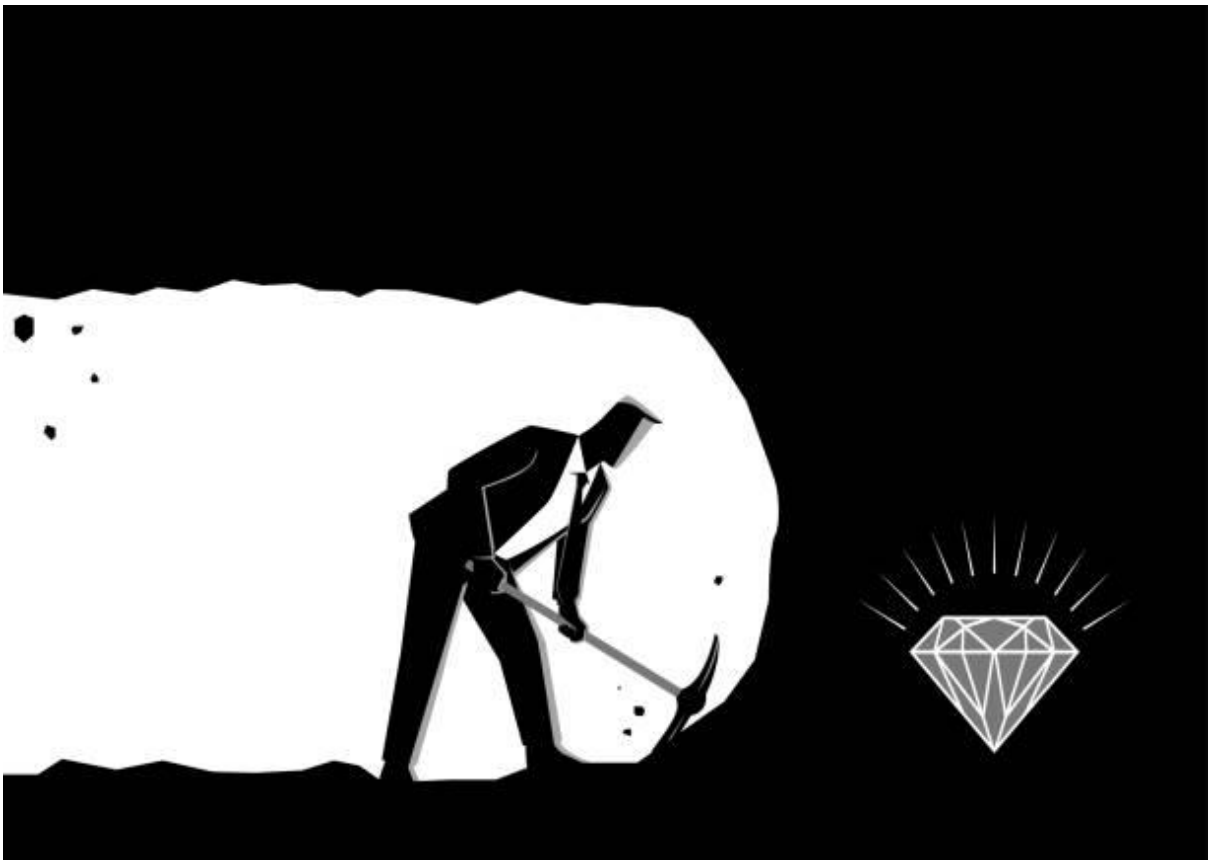
In addition to these protection methods, each block is also credited with a security strength. This strength depends on the block's hash and is an approximate measure of how much computational effort and power was dedicated to creating the block. The function to calculate the block's strength is proprietary and might be improved in the future. In short, the function counts the amount of leading zeros in the hash, and takes two to the power of that amount and subtracts one.

```
uint64_t Calculate_Block_Strength(_Block * block) {
        char * digest = Hash_SHA256(block, sizeof(_Block));

        int i = 0;
        while (digest[i] == '0') { i++; }

        free(digest);

        return (uint64_t)(pow(2, i) - 1);
}
```

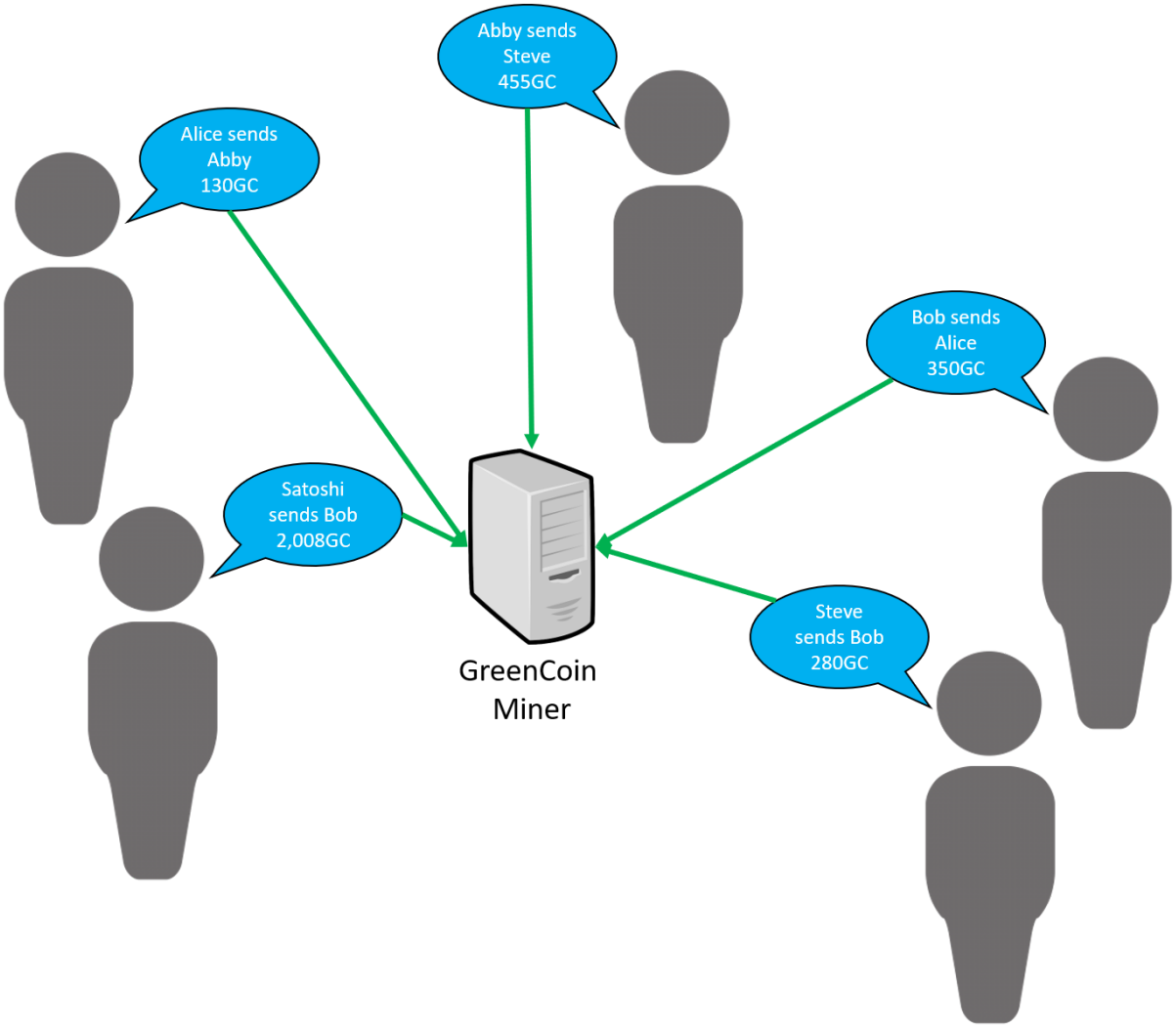
Take 'n' to be the amount of preceding zeros: $S = 2^n - 1$. This means that a block which just has a random hash, say: "ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb", will have strength: $S = 2^n - 1 = 2^0 - 1 = 0$, since absolutely no extra work was put into generating this hash. A hash which had some work put into it, say "00978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb", will have an exponentially higher strength: $S = 2^n - 1 = 2^2 - 1 = 3$. The more leading zeros, the more effort was probably put into generating the hash, the stronger the block. The idea behind this is to make it "cost" a block verifier something to publish a block, since the network will only accept blocks with a strength greater than some threshold. This decreases a hacker's motive to publish fake blocks, since they would need to spend significant power and energy to generate a good enough hash digest, and if they are caught, all that work is wasted. Therefore, it is more worthwhile to follow the rules of the network and generate proper blocks.

Mining

Mining cryptocurrencies is a core aspect of blockchain. “Mining” is actually the action of verifying blocks, and finding a nonce which produces a strong enough block. This action is called “mining” since it is similar to mining diamonds; a great amount of time and energy is spent searching for the diamond, and with some luck, a great reward is given. Miners compete to find a nonce which produces a strong enough block, and the first to find one is awarded all the fees on the block, and some additional newly minted coins.



A miner will first listen for transactions occurring on the network and save valid ones. Once the miner has accumulated enough transactions, 64 for GreenCoin, the miner will start to guess 256 bit numbers to append to the block in order to make its hash have the most leading zeros. The guessing and hashing part of mining takes a long time since it is computationally expensive, and requires quite a bit of electricity. This “cost” is placed in order to offset the time spent verifying the transactions, as it is pointless for a miner to skip checking the transactions and only guess hashes, since there is a good chance their block will be rejected, and then all the time and energy they spent will be lost.



The miner will verify each and every transaction they recieve by checking the following:

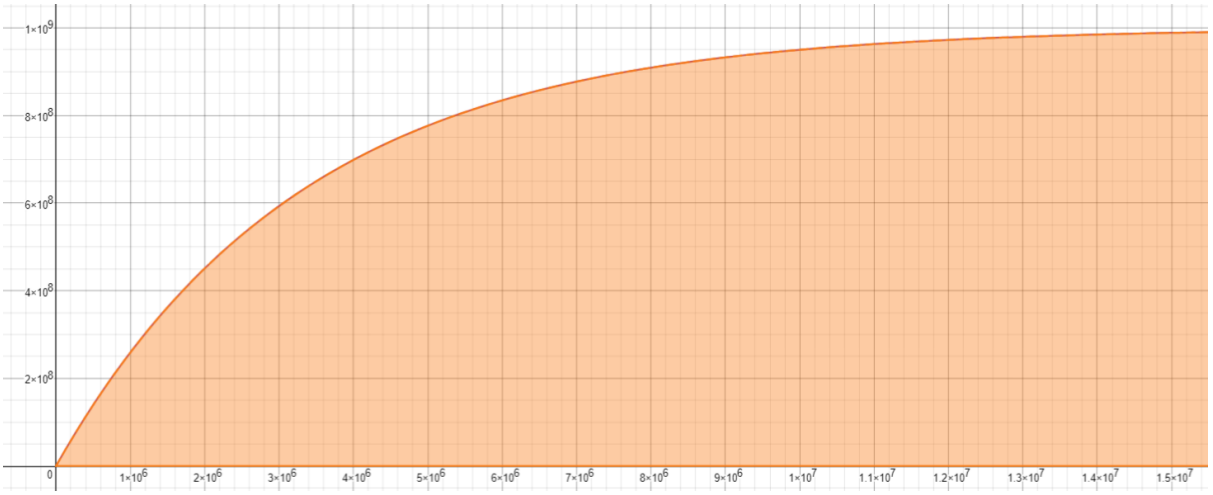
1. The transaction's block index matches the current block.
2. The transaction does not exist in the exact same form on the block already - it is not a duplicate.
3. The signature on the transaction is valid. This is done using the DSA signature verification function.
4. The sender has sufficient funds to pay the transaction value and the fee.
    a. To calculate the sender's funds, the miner will go over every single block in the history of the blockchain and calculate how many coins the wallet has. Adding funds sent to them, and subtracting funds they sent, factoring in mining compensation.

If all these criteria are met, the miner will append the transaction to the current working block. If this was the 64th transaction, the miner will validate the block - find the proper nonce - and broadcast the signed block.

A miner's compensation is one of the most important factors of a blockchain protocol. On GreenCoin, a miner is rewarded the sum of all the transaction fees on the block, plus

some newly minted coins. In order to control the maximum amount of GreenCoins in circulation, to avoid inflation, there are diminishing returns on miner's fees.

Mining fees follow an infinite convergent series pattern, which eventually produces no more than 1,000,000,000 coins.



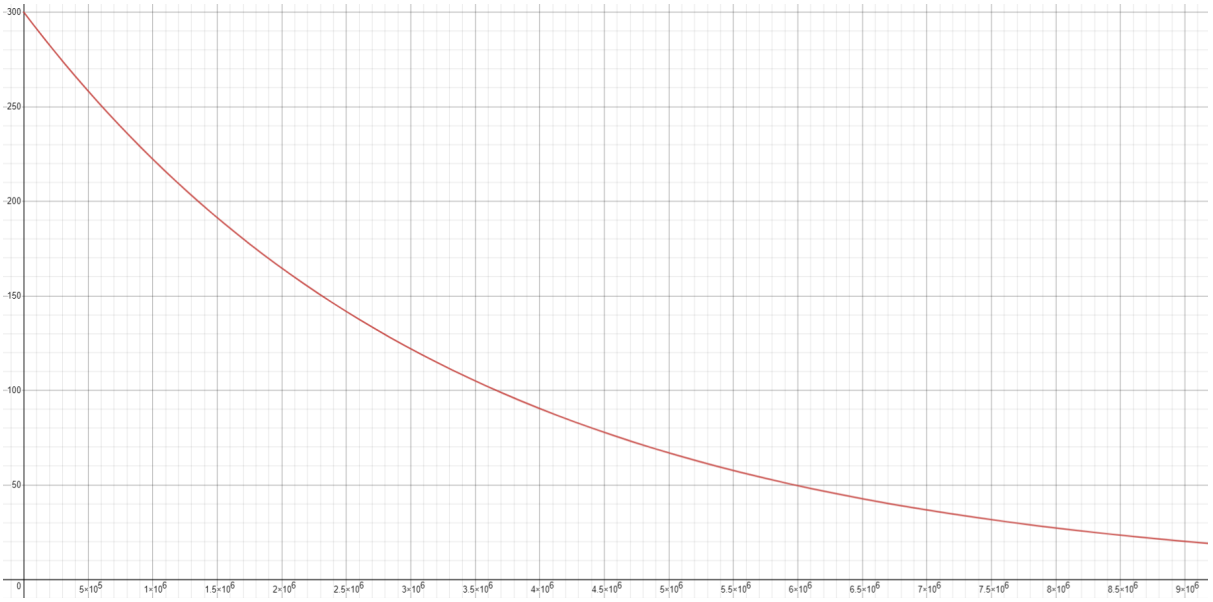
The formula for the total amount of coins is as follows (according to the formula for the sum of an infinite convergent series):

$$S = \frac{a_1}{q-1} = \frac{300}{1-0.9999997} = 1,000,000,000$$

.

Since we define that the first mining reward will be 300GC, and each miner will receive 0.9999997 in mining fees of what the previous miner received. The n^{th} miner will receive: $a_1 \cdot q^{n-1} = 300 \cdot 0.9999997^{n-1}$GC.

Visualized, the x^{th} miner will receive y coins in miner's fees:



And the total coins in circulation after the n^{th} mining will be

$$\frac{a_1 \cdot \left(q^n - 1\right)}{q - 1} = \frac{300 \cdot \left(0.9999997^n - 1\right)}{0.9999997 - 1} \text{GC.}$$

Code

BlockChain.h + BlockChain.c

```
#pragma once


#ifndef __BLOCKCHAIN_H
#define __BLOCKCHAIN_H



#define MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER 64
//const uint16_t MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER = 64;


#include "Transaction.h"
#include "../General/FileIO.h"
#include <time.h>


#define INITIAL_BLOCK_MINING_FEE  300
//const uint16_t INITIAL_BLOCK_MINING_FEE = 300;
#define MINING_FEE_DECAY                0.9999997
//const double MINING_FEE_DECAY = 0.9999997;


uint64_t NETWORK_BLOCK_REQUEST_GLOBAL_PLACEHOLDER;


char BLOCK_HISTORY_DIRECTORY_PATH[256];
```

```
uint64_t MIN_STRENGTH;


_Wallet_Address LOCAL_NOTARY_SIGNING_ADDRESS;


typedef struct _TimeStamp {
        uint32_t Unix_Time;
} _TimeStamp;
static char DateTimeBuffer[128];
char * HumanFormatDateTime(_TimeStamp * timestamp);
char * HumanFormatDateTimeInt(uint32_t timestamp);


typedef struct _Hash {
        uint32_t Bytes[8];
} _Hash;
void Print_Hash(FILE * fstream, _Hash * hash);


typedef uint32_t uint256_t[8];
void Print_uint256(FILE * fstream, uint256_t * r);


typedef struct _Block _Block;


struct _Block {


        /*      The index of the block in the chain.
```

```
        *       This will be used to decide which chain of blocks is the truth.
        *       The block index will increase by one each block.

        *       If two blocks have the same index, a second chain "of truth" will be created.
        *       Once one chain exceeds the other chain's length by a predefined (or relative) amount, the shorter chain will be invalidated.
        */
        uint64_t Block_Index;



        /*
        *       Information about the creation time of the block.
        *       This will be the time the first information was written to the block, not the time it was signed.
        */
        _TimeStamp Time_Stamp;



        /*
        *       The calculated hash of the previous block.
        *       This is used to make sure the hashes are properly interconnected.
        */
        _Hash Previous_Block_Hash;
```

```
        /*
        *       An array of the transactions on the ledger.
        */
        _Transaction
Transactions[MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER];


        /*
        *       The wallet address of the notary who verified and signed the current block.
        *       This address will be awarded all the fees from the transactions on this block.
        */
        _Wallet_Address Notary_Address;



        /*
        *       256 bit number which is used in order to "control" the resulting hash
        */
        uint256_t Block_Validation;



        ///*
        //*     A linked chain of blocks, as Unary Nodes.
        //*/
        //_Block * Next_Block;
};
```

```
_Block * live_block;


uint64_t Calculate_Block_Strength(_Block * block);


_Block * Create_Block(uint64_t new_block_index, char * previous_block_hash_ptr);


error_t Validate_Block(void * wsadata, void * socket, _Block * block, uint64_t
desired_strength);


error_t Append_Transaction(void * wsadata, void * socket, _Block * block, _Transaction *
transaction);


double Calculate_Block_Total_Miner_Fees(_Block * block);


void Print_Block(FILE * fstream, _Block * block);


int Block_Exists(_Block * block, int block_size);

int Block_Index_Exists(uint64_t ind);

error_t Open_Block_File(FILE ** f, uint64_t block_ind);


error_t Export_Block(FILE * fstream, _Block * block);

error_t Export_To_File(char * dir_path, _Block * block);




error_t Load_Local_Notary_Signing_Address();
```

```
error_t Load_Block_History_Path();

error_t Create_First_Block(void * wsadata, void * socket);

error_t Verify_Block(void * wsadata, void * socket, _Block * block, int block_size, int from_history);

//void BlockChain_Demo();

#endif // !__BLOCKCHAIN_H
```

```
#include "BlockChain.h"

#include "../Wallet/Wallet.h"
#include "../General/Print/PrettyPrint.h"

uint64_t NETWORK_BLOCK_REQUEST_GLOBAL_PLACEHOLDER = 0;

char BLOCK_HISTORY_DIRECTORY_PATH[256] = "";

_Wallet_Address LOCAL_NOTARY_SIGNING_ADDRESS = {
        0x933BB069, 0x3402D6A8, 0x5168FBEC , 0x6082544A
  , 0x59A54893 , 0x875DE25E , 0xE1210B1E , 0x147A4979
  , 0xDEA427CF , 0x95466511 , 0x396D7B4E , 0x5EFF0522
```

```
  , 0xA7E7D606 , 0x5FBAE2DD , 0xB9E568F5 , 0xD4DE08D5
  , 0x9775927E , 0xC697788F , 0xE57FE89D , 0xA8E429E3
  , 0x7916307B , 0x3F71B0CA , 0xA7CF169C , 0x60A3F1BE
  , 0x5D811C74 , 0x8FCC06B3 , 0xF40731A3 , 0xDF65D263
  , 0x4C9C0A93 , 0x7D1B1269 , 0x82583063 , 0x112B715B };




char * HumanFormatDateTime(_TimeStamp * timestamp) {
        time_t rawtime = timestamp->Unix_Time;
        struct tm  ts;
        // Format time, "ddd yyyy-mm-dd hh:mm:ss zzz"
        localtime_s(&ts, &rawtime);
        strftime(DateTimeBuffer, sizeof(DateTimeBuffer), "%Y-%m-%d %H:%M:%S",
&ts);
        return DateTimeBuffer;
}
char * HumanFormatDateTimeInt(uint32_t timestamp) {
        time_t rawtime = timestamp;
        struct tm  ts;
        // Format time, "ddd yyyy-mm-dd hh:mm:ss zzz"
        localtime_s(&ts, &rawtime);
        strftime(DateTimeBuffer, sizeof(DateTimeBuffer), "%Y-%m-%d %H:%M:%S",
&ts);
        return DateTimeBuffer;
```

```
}

void Print_Hash(FILE * fstream, _Hash * hash) {
        fprintf(fstream, "  ");
        for (int i = 7; i >= 0; i--) {
        fprintf(fstream, "%.8X ", hash->Bytes[i]);
        if (i % 4 == 0) { fprintf(fstream, "\n  "); }
        }
        fprintf(fstream, "\n");
}

void Print_uint256(FILE * fstream, uint256_t * r) {
        fprintf(fstream, "  ");
        for (int i = 7; i >= 0; i--) {
        fprintf(fstream, "%.8X ", (*r)[i]);
        if (i % 4 == 0) { fprintf(fstream, "\n  "); }
        }
        fprintf(fstream, "\n");
}

uint64_t Calculate_Block_Strength(_Block * block) {
        char * digest = Hash_SHA256(block, sizeof(_Block));

        int i = 0;
```

```
        while (digest[i] == '0') { i++; }

        free(digest);

        return (uint64_t)(pow(2, i) - 1);
}

_Block * Create_Block(uint64_t new_block_index, char * previous_block_hash_ptr) {
        _Block * block = (_Block*)calloc(1, sizeof(_Block));

        block->Time_Stamp.Unix_Time = time(NULL);

        block->Block_Index = new_block_index;

        BN hs; BN_Init_Stack(&hs);
        BN_Import_Hex_String(&hs, previous_block_hash_ptr, 64, BN_BIG_ENDIAN);
        memcpy(block->Previous_Block_Hash.Bytes, hs.data, 8 * sizeof(uint32_t));

        free(hs.data);

        memset(block->Block_Validation, 0, sizeof(uint256_t));

        return block;
}
```

```
error_t Validate_Block(void * wsadata, void * socket, _Block * block, uint64_t
desired_strength) {

	//memcpy(block->Notary_Address, y->data, y->size * BN_INT_SIZE);

	memcpy(block->Notary_Address, LOCAL_NOTARY_SIGNING_ADDRESS,
sizeof(LOCAL_NOTARY_SIGNING_ADDRESS));


	BN c;

	BN_Init_Stack(&c);

	BN_Resize(&c, 32 / BN_INT_SIZE);

	memcpy(c.data, block->Block_Validation, sizeof(uint256_t));


	uint64_t strength = Calculate_Block_Strength(block);

	//printf("Block strength: %lu\n", strength);


	while (strength < desired_strength) {

	BN_Add_Int(&c, &c, 1);

	memcpy(block->Block_Validation, c.data, sizeof(uint256_t));

	strength = Calculate_Block_Strength(block);

	//printf("Block strength: %u\n", strength);

	}


	free(c.data);
```

```
        printf_Success("Block has been signed with strength: %lu!\n", strength);

        Export_To_File(BLOCK_HISTORY_DIRECTORY_PATH, block);

        Network_Broadcast_Block(wsadata, socket, block, sizeof(_Block));

        return ERROR_NONE;
}

error_t Append_Transaction(void * wsadata, void * socket, _Block * block, _Transaction *
transaction) {
        //uint32_t transaction_index = transaction->Index;

        //_Transaction * target = &(block->Transactions[transaction_index]);
        _Transaction * target = &(block->Transactions[0]);

        if (transaction->Block_Index != block->Block_Index) { return
ERROR_BLOCK_TRANSACTION_FAILED; }

        _Transaction zero_transaction; memset(&zero_transaction, 0,
sizeof(_Transaction));

        int index = 0;
        while (memcmp(&(block->Transactions[index]), &zero_transaction,
sizeof(_Transaction)) != 0) {
        index++;
```

```
        }


        target = &(block->Transactions[index]);


        // Make sure that there aren't two duplicate transactions on a block.

        // This protects the system from a fraud duplicating a valid transaction.

        for (int i = 0; i < index; i++) {

        if (memcmp(transaction, &(block->Transactions[i]), sizeof(_Transaction)) == 0) {
return ERROR_BLOCK_TRANSACTION_DUPLICATE; }

        }


        if (index >= MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER) {
return ERROR_BLOCK_TRANSACTION_SLOT_IN_USE; }


        //if (memcmp(target, &zero_transaction, sizeof(_Transaction)) != 0) { return
ERROR_BLOCK_TRANSACTION_SLOT_IN_USE; }


        if (Verify_Transaction(transaction) != SIGNATURE_VALID) { return
ERROR_BLOCK_TRANSACTION_SIGNATURE_INVALID; }


        double value = Calculate_Wallet_Value(BLOCK_HISTORY_DIRECTORY_PATH,
transaction->Sender, block->Block_Index);


        if (value < transaction->Value + transaction->Fee) { return
ERROR_BLOCK_TRANSACTION_INSUFFICIENT_FUNDS; }


        memcpy(target, transaction, sizeof(_Transaction));
```

```
		printf_Success("A valid transaction has been received and processed.\n");

		if (index == MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER - 1)
{
		// Block is full. You may now sign it.
		Validate_Block(wsadata, socket, live_block, 7);

		char * hash = Hash_SHA256(live_block, sizeof(_Block));
		live_block = Create_Block(live_block->Block_Index + 1, hash);
		free(hash);
		}

		return ERROR_BLOCK_TRANSACTION_NONE;
}

double Calculate_Block_Total_Miner_Fees(_Block * block) {
		double total = 0;
		double mining_fee = (double)INITIAL_BLOCK_MINING_FEE *
pow((double)MINING_FEE_DECAY, block->Block_Index);
		total += mining_fee;

		for (int i = 0; i < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER;
i++) {
		total += block->Transactions[i].Fee;
```

```
	}


	return total;

}


void Print_Block(FILE * fstream, _Block * block) {

	fprintf(fstream, "=== === === Block #%lu === === ===\n",
block->Block_Index);

	fprintf(fstream, "\tCreation Time: %s\n",
HumanFormatDateTime(&(block->Time_Stamp)));


	if (block->Block_Index != 0) {

	fprintf(fstream, "\t--- Begin Transactions ---\n");

	for (int i = 0; i < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER;
i++) {

		Print_Transaction(fstream, &(block->Transactions[i]));

	}

	fprintf(fstream, "\t--- End Transactions ---\n");

	}

	else {

	fprintf(fstream, "%s", block->Transactions);

	}


	fprintf(fstream, "\tPrevious block hash: \n");

	Print_Hash(fstream, &(block->Previous_Block_Hash));
```

```
	fprintf(fstream, "\tNotary Address: \n");
	Print_Transaction_Wallet_Address(fstream, block->Notary_Address);

	fprintf(fstream, "\tVerification: \n");
	Print_uint256(fstream, (block->Block_Validation));

	fprintf(fstream, "\n=== === === End Block #%lu === === ===\n",
block->Block_Index);
}

/*
void BlockChain_Demo() {
	char * prev_block_hash = Hash_SHA256("", 0);
	char prev_block_hash_data[64];// = Hash_SHA256("", 0);

	printf("Please enter your public key (as base64): \n");
	char key_64[180]; fgets(key_64, 180, stdin);

	byte * key;
	B64_Decode(key_64, &key);

	DSA_Public_Key * pub_key=0; DSA_Init_Public_Key(&pub_key);
	BN_Resize(pub_key, 32);
	memcpy(pub_key->data, key, sizeof(_Wallet_Address));
```

```
        free(key);



        char file_path[256];

        char * buffer;

        _Block * block;

        _Transaction transaction_tmp;

        size_t length = 0;


        uint64_t c = 303;

        while (1) {

        block = Create_Block(c++, prev_block_hash);

        free(prev_block_hash);


        for (uint32_t i = 0; i <
MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER; i++) {

                sprintf_s(file_path, 256,
"C:\\Users\\stein\\Desktop\\GreenCoin\\Globals\\Demo_Transactions\\Test_%u.GCT", i);


                length = Load_File(file_path, &buffer);

                if (length != -1) {

                memcpy(&transaction_tmp, buffer + 4, sizeof(_Transaction));


                Append_Transaction(block, &transaction_tmp);
```

```
                free(buffer);
                }

        }

        Validate_Block(block, 1);

        Export_To_File("C:\\Users\\stein\\Desktop\\GreenCoin\\Demo\\Blocks", block);

        //free(prev_block_hash);
        prev_block_hash = Hash_SHA256(block, sizeof(_Block));
        //memcpy(prev_block_hash_data, prev_block_hash, 64);
        free(block);
        }

        //Print_BlockChain(stderr, block, params);
}
*/

error_t Export_Block(FILE * fstream, _Block * block) {
        size_t count = fwrite(block, sizeof(_Block), 1, fstream);
        if (count == 1) {
        return ERROR_NONE;
        }
```

```
        else {
        return ERROR_WRITE_TO_FILE;
        }
}
error_t Export_To_File(char * dir_path, _Block * block) {
        FILE * f;
        char buffer[256] = { 0 };
        sprintf_s(buffer, 256, "%s\\%lu.GCB", dir_path, block->Block_Index);
        fopen_s(&f, buffer, "wb");
        if (f == NULL) { return ERROR_OPEN_FILE; }
        error_t error = ERROR_NONE;
        error |= !(fwrite(GCB_MAGIC, sizeof(char), 4, f) == 4);
        error |= Export_Block(f, block);
        fclose(f);
        return error;
}


error_t Load_Local_Notary_Signing_Address() {
        FILE* f;
        errno_t err = fopen_s(&f, "Local_Notary_Address.GCWA", "rb");
        if (err == 0) {
        fread_s(LOCAL_NOTARY_SIGNING_ADDRESS,
sizeof(LOCAL_NOTARY_SIGNING_ADDRESS),
sizeof(LOCAL_NOTARY_SIGNING_ADDRESS), 1, f);
        }
```

```
        else {
        err = fopen_s(&f, "Local_Notary_Address.GCWA", "wb");
        printf("Please enter the address to be used when mining blocks (as base64): \n");
        char buf[1024];
        fgets(buf, 1024, stdin);
        byte* out;
        B64_Decode(buf, &out);
        memcpy(LOCAL_NOTARY_SIGNING_ADDRESS, out,
sizeof(LOCAL_NOTARY_SIGNING_ADDRESS));
        fwrite(LOCAL_NOTARY_SIGNING_ADDRESS,
sizeof(LOCAL_NOTARY_SIGNING_ADDRESS), 1, f);
        free(out);
        }
    fclose(f);
        printf("Mining address is: \n");
        Print_Transaction_Wallet_Address(stdout,
LOCAL_NOTARY_SIGNING_ADDRESS);
}


error_t Load_Block_History_Path() {
        FILE * f;
        errno_t err = fopen_s(&f, "Block_History_Path.GCTXT", "r");
        if (err == 0) {
        fread_s(BLOCK_HISTORY_DIRECTORY_PATH, 256, sizeof(char), 256, f);
        }
```

```
	else {
	printf("Would you like to define a path for the BlockChain history? ( Y / N )");
	char buf[32];
	fgets(buf, sizeof(buf), stdin);
	int define = (strcmp(buf, "Y\n") == 0) || (strcmp(buf, "y\n") == 0) || (strcmp(buf, "Yes\n") == 0) || (strcmp(buf, "yes\n") == 0);
	if (!define) { return ERROR_NONE; }

	err = fopen_s(&f, "Block_History_Path.GCTXT", "w");
	printf("Please enter directory path for Block History: ");
	fgets(BLOCK_HISTORY_DIRECTORY_PATH, 256, stdin);
	fwrite(BLOCK_HISTORY_DIRECTORY_PATH, sizeof(char), 256, f);
	}
	fclose(f);

	printf("Blockchain is saved to: %s\\\n", BLOCK_HISTORY_DIRECTORY_PATH);
	return ERROR_NONE;
}

error_t Create_First_Block(void * wsadata, void * socket) {
	char * null = NULL;
	char * hash = Hash_SHA256(null, 0);
	_Block * b = Create_Block(0, hash);
	free(hash);
```

```
	b->Time_Stamp.Unix_Time = time(NULL);



	FILE * f;
	fopen_s(&f, "C:\\Users\\stein\\Desktop\\GreenCoin\\Globals\\INIT_MESSAGE.txt", "r");
	fseek(f, 0, SEEK_END);
	long size = ftell(f);
	fseek(f, 0, SEEK_SET);
	char * Initial_Block_Message = (char*)calloc(size+1, sizeof(char));
	fread(Initial_Block_Message, sizeof(char), size, f);
	fclose(f);


	memcpy(b->Transactions, Initial_Block_Message, size + 1);
	free(Initial_Block_Message);


	Validate_Block(wsadata, socket, b, 7);


	Export_To_File(BLOCK_HISTORY_DIRECTORY_PATH, b);


	free(b);


	return ERROR_NONE;
}
```

```
error_t Open_Block_File(FILE ** f, uint64_t block_ind) {
	char buffer[256] = { 0 };
	sprintf_s(buffer, 256, "%s\\%lu.GCB", BLOCK_HISTORY_DIRECTORY_PATH, block_ind);

	fopen_s(f, buffer, "rb");
	if (*f == NULL) { return ERROR_FAILED; }

	return ERROR_NONE;
}

int Block_Index_Exists(uint64_t ind) {
	char buffer[256] = { 0 };
	sprintf_s(buffer, 256, "%s\\%lu.GCB", BLOCK_HISTORY_DIRECTORY_PATH, ind);

	FILE * f;
	fopen_s(&f, buffer, "rb");
	if (f == NULL) { return 0; }
	fseek(f, 0, SEEK_END);
	int size = ftell(f);
	fseek(f, 0, SEEK_SET);

	if (size != sizeof(_Block)+4) { /* A different block with the same index exists */
```

```
return 2; }


	fclose(f);


	return 1;
}
int Block_Exists(_Block * block, int block_size) {
	char buffer[256] = { 0 };
	sprintf_s(buffer, 256, "%s\\%lu.GCB", BLOCK_HISTORY_DIRECTORY_PATH, block->Block_Index);


	FILE * f;
	errno_t err =fopen_s(&f, buffer, "rb");
	if (f == NULL || err != 0) { return 0; }
	fseek(f, 0, SEEK_END);
	int size = ftell(f);
	fseek(f, 0, SEEK_SET);


	if (size != block_size+sizeof(GCB_MAGIC)) { /* A different block with the same index exists */ return 2; }


	char * buf = (char*)malloc(size);
	fread(buf, 1, size, f);
	fclose(f);
```

```
        int is_same = memcmp(buf + 4, block, size - 4) == 0;


        if (is_same) { return 1; }
        else {
        /* A different block with the same index exists */
        return 2;
        }


        return -1;
}


uint64_t MIN_STRENGTH = 7;
error_t Verify_Block(void * wsadata, void * socket, _Block * block, int block_size, int
from_history) {


        // 0 -> No block by this name.
        // 1 -> This exact block exists => duplicate.
        // 2 -> Inconsistent blocks. Two different blockchains exist.
        int block_exists = Block_Exists(block, block_size);


        if (block_exists == 1) { return ERROR_NONE; }


        // Check that the block produces the desired hash
        uint64_t strength = Calculate_Block_Strength(block);
        if (strength < MIN_STRENGTH) {
```

```
        return ERROR_BLOCK_WEAK;

        }


        error_t error = ERROR_NONE;

        if (block->Block_Index > 0 && !from_history) {

        // Verify each transaction

        for (int index = 0; index < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER; index++) {

                _Transaction * transaction = &(block->Transactions[index]);


                if (Verify_Transaction(transaction) != SIGNATURE_VALID) { error |= ERROR_BLOCK_TRANSACTION_SIGNATURE_INVALID; }


                /*double value = Calculate_Wallet_Value(BLOCK_HISTORY_DIRECTORY_PATH, transaction->Sender, block->Block_Index);


                if (value < transaction->Value + transaction->Fee) { error |= ERROR_BLOCK_TRANSACTION_INSUFFICIENT_FUNDS; }*/


                if (Is_Wallet_Value_Greater(BLOCK_HISTORY_DIRECTORY_PATH, transaction->Sender, block->Block_Index, transaction->Value + transaction->Fee) != 1) {

                error |= ERROR_BLOCK_TRANSACTION_INSUFFICIENT_FUNDS;

                }

        }

        }
```

```
        if (block->Block_Index > 0) {

        FILE* f;

        Open_Block_File(&f, block->Block_Index - 1);

        if (f == NULL) { return ERROR_FAILED; }

        fseek(f, 4, SEEK_SET);

        _Block b; fread(&b, sizeof(b), 1, f);

        fclose(f);

        char* prev_hash = Hash_SHA256(&b, sizeof(b));


        BN hs; BN_Init_Stack(&hs);

        BN_Import_Hex_String(&hs, prev_hash, 64, BN_BIG_ENDIAN);

        int prev_hash_valid = (memcmp(hs.data, &(block->Previous_Block_Hash),
sizeof(_Hash))==0);


        free(hs.data);


        if (prev_hash_valid != 1) { error |= ERROR_FAILED; }

        free(prev_hash);

        }


        if (error == ERROR_NONE) {

        BlockChainLength = block->Block_Index+1;


        NETWORK_BLOCK_REQUEST_GLOBAL_PLACEHOLDER =
block->Block_Index;
```

```
        Export_To_File(BLOCK_HISTORY_DIRECTORY_PATH, block);


        Network_Broadcast_Block(wsadata, socket, block, block_size);


        char* hash = Hash_SHA256(block, sizeof(_Block));
        live_block = Create_Block(BlockChainLength, hash);
        free(hash);
        }
}
```

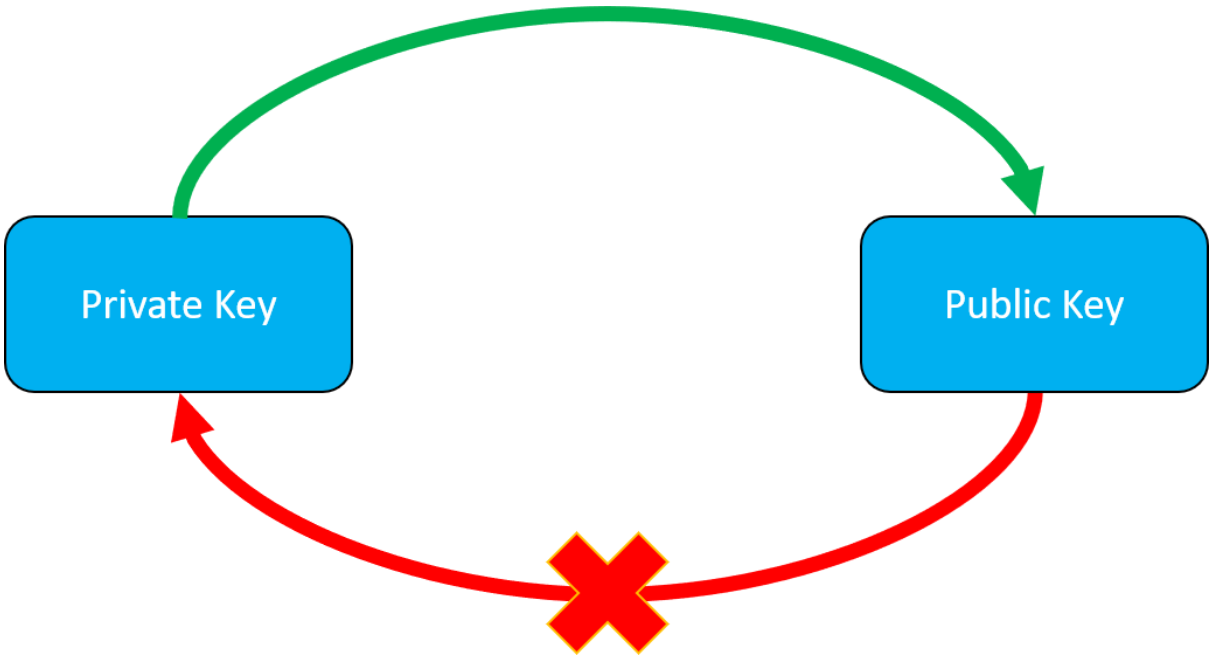
Wallets

Wallets are the name given to public keys, since they act like real world wallets; money can be added to them, and money can be taken out of them. The wallet address is synonymous with the public key.

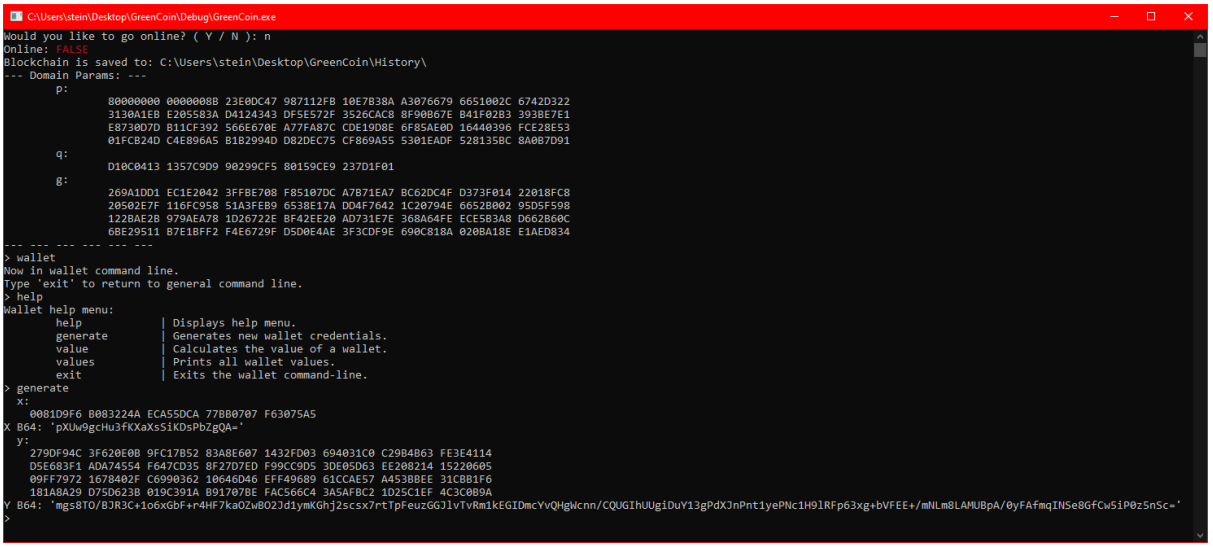
A wallet can be described like a bank account: the private key is the account password, and the public key is the bank account number to which people route money sent to. Except that in the context of digital currencies, the private key is a randomly generated value, and the public key is a number mathematically computed out of the private key, which cannot be reversed. This pair of values, the private-public key pair, is later used to generate and verify DSA signatures.

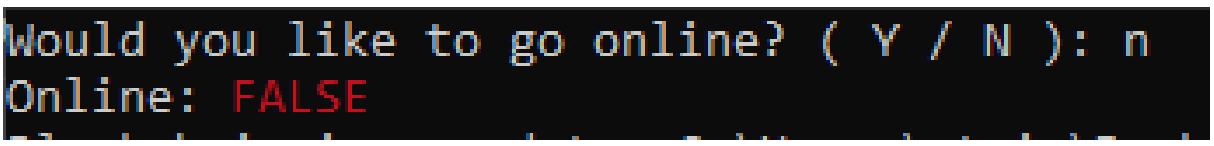
The private key of a wallet should be kept secret at all times, and preferably should not be stored on any hackable device, such as a computer, smartphone, or similar. A private key can literally be written down on a piece of paper, in any encoding format, bitwise, hexadecimal, base-64, even a QR code. The public key can, and will, be freely shared. The public key is what other users will need in order to transfer funds to the wallet. Never the private key!

Losing the private key of a wallet will render the wallet completely useless, and all the funds associated with that wallet will be lost forever and can never be retrieved. This is since, unlike a normal bank password, the private key of a wallet cannot ever be changed. Additionally, a private key can obviously never be retrieved even if one has the public key, as if this was possible, it would render the entire DSA system useless, and collapse all BlockChain protocols.



The private key of a wallet is any number satisfying $1 \leq x \leq q - 1$. This number should be randomly generated. Statistically, the chance of two people generating the same private key, assuming they used truly random generation, is so small it is insignificant. Since it is so simple, a private key can be computed by hand, or on an offline computer system. This is recommended in order to decrease the possibility that a hacker will steal the private key. Therefore, there is a special mode for the GreenCoin software where everything is offline, which is designed specifically in order to generate a new wallet - a new private-public key pair. This is done the following way:





```
> wallet
Now in wallet command line.
Type 'exit' to return to general command line.
> help
Wallet help menu:
        help            | Displays help menu.
        generate        | Generates new wallet credentials.
        value           | Calculates the value of a wallet.
        values          | Prints all wallet values.
        exit            | Exits the wallet command-line.
> generate
```

Base-64 Encoding

Base-64 is an encoding format which converts arbitrary binary data into a human-readable string of characters. These characters consist of 65 ASCII (printable) characters:

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Base-64 splits data into sets of 6 bits, instead of 8 bits like a byte, which allows for $2^6 = 64$combinations. This allows all combinations to be represented by legible characters, since there are 26 Latin characters, times two since they can be uppercase, 10 digits, and two more special symbols, and a padding character. This is in contrary to ASCII 8 bit encoding which contains many eligible (unprintable) characters:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Since computer data is traditionally stored in bytes, sets of 8 bits, some data will not necessarily divide into 6 bits without a remainder. In order to address this, base-64 adds padding characters; '=' characters at the end of the base-64 string, each of which means an additional 2 bits of 0 tacked onto the original data. These padding bits are discarded when decoding, and do not affect the original data in any way.

Base-64 is useful for representing wallet private and public keys since these keys are pure binary data which cannot necessarily be represented as ASCII (base-256). Therefore, these values can either be shown in raw binary form, which would be unbelievably long, hexadecimal form, which is much more elegant yet still quite long, or base-64 encoding, which is the most concise encoding which can still be printed and is legible.

Therefore, when generating a wallet, the GreenCoin software will also print the keys in base-64 format in addition to their hexadecimal representation, in a form which is easily copied or written down on paper. The software will also request wallet addresses and private keys only in base-64 encoding, since this is much easier to enter, and removes the confusion between little and big endian.

```
Please enter your public key (as base64):
yQXNOtIfvBL+mJFNzZJZGGgtqdTtsexoZazVfMMYHIsuR9gyQYIzn759pL14+qyeGdooUdewGczSYyugHI1FQu9AWgUKt6RGdfnUBkWp4DXhqFVac5ZnJpDe
HPOpTpBsN2riMtJxKLZfFW+VYhIARaOkqpbTud13sLPUQz1+jQ0=
Please enter the reciever's public key (as base64):
3EEO8LXVh8lqibPY8hu0oEGhuBENLV8TyBC4698vy9D591DWD1FLi84ySBDttTbQzX2RMWTF6/lP1IRTacJAUdXjGBOWniU9g1BRjkGTLPGY+HrZly5FHWcr
kpO5WrRef+8LbrOW74/jfMoXJe42cqBS36oIquLGhDuyOgHckis=
How much would you like to send?
100
Please enter your private key (as base64) in order to sign this transaction:
+UKTs0dcf73nAGzrH0/AUlKYLmY=
```

Code

Base64.h + Base64.c

```
#pragma once

#ifndef __BASE_64_H
#define __BASE_64_H

#include "../General.h"

const char BASE64_CHAR_SEQUENCE[64];
static const signed char BASE64_REVERSE_SEQUENCE[256];

// Returns the size of something to be converted  ~ 4/3 * len
size_t B64_Encoded_Size(size_t len);

// Returns the size of something to be decoded  ~ 3/4 * len
size_t B64_Decoded_Size(char * base64_array);

/* Encode bytes to base64 char array
   Returns the length of the base64 char array.
   Allocates buffer. */
size_t B64_Encode(byte * input, size_t input_length, char ** output);

/* Decodes base64 char array to byte array
   Returns the length of the char array.
   Allocates buffer. */
size_t B64_Decode(char * input, byte ** output);

#endif // !__BASE_64
```

```
#include "Base64.h"

const char BASE64_CHAR_SEQUENCE[64] = {
  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
```

```
  'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
  'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
  'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
  'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
  'w', 'x', 'y', 'z', '0', '1', '2', '3',
  '4', '5', '6', '7', '8', '9', '+', '/'
};

static const signed char BASE64_REVERSE_SEQUENCE[256] = {
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, -1, -1, -1, -1, -1, -1,
-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1, -1, -1, -1, -1,
-1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1
};

size_t B64_Encoded_Size(size_t len) {
	if (len % 3 != 0) {
	len += 3 - (len % 3);
	}
	len /= 3;
	len *= 4;

	return len;
}

size_t B64_Decoded_Size(char * base64_array) {

	size_t length = strlen(base64_array);

	if (length < 1) { return 0; }
```

```
	int padding_count = (int)(base64_array[length - 1] == '=') +
(int)(base64_array[length - 2] == '=');

	length = ((length / 4) * 3) - padding_count;

	return length;
}

size_t B64_Encode(byte * input, size_t input_length, char ** output)
{
	char * out, * pos;

	byte * end, * in;

	size_t out_len;

	out_len = 4 * ((input_length + 2) / 3);

	if (out_len < input_length) {
	return -1;
	}

	*output = (char*)malloc(out_len + 1);
	out = *output;

	end = input + input_length;
	in = input;
	pos = out;

	while (end - in >= 3) {
	*pos++ = BASE64_CHAR_SEQUENCE[in[0] >> 2];
	*pos++ = BASE64_CHAR_SEQUENCE[((in[0] & 0x03) << 4) | (in[1] >> 4)];
	*pos++ = BASE64_CHAR_SEQUENCE[((in[1] & 0x0f) << 2) | (in[2] >> 6)];
	*pos++ = BASE64_CHAR_SEQUENCE[in[2] & 0x3f];
	in += 3;
	}

	if (end - in) {
	*pos++ = BASE64_CHAR_SEQUENCE[in[0] >> 2];
	if (end - in == 1) {
		*pos++ = BASE64_CHAR_SEQUENCE[(in[0] & 0x03) << 4];
		*pos++ = '=';
```

```
        }
        else {
                *pos++ = BASE64_CHAR_SEQUENCE[((in[0] & 0x03) << 4) |
                (in[1] >> 4)];
                *pos++ = BASE64_CHAR_SEQUENCE[(in[1] & 0x0f) << 2];
        }
        *pos++ = '=';
        }

        *pos = 0x0;

        return out_len;
}

size_t B64_Decode(char * input, byte ** output) {
        size_t input_length = strlen(input);
        size_t length = B64_Decoded_Size(input);

        *output = (byte*)malloc(length+1);
        char * out = *output;

        char * pos = out;
        char * end = out + length;
        char * in = input;

        while (end - pos >= 3) {
        // 4 base64 -> 3 char
        // 4 * 6bits => 3 * 8bits = 24bits
        pos[0] = ((BASE64_REVERSE_SEQUENCE[in[0]] << 2) |
(BASE64_REVERSE_SEQUENCE[in[1]] >> 4));
        pos[1] = (((BASE64_REVERSE_SEQUENCE[in[1]] & 0x0f) << 4) |
(BASE64_REVERSE_SEQUENCE[in[2]] >> 2));
        pos[2] = (((BASE64_REVERSE_SEQUENCE[in[2]] & 0x03) << 6) |
(BASE64_REVERSE_SEQUENCE[in[3]]));

        pos += 3;
        in += 4;
        }

        if (end - pos) {
        // =
        int padding_count = (int)(input[input_length - 1] == '=') + (int)(input[input_length -
2] == '=');
```

```
        if (padding_count == 1) {
                pos[0] = ((BASE64_REVERSE_SEQUENCE[in[0]] << 2) |
(BASE64_REVERSE_SEQUENCE[in[1]] >> 4));
                pos[1] = (((BASE64_REVERSE_SEQUENCE[in[1]] & 0x0f) << 4) |
(BASE64_REVERSE_SEQUENCE[in[2]] >> 2));
                pos += 2;
                in += 2;
        }
        else if (padding_count == 2) {
                pos[0] = ((BASE64_REVERSE_SEQUENCE[in[0]] << 2) |
(BASE64_REVERSE_SEQUENCE[in[1]] >> 4));
                pos++;
                in += 2;
        }
        else {
                // Error?
        }
        }

        pos[0] = 0x0;

        return length;
}
```

Code

Wallet.h + Wallet.c

```
#pragma once


#ifndef __WALLET_H

#define __WALLET_H



#include "../Cryptography/DSA/DSA.h"

#include "../General/Base64/Base64.h"

#include "../Cryptography/Hash/SHA256.h"


command_t COMMAND_GENERATE_WALLET;

command_t COMMAND_EXIT;

command_t COMMAND_WALLET_VALUE;


typedef uint_t _Wallet_Address[32];



double Calculate_Wallet_Value(char * dir_path, _Wallet_Address pk, uint64_t
up_to_block_index);


int Is_Wallet_Value_Greater(char * dir_path, _Wallet_Address pk, uint64_t
up_to_block_index, double thresh);
```

```
int Wallet_Chain_Contains_Address(struct Wallet_Piece * wallet_chain, _Wallet_Address wallet_address);

int Wallet_Add_Address_To_Chain(struct Wallet_Piece * wallet_chain, _Wallet_Address wallet_address);

int Wallet_Chain_Recursive_Free(struct Wallet_Piece * wallet_chain);

void Wallet_Calculate_Values();


void Wallet_Calculate_Value_CommandLine();


void Print_Demo_Keys();


DSA_Public_Key * Get_Random_Public_Key();


int wallet_exit;

int Wallet_Exit();


int Wallet_Help();


int Wallet_CommandLine_General();


#endif // !__WALLET_H
```

```
#include "Wallet.h"
```

```
#include "../BlockChain/BlockChain.h"

command_t WALLET_COMMAND_HELP = {
        "help",
        "Displays help menu.",
        Wallet_Help
};
command_t WALLET_COMMAND_GENERATE_WALLET = {
        "generate",
        "Generates new wallet credentials.",
        Print_Demo_Keys
};
command_t WALLET_COMMAND_WALLET_VALUE = {
        "value",
        "Calculates the value of a wallet.",
        Wallet_Calculate_Value_CommandLine
};
command_t WALLET_COMMAND_ALL_VALUES = {
        "values",
        "Prints all wallet values.",
        Wallet_Calculate_Values
};
command_t WALLET_COMMAND_EXIT = {
```

```
        "exit",

        "Exits the wallet command-line.",

        Wallet_Exit

};


command_t * WALLET_COMMANDS[5] = { &WALLET_COMMAND_HELP,
&WALLET_COMMAND_GENERATE_WALLET,
&WALLET_COMMAND_WALLET_VALUE,
&WALLET_COMMAND_ALL_VALUES, &WALLET_COMMAND_EXIT };



double Calculate_Wallet_Value(char * dir_path, _Wallet_Address pk, uint64_t
up_to_block_index) {

        double value = 0;


        char fp[256] = { 0 };

        char magic[4] = { 0 };


        _Block block;


        for (uint64_t i = 0; i < up_to_block_index; i++) {


        sprintf_s(fp, 256, "%s\\%llu.GCB", dir_path, i);


        FILE * f;
```

```
		fopen_s(&f, fp, "rb");


		if (f == NULL) { break; }


		fread(magic, sizeof(char), 4, f);


		if (memcmp(magic, GCB_MAGIC, 4)) { printf("Malformatted file in
directory!\n'%s' is invalid!\n", fp); break; }



		fread(&block, sizeof(char), sizeof(_Block), f);


		fclose(f);


		for (int t = 0; t < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER;
t++) {
			_Transaction * transaction = &(block.Transactions[t]);


			value += Calculate_Transaction_Change_To_Wallet(transaction, pk);
		}


		if (memcmp(block.Notary_Address, pk, sizeof(_Wallet_Address)) == 0) {
			value += Calculate_Block_Total_Miner_Fees(&block);
		}
		}
```

```
	return value;

}


int Is_Wallet_Value_Greater(char * dir_path, _Wallet_Address pk, uint64_t
up_to_block_index, double thresh) {

	double value = 0;


	char fp[256] = { 0 };

	char magic[4] = { 0 };


	_Block block;


	uint64_t i = max(0, up_to_block_index - 1);

	while (i != 0) {


	sprintf_s(fp, 256, "%s\\%llu.GCB", dir_path, i);


	FILE * f;

	fopen_s(&f, fp, "rb");


	if (f == NULL) { break; }


	fread(magic, sizeof(char), 4, f);
```

```
        if (memcmp(magic, GCB_MAGIC, 4)) { printf("Malformatted file in
directory!\n'%s' is invalid!\n", fp); break; }



        fread(&block, sizeof(char), sizeof(_Block), f);


        fclose(f);


        for (int t = 0; t < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER;
t++) {
                _Transaction * transaction = &(block.Transactions[t]);


                value += Calculate_Transaction_Change_To_Wallet(transaction, pk);
        }


        if (memcmp(block.Notary_Address, pk, sizeof(_Wallet_Address)) == 0) {
                value += Calculate_Block_Total_Miner_Fees(&block);
        }


        if (value >= thresh) { return 1; }


        i--;
        }


        value += Calculate_Wallet_Value(dir_path, pk, 0);
```

```
	return (value >= thresh);

}


struct Wallet_Piece {

	_Wallet_Address address;

	void * next;

};

int Wallet_Chain_Contains_Address(struct Wallet_Piece * wallet_chain, _Wallet_Address wallet_address) {


	struct Wallet_Piece * ptr = wallet_chain;


	while (ptr->next != NULL) {

	ptr = ptr->next;


	if (memcmp(ptr->address, wallet_address, sizeof(_Wallet_Address)) == 0) {

		return 1;

	}

	}


	return 0;

}

int Wallet_Add_Address_To_Chain(struct Wallet_Piece * wallet_chain, _Wallet_Address wallet_address) {
```

```
	struct Wallet_Piece * ptr = wallet_chain;


	while (ptr->next != NULL) {

	ptr = ptr->next;

	}


	ptr->next = (struct Wallet_Piece *)calloc(1, sizeof(struct Wallet_Piece));

	memcpy(((struct Wallet_Piece *)(ptr->next))->address, wallet_address,
sizeof(_Wallet_Address));

}

int Wallet_Chain_Recursive_Free(struct Wallet_Piece * wallet_chain) {

	if (wallet_chain->next != NULL) {

	Wallet_Chain_Recursive_Free(wallet_chain->next);

	}

	free(wallet_chain);

	return 0;

}

void Wallet_Calculate_Values() {

	double coins_in_circulation = (double)INITIAL_BLOCK_MINING_FEE;

	coins_in_circulation *= (pow((double)MINING_FEE_DECAY,
BlockChainLength) - 1);

	coins_in_circulation /= ((double)MINING_FEE_DECAY - 1);


	printf("There are currently %.20f GreenCoins in circulation.\n",
coins_in_circulation);
```

```
	struct Wallet_Piece * wallet_chain = (struct Wallet_Piece*)calloc(1, sizeof(struct Wallet_Piece));


	_Block b;

	_Wallet_Address wallet_address;

	char * address_base64_buffer;

	FILE * f;

	for (uint64_t block_ind = 0; block_ind < BlockChainLength; block_ind++) {

	error_t err = Open_Block_File(&f, block_ind);

	if (err != ERROR_NONE) { break; }

	fseek(f, 4, SEEK_SET);

	fread(&b, sizeof(_Block), 1, f);

	fclose(f);


	for (int trans_ind = 0; trans_ind < MAXIMUM_AMOUNT_OF_TRANSACTIONS_ON_LEDGER; trans_ind++) {

		memcpy(wallet_address, b.Transactions[trans_ind].Sender, sizeof(_Wallet_Address));


		if (!Wallet_Chain_Contains_Address(wallet_chain, wallet_address)) {

		Wallet_Add_Address_To_Chain(wallet_chain, wallet_address);

		}


		memcpy(wallet_address, b.Transactions[trans_ind].Receiver, sizeof(_Wallet_Address));
```

```
                if (!Wallet_Chain_Contains_Address(wallet_chain, wallet_address)) {
                Wallet_Add_Address_To_Chain(wallet_chain, wallet_address);
                }
        }

        memcpy(wallet_address, b.Notary_Address, sizeof(_Wallet_Address));

        if (!Wallet_Chain_Contains_Address(wallet_chain, wallet_address)) {
                Wallet_Add_Address_To_Chain(wallet_chain, wallet_address);
        }
        }

        struct Wallet_Piece * ptr = wallet_chain;
        while (ptr->next != NULL) {
        ptr = ptr->next;

        double value = Calculate_Wallet_Value(BLOCK_HISTORY_DIRECTORY_PATH,
ptr->address, -1);

        B64_Encode(ptr->address, sizeof(_Wallet_Address), &address_base64_buffer);
        printf("%s\t : %.2f%%\t : Value: %.20f\n", address_base64_buffer,
((100.0*value)/coins_in_circulation), value);
        free(address_base64_buffer);
        }
```

```
	Wallet_Chain_Recursive_Free(wallet_chain);
}

void Wallet_Calculate_Value_CommandLine() {
	printf("Please enter wallet address (as base64): \n");
	char buffer[256];
	fgets(buffer, sizeof(buffer), stdin);
	byte * address;
	B64_Decode(buffer, &address);

	double value = Calculate_Wallet_Value(BLOCK_HISTORY_DIRECTORY_PATH, address, -1);

	printf("This wallet has %.20f GreenCoin.\n", value);
}

void Print_Demo_Keys() {
	BN * p = params->p;
	BN * q = params->q;
	BN * G = params->G;

	DSA_Private_Key * priv_key = 0;
	DSA_Public_Key * pub_key = 0;
```

```
        DSA_Init_Private_Key(&priv_key);
        DSA_Init_Public_Key(&pub_key);

        //DSA_Load_Private_Key(priv_key, p, q, G);
        //DSA_Load_Public_Key(pub_key, p, q, G);

        char * buffer;

        DSA_Create_Keys(priv_key, pub_key);
        printf("  x:\r\n");
        TRACE_MPI("          ", (priv_key));
        B64_Encode(priv_key->data, priv_key->size * BN_INT_SIZE, &buffer);
        printf("X B64: '%s'\n", buffer);
        free(buffer);

        printf("  y:\r\n");
        TRACE_MPI("          ", (pub_key));
        B64_Encode(pub_key->data, pub_key->size * BN_INT_SIZE, &buffer);
        printf("Y B64: '%s'\n", buffer);
        free(buffer);
}

DSA_Public_Key * Get_Random_Public_Key() {
        BN * p = params->p;
```

```
BN * q = params->q;
BN * G = params->G;

DSA_Private_Key * priv_key = 0;
DSA_Public_Key * pub_key = 0;

DSA_Init_Private_Key(&priv_key);
DSA_Init_Public_Key(&pub_key);

//DSA_Load_Private_Key(priv_key, p, q, G);
//DSA_Load_Public_Key(pub_key, p, q, G);

char * buffer;

DSA_Create_Keys(priv_key, pub_key);
/*printf("  x:\r\n");
TRACE_MPI("          ", (priv_key->x));
B64_Encode(priv_key->x->data, priv_key->x->size * BN_INT_SIZE, &buffer);
printf("X B64: '%s'\n", buffer);
free(buffer);

printf("  y:\r\n");
TRACE_MPI("          ", (pub_key->y));
B64_Encode(pub_key->y->data, pub_key->y->size * BN_INT_SIZE, &buffer);
```

```
        printf("Y B64: '%s'\n", buffer);
        free(buffer);*/

        return pub_key;
}

int wallet_exit = 0;
int Wallet_Exit() {
        wallet_exit = 1;
        return wallet_exit;
}

int Wallet_Help() {
        printf("Wallet help menu: \n");
        for (int i = 0; i < sizeof(WALLET_COMMANDS) /
sizeof(WALLET_COMMANDS[0]); i++) {
        command_t * command = WALLET_COMMANDS[i];
        printf("\t%s", command->command_text);
        int text_len = strlen(command->command_text);
        for (int j = 0; j < 3 - floor((text_len) / 4.0); j++) { printf("\t"); }
        printf("| %s\n", command->command_description);
        }
        return 0;
}
```

```
int Wallet_CommandLine_General() {

	printf("Now in wallet command line.\nType 'exit' to return to general command line.\n");


	wallet_exit = 0;


	char buffer[1024] = { 0 };


	while (!wallet_exit) {

	printf("> ");

	memset(buffer, 0, sizeof(buffer));

	fgets(buffer, sizeof(buffer), stdin);

	if (buffer[strlen(buffer) - 1] == '\n') { buffer[strlen(buffer) - 1] = 0x00; }


	for (int i = 0; i < sizeof(WALLET_COMMANDS) / sizeof(WALLET_COMMANDS[0]); i++) {

		if (strcmp(buffer, WALLET_COMMANDS[i]->command_text) == 0) {

		WALLET_COMMANDS[i]->function(NULL, NULL);

		}

	}

	}


	printf("Exiting wallet command line.\n");

}
```

Network

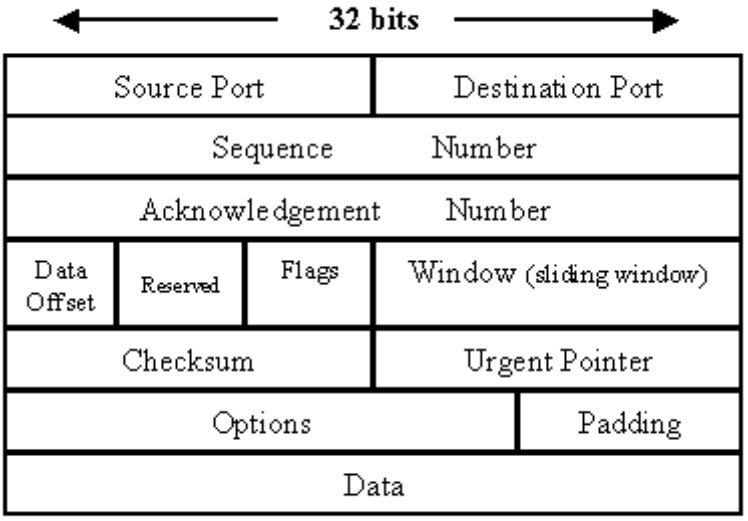
Network communication is one of the most important functions of modern computer systems. Network communication over the internet allows computers which are physically very far apart to communicate with each other.

Some basic computer network concepts:

Basic Network Concepts:

Packets

Network packets are formatted payloads of data which are transmitted over a network. These packets will follow the rules of the protocol through which they are being transmitted, and will generally contain important metadata, such as sender and receiver IPs and ports.



Additionally, a packet, in contrast to a normal bunch of data, will have some form of error detection and / or correction. This error detection could be implemented using a simple checksum - a hash of the data appended to the original data, with which one can compare the hash of the received data. Error correction has more applicability, and is slightly more advanced. The inception of error correcting schemes is credited to Richard Hamming, who invented the Hamming Code, which can detect up to two errors and correct one using linear algebra matrix multiplication - a computation modern computers can easily perform.

$$H_{HammingSEC-DED} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \dots & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \dots & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \dots & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

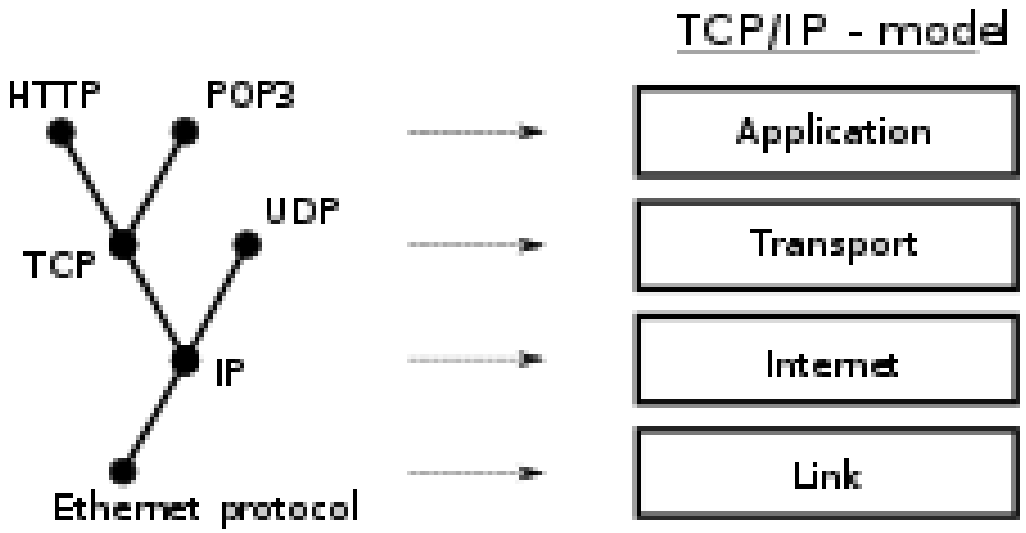
(a)

$$H_{HsiaoSEC-DED} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & \dots & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \dots & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & \dots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)

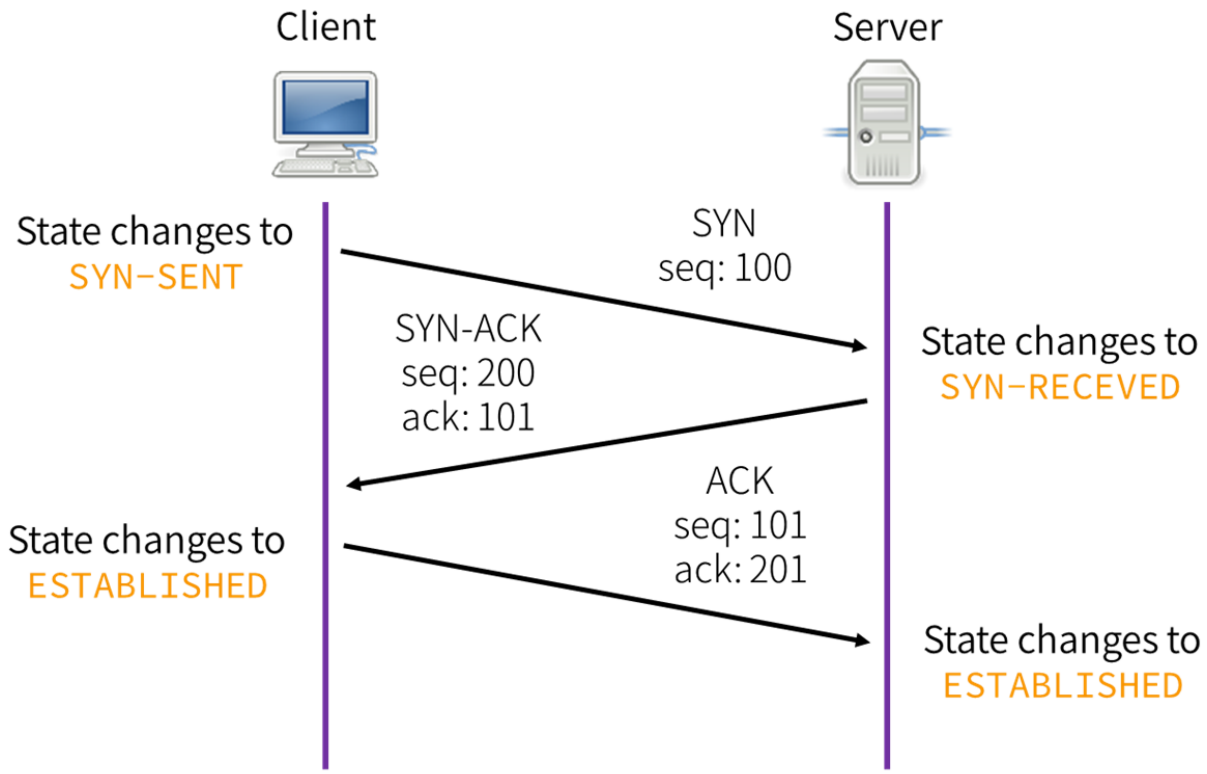
Internet Protocol Suite

Internet Protocol Suite is a collection of protocols used to communicate over the internet. This collection can also be called TCP/IP. The two important protocols in the suite are TCP, Transmission control Protocol, and UDP, User Datagram Protocol. From these protocols, other more application specific protocols have branched:



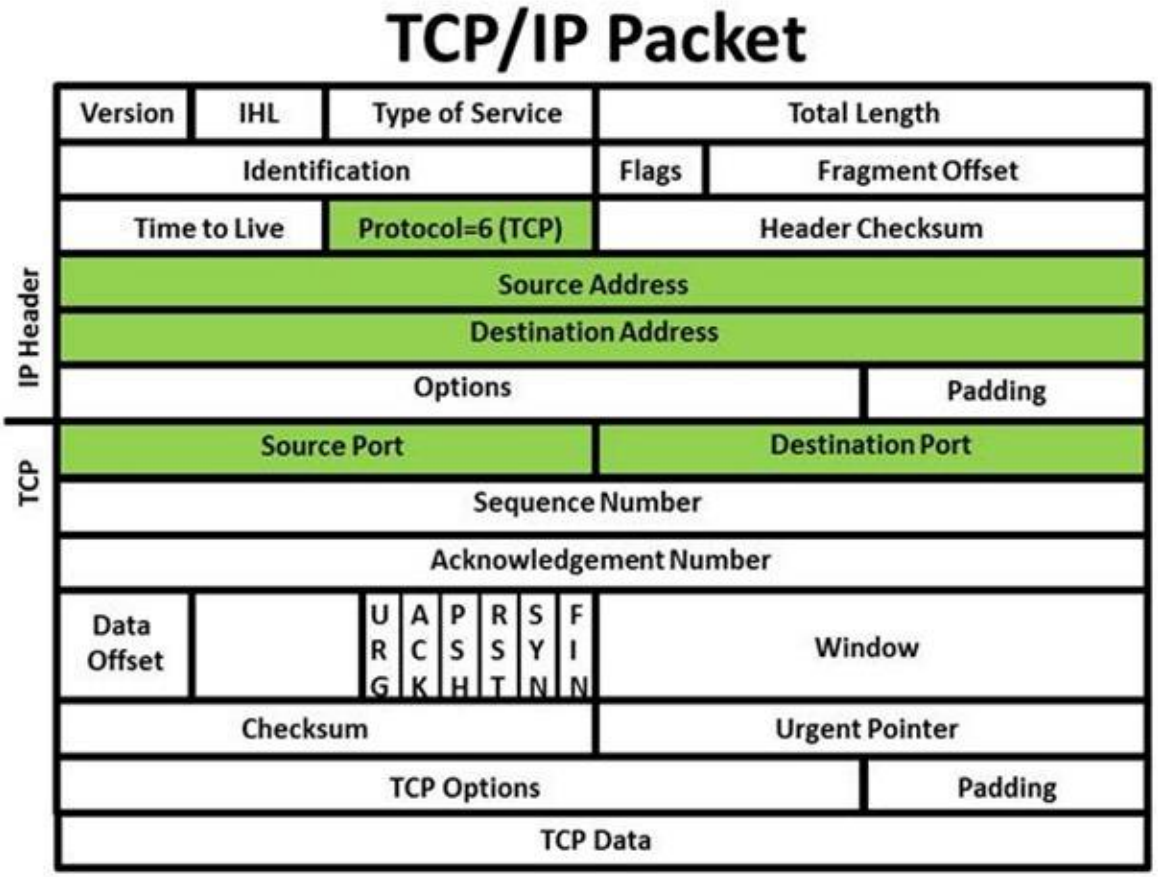
TCP - Transmission Control Protocol

TCP, the protocol GreenCoin uses for all communication, is a very reliable protocol which requires a connection between the client and server. The TCP connection is implemented using a three-way handshake:



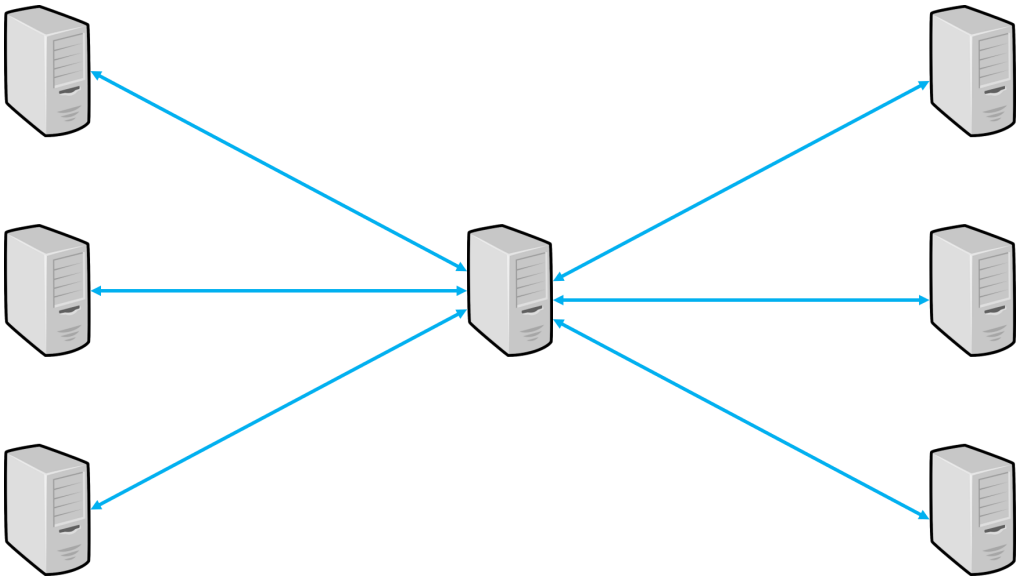
The redundancy checks in the TCP protocol allow us to know whether a packet arrived at its destination, and be sure that the packet was received accurately. In contrast, UDP (User Datagram Protocol), which is much faster, does not provide us this information. Therefore, GreenCoin uses TCP, to ensure all information is received exactly as it was sent, since each and every bit is critical when working with high level encryption. UDP, the runner-up, is more often used when a constant stream of data which is updated at a very high frequency is required, such as in live multiplayer games or video/audio streaming.

Each TCP packet contains a whole load of important metadata which aides the application in signal interference checking:



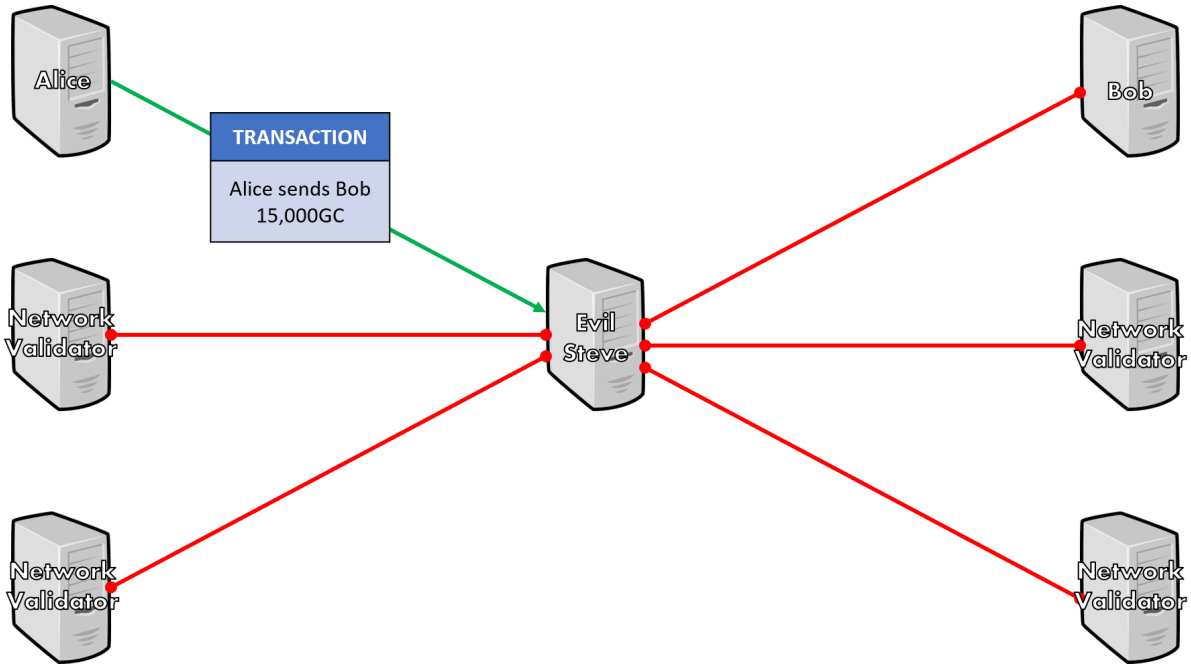
P2P - Peer to Peer

A critical point in the whole concept of a decentralized digital currency is, obviously, that it is decentralized. This means that network data should never go through only a single node; there should never be a single entity through which all data goes.



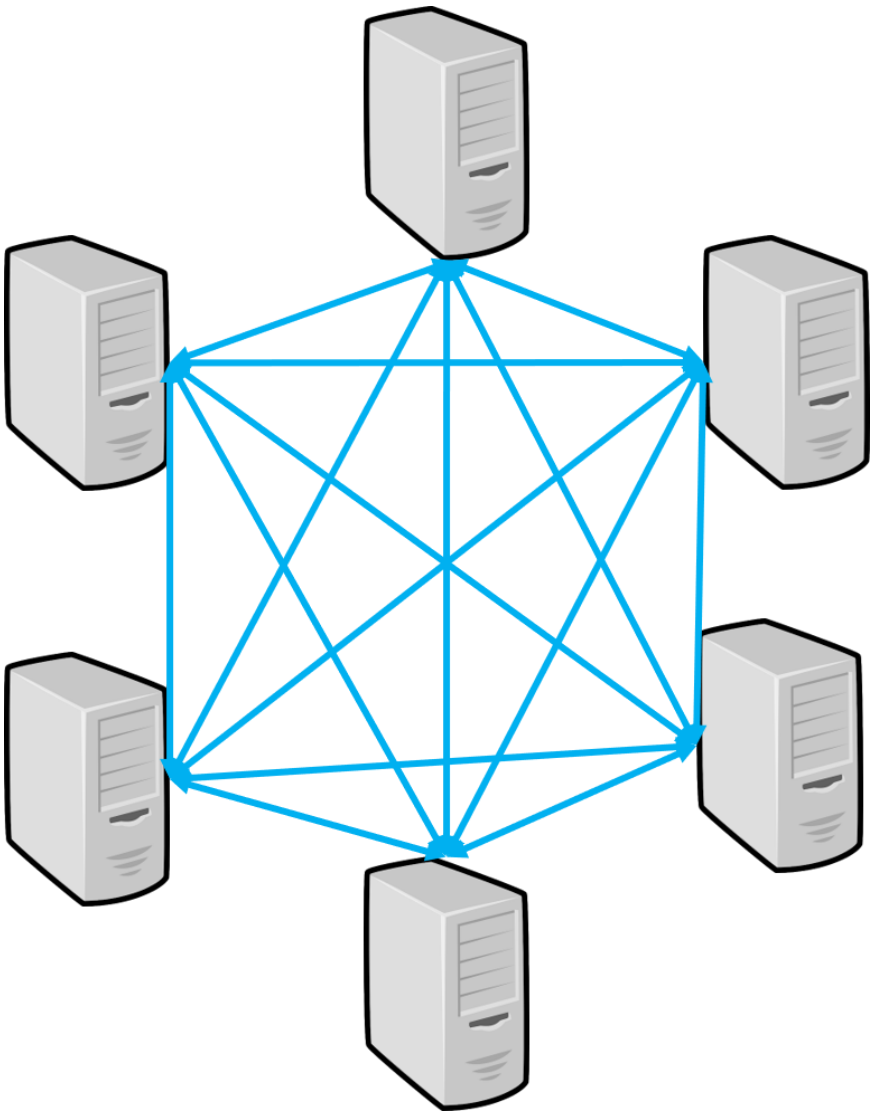
If all data goes through a single entity, and this entity happens to have nefarious intentions, something like this could happen:

- Assume Steve hates Bob with a burning passion.
- Alice and Bob conduct some business, and Alice agrees to pay Bob 15,000GC.
- Alice sends the transaction over the network, possibly multiple times.
- Steve, who is the bottleneck of the network, rejects the transaction and does not forward it.

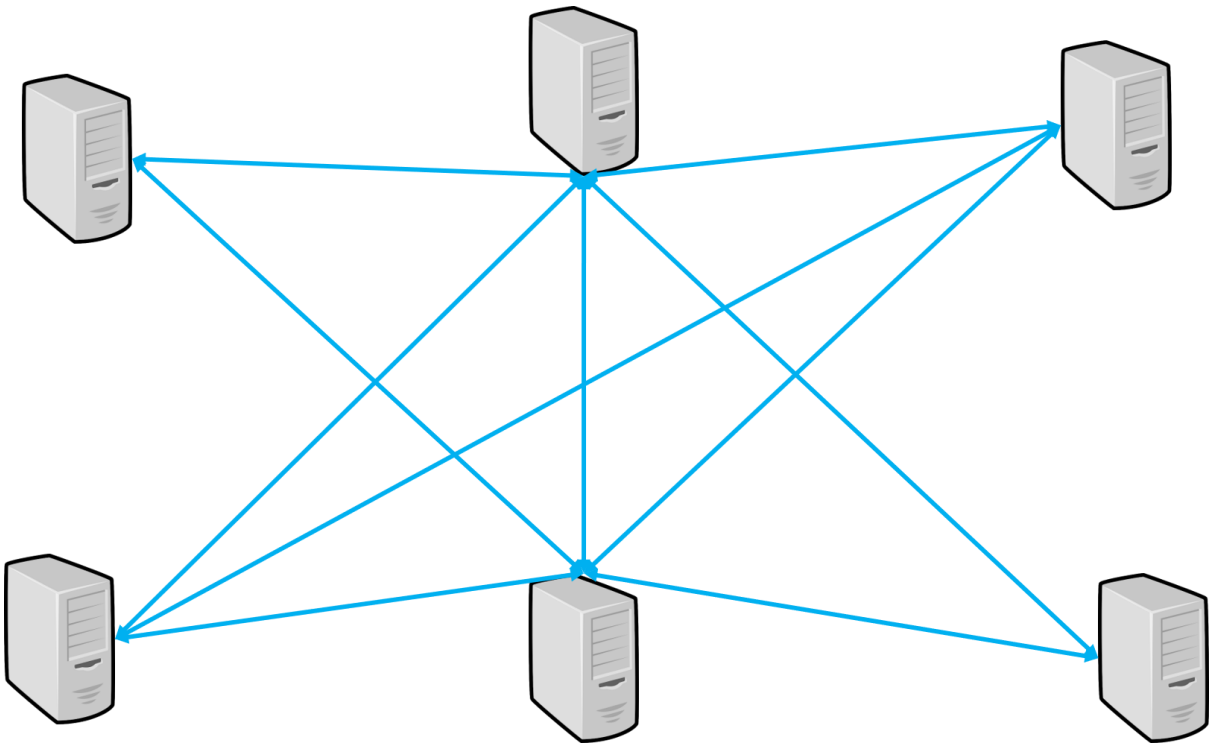


Steve could now practically perform a siege on Bob, cutting him off from the rest of the network. If Steve is actually a dictatorship or other authoritarian regime, this could result as being a very severe issue, where Steve dwindles rivals and opposition of funds.

A proper decentralized network should look more interconnected, where ideally all nodes are directly interconnected:



While this fully interconnected network layout is absolutely ideal and would completely eliminate any single entity's ability to control communication, it is not very practical and on a large scale will probably not be achievable. A more realistic network layout would be a "mesh" network, a network layout where nodes are interconnected randomly in a way where any two nodes can communicate through some other nodes. A good mesh could look like this:

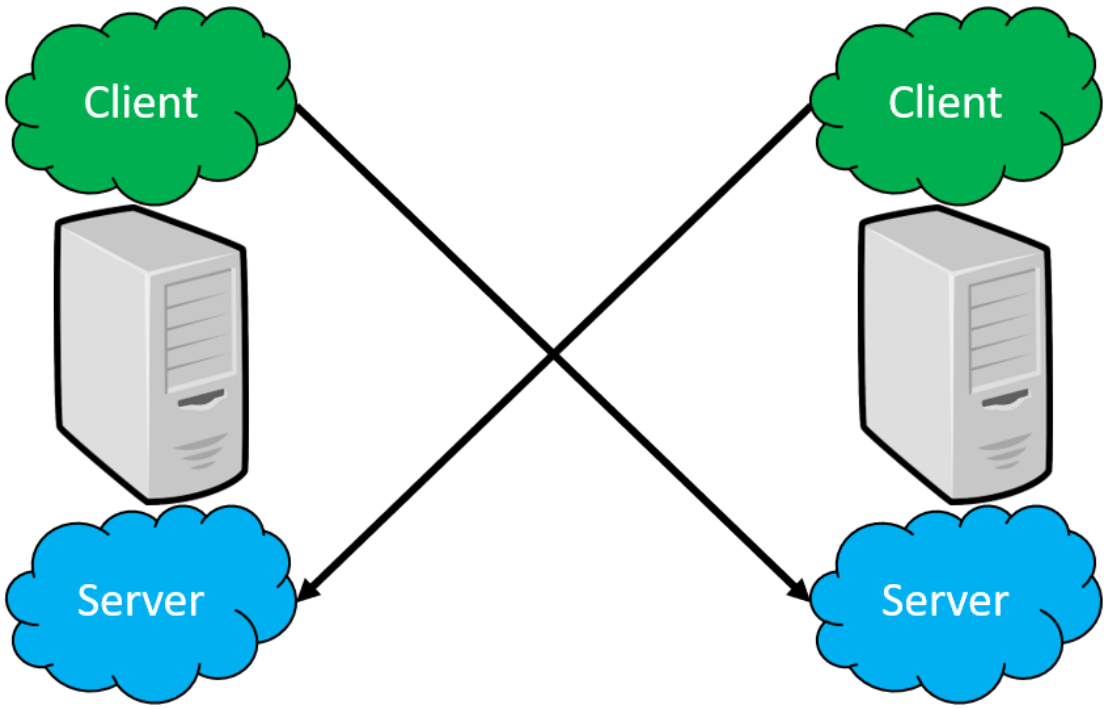


In this layout, not every two nodes are connected, but there are at least two paths of connections between every two nodes, meaning that if most of the nodes are reliable and compliant, communication will always be achievable, even if a few entities are malicious.

GreenCoin's implementation of a peer-to-peer network is as follows:

Each node hosts a main TCP server on a globally defined port (22110). This server has some predefined and commonly understood functions, such as returning "MOON" when receiving "GreenCoin". This functionality can be used to quickly check if a node is running a GreenCoin server.

When the server receives a P2P connection request, defined as P2P_CHANNEL_CONNECTION_INIT_MESSAGE in Network.h and is the phrase "Ahoy!" (ASCII { 0x41, 0x68, 0x6f, 0x79, 0x21 }), the receiver opens a new socket and connects to the sender's main server. On the sender's side, a new socket is opened and connects to the receiver's main server. This mutual connection allows either peer to send messages through their client connection to the other peer's server, and neither node is only a server or only a client.



This method means that each node opens plenty of sockets: one for their main server, and another socket for every network peer they connect to, ideally, many. To organize all the peer sockets, a dynamically allocated linked list is used. This list contains structures which hold the IP of the peer and the client socket used to send them data. There is a function implementation for broadcasting to all peers, which is the primary use of the network. This function loops over the peer socket list and sends the data to each one. Alternatively, if a message is destined for only one peer, say they requested some information, the loop searches for a peer with the desired IP address and sends only them the message.

Code

Network.h + Network.c

```
#pragma once


#ifndef __NETWORK_H
#define __NETWORK_H


#include "../General/error.h"
#include "../General/FileIO.h"


#include "../BlockChain/BlockChain.h"


#include <string.h>


#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <WinSock2.h>
//#include <ws2tcpip.h>
#include <stdio.h>


#include "../General/Print/PrettyPrint.h"


/*
	=== Node Communications Protocol Manual ===
```

```
        1) All communication is on TCP.

        2) A connected node will always respond "MOON" {0x4d, 0x4f, 0x4f, 0x4e} to the message "GreenCoin" {0x47, 0x72, 0x65, 0x65, 0x6e, 0x43, 0x6f, 0x69, 0x6e}.

        3) Start a P2P channel by sending the string "Ahoy!" to a connected node on port 22110.

*/


char P2P_CHANNEL_CONNECTION_INIT_MESSAGE[5];

char CONNECTED_TEST_MESSAGE[10];

char CONNECTED_TEST_MESSAGE_RESPONSE[4];


char TRANSACTION_BROADCAST_MAGIC[4];

char BLOCK_BROADCAST_MAGIC[4];


char BLOCK_REQUEST_MAGIC[4];


char BLOCKCHAIN_QUERY_MAGIC[4];


unsigned int GENERAL_SERVER_PORT;// = 22110;


char Network_Nodes_List_File_Path[256];


unsigned char LOCALHOST_IP[4];
```

```
typedef struct Main_Server_Thread_Params Main_Server_Thread_Params;

typedef struct P2P_Thread_Params P2P_Thread_Params;


typedef struct Network_TCP_Connect_Thread_Params
Network_TCP_Connect_Thread_Params;


typedef struct Node_Peer Node_Peer;

// A linked list of nodes holding pointers to the sockets for P2P connections

Node_Peer * Node_List;// = NULL;


void Copy_Node_To_List(SOCKET * socket, unsigned char * ip);

void Copy_Socket_To_List(SOCKET * socket);


error_t Network_Init(WSADATA * ptr_WSA_Data, SOCKET * ptr_Sending_Socket);


error_t Network_Print_IP();


unsigned long Network_Node_Addr_Format(unsigned char * node_addr);


error_t Network_Send_Message_Await_Return(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * node_addr, char * message, int message_len, char **
output, int * output_len);


error_t Network_Query_Node_Nodes_List(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * node_addr);
```

```
error_t Network_Locate_Nodes(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket);



error_t Network_TCP_Connect(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * node_addr);

DWORD WINAPI Network_TCP_Connect_Thread(void * params);



DWORD WINAPI Network_Main_Server_Thread(/*Main_Server_Thread_Params*/ void
* param);

error_t Network_Main_Server(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * server_addr);



error_t Network_Add_New_Peer_To_Node_File(unsigned char * ip);

DWORD WINAPI Network_P2P_Thread(void * param);

error_t Network_P2P(WSADATA * ptr_WSA_Data, SOCKET * ptr_Sending_Socket,
SOCKET * ptr_Peer_Socket);



error_t Network_Broadcast_Transaction(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * transaction, int transaction_size);

error_t Network_Broadcast_Block(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * block, int block_size);



error_t Network_Transaction_Recieved(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * transaction, int transaction_size);

error_t Network_Block_Recieved(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, _Block * block, int block_size);



error_t Network_Block_Request(WSADATA * ptr_WSA_Data, SOCKET *
```

```
ptr_Sending_Socket, void * data, int data_size, unsigned char * client_ip);


error_t Network_Request_Block(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, uint64_t block_ind);


int Network_Send_To_Peer(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * data, int data_size, unsigned char * client_ip);


void Network_CommandLine_Init(WSADATA ** wsadata, SOCKET ** socket);

HANDLE Network_CommandLine_Server(WSADATA * wsadata, SOCKET * socket);


error_t Network_Connect_To_Known_Peers();


void Recursive_Free(struct piece_t * ptr);

uint64_t Network_Request_Blockchain_Length(WSADATA * wsadata, SOCKET *
socket);

void Network_Request_Blockchain_Length_Print(WSADATA * wsadata, SOCKET *
socket);


error_t Network_CommandLine_Request_Blocks(WSADATA * wsadata, SOCKET *
socket);


HANDLE Network_Demo(WSADATA * wsadata, SOCKET * socket);


#endif
```

```
#include "Network.h"

char P2P_CHANNEL_CONNECTION_INIT_MESSAGE[5] = "Ahoy!";
char CONNECTED_TEST_MESSAGE[10] = "GreenCoin";
char CONNECTED_TEST_MESSAGE_RESPONSE[4] = "MOON";

char TRANSACTION_BROADCAST_MAGIC[4] = { "GCT3" };
char BLOCK_BROADCAST_MAGIC[4] = { "GCB3" };

char BLOCK_REQUEST_MAGIC[4] = { "GCBR" };

char BLOCKCHAIN_QUERY_MAGIC[4] = { "GCBL" };

unsigned int GENERAL_SERVER_PORT = 22110;

unsigned char LOCALHOST_IP[4] = { 127, 0, 0, 1 };

char Network_Nodes_List_File_Path[256] = "Nodes.GCNL";

struct Main_Server_Thread_Params {
        WSADATA * ptr_WSA_Data;
        SOCKET * ptr_Sending_Socket;
        unsigned char * server_addr;
```

```
	int ret;
};

struct P2P_Thread_Params {
	WSADATA * ptr_WSA_Data;
	SOCKET * ptr_Sending_Socket;
	SOCKET * client_socket;
};

struct Node_Peer {
	unsigned char ip[4];
	SOCKET * socket;
	Node_Peer * next_node;
};
Node_Peer * Node_List = NULL;

struct Network_TCP_Connect_Thread_Params {
	WSADATA * ptr_WSA_Data;
	SOCKET * ptr_Sending_Socket;
	unsigned char node_addr[4];
};

void Copy_Node_To_List(SOCKET * socket, unsigned char * ip) {
	Node_Peer * ptr = Node_List;
```

```
        if (ptr->socket == NULL) {
        ptr->socket = (SOCKET *)malloc(sizeof(SOCKET));
        memcpy(ptr->socket, socket, sizeof(SOCKET));
        memcpy(ptr->ip, ip, 4);
        }
        else {
        while (ptr->next_node != NULL) { ptr = ptr->next_node; }

        ptr->next_node = (Node_Peer*)calloc(1, sizeof(Node_Peer));

        Node_Peer * new_node = ptr->next_node;
        new_node->next_node = NULL;
        new_node->socket = (SOCKET *)malloc(sizeof(SOCKET));
        memcpy(new_node->socket, socket, sizeof(SOCKET));
        memcpy(new_node->ip, ip, 4);
        }
}

void Copy_Socket_To_List(SOCKET * socket) {
        unsigned char tmp_ip[4] = { 0 };
        SOCKADDR_IN addr;
        int addr_len = sizeof(addr);
        getpeername(*socket, &addr, &addr_len);
```

```
		memcpy(tmp_ip, &addr.sin_addr.S_un, sizeof(tmp_ip));

		Copy_Node_To_List(socket, tmp_ip);
}

error_t Network_Init(WSADATA * ptr_WSA_Data, SOCKET * ptr_Sending_Socket) {
		printf_Info("Now initializing network WSA!\n");

		WSADATA			WSA_Data;
		SOCKET		Sending_Socket;

		// Server/Reciever address
		SOCKADDR_IN		Server_Addr, This_Sender_Info;

		WSAStartup(MAKEWORD(2, 2), &WSA_Data);
		printf_Info("Winsock DLL status is %s.\n", WSA_Data.szSystemStatus);

		// Create socket to make a client connection
		Sending_Socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
		if (Sending_Socket == INVALID_SOCKET) {
		printf_Error("Could not create socket! Error code: %ld\n", WSAGetLastError());

		// Do the clean up
```

```
        WSACleanup();


        // Exit with error
        return ERROR_NETWORK_FAILED;
        }
        else {
        printf_Success("Socket creation successful.\n");
        }


        BOOL bOptVal = FALSE;
        int bOptLen = sizeof(BOOL);
        int iOptVal = 0;
        int iOptLen = sizeof(int);
        bOptVal = TRUE;
        int iResult = setsockopt(Sending_Socket, SOL_SOCKET, SO_KEEPALIVE, (char
*)&bOptVal, bOptLen);


        memcpy(ptr_WSA_Data, &WSA_Data, sizeof(WSADATA));
        memcpy(ptr_Sending_Socket, &Sending_Socket, sizeof(SOCKET));


        Node_List = (Node_Peer*)malloc(sizeof(Node_Peer));
        Node_List->next_node = NULL;
        Node_List->socket = NULL;


        return ERROR_NETWORK_NONE;
```

```
}

error_t Network_Print_IP() {
        char hostbuffer[256];
        char *IPbuffer;
        struct hostent *host_entry;
        int hostname;

        // To retrieve hostname
        hostname = gethostname(hostbuffer, sizeof(hostbuffer));

        // To retrieve host information
        host_entry = gethostbyname(hostbuffer);

        printf("--- Your local network info ---\n");
        printf("Hostname: %s\n", hostbuffer);

        char** addr = host_entry->h_addr_list;

        // To convert an Internet network
        // address into ASCII string
        while (*addr!=NULL) {
        IPbuffer = inet_ntoa(*((struct in_addr*)
                *addr));
```

```
		printf("\tIP: %s\n", IPbuffer);
		addr++;
		}
		printf("--- --- --- --- --- --- --- ---\n");
}

unsigned long Network_Node_Addr_Format(unsigned char * node_addr) {
		return (unsigned long)(node_addr[0] | node_addr[1] << 8 | node_addr[2] << 16 | node_addr[3] << 24);
}

error_t Network_Send_Message_Await_Return(WSADATA * ptr_WSA_Data, SOCKET * ptr_Sending_Socket, unsigned char * node_addr, char * message, int message_len, char ** output, int * output_len) {
		// Server/Reciever address
		SOCKADDR_IN		Server_Addr, This_Sender_Info;

		// Server/Reciever port to connect to
		unsigned int  Port = 10087;
		int Return_Code;

		int Bytes_Sent, nlen;

		// IPv4
```

```
        Server_Addr.sin_family = AF_INET;


        // Port no.

        Server_Addr.sin_port = htons(Port);


        // The IP address

        Server_Addr.sin_addr.s_addr =
Network_Node_Addr_Format(node_addr);//inet_addr("127.0.0.1");


        Return_Code = connect(*ptr_Sending_Socket, (SOCKADDR*)&Server_Addr,
sizeof(Server_Addr));

        if (Return_Code != 0) {

        printf("Client: connect() failed! Error code: %ld\n", WSAGetLastError());

        // Exit with error

        return -1;

        }

        else {

        printf("Client: connect() is OK, got connected...\n");

        printf("Client: Ready for sending and/or receiving data...\n");

        }



        // At this point you can start sending or receiving data on

        // the socket SendingSocket.
```

```
// Some info on the receiver side...
int ilen = sizeof(Server_Addr);
getsockname(*ptr_Sending_Socket, (SOCKADDR *)&Server_Addr, &ilen);
printf("Client: Receiver IP(s) used: %s\n", inet_ntoa(Server_Addr.sin_addr));
//printf("Client: Receiver IP(s) used: %s\n", inet_ntop(Server_Addr.sin_addr));

printf("Client: Receiver port used: %d\n", htons(Server_Addr.sin_port));
// Sends some data to server/receiver...
Bytes_Sent = send(*ptr_Sending_Socket, message, message_len, 0);

if (Bytes_Sent == SOCKET_ERROR) {
printf("Client: send() error %ld.\n", WSAGetLastError());
}
else {
printf("Client: send() is OK - bytes sent: %ld\n", Bytes_Sent);
// Some info on this sender side...
// Allocate the required resources
memset(&This_Sender_Info, 0, sizeof(This_Sender_Info));
nlen = sizeof(This_Sender_Info);

getsockname(*ptr_Sending_Socket, (SOCKADDR *)&This_Sender_Info, &nlen);
//printf("Client: Sender IP(s) used: %s\n", inet_ntop(This_Sender_Info.sin_addr));
printf("Client: Sender IP(s) used: %s\n", inet_ntoa(This_Sender_Info.sin_addr));
printf("Client: Sender port used: %d\n", htons(This_Sender_Info.sin_port));
```

```
		printf("Client: Those bytes represent: \"%s\"\n", message);


		char buffer[1024] = { 0 };
		int Bytes_Recieved = 0;
		while (!Bytes_Recieved) {
			Bytes_Recieved = recv(*ptr_Sending_Socket, buffer, sizeof(buffer), 0);
		}
		printf("Recieved %d bytes: '%s'\n", Bytes_Recieved, buffer);
		}
}


error_t Network_Query_Node_Nodes_List(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * node_addr) {


}


error_t Network_Locate_Nodes(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket) {
		printf_Info("Locating network nodes...\n");


		char * buffer;
		int size = Load_File(Network_Nodes_List_File_Path, &buffer);


		HANDLE * handles = (HANDLE*)calloc(size / 5, sizeof(HANDLE));
		int handle_ind = 0;
```

```
		if (size > 0) {
		// For each entry
		int ind = 0;
		// 4 bytes for address, 1 byte for status
		unsigned char addr_bytes[5];
		while (ind < size) {
			memcpy(addr_bytes, buffer + ind, 5); ind += 5;
			if (memcmp(addr_bytes, LOCALHOST_IP, 4) == 0) { continue; }
			if (addr_bytes[4] == 0xff) {
			// A node with status 0xff is extremely weak, and therefore will not be checked.
			}
			else {
			Network_TCP_Connect_Thread_Params * params = (Network_TCP_Connect_Thread_Params*)malloc(sizeof(Network_TCP_Connect_Thread_Params));
			params->ptr_WSA_Data = ptr_WSA_Data;
			params->ptr_Sending_Socket = ptr_Sending_Socket;
			memcpy(params->node_addr, addr_bytes, sizeof(params->node_addr));

			handles[handle_ind++] = CreateThread(NULL, 0, Network_TCP_Connect_Thread, params, 0, NULL);
			 //Network_TCP_Connect(ptr_WSA_Data, ptr_Sending_Socket, addr_bytes);
```

```
                }
        }
        }

        if (size > 0) {
        free(buffer);
        free(handles);
        }
}

error_t Network_TCP_Connect(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * node_addr) {
        SOCKADDR_IN my_info;
        SOCKET my_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (my_sock == INVALID_SOCKET) {
        //printf("Client: socket() failed! Error code: %ld\n", WSAGetLastError());
        // Exit with error
        return ERROR_NETWORK_FAILED;
        }
        else {
        //printf("Client: socket() successful.\n");
        }

        // IPv4
        my_info.sin_family = AF_INET;
```

```
	// Port no.

	my_info.sin_port = htons(GENERAL_SERVER_PORT);

	// The IP address

	my_info.sin_addr.s_addr = Network_Node_Addr_Format(node_addr);

	char * server_ip = inet_ntoa(my_info.sin_addr);


	DWORD recv_time_out = 1500;

	setsockopt(my_sock, SOL_SOCKET, SO_RCVTIMEO, &recv_time_out,
sizeof(recv_time_out));


	int err = connect(my_sock, (SOCKADDR*)&my_info, sizeof(my_info));

	if (err != 0) { /*printf_Error("Could not connect to server node %s!\n",
server_ip);*/ }

	else {

	printf_Success("Succesfuly connected to %s\n", server_ip);

	Copy_Socket_To_List(&my_sock);

	}


	//free(server_ip);

}

DWORD WINAPI Network_TCP_Connect_Thread(void * params) {

	Network_TCP_Connect_Thread_Params * thread_params =
(Network_TCP_Connect_Thread_Params*)params;

	error_t err =
Network_TCP_Connect((WSADATA*)(thread_params->ptr_WSA_Data),
(SOCKET*)(thread_params->ptr_Sending_Socket), (unsigned char
*)(thread_params->node_addr));
```

```
	free(params);

	return err;

}


DWORD WINAPI Network_Main_Server_Thread(/*Main_Server_Thread_Params*/ void
* param) {

	//WSADATA * ptr_wsadata =
((Main_Server_Thread_Params*)params)->ptr_WSA_Data;

	int ret =
Network_Main_Server(((Main_Server_Thread_Params*)param)->ptr_WSA_Data,
((Main_Server_Thread_Params*)param)->ptr_Sending_Socket,
((Main_Server_Thread_Params*)param)->server_addr);

	(((Main_Server_Thread_Params*)param)->ret) = ret;

	return 0;

}


error_t Network_Main_Server(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, unsigned char * server_addr) {

	int iResult = 0;

	struct addrinfo * result = NULL;


	SOCKADDR_IN	Server_Addr;

	unsigned int  Port = GENERAL_SERVER_PORT;

	// IPv4

	Server_Addr.sin_family = AF_INET;

	// Port no.

	Server_Addr.sin_port = htons(Port);
```

```
        // The IP address

        Server_Addr.sin_addr.s_addr =
Network_Node_Addr_Format(server_addr);//inet_addr("127.0.0.1");



        iResult = bind(*ptr_Sending_Socket, (SOCKADDR*)&Server_Addr,
sizeof(Server_Addr));

        if (iResult == SOCKET_ERROR) {

        printf_Error("Bind failed with error: %d\nCould not start server!\n",
WSAGetLastError());

        return 1;

        }



        iResult = listen(*ptr_Sending_Socket, SOMAXCONN);

        if (iResult == SOCKET_ERROR) {

        printf_Error("Listen failed with error: %d\nCould not start server!\n",
WSAGetLastError());

        return 1;

        }



        printf_Info("Server is active. Awaiting connection.\n");



        while (1) {

        // Accept a client socket

        SOCKET ClientSocket = accept(*ptr_Sending_Socket, NULL, NULL);

        if (ClientSocket == INVALID_SOCKET) {

                //printf("accept failed with error: %d\n", WSAGetLastError());
```

```
		//return 1;
	}


	unsigned char client_ip[4];
	SOCKADDR_IN addr;
	int addr_len = sizeof(addr);
	getpeername(ClientSocket, &addr, &addr_len);
	memcpy(client_ip, &addr.sin_addr.S_un, sizeof(client_ip));




	// Receive until the peer shuts down the connection
	char recvbuf[32768] = { 0 };
	int recvbuflen = sizeof(recvbuf);
	int iSendResult = 0;
	do {
		memset(recvbuf, 0, recvbuflen);
		iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
		if (iResult > 0) {
		//printf("Bytes received: %d\n", iResult);
		//printf("Recieved: '%s'\n", recvbuf);

		if (strcmp(recvbuf, P2P_CHANNEL_CONNECTION_INIT_MESSAGE)
== 0) {
			// The client has sent the
```

```
P2P_CHANNEL_CONNECTION_INIT_MESSAGE,
                        //  meaning they want to begin communicating over P2P.
                        P2P_Thread_Params param;
                        param.ptr_WSA_Data = ptr_WSA_Data;
                        param.ptr_Sending_Socket = ptr_Sending_Socket;
                        param.client_socket = &ClientSocket;
                        HANDLE thread = CreateThread(0,0,Network_P2P_Thread,
&param,0,0);
                }
                else if (strcmp(recvbuf, CONNECTED_TEST_MESSAGE) == 0) {
                        // The client has queried the server with the connection test message.
                        // Return the protocol-wide response to confirm that the server is
valid.
                        iSendResult = send(ClientSocket,
CONNECTED_TEST_MESSAGE_RESPONSE,
sizeof(CONNECTED_TEST_MESSAGE_RESPONSE), 0);
                        if (iSendResult == SOCKET_ERROR) {
                                printf_Error("Failed to respond %s to querying client! Error
code: %d\n", CONNECTED_TEST_MESSAGE_RESPONSE, WSAGetLastError());
                                closesocket(ClientSocket);
                                //return 1;
                        }
                }
                else if (memcmp(recvbuf, TRANSACTION_BROADCAST_MAGIC,
sizeof(TRANSACTION_BROADCAST_MAGIC)) == 0) {
                        Network_Transaction_Recieved(ptr_WSA_Data,
ptr_Sending_Socket, recvbuf + sizeof(TRANSACTION_BROADCAST_MAGIC), iResult
- sizeof(TRANSACTION_BROADCAST_MAGIC));
```

```
                        //Print_Transaction(stderr, recvbuf +
sizeof(TRANSACTION_BROADCAST_MAGIC));

                }

                else if (memcmp(recvbuf, BLOCK_BROADCAST_MAGIC,
sizeof(BLOCK_BROADCAST_MAGIC)) == 0) {

                        Network_Block_Recieved(ptr_WSA_Data, ptr_Sending_Socket,
recvbuf + sizeof(BLOCK_BROADCAST_MAGIC), sizeof(_Block));

                        //Print_Block(stderr, recvbuf +
sizeof(BLOCK_BROADCAST_MAGIC));

                }

                else if (memcmp(recvbuf, BLOCK_REQUEST_MAGIC,
sizeof(BLOCK_REQUEST_MAGIC)) == 0) {

                        Network_Block_Request(ptr_WSA_Data, ptr_Sending_Socket,
recvbuf + sizeof(BLOCK_REQUEST_MAGIC), iResult -
sizeof(BLOCK_REQUEST_MAGIC), client_ip);

                }else if (memcmp(recvbuf, BLOCKCHAIN_QUERY_MAGIC,
sizeof(BLOCKCHAIN_QUERY_MAGIC)) == 0) {

                        uint64_t blockchain_length = Get_Blockchain_Length();

                        char buffer[8] = { 0 }; memcpy(buffer, &blockchain_length,
sizeof(blockchain_length));

                        send(ClientSocket, buffer, sizeof(buffer), 0);

                }

                }

                else if (iResult == 0) {

                printf_Info("Connection to %hhu.%hhu.%hhu.%hhu closing...\n",
client_ip[0],client_ip[1],client_ip[2],client_ip[3]);

                }

                else {
```

```
                printf_Error("Recieve failed with error: %d from client
%hhu.%hhu.%hhu.%hhu\n", WSAGetLastError(), client_ip[0], client_ip[1], client_ip[2],
client_ip[3]);

                closesocket(ClientSocket);

                //return 1;

                }


        } while (iResult > 0);


        // shutdown the connection since we're done

        iResult = shutdown(ClientSocket, SD_SEND);

        if (iResult == SOCKET_ERROR) {

                printf_Error("Client (%hhu.%hhu.%hhu.%hhu) shutdown failed with error:
%d\n", client_ip[0], client_ip[1], client_ip[2], client_ip[3], WSAGetLastError());

                closesocket(ClientSocket);

                //return 1;

        }

        }


        return ERROR_NETWORK_NONE;

}


error_t Network_Add_New_Peer_To_Node_File(unsigned char * ip) {

        FILE * f;

        errno_t err = fopen_s(&f, Network_Nodes_List_File_Path, "r+b");
```

```
        if (err != 0) { return ERROR_FAILED; }


        unsigned char exbuf[5];
        int exists = 0;
        while (fread(exbuf, 1, 5, f) && !exists) {
        exists |= (memcmp(exbuf, ip, 4) == 0);
        }


        if (!exists) {
        unsigned char o = 0x01;
        fwrite(ip, 1, 4, f);
        fwrite(&o, 1, 1, f);
        }


        fclose(f);
}
DWORD WINAPI Network_P2P_Thread(void * param) {
        return Network_P2P(((P2P_Thread_Params*)param)->ptr_WSA_Data,
((P2P_Thread_Params*)param)->ptr_Sending_Socket,
((P2P_Thread_Params*)param)->client_socket);
}
error_t Network_P2P(WSADATA * ptr_WSA_Data, SOCKET * ptr_Sending_Socket,
SOCKET * ptr_Peer_Socket) {
        SOCKADDR_IN client_info;
        int client_info_len = sizeof(client_info);
```

```
        getpeername(*ptr_Peer_Socket, &client_info, &client_info_len);


        char * peer_ip_address = inet_ntoa(client_info.sin_addr);


        printf_Info("Client '%s' has requested to open a P2P channel!\n", peer_ip_address);


        SOCKADDR_IN my_info;
        SOCKET my_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (my_sock == INVALID_SOCKET) {
        printf_Error("Could not create socket for client %s! Error code: %ld\n",
peer_ip_address, WSAGetLastError());
        // Exit with error
        return ERROR_NETWORK_FAILED;
        }
        else {
        printf_Success("Client (%s) socket creation successful.\n", peer_ip_address);
        }


        // IPv4
        my_info.sin_family = AF_INET;
        // Port no.
        my_info.sin_port = htons(GENERAL_SERVER_PORT);
        // The IP address
        memcpy(&my_info.sin_addr.s_addr, &client_info.sin_addr,
sizeof(client_info.sin_addr));
```

```
	int err = connect(my_sock, (SOCKADDR*)&my_info, sizeof(my_info));

	if (err != 0) { printf_Error("Could not connect to peer %s!\n", peer_ip_address); }

	else {

	printf_Success("Succesfuly connected to %s\n", peer_ip_address);

	Copy_Socket_To_List(&my_sock);

	unsigned char buf[4] = { client_info.sin_addr.S_un.S_un_b.s_b1,
client_info.sin_addr.S_un.S_un_b.s_b2, client_info.sin_addr.S_un.S_un_b.s_b3,
client_info.sin_addr.S_un.S_un_b.s_b4 };

	Network_Add_New_Peer_To_Node_File(buf);

	}


	return ERROR_NONE;

}


error_t Network_Broadcast_Transaction(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * transaction, int transaction_size) {

	char * message = malloc(transaction_size + 4);

	memcpy(message, "GCT3", 4);

	memcpy(message + 4, transaction, transaction_size);


	if (Node_List->socket == NULL) {

	printf_Error("No nodes to broadcast to!\n");

	return ERROR_FAILED;

	}
```

```
Node_Peer * ptr = Node_List;
do {
if (ptr->socket != NULL) {
        SOCKADDR_IN client_info;
      int client_info_len = sizeof(client_info);
        getpeername(*(ptr->socket), &client_info, &client_info_len);

        char * peer_ip_address = inet_ntoa(client_info.sin_addr);

        int res = send(*(ptr->socket), message, transaction_size + 4, 0);
        if (res != transaction_size + 4) {
        printf_Error("Error broadcasting transaction to %s\n", peer_ip_address);
        }
        else {
        printf_Success("Broadcasted the transaction to %s\n", peer_ip_address);
        }
}
ptr = ptr->next_node;
//free(peer_ip_address);
} while (ptr != NULL);

free(message);
```

```
	return ERROR_NONE;

}


error_t Network_Broadcast_Block(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * block, int block_size) {

	char * message = malloc(block_size + 4);

	memcpy(message, "GCB3", 4);

	memcpy(message + 4, block, block_size);


	if (Node_List->socket == NULL) {

	printf_Error("No nodes to broadcast to!\n");

	free(message);

	return ERROR_FAILED;

	}


	Node_Peer * ptr = Node_List;

	do {

	if (ptr->socket != NULL) {

		SOCKADDR_IN client_info;

		int client_info_len = sizeof(client_info);

		getpeername(*(ptr->socket), &client_info, &client_info_len);


		char * peer_ip_address = inet_ntoa(client_info.sin_addr);


		int res = send(*(ptr->socket), message, block_size + 4, 0);
```

```
                if (res != block_size + 4) {
                printf_Error("Error broadcasting block to %s\n", peer_ip_address);
                }
                else {
                printf_Success("Broadcasted the block to %s\n", peer_ip_address);
                }
        }
        ptr = ptr->next_node;
        //free(peer_ip_address);
        } while (ptr != NULL);

        free(message);

        return ERROR_NONE;
}

error_t Network_Transaction_Recieved(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * transaction, int transaction_size) {
        return Append_Transaction(ptr_WSA_Data, ptr_Sending_Socket, live_block,
transaction);
}

error_t Network_Block_Recieved(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, _Block * block, int block_size) {
        uint64_t ind = 0;
```

```
        while (Block_Index_Exists(ind)) { ind++; }
        BlockChainLength = ind;
        // Request future blocks
        while (ind < block->Block_Index) {
        printf_Info("Requesting block #%llu from network.\n", ind);
        Network_Request_Block(ptr_WSA_Data, ptr_Sending_Socket, ind++);
        }
        return Verify_Block(ptr_WSA_Data, ptr_Sending_Socket, block, block_size, 1);
}

error_t Network_Block_Request(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * data, int data_size, unsigned char * client_ip) {
        uint64_t block_ind = -1;
        if (data_size != sizeof(block_ind)) {
        return ERROR_FAILED;
        }

        memcpy(&block_ind, data, data_size);

        FILE * f;
        if (Open_Block_File(&f, block_ind) != ERROR_NONE) {
        return ERROR_FAILED;
        }

        _Block b;
```

```
	fseek(f, sizeof(BLOCK_BROADCAST_MAGIC), SEEK_SET);

	int read = fread(&b, sizeof(b), 1, f);

	if (read != 1) {

	return ERROR_FAILED;

	}


	char * message = malloc(sizeof(b) + 4);

	memcpy(message, "GCB3", 4);

	memcpy(message + 4, &b, sizeof(b));


	int res = Network_Send_To_Peer(ptr_WSA_Data, ptr_Sending_Socket, message,
sizeof(b) + 4, client_ip);


	//int res = send(*ptr_Sending_Socket, message, sizeof(b) + 4, 0);

	if (res != sizeof(b) + 4) {

	free(message);

	return ERROR_FAILED;

	}


	free(message);

	return ERROR_NONE;

}


error_t Network_Request_Block(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, uint64_t block_ind) {
```

```
	char * message = (char*)malloc(sizeof(BLOCK_REQUEST_MAGIC) +
sizeof(block_ind));

	memcpy(message, BLOCK_REQUEST_MAGIC,
sizeof(BLOCK_REQUEST_MAGIC));

	memcpy(message + sizeof(BLOCK_REQUEST_MAGIC), &block_ind,
sizeof(block_ind));



	NETWORK_BLOCK_REQUEST_GLOBAL_PLACEHOLDER = 0;



	int off = 0;

	while (NETWORK_BLOCK_REQUEST_GLOBAL_PLACEHOLDER !=
block_ind) {



	if (off == 0) {

		Node_Peer * node = Node_List;

		if (node == NULL) { printf("There are no nodes to request from!\n"); return
ERROR_FAILED; }

		do {

		if (node->socket != NULL) {

			send(*(node->socket), message,
sizeof(BLOCK_REQUEST_MAGIC) + sizeof(block_ind), 0);

		}

		node = node->next_node;

		} while (node != NULL);

	}
```

```
		Sleep(100);
		off = (off + 1) % 10;
		}

		free(message);
}

int Network_Send_To_Peer(WSADATA * ptr_WSA_Data, SOCKET *
ptr_Sending_Socket, void * data, int data_size, unsigned char * client_ip) {
		Node_Peer * node = Node_List;
		int ires = 0;
		do {
		if (node->socket != NULL) {
			if (memcmp(node->ip, client_ip, 4) == 0) {
			// Found peer!
			ires = send(*(node->socket), data, data_size, 0);
			if (ires == data_size){
				return ires;
			}
			}
			node = node->next_node;
		}
		} while (node != NULL);
		return ires;
}
```

```
struct piece_t {
	uint64_t length; // The length of the chain
	uint32_t consensus; // How many nodes support this length
	struct piece_t * next_piece;
};
void Recursive_Free(struct piece_t * ptr) {
	if (ptr->next_piece != NULL) {
	Recursive_Free(ptr->next_piece);
	}
	free(ptr);
}
uint64_t Network_Request_Blockchain_Length(WSADATA * wsadata, SOCKET *
socket) {
	struct piece_t * first_piece = (struct piece_t *)calloc(1, sizeof(struct piece_t));
	first_piece->next_piece = NULL;

	int res = 0;
	Node_Peer * ptr = Node_List;
	if (ptr == NULL || ptr->socket == NULL) {
	printf_Error("No nodes to request length from!\n");
	return 0;
	}
	do {
	res = send(*(ptr->socket), BLOCKCHAIN_QUERY_MAGIC,
```

```
sizeof(BLOCKCHAIN_QUERY_MAGIC), 0);
        if (res != SOCKET_ERROR) {
                char buffer[8] = { 0 };
                res = recv(*(ptr->socket), buffer, sizeof(buffer), 0);
                if (res != SOCKET_ERROR) {
                uint64_t tmp = 0;
                memcpy(&tmp, buffer, sizeof(tmp));


                struct piece_t * ptr = first_piece;
                int found = 0;
                while (ptr->next_piece != NULL && !found) {
                        ptr = ptr->next_piece;


                        if (ptr->length == tmp) { ptr->consensus++; found = 1; }
                }
                if (!found) {
                        ptr->next_piece = (struct piece_t *)calloc(1, sizeof(struct piece_t));
                        ptr->next_piece->next_piece = NULL;
                        ptr->next_piece->length = tmp;
                        ptr->next_piece->consensus = 1;
                }
                }
        }


        ptr = ptr->next_node;
```

```
	} while (ptr != NULL);


	uint64_t length = 0;

	uint32_t consensus = 0;

	struct piece_t * piece = first_piece;

	while (piece->next_piece != NULL) {

	piece = piece->next_piece;

	if (piece->consensus > consensus || (piece->consensus == consensus &&
piece->length > length)) { consensus = piece->consensus; length = piece->length; }

	}


	Recursive_Free(first_piece);


	return length;

}

void Network_Request_Blockchain_Length_Print(WSADATA * wsadata, SOCKET *
socket) {

	uint64_t length = Network_Request_Blockchain_Length(wsadata, socket);


	printf("The current blockchain with the highest consensus is %llu blocks long.\n",
length);

}


void Network_CommandLine_Init(WSADATA ** wsadata, SOCKET ** socket) {

	*wsadata = (WSADATA *)malloc(sizeof(WSADATA));
```

```
	*socket = (SOCKET *)malloc(sizeof(SOCKET));


	Network_Init(*wsadata, *socket);

}


error_t Network_Connect_To_Known_Peers() {

	printf_Info("Requesting P2P channels...\n");


	Node_Peer * node = Node_List;

	do {

	if (node->socket != NULL) {

		printf_Info("Openning P2P with %hhu.%hhu.%hhu.%hhu\n", node->ip[0],
node->ip[1], node->ip[2], node->ip[3]);

		send(*(node->socket),
P2P_CHANNEL_CONNECTION_INIT_MESSAGE,
sizeof(P2P_CHANNEL_CONNECTION_INIT_MESSAGE), 0);

	}

	node = node->next_node;

	} while (node != NULL);


	return ERROR_NONE;

}


HANDLE Network_CommandLine_Server(WSADATA * wsadata, SOCKET * socket) {
```

```
	Main_Server_Thread_Params * param = malloc(sizeof(Main_Server_Thread_Params));

	param->ptr_WSA_Data = wsadata;

	param->ptr_Sending_Socket = socket;

	param->server_addr = LOCALHOST_IP;


	DWORD thread_id;

	HANDLE thread = CreateThread(NULL, 0, Network_Main_Server_Thread, param, 0, &thread_id);




	Network_Connect_To_Known_Peers();



	uint64_t ind = 0;

	while (Block_Index_Exists(ind)) { ind++; }

	BlockChainLength = ind;


	printf_Info("Local blockchain length: %llu.\n", ind);


	uint64_t length = Network_Request_Blockchain_Length(wsadata, socket);

	printf_Info("Network blockchain length: %llu.\n", length);
```

```
// Request future blocks
while (ind < length) {
printf_Info("Requesting block #%llu from network.\n", ind);
Network_Request_Block(wsadata, socket, ind++);
}

Sleep(3000);

ind = 0;
while (Block_Index_Exists(ind)) { ind++; }
BlockChainLength = ind;

printf_Info("New local blockchain length: %llu.\n", BlockChainLength);

FILE* f;
Open_Block_File(&f, BlockChainLength - 1);
if (f != NULL) {
_Block* b;

FILE* f;
Open_Block_File(&f, BlockChainLength - 1);
char* block_buffer = (char*)malloc(sizeof(_Block));
fseek(f, 4, SEEK_SET);
fread(block_buffer, sizeof(_Block), 1, f);
```

```
        char* hash = Hash_SHA256(block_buffer, sizeof(_Block));

         b = Create_Block(BlockChainLength, hash);

        fclose(f);
        free(hash);

        live_block = b;
        }
        else {
        Create_First_Block(wsadata, socket);
        }

}

error_t Network_CommandLine_Request_Blocks(WSADATA * wsadata, SOCKET *
socket) {
        uint64_t ind = 0;
        while (Block_Index_Exists(ind)) { ind++; }
        BlockChainLength = ind;

        printf_Info("Local blockchain length: %llu.\n", ind);

        uint64_t length = Network_Request_Blockchain_Length(wsadata, socket);
```

```
        printf_Info("Network blockchain length: %llu.\n", length);


        // Request future blocks
        while (ind < length) {
        printf_Info("Requesting block #%llu from network.\n", ind);
        Network_Request_Block(wsadata, socket, ind++);
        }


        Sleep(3000);


        ind = 0;
        while (Block_Index_Exists(ind)) { ind++; }
        BlockChainLength = ind;


        printf_Info("New local blockchain length: %llu.\n", BlockChainLength);
}


HANDLE Network_Demo(WSADATA * wsadata, SOCKET * socket) {


        wsadata = (WSADATA *)malloc(sizeof(WSADATA));
        socket = (SOCKET *)malloc(sizeof(SOCKET));


        printf_Info("Initializing server on ip: %hhu.%hhu.%hhu.%hhu\n",
LOCALHOST_IP[0], LOCALHOST_IP[1], LOCALHOST_IP[2], LOCALHOST_IP[3]);
```

```
	Network_Init(wsadata, socket);


	int a = 0;


	Main_Server_Thread_Params * param =
malloc(sizeof(Main_Server_Thread_Params));
	param->ptr_WSA_Data = wsadata;
	param->ptr_Sending_Socket = socket;
	param->server_addr = LOCALHOST_IP;
	param->ret = a;


	DWORD thread_id;
	//HANDLE thread = CreateThread(NULL, 0, fun, NULL, 0, &thread_id);
	HANDLE thread = CreateThread(NULL, 0, Network_Main_Server_Thread, param,
0, &thread_id);


	char AHOY[5] = "Ahoy!";


	Node_Peer * node = Node_List;
	do {
	if (node->socket != NULL) {
		send(*(node->socket), AHOY, sizeof(AHOY), 0);
		node = node->next_node;
	}
	} while (node != NULL && node->next_node != NULL);
```

```
        /*node = Node_List;

        uint64_t block_chain_length = 0;

        while (block_chain_length == 0 && node->next_node != NULL && node->socket
!= NULL) {

        send(*(node->socket), BLOCKCHAIN_QUERY_MAGIC,
sizeof(BLOCKCHAIN_QUERY_MAGIC), 0);

        uint64_t block_chain_length_tmp = 0;

        recv(*(node->socket), &block_chain_length_tmp, sizeof(block_chain_length_tmp),
0);

        block_chain_length = max(block_chain_length, block_chain_length_tmp);

        }*/


        uint64_t ind = 0;

        while (Block_Index_Exists(ind)) { ind++; }


        // Request future blocks


        while (ind == 0 || Block_Index_Exists(fmax(0, ind - 5))) {

        Network_Request_Block(wsadata, socket, ind++);

        Sleep(100);

        }


        /*if (thread) {

        // Optionally do stuff, such as wait on the thread.
```

```
		printf("Waiting...\n");

		while (params.ret == 0) {

			WaitForSingleObject(thread, 1000 * 20);

		}

		printf("Done waiting!\n");

		}*/


		//Network_Send_Message_Await_Return(&wsadata, &socket, node_addr, message,
message_len, &output, &output_len);


		return thread;//Network_TCP_Handshake();

}
```

Command Line

To make GreenCoin user friendly and usable, the GreenCoin application has intuitive command line functionality. Through the command line, a user can call functions, such as transaction making, wallet generating, and many more. In order to make the command line easy to use for new users, there is an in-depth help menu which explains each command, called using the simple command “help”.

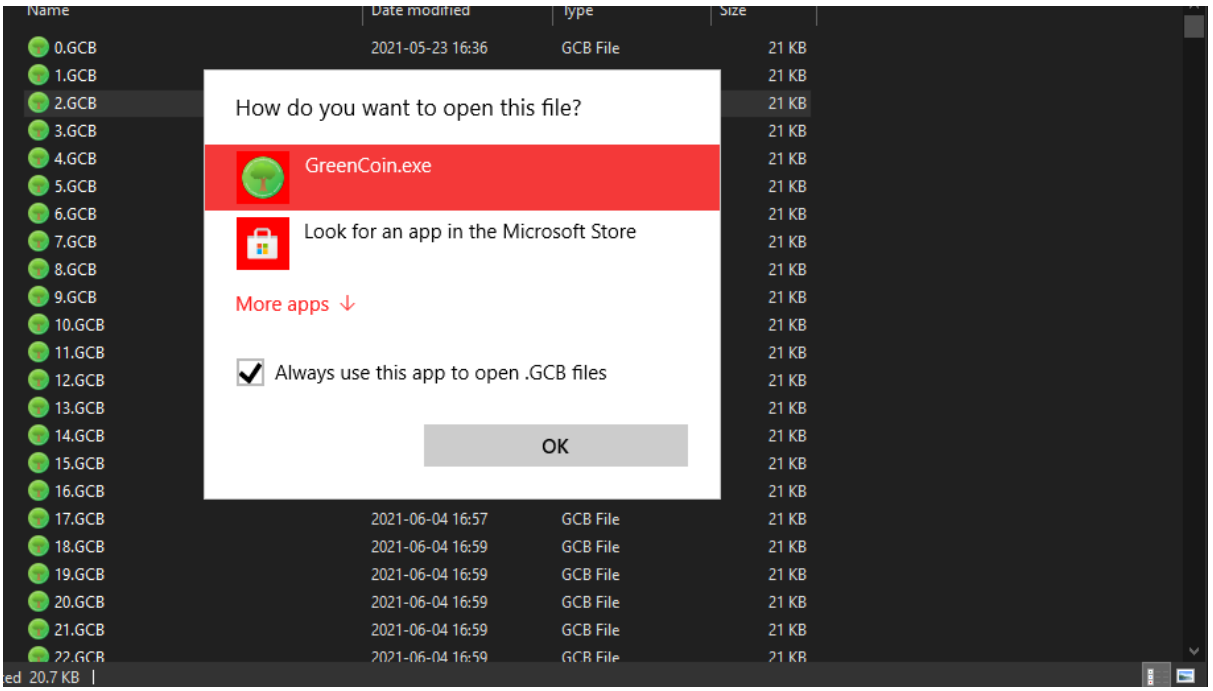
Code wise, to organize the command line, a structure “command_t” was used. This structure has three parts:

- command_text
  - A 64 character long string buffer which holds the exact command which is needed to execute the function.
  - This is the actual command a user needs to write in the console.
- command_description
  - A 256 character long string buffer which contains an explanation of what happens when a user types the command.
  - This description contains a simple explanation of the function the command calls.
- function
  - The function part of the command_t structure is an interesting piece of code. The definition of this part is: int(*function)(void *, void *);
    - Meaning a pointer to a function which returns an integer and takes two void pointers as parameters.
  - This holds a pointer to the function the command calls.
  - The void pointers are used to pass common arguments to the functions, such as the main network information (wsadata and server socket) which most functions need and use.

This setup allows us to clearly define commands using these three parts, removes the need for a set of “if-else if-else” statements to check for each command, and allows us to easily print out help information which does not need to be manually updated each time a new command is implemented.

The command line is actually, currently, split into two parts; general command line, and a wallet command line. The general command line is where everything begins and goes through, while the wallet command line, which is accessed from the general command line, executes commands which are related to wallets (generation, value/balance checking, etc.). The wallet specific command line is implemented in Wallet.h + Wallet.c, but uses all the same methods and structures as the general command line.

Moreover, the GreenCoin program also contains a very rudimentary external command line. This external command line is used when doing something like opening a file with GreenCoin.exe:



The external command line will process the command, which is passed in the “char ** argv” parameter in main, and will recognize the command as the file path of a GreenCoin Block. The command line will then read the file, check its magic, which should be “GCB3”, and then call a preview item function which will print the block in a readable format to the console.

This external command line is useful since the GreenCoin software automatically saves all transactions the user executes and all blocks verified to the file system. The external command line allows users to easily open and view these transactions and blocks by simply defining GreenCoin.exe as the default processing program for “.GCB”, “.GCT”, and “.GCNL” files (Functionality for previewing other GreenCoin files can be implemented in the future).

Code

GeneralCommandLine.h + GeneralCommandLine.c

```
#pragma once


#ifndef __GENERAL_COMMAND_LINE_H

#define __GENERAL_COMMAND_LINE_H
```

```
#include "../General/General.h"
#include "../BlockChain/BlockChain.h"
#include "../Network/Network.h"



#pragma region Commands

command_t COMMAND_HELP;
command_t COMMAND_CREATE_TRANSACTION;
command_t COMMAND_START_SERVER;
command_t COMMAND_REQUEST_BLOCKS;
command_t COMMAND_WALLET;
command_t COMMAND_BLOCKCHAIN_LENGTH;
command_t COMMAND_SET_NOTARY;

command_t * COMMANDS[7];

#pragma endregion

char INPUT_BUFFER[1024];
int Query_User_Input();

int Help_Menu();
```

```
void Print_Node_List(byte * buffer, long * buffer_size);


void Preview_Item(byte * buffer, long * buffer_size);


int Command_Line();


#endif // !__GENERAL_COMMAND_LINE_H
```

```
#include "GeneralCommandLine.h"



#pragma region Commands


command_t COMMAND_HELP = {
"help",
"Displays help menu.",
Help_Menu
};
command_t COMMAND_CREATE_TRANSACTION = {
        "transact",
        "Opens transaction command-line where transactions are made and executed in order to transfer funds between wallets.",
        Transaction_Demo
```

```
};

command_t COMMAND_START_SERVER = {

        "server",

        "Starts local server which accepts P2P connections from other nodes. Starting a server is a prerequisite to block mining.",

        Network_CommandLine_Server

};

command_t COMMAND_REQUEST_BLOCKS = {

        "gcbh",

        "GreenCoin Block History. Requests blockchain history from all known nodes. Use this to keep up to date with the blockchain.",

        Network_CommandLine_Request_Blocks

};

command_t COMMAND_WALLET = {

        "wallet",

        "Opens wallet command-line (avaliable offline). Use this to monitor wallet funds and create wallets.",

        Wallet_CommandLine_General

};

command_t COMMAND_BLOCKCHAIN_LENGTH = {

        "gcbl",

        "GreenCoin BlockChain Length. Requests the blockchain length from all known nodes.",

        Network_Request_Blockchain_Length_Print

};

command_t COMMAND_SET_NOTARY = {
```

```
        "notary",
        "Set the local notary signing address. This address will be used when mining blocks to award you your fee.",
        Load_Local_Notary_Signing_Address
};


command_t * COMMANDS[7] = { &COMMAND_HELP, &COMMAND_CREATE_TRANSACTION, &COMMAND_START_SERVER, &COMMAND_REQUEST_BLOCKS, &COMMAND_BLOCKCHAIN_LENGTH, &COMMAND_WALLET, &COMMAND_SET_NOTARY };



#pragma endregion


int Help_Menu() {
        printf("General help menu: \n");
        for (int i = 0; i < sizeof(COMMANDS) / sizeof(COMMANDS[0]); i++) {
        command_t * command = COMMANDS[i];
        printf("\t%s", command->command_text);
        int text_len = strlen(command->command_text);
        for (int j = 0; j < 3 - floor((text_len) / 4.0); j++) { printf("\t"); }
        printf("| %s\n", command->command_description);
        }
        return 0;
}
```

```
void Print_Node_List(byte * buffer, long * buffer_size) {

	/*

	The value of a node is a representation of how often they are online and if they send fraudulent blocks/transactions.

	A node with a high value means it has been unreliable.

	After a certain threshold, a node should no longer be connected to, and should be banned.

	This will essentially remove malicious nodes from the network as after a few nefarious actions, they will be banned from the network indefinetly.

	*/

	long ind = 0;

	unsigned char * ptr = buffer;


	fprintf(stderr, "Known network nodes: \n");


	while (ind < (*buffer_size) - 4) {


	fprintf(stderr, "IP: %hhu.%hhu.%hhu.%hhu \t|\t Value: %hhu\n", ptr[0], ptr[1], ptr[2], ptr[3], ptr[4]);


	ind += 5;

	ptr += 5;

	}

}


void Preview_Item(byte * buffer, long * buffer_size) {
```

```
        if (memcmp(buffer, GCT_MAGIC, 4) == 0) {
        // GCT file
        Print_Transaction(stderr, (_Transaction*)(buffer+4));
        }
        else if (memcmp(buffer, GCB_MAGIC, 4) == 0) {
        // GCB file
        Print_Block(stderr, (_Block*)(buffer + 4));
        }
        else if (memcmp(buffer, GCNL_MAGIC, 4) == 0) {
        // GCNL file
        Print_Node_List(buffer + 4, buffer_size);
        }

}

int Query_User_Input() {
        memset(INPUT_BUFFER, 0, sizeof(INPUT_BUFFER));
        char * ptr = fgets(INPUT_BUFFER, sizeof(INPUT_BUFFER), stdin);
        if (ptr == NULL) { return -1; }
        return 0;
}

int Command_Line() {
```

```
	srand(time(NULL));


	printf("Would you like to go online? ( Y / N ): ");

	Query_User_Input();

	int is_online = (strcmp(INPUT_BUFFER, "Y\n") == 0) ||
(strcmp(INPUT_BUFFER, "y\n") == 0) || (strcmp(INPUT_BUFFER, "Yes\n") == 0) ||
(strcmp(INPUT_BUFFER, "yes\n") == 0);

	printf("Online: ");

	if (is_online) { printf_Success("TRUE\n"); }

	else { printf_Error("FALSE\n"); }


	WSADATA * wsadata = NULL;

	SOCKET * socket = NULL;

	if (is_online) {

	Network_CommandLine_Init(&wsadata, &socket);

	Network_Print_IP();


	// Get ip

	printf("Please enter your ip: ");

	char buf[64] = { 0 };

	fgets(buf, sizeof(buf), stdin);

	unsigned long ip = inet_addr(buf);

	memcpy(LOCALHOST_IP, &ip, sizeof(ip));


		printf_Info("Initializing server on ip: %hhu.%hhu.%hhu.%hhu\n",
```

```
LOCALHOST_IP[0], LOCALHOST_IP[1], LOCALHOST_IP[2], LOCALHOST_IP[3]);
        Network_Locate_Nodes(&wsadata, &socket);
        }
        Load_Block_History_Path();


        Load_Domain_Parameters();




        uint64_t ind = 0;
        while (Block_Index_Exists(ind)) { ind++; }
        BlockChainLength = ind;


        HANDLE server_handle;


        char buffer[1024] = { 0 };
        while (1) {
        printf("> ");
        memset(buffer, 0, sizeof(buffer));
        fgets(buffer, sizeof(buffer), stdin);
        if (buffer[strlen(buffer) - 1] == '\n') { buffer[strlen(buffer) - 1] = 0x00; }


        for (int i = 0; i < sizeof(COMMANDS) / sizeof(COMMANDS[0]); i++) {
                if (strcmp(buffer, COMMANDS[i]->command_text) == 0) {
```

```
                COMMANDS[i]->function(wsadata, socket);
                }
        }
        }
}
```

ExternalCommandLine.h + ExternalCommandLine.c

```
#pragma once


#ifndef __EXTERNAL_COMMAND_LINE_H
#define __EXTERNAL_COMMAND_LINE_H


#include "../General/FileIO.h"
#include "../General/General.h"
#include "GeneralCommandLine.h"


int Execute_External_Commands(int argc, char ** argv);




#endif // !__EXTERNAL_COMMAND_LINE_H
```

```
#include "ExternalCommandLine.h"


int Execute_External_Commands(int argc, char ** argv) {
	printf("'%s'\n", argv[0]);


	Load_Domain_Parameters();
```

```
        if (argc == 2) {
        if (File_Valid(argv[1])) {
                // Open file with GreenCoin.exe
                char * file_path = argv[1];


                byte * buffer;
                long buffer_size = Load_File(file_path, &buffer);
                if (buffer_size > 0) {
                Preview_Item(buffer, &buffer_size);
                }
                else {
                printf("Could not load or read file!\n");
                }
        }
        }
}
```

Extras

To accommodate all of the GreenCoin software functionality, a few extra helping functions and structures were used. Since GreenCoin contains no external library dependencies (other than standard libraries such as stdint.h, math.h, string.h, stdio.h, and winsock2.h), a lot of functionality had to be manually implemented.

debug.h

This header file contains global declarations of debugging levels which are used by the functions in BNMath. In this file, a developer can choose which level of in-depth debugging they wish to use, and the various print functions used for debugging will either activate or not depending on this.

```
#pragma once
#ifndef __DEBUG_H
#define __DEBUG_H

#include "General.h"
#include "../Cryptography/BNMath/BNMath.h"

#include <stdio.h>

//Trace level definitions
#define TRACE_LEVEL_OFF       0
#define TRACE_LEVEL_FATAL     1
#define TRACE_LEVEL_ERROR     2
#define TRACE_LEVEL_WARNING   3
```

```
#define TRACE_LEVEL_INFO       4
#define TRACE_LEVEL_DEBUG   5
#define TRACE_LEVEL_VERBOSE  6

//Default trace level
#ifndef TRACE_LEVEL
#define TRACE_LEVEL TRACE_LEVEL_OFF
#endif

//Trace output redirection
#ifndef TRACE_PRINTF
#define TRACE_PRINTF(...)           fprintf(stderr, __VA_ARGS__)
#endif

#ifndef TRACE_ARRAY
#define TRACE_ARRAY(p, a, n)      fprintf(stderr, "Trace Array")
#endif

#ifndef TRACE_MPI
#define TRACE_MPI(p, a)              BN_Dump(stderr, p, a)
#endif

//Debugging macros
#if (TRACE_LEVEL >= TRACE_LEVEL_FATAL)
```

```
#define TRACE_FATAL(...) TRACE_PRINTF(__VA_ARGS__)
#define TRACE_FATAL_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)
#define TRACE_FATAL_MPI(p, a) TRACE_MPI(p, a)
#else
#define TRACE_FATAL(...)
#define TRACE_FATAL_ARRAY(p, a, n)
#define TRACE_FATAL_MPI(p, a)
#endif

#if (TRACE_LEVEL >= TRACE_LEVEL_ERROR)
#define TRACE_ERROR(...) TRACE_PRINTF(__VA_ARGS__)
#define TRACE_ERROR_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)
#define TRACE_ERROR_MPI(p, a) TRACE_MPI(p, a)
#else
#define TRACE_ERROR(...)
#define TRACE_ERROR_ARRAY(p, a, n)
#define TRACE_ERROR_MPI(p, a)
#endif

#if (TRACE_LEVEL >= TRACE_LEVEL_WARNING)
#define TRACE_WARNING(...) TRACE_PRINTF(__VA_ARGS__)
#define TRACE_WARNING_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)
#define TRACE_WARNING_MPI(p, a) TRACE_MPI(p, a)
#else
```

```
#define TRACE_WARNING(...)
#define TRACE_WARNING_ARRAY(p, a, n)
#define TRACE_WARNING_MPI(p, a)
#endif

#if (TRACE_LEVEL >= TRACE_LEVEL_INFO)
#define TRACE_INFO(...) TRACE_PRINTF(__VA_ARGS__)
#define TRACE_INFO_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)
#define TRACE_INFO_NET_BUFFER(p, b, o, n)
#define TRACE_INFO_MPI(p, a) TRACE_MPI(p, a)
#else
#define TRACE_INFO(...)
#define TRACE_INFO_ARRAY(p, a, n)
#define TRACE_INFO_NET_BUFFER(p, b, o, n)
#define TRACE_INFO_MPI(p, a)
#endif

#if (TRACE_LEVEL >= TRACE_LEVEL_DEBUG)
#define TRACE_DEBUG(...) TRACE_PRINTF(__VA_ARGS__)
#define TRACE_DEBUG_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)
#define TRACE_DEBUG_NET_BUFFER(p, b, o, n)
#define TRACE_DEBUG_MPI(p, a) TRACE_MPI(p, a)
#else
#define TRACE_DEBUG(...)
```

```
#define TRACE_DEBUG_ARRAY(p, a, n)

#define TRACE_DEBUG_NET_BUFFER(p, b, o, n)

#define TRACE_DEBUG_MPI(p, a)

#endif


#if (TRACE_LEVEL >= TRACE_LEVEL_VERBOSE)

#define TRACE_VERBOSE(...) TRACE_PRINTF(__VA_ARGS__)

#define TRACE_VERBOSE_ARRAY(p, a, n) TRACE_ARRAY(p, a, n)

#define TRACE_VERBOSE_NET_BUFFER(p, b, o, n)

#define TRACE_VERBOSE_MPI(p, a) TRACE_MPI(p, a)

#else

#define TRACE_VERBOSE(...)

#define TRACE_VERBOSE_ARRAY(p, a, n)

#define TRACE_VERBOSE_NET_BUFFER(p, b, o, n)

#define TRACE_VERBOSE_MPI(p, a)

#endif




#endif // !__DEBUG_H
```

FileIO.h + FileIO.c

These files contain helper functions which process file system operations. Specifically, checking whether a file is valid, calculating a file's byte size, and loading a file into a buffer.

```
#pragma once


#ifndef __FILE_IO_H
#define __FILE_IO_H


#include <stdio.h>


int File_Valid(char * file_path);
int Get_File_Size(char * file_path);
int Load_File(char * file_path, void ** buffer); // Allocates the neccessary amount of memory to the buffer. Returns the size of the loaded file - size of the buffer.




#endif // !__FILE_IO_H
```

```
#include "FileIO.h"


int File_Valid(char * file_path) {
	FILE * f;
	fopen_s(&f, file_path, "rb");
```

```
	int valid = f != NULL;
	fclose(f);
	return valid;
}
long Get_File_Size(char * file_path) {
	FILE * f;
	fopen_s(&f, file_path, "rb");
	if (f == NULL) { return -1; }
	fseek(f, 0, SEEK_END);
	long size = ftell(f);
	fclose(f);
	return size;
}
long Load_File(char * file_path, void ** buffer) {

	//long file_size = Get_File_Size(file_path);
	//if (file_size < 0) { return file_size; }

	FILE * f;
	fopen_s(&f, file_path, "rb");
	if (f == NULL) { return -1; }

	fseek(f, 0, SEEK_END);
	long file_size = ftell(f);
```

```
	fseek(f, 0, SEEK_SET);
	if (file_size < 0 && file_size != -1) { fclose(f); return file_size; }

	*buffer = malloc(file_size);

	if (*buffer == NULL) { fclose(f); return -1; }

	long read = fread(*buffer, sizeof(char), file_size, f);

	if (read != file_size) { fclose(f); free(*buffer); return -1; }

	fclose(f);

	return file_size;
}
```

General.h + General.c

As the name states, these files contain general functions which are used globally throughout the GreenCoin program. Additionally, General contains definitions of magics and structures.

```
#pragma once


#ifndef __GENERAL_H
#define __GENERAL_H


#include <string.h>
#include <stdlib.h>
#include "../Cryptography/DSA/DSA.h"


typedef unsigned char byte;


typedef struct Command command_t;
struct Command {
	char command_text[64];
	char command_description[256];
	int(*function)(void *, void *);
};


// Magics
char GCT_MAGIC[4];// = { 'G', 'C', 'T', 0x33 }; // GreenCoin Transaction
```

```
char GCB_MAGIC[4];// = { 'G', 'C', 'B', 0x33 }; // GreenCoin Block
char GCNL_MAGIC[4];// = { 'G', 'C', 'N', 'L' }; // GreenCoin Node List

uint64_t BlockChainLength;
uint64_t Get_Blockchain_Length();

void Load_Domain_Parameters();

#endif // !__GENERAL_H
```

```
#include "General.h"

char GCT_MAGIC[4]= { 'G', 'C', 'T', 0x33 }; // GreenCoin Transaction
char GCB_MAGIC[4] = { 'G', 'C', 'B', 0x33 }; // GreenCoin Block
char GCNL_MAGIC[4] = { 'G', 'C', 'N', 'L' }; // GreenCoin Node List


uint64_t Get_Blockchain_Length() {
        return BlockChainLength;
}

// Loads the domain parameters into 'params'
```

```
void Load_Domain_Parameters() {
	uint p; BN_Init(&p);
	uint q; BN_Init(&q);

	uint_t N = 5;
	uint_t L = 31;

	//BN_Set_Value(q, 6071717768788351249);
	//DSA_Generate_P(p, q, L);

	//char * q_file_path = "Globals\\q.hex";
	//char * p_file_path = "Globals\\p.hex";

	//FILE * fq;
	//FILE * fp;
	//fopen_s(&fq, q_file_path, "rb");
	//fopen_s(&fp, p_file_path, "rb");

	//if (fq == NULL || fp == NULL) { printf("Could not read p or q hex files!\n");
return; }

	char q_bin[20] = { 0xD1, 0x0C, 0x04, 0x13, 0x13, 0x57, 0xC9, 0xD9, 0x90, 0x29,
0x9C, 0xF5, 0x80, 0x15, 0x9C, 0xE9, 0x23, 0x7D, 0x1F, 0x01 };
	char p_bin[128] = { 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x8B, 0x23, 0xE0,
0xDC, 0x47, 0x98, 0x71, 0x12, 0xFB, 0x10, 0xE7, 0xB3, 0x8A, 0xA3, 0x07, 0x66, 0x79,
0x66, 0x51, 0x00, 0x2C, 0x67, 0x42, 0xD3, 0x22, 0x31, 0x30, 0xA1, 0xEB, 0xE2, 0x05,
```

```
0x58, 0x3A, 0xD4, 0x12, 0x43, 0x43, 0xDF, 0x5E, 0x57, 0x2F, 0x35, 0x26, 0xCA, 0xC8,
0x8F, 0x90, 0xB6, 0x7E, 0xB4, 0x1F, 0x02, 0xB3, 0x39, 0x3B, 0xE7, 0xE1, 0xE8, 0x73,
0x0D, 0x7D, 0xB1, 0x1C, 0xF3, 0x92, 0x56, 0x6E, 0x67, 0x0E, 0xA7, 0x7F, 0xA8, 0x7C,
0xCD, 0xE1, 0x9D, 0x8E, 0x6F, 0x85, 0xAE, 0x0D, 0x16, 0x44, 0x03, 0x96, 0xFC, 0xE2,
0x8E, 0x53, 0x01, 0xFC, 0xB2, 0x4D, 0xC4, 0xE8, 0x96, 0xA5, 0xB1, 0xB2, 0x99, 0x4D,
0xD8, 0x2D, 0xEC, 0x75, 0xCF, 0x86, 0x9A, 0x55, 0x53, 0x01, 0xEA, 0xDF, 0x52, 0x81,
0x35, 0xBC, 0x8A, 0x0B, 0x7D, 0x91 };


        //fread(q_bin, sizeof(char), 20, fq);

        //fread(p_bin, sizeof(char), 128, fp);


        //fclose(fq);

        //fclose(fp);


        BN_Import(q, q_bin, 20, BN_BIG_ENDIAN);

        BN_Import(p, p_bin, 128, BN_BIG_ENDIAN);



        uint G; BN_Init(&G);

        uint h; BN_Init(&h);

        BN_Set_Value(h, 2);


        DSA_Generate_G(G, p, q, h);


        BN_Free(h);


        params = (DSA_Domain_Parameters*)malloc(sizeof(DSA_Domain_Parameters));
```

```
        params->G = G;
        params->p = p;
        params->q = q;

        printf("--- Domain Params: ---\n");
        printf("\tp:\r\n");
        TRACE_MPI("\t\t", (params->p));
        printf("\tq:\r\n");
        TRACE_MPI("\t\t", (params->q));
        printf("\tg:\r\n");
        TRACE_MPI("\t\t", (params->G));
        printf("--- --- --- --- --- ---\n");

        //return params;
}
```

PrettyPrint.h + PrettyPrint.c + ANSI_Color_Codes.h

In order to implement colorful printing in the default windows console, some special print functions were implemented. These functions use ANSI escape codes to trigger coloring in the console. Each use case has its own color; Error: Red, Success: Green, Info: Yellow.

```
#pragma once


#ifndef __PrettyPrint
#define __PrettyPrint


#include <Windows.h>


#include "ANSI_Color_Codes.h"


#ifndef printf_Error
#define printf_Error(...)        Print_Error();fprintf(stderr, __VA_ARGS__);Print_Reset();
#endif
#ifndef printf_Success
#define printf_Success(...)      Print_Success();fprintf(stderr,
__VA_ARGS__);Print_Reset();
#endif
#ifndef printf_Info
#define printf_Info(...)         Print_Info();fprintf(stderr, __VA_ARGS__);Print_Reset();
#endif


void Print_Red();
```

```
void Print_Green();

void Print_Yellow();

void Print_Reset();


void Print_Error();

void Print_Success();

void Print_Info();


void setupConsole(void);

void restoreConsole(void);


#endif // !__PrettyPrint
```

```
#include "PrettyPrint.h"


void Print_Red() { printf(RED); }

void Print_Green() { printf(GRN); }

void Print_Yellow() { printf(YEL); }

void Print_Reset() { printf(reset); }


void Print_Error() { Print_Red(); }

void Print_Success() { Print_Green(); }
```

```
void Print_Info() { Print_Yellow(); }



#ifdef _WIN32
// Some old MinGW/CYGWIN distributions don't define this:
#ifndef ENABLE_VIRTUAL_TERMINAL_PROCESSING
#define ENABLE_VIRTUAL_TERMINAL_PROCESSING  0x0004
#endif

static HANDLE stdoutHandle;
static DWORD outModeInit;

void setupConsole(void) {
		DWORD outMode = 0;
		stdoutHandle = GetStdHandle(STD_OUTPUT_HANDLE);

		if (stdoutHandle == INVALID_HANDLE_VALUE) {
		exit(GetLastError());

	}

		if (!GetConsoleMode(stdoutHandle, &outMode)) {
		exit(GetLastError());
```

```
        }

                outModeInit = outMode;

                // Enable ANSI escape codes
                outMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;

                if (!SetConsoleMode(stdoutHandle, outMode)) {
                exit(GetLastError());

        }

}

void restoreConsole(void) {
                // Reset colors
                printf("\x1b[0m");

                // Reset console mode
                if (!SetConsoleMode(stdoutHandle, outModeInit)) {
                exit(GetLastError());

        }
```

```
}
#else
void setupConsole(void) {}


void restoreConsole(void) {
                    // Reset colors
                    printf("\x1b[0m");


}
#endif
```

```
#pragma once


/*
 * This is free and unencumbered software released into the public domain.
 *
 * For more information, please refer to <https://unlicense.org>
 */


 //Regular text
#define BLK "\x1B[0;30m"
#define RED "\x1B[0;31m"
#define GRN "\x1B[0;32m"
#define YEL "\x1B[0;33m"
```

```
#define BLU "\x1B[0;34m"

#define MAG "\x1B[0;35m"

#define CYN "\x1B[0;36m"

#define WHT "\x1B[0;37m"


//Regular bold text

#define BBLK "\x1B[1;30m"

#define BRED "\x1B[1;31m"

#define BGRN "\x1B[1;32m"

#define BYEL "\x1B[1;33m"

#define BBLU "\x1B[1;34m"

#define BMAG "\x1B[1;35m"

#define BCYN "\x1B[1;36m"

#define BWHT "\x1B[1;37m"


//Regular underline text

#define UBLK "\x1B[4;30m"

#define URED "\x1B[4;31m"

#define UGRN "\x1B[4;32m"

#define UYEL "\x1B[4;33m"

#define UBLU "\x1B[4;34m"

#define UMAG "\x1B[4;35m"

#define UCYN "\x1B[4;36m"

#define UWHT "\x1B[4;37m"
```

```
//Regular background
#define BLKB "\x1B[40m"
#define REDB "\x1B[41m"
#define GRNB "\x1B[42m"
#define YELB "\x1B[43m"
#define BLUB "\x1B[44m"
#define MAGB "\x1B[45m"
#define CYNB "\x1B[46m"
#define WHTB "\x1B[47m"

//High intensty background
#define BLKHB "\x1B[0;100m"
#define REDHB "\x1B[0;101m"
#define GRNHB "\x1B[0;102m"
#define YELHB "\x1B[0;103m"
#define BLUHB "\x1B[0;104m"
#define MAGHB "\x1B[0;105m"
#define CYNHB "\x1B[0;106m"
#define WHTHB "\x1B[0;107m"

//High intensty text
#define HBLK "\x1B[0;90m"
#define HRED "\x1B[0;91m"
```

```
#define HGRN "\x1B[0;92m"

#define HYEL "\x1B[0;93m"

#define HBLU "\x1B[0;94m"

#define HMAG "\x1B[0;95m"

#define HCYN "\x1B[0;96m"

#define HWHT "\x1B[0;97m"


//Bold high intensity text

#define BHBLK "\x1B[1;90m"

#define BHRED "\x1B[1;91m"

#define BHGRN "\x1B[1;92m"

#define BHYEL "\x1B[1;93m"

#define BHBLU "\x1B[1;94m"

#define BHMAG "\x1B[1;95m"

#define BHCYN "\x1B[1;96m"

#define BHWHT "\x1B[1;97m"


//Reset

#define reset "\x1B[0m"
```

GreenCoin.cpp + Source.c

Since the whole GreenCoin program is in pure C, which Visual Studio does not natively support as a project language, GreenCoin.cpp, which contains the program's main function, calls an external C-only function in Source.c. From then on, everything is pure C, and the pure C main function is in Source.c.

```
#include <iostream>

extern "C" void main_c(int argc,char**argv);int main(int
argc,char**argv){main_c(argc,argv);}
```

```
#include "src/BlockChain/BlockChain.h"

#include "src/CommandLine/ExternalCommandLine.h"

#include "src/Network/Network.h"


#include <stdio.h>


#include "src/General/Print/PrettyPrint.h"


int Demo() {


	srand(time(NULL));


	// Get ip

	printf("Please enter your ip: ");

	char buf[64] = { 0 };
```

```
fgets(buf, sizeof(buf), stdin);
unsigned long ip = inet_addr(buf);
memcpy(LOCALHOST_IP, &ip, sizeof(ip));

Load_Block_History_Path();

Load_Domain_Parameters();

Load_Local_Notary_Signing_Address();

WSADATA * wsadata = NULL;
SOCKET * socket = NULL;

HANDLE server_thread = Network_Demo(wsadata, socket);

uint64_t ind = 0;
while (Block_Index_Exists(ind)) { ind++; }



FILE* f;
Open_Block_File(&f, ind - 1);
if (f != NULL) {
_Block* b;
```

```
        FILE* f;
        Open_Block_File(&f, ind - 1);
        char* block_buffer = (char*)malloc(sizeof(_Block));
        fseek(f, 4, SEEK_SET);
        fread(block_buffer, sizeof(_Block), 1, f);

        char* hash = Hash_SHA256(block_buffer, sizeof(_Block));



        b = Create_Block(ind, hash);

        fclose(f);
        free(hash);

        live_block = b;
        }
        else {
        Create_First_Block(wsadata, socket);
        }

        while (1) {
        Transaction_Demo_Spam(wsadata, socket);
        }
}
```

```
int main_c(int argc, char ** argv) {

	setupConsole();

	if (argc > 1) { Execute_External_Commands(argc, argv); }



	else {
	Command_Line();
	//Demo();
	}

	if (argc > 1) {
	printf("Press ENTER to close.\n");
	getchar();
	}

	restoreConsole();
}
```

Works Cited

"4.6 — Fixed-Width Integers and Size_t | Learn C++." *Learn CPP*, www.learncpp.com/cpp-tutorial/fixed-width-integers-and-size-t. Accessed 9 June 2021.

"7.15 Using the Digital Signature Algorithm (DSA)." *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*, flylib.com/books/en/2.878.1.133/1. Accessed 27 May 2021.

Anssi-Fr. "ANSSI-FR/Libecc." *GitHub*, github.com/ANSSI-FR/libecc/blob/master/src/sig/ecdsa.c. Accessed 27 May 2021.

"Are There Public p and q Numbers for Use in DSA?" *Cryptography Stack Exchange*, 11 Nov. 2013, crypto.stackexchange.com/questions/11655/are-there-public-p-and-q-numbers-for-use-in-dsa.

"Ascii Table - ASCII Character Codes and Html, Octal, Hex and Decimal Chart Conversion." *ASCII Table*, www.asciitable.com. Accessed 27 May 2021.

"---." *ASCII Table*, www.asciitable.com. Accessed 12 June 2021.

"Base64 Decode and Encode - Online." *Base64 Decode*, www.base64decode.org. Accessed 27 May 2021.

"The Best Way to Check If a File Exists Using Standard C/C++." *Tutorials Point*, www.tutorialspoint.com/the-best-way-to-check-if-a-file-exists-using-standard-c-cplus plus. Accessed 27 May 2021.

"Bit Shifting (Left Shift, Right Shift) | Interview Cake." *Interview Cake: Programming Interview Questions and Tips*, www.interviewcake.com/concept/java/bit-shift. Accessed 27 May 2021.

"Bitcoin (BTC) and United States Dollar (USD) Year 2021 Exchange Rate History. Free Currency Rates (FCR)." *Copyright Mobilesoftjungle.Com, MobileSoftJungle Ltd.*, freecurrencyrates.com/en/exchange-rate-history/BTC-USD/2021. Accessed 9 June 2021.

"C: Is There Anyway i Can Get the modulo Operator to Work on Non Integer Values?" *Stack Overflow*, 26 June 2020, stackoverflow.com/questions/62597353/c-is-there-anyway-i-can-get-the-modulo-operator-to-work-on-non-integer-values.

"C Library Function - Ceil() - Tutorialspoint." *Tutorials Point*, www.tutorialspoint.com/c_standard_library/c_function_ceil.htm. Accessed 27 May 2021.

"C Library Function - Fgets() - Tutorialspoint." *Tutorials Point*, www.tutorialspoint.com/c_standard_library/c_function_fgets.htm. Accessed 9 June 2021.

"C Library Function - Fopen() - Tutorialspoint." *Tutorials Point*, www.tutorialspoint.com/c_standard_library/c_function_fopen.htm. Accessed 27 May 2021.

"Compilation Error, 'Uses Undefined Struct.'" *Stack Overflow*, 12 Jan. 2013, stackoverflow.com/questions/14298417/compilation-error-uses-undefined-struct.

"Digital Signature Algorithm (DSA and ECDSA) — PyCryptodome 3.9.9 Documentation." *PyCryptodome 3.9.9 Documentation*, pycryptodome.readthedocs.io/en/latest/src/signature/dsa.html#:%7E:text=For%20ECDSA%2C%20the%20signature%20is,bytes%20for%20P%2D256. Accessed 27 May 2021.

Digital Signature Standard (DSS), U.S. Dept. of Commerce, National Institute of Standards and Technology, 2009.

"DSA Generate Keys, Generate Signature and Verify Signature File." *8gwifi*, 4 Mar. 2018, 8gwifi.org/DSAFunctionality?keysize=512.

"Dsa.c Source Code - DSA (Digital Signature Algorithm)." *ORYX*, oryx-embedded.com/doc/dsa_8c_source.html. Accessed 27 May 2021.

Edpresso Team. "C: Reading Data from a File Using Fread()." *Educative: Interactive Courses for Software Developers*, 23 Apr. 2021, www.educative.io/edpresso/c-reading-data-from-a-file-using-fread.

"Elliptic Curve Digital Signature Algorithm - Bitcoin Wiki." *Bitcoin Wiki*, 2011, en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

Eric O Meehan. "Creating a Peer to Peer Network in C." *YouTube*, 14 Mar. 2021, www.youtube.com/watch?v=oHBi8k31fgM.

"Free Cartoon Pictures Of Trees, Download Free Cartoon Pictures Of Trees Png Images, Free ClipArts on Clipart Library." *Clipart Library*, clipart-library.com/cartoon-pictures-of-trees.html. Accessed 27 May 2021.

"Generate Random Prime Numbers." *A Security Site*, asecuritysite.com/encryption/random3?val=8. Accessed 27 May 2021.

Hertig, Alyssa. "What Is Bitcoin Halving? Here's Everything You Need to Know –." *CoinDesk*, 17 Dec. 2020, www.coindesk.com/bitcoin-halving-explainer.

"How Do Banks Become Insolvent?" *Positive Money*, 9 Nov. 2018, positivemoney.org/how-money-works/advanced/how-do-banks-become-insolvent/#:%7E:text=For%20a%20bank%2C%20being%20insolvent,the%20availability%20of%20deposit%20insurance.

"How Do I Base64 Encode (Decode) in C?" *Stack Overflow*, 4 Dec. 2008, stackoverflow.com/questions/342409/how-do-i-base64-encode-decode-in-c.

"How Is the Generator Point 'G' in ECDSA Generated?" *Reddit*, 28 Aug. 2019, www.reddit.com/r/cryptography/comments/cwh723/how_is_the_generator_point_g_in_ecdsa_generated.

"Index Of /." *Apache*, svn.apache.org. Accessed 27 May 2021.

Jakob Jenkov. "Peer-to-Peer (P2P) Networks - Basic Algorithms." *YouTube*, 20 Feb. 2012, www.youtube.com/watch?v=kXyVqk3EbwE.

Leandro Junes. "Applied Cryptography: The Digital Signature Algorithm - Part 1." *YouTube*, 1 Dec. 2016, www.youtube.com/watch?v=PQ8AruHaoLo.

Lisk. "What Is a Peer to Peer Network? Blockchain P2P Networks Explained." *YouTube*, 15 Jan. 2019, www.youtube.com/watch?v=ie-qRQIQT4I.

Littlstar. "Littlstar/B64.c." *GitHub*, github.com/littlstar/b64.c/blob/master/decode.c. Accessed 27 May 2021.

LiveOverflow. "Breaking ECDSA (Elliptic Curve Cryptography) - Rhme2 Secure Filesystem v1.92r1 (Crypto 150)." *YouTube*, 19 May 2017, www.youtube.com/watch?v=-UcCMjQab4w.

"Mastering Bitcoin." *O'Reilly Online Learning*, www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch08.html. Accessed 27 May 2021.

"National Institute of Standards and Technology." *NIST*, 20 May 2021, www.nist.gov.

Noogin. "The Financial Crisis and History of Bitcoin - Noogin." *Medium*, 15 May 2018, medium.com/@noogin/the-financial-crisis-and-history-of-bitcoin-27ebdb932b99#:%7E:text=Bitcoin%20was%20invented%20in%20the,the%20expense%20of%20their%20taxpayers.

"Reverse Sha256 Hash." *Hashtoolkit.Com*, hashtoolkit.com/reverse-sha256-hash/65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5. Accessed 27 May 2021.

"Set an Exe Icon for My Program." *Stack Overflow*, 6 Mar. 2010, stackoverflow.com/questions/2393863/set-an-exe-icon-for-my-program/26130514.

"SHA-256 Hash Calculator." *Xorbin*, xorbin.com/tools/sha256-hash-calculator. Accessed 27 May 2021.

"SHA256 Online." *Github*, emn178.github.io/online-tools/sha256.html. Accessed 27 May 2021.

"---." *Github*, emn178.github.io/online-tools/sha256.html. Accessed 27 May 2021.

Simply Explained. "How Bitcoin Wallets Work (Public & Private Key Explained)." *YouTube*, 6 Aug. 2019, www.youtube.com/watch?v=GSTiKjnBaes.

---. "How Bitcoin Wallets Work (Public & Private Key Explained)." *YouTube*, 6 Aug. 2019, www.youtube.com/watch?v=GSTiKjnBaes.

Statista. "Bitcoin (BTC) Price History from 2013 to June 8, 2021." *Statista*, 8 June 2021, www.statista.com/statistics/326707/bitcoin-price-index.

---. "United States - Monthly Inflation Rate in April 2021." *Statista*, 12 May 2021, www.statista.com/statistics/273418/unadjusted-monthly-inflation-rate-in-the-us.

Stevewhims. "Complete Winsock Server Code - Win32 Apps." *Microsoft Docs*, 31 May 2018, docs.microsoft.com/en-us/windows/win32/winsock/complete-server-code.

Terrazzoni, Jean-Baptiste. "Implementing the Sha256 and Md5 Hash Functions in C." *Medium*, 28 Apr. 2021, medium.com/a-42-journey/implementing-the-sha256-and-md5-hash-functions-in-c-78c17e657794.

"Transmission Control Protocol (TCP) (Article)." *Khan Academy*, www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp. Accessed 9 June 2021.

"Typedef Fixed Length Array." *Stack Overflow*, 24 Dec. 2010, stackoverflow.com/questions/4523497/typedef-fixed-length-array/4523537.

"What Is an Urgent Pointer in TCP? - Quora." *Quora*, www.quora.com/What-is-an-urgent-pointer-in-TCP. Accessed 12 June 2021.

"What Is the Difference between Fwrite and Fprintf in C? - Quora." *Quora*, www.quora.com/What-is-the-difference-between-fwrite-and-fprintf-in-C#:%7E:text=What%20is%20the%20difference%20between%20a%20fprintf%20()%20and%20a,other%20arrays. Accessed 27 May 2021.

Wikipedia contributors. "Base64." *Wikipedia*, 10 June 2021, en.wikipedia.org/wiki/Base64.

---. "Bitcoin Scalability Problem." *Wikipedia*, 25 May 2021, en.wikipedia.org/wiki/Bitcoin_scalability_problem.

---. "C Data Types." *Wikipedia*, 30 Apr. 2021, en.wikipedia.org/wiki/C_data_types.

---. "Computer Network." *Wikipedia*, 10 June 2021, en.wikipedia.org/wiki/Computer_network.

---. "Convergent Series." *Wikipedia*, 8 May 2021, en.wikipedia.org/wiki/Convergent_series.

---. "Digital Signature Algorithm." *Wikipedia*, 17 May 2021, en.wikipedia.org/wiki/Digital_Signature_Algorithm.

---. "Double-Precision Floating-Point Format." *Wikipedia*, 24 May 2021, en.wikipedia.org/wiki/Double-precision_floating-point_format.

---. "Elliptic Curve Digital Signature Algorithm." *Wikipedia*, 26 May 2021, en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

---. “Elliptic-Curve Cryptography.” *Wikipedia*, 4 May 2021, en.wikipedia.org/wiki/Elliptic-curve_cryptography.

---. “Error Detection and Correction.” *Wikipedia*, 8 June 2021, en.wikipedia.org/wiki/Error_detection_and_correction.

---. “Hash Function.” *Wikipedia*, 27 May 2021, en.wikipedia.org/wiki/Hash_function#:%7E:text=A%20hash%20function%20is%20any,table%20called%20a%20hash%20table.

---. “Modular Exponentiation.” *Wikipedia*, 15 Jan. 2021, en.wikipedia.org/wiki/Modular_exponentiation.

---. “Natural Number.” *Wikipedia*, 26 May 2021, en.wikipedia.org/wiki/Natural_number.

---. “One-Way Function.” *Wikipedia*, 19 May 2021, en.wikipedia.org/wiki/One-way_function.

---. “Peer-to-Peer.” *Wikipedia*, 9 May 2021, en.wikipedia.org/wiki/Peer-to-peer.

---. “Port (Computer Networking).” *Wikipedia*, 11 June 2021, en.wikipedia.org/wiki/Port_(computer_networking).

---. “Printf Format String.” *Wikipedia*, 29 Apr. 2021, en.wikipedia.org/wiki/Printf_format_string.

---. “Satoshi Nakamoto.” *Wikipedia*, 24 May 2021, en.wikipedia.org/wiki/Satoshi_Nakamoto.

---. “SHA-2.” *Wikipedia*, 19 May 2021, en.wikipedia.org/wiki/SHA-2.

---. “Transmission Control Protocol.” *Wikipedia*, 12 June 2021, en.wikipedia.org/wiki/Transmission_Control_Protocol.

Links:

https://blockgeeks.com/guides/cryptocurrency-wallet-guide/
https://www.youtube.com/watch?v=GSTiKjnBaes
https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm
https://www.google.com/search?q=double&rlz=1C1CHBF_enCH781CH781&oq=double&aqs=chrome.0.0i433j0l2j46i175i199j0l4j46l2.881j0j7&sourceid=chrome&ie=UTF-8
https://pycryptodome.readthedocs.io/en/latest/src/signature/dsa.html#:~:text=For%20ECDSA%2C%20the%20signature%20is,bytes%20for%20P%2D256).
https://stackoverflow.com/questions/14298417/compilation-error-uses-undefined-struct
https://github.com/ANSSI-FR/libecc/blob/master/src/sig/ecdsa.c
https://en.wikipedia.org/wiki/C_data_types
https://en.wikipedia.org/wiki/Double-precision_floating-point_format
https://www.youtube.com/watch?v=-UcCMjQab4w
https://xorbin.com/tools/sha256-hash-calculator
https://medium.com/a-42-journey/implementing-the-sha256-and-md5-hash-functions-in-c-78c17e657794
https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
https://www.reddit.com/r/cryptography/comments/cwh723/how_is_the_generator_point_g_in_ecdsa_generated/
https://csrc.nist.gov/csrc/media/publications/fips/186/3/archive/2009-06-25/documents/fips_186-3.pdf
https://stackoverflow.com/questions/4523497/typedef-fixed-length-array/4523537
https://www.nist.gov/
https://flylib.com/books/en/2.878.1.133/1/
https://oryx-embedded.com/doc/dsa_8c_source.html
https://www.interviewcake.com/concept/java/bit-shift
https://www.tutorialspoint.com/c_standard_library/c_function_ceil.htm
https://emn178.github.io/online-tools/sha256.html
https://8gwifi.org/DSAFunctionality?keysize=512
https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
https://en.wikipedia.org/wiki/Modular_exponentiation
https://stackoverflow.com/questions/62597353/c-is-there-anyway-i-can-get-the-modulo-operator-to-work-on-non-integer-values
https://asecuritysite.com/encryption/random3?val=8
https://crypto.stackexchange.com/questions/11655/are-there-public-p-and-q-numbers-for-use-in-dsa
https://www.tutorialspoint.com/the-best-way-to-check-if-a-file-exists-using-standard-c-cplusplus
https://www.tutorialspoint.com/c_standard_library/c_function_fopen.htm
https://www.educative.io/edpresso/c-reading-data-from-a-file-using-fread
https://github.com/littlstar/b64.c/blob/master/decode.c
https://stackoverflow.com/questions/342409/how-do-i-base64-encode-decode-in-c

https://www.base64decode.org/
https://svn.apache.org/repos/asf/subversion/trunk/subversion/libsvn_subr/base64.c
https://www.youtube.com/watch?v=PQ8AruHaoLo
http://www.asciitable.com/
https://en.wikipedia.org/wiki/Printf_format_string
https://www.quora.com/What-is-the-difference-between-fwrite-and-fprintf-in-C#:~:text=What%20is%20the%20difference%20between%20a%20fprintf%20()%20and%20a,other%20arrays)%20output%20to%20files.
https://en.wikipedia.org/wiki/Convergent_series
https://en.wikipedia.org/wiki/Bitcoin_scalability_problem
https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch08.html
https://en.wikipedia.org/wiki/Satoshi_Nakamoto
https://en.wikipedia.org/wiki/Peer-to-peer
https://docs.microsoft.com/en-us/windows/win32/winsock/complete-server-code
https://www.youtube.com/watch?v=ie-qRQIQT4I
https://www.youtube.com/watch?v=kXyVqk3EbwE
https://www.youtube.com/watch?v=oHBi8k31fgM
https://en.wikipedia.org/wiki/Hash_function#:~:text=A%20hash%20function%20is%20any,table%20called%20a%20hash%20table.
https://en.wikipedia.org/wiki/SHA-2
https://stackoverflow.com/questions/2393863/set-an-exe-icon-for-my-program/26130514
http://clipart-library.com/cartoon-pictures-of-trees.html
https://hashtoolkit.com/reverse-sha256-hash/65e84be33532fb784c48129675f9eff3a682b27168c0ea744b2cf58ee02337c5
https://emn178.github.io/online-tools/sha256.html
https://en.wikipedia.org/wiki/One-way_function
https://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm
https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:the-internet/xcae6f4a7ff015e7d:transporting-packets/a/transmission-control-protocol--tcp
https://positivemoney.org/how-money-works/advanced/how-do-banks-become-insolvent/#:~:text=For%20a%20bank%2C%20being%20insolvent,the%20availability%20of%20deposit%20insurance.
https://medium.com/@noogin/the-financial-crisis-and-history-of-bitcoin-27ebdb932b99#:~:text=Bitcoin%20was%20invented%20in%20the,the%20expense%20of%20their%20taxpayers.
https://www.statista.com/statistics/326707/bitcoin-price-index/
https://www.statista.com/statistics/273418/unadjusted-monthly-inflation-rate-in-the-us/
https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.coindesk.com%2Fbitcoin-halving-explainer&psig=AOvVaw1nb0W09i4DRVE1hpwIjUMP&ust=1623195250199000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCJDEztbXhvECFQAAAAAdAAAAABAD
https://freecurrencyrates.com/en/exchange-rate-history/BTC-USD/2021
https://www.learncpp.com/cpp-tutorial/fixed-width-integers-and-size-t/
https://en.wikipedia.org/wiki/Natural_number
https://en.wikipedia.org/wiki/Base64
http://www.asciitable.com/

https://en.wikipedia.org/wiki/Computer_network
https://en.wikipedia.org/wiki/Error_detection_and_correction
https://en.wikipedia.org/wiki/Port_(computer_networking)
https://www.quora.com/What-is-an-urgent-pointer-in-TCP
https://en.wikipedia.org/wiki/Transmission_Control_Protocol