An Educational Guide to Creating Your Own Cryptocurrency

Paul Medeiros and Leonidas Deligiannidis

1 Introduction

Over the course of the past decade, many online transactions often required what is known as an "intermediary"—a third party that guarantees the secure exchange of both the goods and information pertaining to the transaction. Additionally, this means that all accountability for the transaction would fall into the hands of the third-party intermediary. This type of security framework is known as a "centralized" framework, as a central authority is responsible for executing a safe data exchange within user transactions. Contrarily, a "decentralized" security framework focuses on eliminating the intermediary, and instead using a "public ledger"—a database of all transaction records shared with all users. This method of transaction allows for the exchanged data between users to become immutable and cryptographically sealed. Additionally, the use of "ledgers" eliminates the chances of losing crucial information during a transaction, giving its users not only immense privacy and security capabilities but also great transparency with all of the transaction data. This type of decentralized security framework is what is used as the foundation for blockchain technology as it is known today. The most common form of blockchain technology, known as "cryptocurrency," utilizes this framework by making all transaction history available to all users while also making all of its data immutable. Each time a series of new transactions is made within the blockchain, a new block containing the new transaction data will be created and added to the existing blockchain, further adding to the long list of immutable data. For this data to be accepted by the blockchain, it must be validated by the blockchain users

P. Medeiros · L. Deligiannidis (✉)
Department of Computer Science and Networking, Wentworth Institute of Technology, Boston, MA, USA
e-mail: medeirosp@wit.edu; deligiannidisl@wit.edu

H. R. Arabnia et al. (eds.), *Advances in Software Engineering, Education, and e-Learning*, Transactions on Computational Science and Computational Intelligence, https://doi.org/10.1007/978-3-030-70873-3_12

themselves through the use of a "proof-of-work" (PoW) algorithm. Miners run the PoW algorithm. This time-consuming process involves the solution to a hard problem [1] involving the computation of hashes which are one-way mathematical functions. While it is very hard to produce the correct hash for a block to be accepted into the blockchain, it is very simple and easy to verify the validity of the hash by any member of the blockchain community.

This report is primarily focused on the creation and deployment of a unique cryptocurrency while utilizing an existing codebase. Unlike the early stages of cryptocurrency that was still in its infancy, the information regarding its development process, especially its use of blockchain technology, has been greatly elaborated upon to make the deployment of a new cryptocurrency more streamlined. Producing a strong blockchain requires several different constituencies such as software developers, miners, exchanges, merchant processing services, web wallet companies, and user/consumers [2]. The process demonstrated here provides a streamlined approach to achieve all the necessary steps in deploying a fresh blockchain without the need of a large software developing team, or even large amounts of third-party software. The primary tools that are utilized in this development process largely center around the use of Bash, a Linux-based scripting language, and the Ubuntu operating system. Additionally, a GitHub account will be especially useful for pushing and pulling new builds of the cryptocurrency when needed.

2 A Working Codebase

We utilized a working cryptocurrency codebase to create our own new cryptocurrency. This codebase allows for anyone to utilize the binaries provided by it to compile their own unique cryptocurrency build. More specifically, this codebase gives developers the necessary tools needed to deploy their own cryptocurrency. Of course, this codebase is not able to generate a new cryptocurrency by itself—there are many necessary alterations of code and unique configuration parameters that must be provided for a new cryptocurrency to be built properly using this codebase. For this project, we used the Litecoin codebase, which is an open-source peer-to-peer cryptocurrency project, and we utilized the public Litecoin GitHub repository for its source code. To gain access to the "Litecoin" repository and fork the necessary documents, a GitHub account needs to be created and then linked to the Ubuntu operating system (which can be done via the Bash terminal setup in Visual Studio Code). This connection with GitHub and Ubuntu is necessary, because if data needs to be pulled and/or be updated from the new forked repository, it can be done through the Ubuntu command line via bash scripting, without having to access the GitHub website directly.

To get the working codebase, we first installed the Ubuntu subsystem on a Windows 10 machine and forked the Litecoin repository from https://github.com/litecoin-project/litecoin and then followed the instructions of the "build-unix" file located in the document folder to install the dependencies.

3 Preliminary Code Modifications

Renaming assets in an existing codebase as a step toward creating a "unique cryptocurrency" is not as questionable as it may seem. Many cryptocurrencies, much like Litecoin itself, often merge their code with changes made from other codebases. In many cases, Litecoin can be seen merging code from Bitcoin, a major cryptocurrency. This can be viewed when going to the Litecoin repository page and checking its commit history and changes. Unsurprisingly, both Bitcoin and Litecoin share enough similarities in their source code to make merging code possible and not very tedious. Practicing something similar to Litecoin, portions of the codebase installed via the Bash terminal can be renamed to include the developer's new cryptocurrency name. Moving forward, it is important to note that there are other aspects of the codebase that must be edited that stretch beyond simply renaming things. The commands shown in Fig. 1 should be entered into the Bash terminal to replace all areas of the code where Litecoin is mentioned.

An imaginary cryptocurrency named "CloudCoin" is used in this example to demonstrate what should be edited in the respected fields. In Fig. 1, any instance of "CloudCoin" should be replaced with the unique cryptocurrency name of the developer's liking. Additionally, the abbreviation of "CloudCoin," which is "CLC," should also be changed to an abbreviation of the new cryptocurrency's name.

Additionally, there are two more name changes that must be made. In Litecoin, and other cryptocurrencies, there are monetary denominations used to represent amounts of cryptocurrency that are smaller than a single coin. In Litecoin, these two denominations are known as "lites" and "photons." Figure 2 shows how to replace these denominations with our own called "clouds" and "raindrops," respectively.

Next, locate the file "chainparams.cpp." This file is one of the major pieces in creating a new blockchain and requires a variety of edits to make a successful

```
find ./ -type f -readable -writable -exec sed -i "s/Litecoin/Cloudcoin/g" {} \;
find ./ -type f -readable -writable -exec sed -i "s/LiteCoin/CloudCoin/g" {} \;
find ./ -type f -readable -writable -exec sed -i "s/LTC/CLC/g" {} \;
find ./ -type f -readable -writable -exec sed -i "s/litecoin/cloudcoin/g" {} \;
find ./ -type f -readable -writable -exec sed -i "s/litecoind/cloudcoind/g" {} \;
```

Fig. 1 The commands to replace any instances of the word "Litecoin" or its variations within the source code. It is important to replace the examples of "CloudCoin" with the developer's own unique cryptocurrency names

```
find ./ -type f -readable -writable -exec sed -i "s/lites/clouds/g" {} \;
find ./ -type f -readable -writable -exec sed -i "s/photons/raindrops/g" {} \;
```

Fig. 2 The commands run via the Bash terminal to replace any instances of the words "lites" or "photons"—the denominations used for Litecoin when the amount of currency is less than one 'Litecoin"

```
pchMessageStart[0] = 0xfb;
pchMessageStart[1] = 0xc0;
pchMessageStart[2] = 0xb6;
pchMessageStart[3] = 0xdb;
```

Fig. 3 Four lines of code that represent the PCH Message Values that are present within the "chainparams.cpp" file. The bytes associated with these lines of code must be changed to unique values to ensure that the different networking protocols present in the blockchain can be successfully handled

```
base58Prefixes[PUBKEY_ADDRESS] = std::vector<unsigned char>(1,48);
base58Prefixes[SECRET_KEY] = std::vector<unsigned char>(1,176);
base58Prefixes[EXT_PUBLIC_KEY] = {0x04, 0x88, 0xB2, 0x1E};
base58Prefixes[EXT_SECRET_KEY] = {0x04, 0x88, 0xAD, 0xE4};
```

Fig. 4 Four lines of code that represent the different public and secret keys that will be present in the cryptocurrency blockchain. All of these values must be unique so that the blockchain can receive accurate data from its users

build. Search in the file for several lines of code that start with the phrase "pchMessageStart," followed by a series of bytes. The bytes present and handle different networking protocols being used to identify the clients of the blockchain. These values must be changed to something unique, because if another cryptocurrency uses the same PCH message values, it will create complications when attempting to identify which cryptocurrency blockchain it is trying to access. The section of code that should be edited should look like Fig. 3. Note that the bytes supplied in the figure are the default values given by Litecoin and should not be used as input.

Like the previous step, the next section of code is also present within the "chainparams.cpp" file but begins with the phrase "base58Prefixes." The bytes associated with these lines of code are used as prefixes for the addresses that can receive data from the cryptocurrency blockchain users. These values must be unique, since sharing these addresses with another cryptocurrency can confuse which cryptocurrency blockchain the data will be sent to. The four lines in Fig. 4 hold the address data for the public key and secret key (as well as the external public and secret keys). The values present within the figure should not be used as input for the code and instead should serve as an example as to the possible values that could be used.

4 Creating the Genesis Block

This is arguably the most important section of the development process, as the following steps are used for creating the first block of the new cryptocurrency blockchain. The first block of a new blockchain, otherwise known as the "genesis

block," is essentially the "origin" of a new cryptocurrency's blockchain. It plays a crucial role not only in creating the new blockchain itself but also for allowing successive blocks to be created and added in the chain. The data structure that exists within each block of the chain is known as the "Merkle root" and must be created alongside the genesis block. The Merkle root consists of what are called "chained hashes." Inside of the "chainparams.cpp" file, the developer can find examples of the genesis block and Merkle root values.

Thankfully, a Python script exists that can help assemble these necessary pieces of data to generate a successful genesis block for the developer. The script, known as "GenesisH0," can be found on GitHub and was utilized for the sake of creating a unique genesis block. Figure 5 shows how to download and install GenesisH0.

In the install directory, there is a python script named "genesis.py," which is the script that will be calculating the nonce and assembling the additional information to create the genesis block for the new blockchain. To use the script, enter the following command shown in Fig. 6 into the terminal—substituting the placeholders with the unique values for the article, public key, and timestamp obtained above (as well as an arbitrary number for the nonce). Be sure to include quotations around certain values as shown in Fig. 6.

It is important to note that this (mining) process can take a long time to complete, as searching for a suitable nonce can be very difficult; we found one within 48 hours. Sometimes, this process can take minutes, while other times, it can take several hours to complete. If the developer finds that they cannot successfully obtain a suitable nonce, either change the arbitrary value given to the nonce or allow the script to run for a longer time.

When the developer sees the message indicating that the genesis hash is found, copy down the nonce and genesis hash values that appear in their respective results. Re-run the previous python script command in Fig. 6—using the new nonce value as the nonce parameter for the script. Running this command should immediately return a result that looks similar to Fig. 7. Values associated with data such as the "Merkle hash," "bits," and other outputs will have unique values when run through the developer's terminal. It is crucial to take note of the values associated with all outputs of the script. Pay close attention to the "Merkle hash," "pubkey," "time,"

```
git clone https://github.com/lhartikk/GenesisH0.git
sudo pip install scrypt construct==2.5.2
```

Fig. 5 Two commands run via the Bash terminal. The first installs the necessary Python script associated with "GenesisH0," while the second installs dependencies that allow the Python script to run

```
python genesis.py -a scrypt -z "Insert Article Here" -p "Public Key" -t timestamp -n nonce
```

Fig. 6 A sample input command for the "genesis.py" script to find a suitable nonce. All placeholders should be substituted for their real corresponding values

```
python genesis.py -a scrypt -z "Insert Article Here" -t timestamp -n nonce
algorithm: scrypt
merkle hash: merkle-hash-value
pszTimestamp: Article-Website Date Title
pubkey: public-key
time: unix-time-value
bits: bit-value
Searching for genesis hash..
genesis hash found!
nonce: nonce-value
genesis hash: genesis-hash-value
```

Fig. 7 The output of the "genesis.py" script when the new suitable nonce is used as the nonce parameter for the command

"bits," "nonce," and "genesis hash" values. This information will aid in developing the next step of the code development process.

5 Primary Code Modifications

Now that the data required to create a new genesis block has been obtained, we need to navigate back to the "chainparams.cpp." First, edit the value associated with the variable "pszTimestamp," and replace it with the name of the article the developer used to create the previous genesis block data.

Scrolling further down the file, there should be a class called "CMainParams." Within it, there is a line of code that refers to creating a genesis block, as well as two lines of code that begin with the word "assert," followed by the words "hashGenesisBlock" or "hashMerkleRoot." The first line of code should include the phrase "CreateGenesisBlock" that holds three genesis block values associated with its Unix time, nonce, and bits. Modify these values with the new values generated from the execution of the "genesisH0" script.

Below the "CreateGenesisBlock" line, the value associated with the line "assert(consensus.hashGenesisBlock)" should be modified to include the genesis hash value the developer obtained from the "genesisH0" script. Additionally, the value associated with "assert(genesis.hashMerkleRoot)" should be modified to include the new Merkle hash.

Cryptocurrencies are typically known to have what are called "decentralized security frameworks." This type of framework eliminates the need for "intermediaries," which are responsible for guaranteeing a secure exchange of data (in this case, money) between the users executing a transaction [3]. A "public ledger" is put

in place of the "intermediary," which is an immutable, cryptographically secured permanent record of all transactions among all users of the blockchain [3].

While this provides a secure alternative to intermediaries, its main strengths lie in its ability to eliminate the chances of information loss, having powerful transaction validation abilities, easy verification processes, and a strong focus on transaction transparency [3]. Interestingly, Litecoin (as well as similar cryptocurrencies, such as Bitcoin) references "dnsseeds" and "seednodes" in its source code, which means that there are multiple active IP addresses that are running to support client interactions with Litecoin (such as transactions). In a way, these could be seen as a form of "intermediaries," but they are in no way required to set up a fresh cryptocurrency blockchain. It's crucial to remove these unnecessary pieces of data, as including them in the new blockchain will send clients of the new blockchain to the addresses provided by the Litecoin source code.

To begin removing the "dnsseeds," navigate back to the "chainparams.cpp" file. Toward the bottom of the file, several lines of code that begin with the phrase "vSeeds.emplace_back" should be present. The developer can either choose to comment out these lines or delete them (doing either will disable these lines of code). Within the same "src," open the "chainparamsseeds.h" file. Edit the method referred to as "pnSeed6_main" by commenting (or deleting) all its accompanying data. The data present in this method is memory associated with the nodes used by the Litecoin source code. Specifically, each line of this method contains data for a unique IP address associated with Litecoin, alongside a port number used by the accompanying address. Because nodes for the new blockchain have not been set up yet (nor can they use the same values provided by Litecoin), these values must be removed from the source code. Additionally, the developer should make sure that the method below "pnSeed6_main," named "pnSeed6_test," is left alone.

6 Deploying the Nodes

Here, a peer-to-peer (P2P) network is used to establish the ability for clients to mine, send, and receive cryptocurrency from the new blockchain. Using this P2P network, the transactions and blocks made through the blockchain will be broadcasted by the nodes and sent to their peers, which then relay further to flood the network if they meet the relay policies [4]. In other words, the P2P network serves as a component that protects its users from "Denial of Service" attacks (DoS) in addition to supporting transactions through Simple Payment Verification (SPV) [4]. The tools used to accomplish this goal were provided through the Microsoft Azure platform and its ability to rapidly deploy multiple virtual machines.

Like Bitcoin, the users and/or computers that will be running one (or multiple) of these nodes will have a direct and authoritative view of the blockchain, with a local copy of all the transactions, independently validated by their own system [5]. This means that if the developer chooses to use their own personal nodes instead of using a service such as Microsoft Azure, then the developer can view the entire history

of the blockchain with other additional privileges. However, running personal nodes will require a permanently connected system in which the system must have enough resources to process all the blockchain transactions [5]. It should also be noted that there may be situations in which two nodes may broadcast different versions of the next block of data simultaneously—which will cause some nodes to receive one or the other versions first [6]. This does not mean anything negative has occurred, but the nodes will continue to compute the work they have been given until the block with the largest amount of work ("largest branch," "longest chain") is identified—to which the other nodes will switch to the branch with the largest amount of work completed [6].

Once the developer has navigated to the Microsoft Azure portal, there are many options that Microsoft provides to its users to deploy a variety of different technologies. One such option is "virtual machines." After selecting the "virtual machines" option, select the "add" option on the page to bring up the setup process for the first virtual machine (these virtual machines will be used as the nodes for the new blockchain). It is recommended when setting up any of the virtual machines to set the virtual machine operating system as Linux, as well as having it run version 16.04 of Ubuntu. After successfully deploying the first virtual machine, a second one with the exact same parameters should also be deployed.

Once both virtual machines are deployed, selecting any of the virtual machines should display information regarding its network protocol, as shown in Fig. 8. Each line should be present in the "inbound port rules" section of the network protocol of the virtual machine, except for the first line. The first line (the lined called "Port_9444") must be manually added to both the "inbound port rules" and the "outbound port rules" of both virtual machines.

To do so, select the "add inbound port rule" option on the page, and change the "destination port ranges" value to the default port number associated with the new blockchain.

Additionally, the developer should change the priority value to 100, as well as the "Name" of the security rule to the default port number. Figure 9 provides an example of what the sample inputs should look like for both virtual machines. It

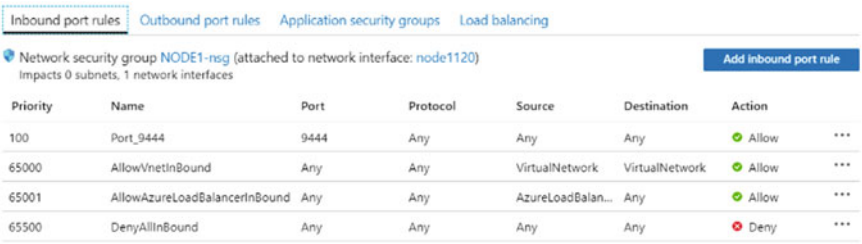

Fig. 8 An example demonstrating the correct input/output port rules associated with both of the deployed virtual machines. The first rule, titled "Port_9444," should be missing after initially deploying both of the virtual machines and must be added manually

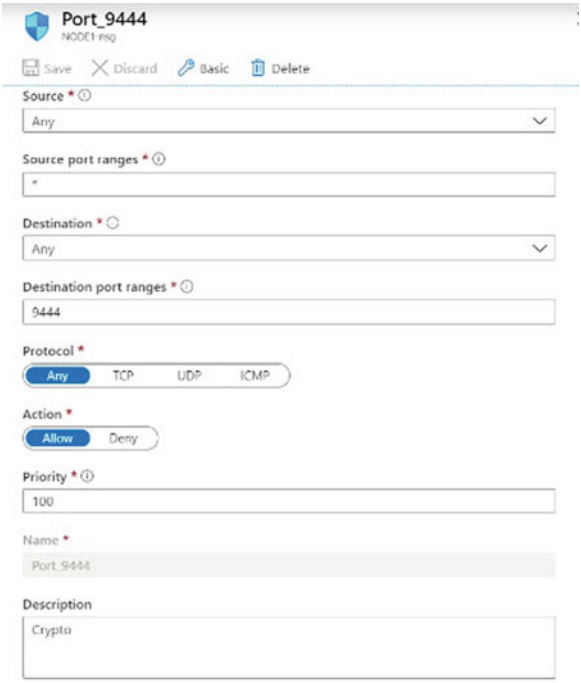

Fig. 9 A sample input for the additional inbound/outbound port rule that must be added to both virtual machines

is also important to note that the "outbound port rules" should be identical to the inbound port rules.

In order for clients of the blockchain to receive updates and submit transactions, they must know the proper nodes and ports to connect to the blockchain. This can easily be done by adding a ".conf" file to the root directory of the project (create a ".conf" file in the "litecoin" directory, which should be the directory that holds all of the files for the developer's current build). The .conf file should be titled whatever the name of the developer's cryptocurrency is. Change the values for the "addnode" section, and supply them with the correct information provided by the Microsoft Azure portal. Typically, the format for the "addnode" values is the name of the node (in this case, the virtual machine's name), the node's location, the phrase ".cloudapp.azure.com:", and the default port number. Additionally, the values for "rcpuser" and "rcppassword" must be changed if the developer wishes to mine their cryptocurrency on a local build of their blockchain. A problem we encountered is the fact that there is no ".conf" file given by the Litecoin source code, meaning there's no file to edit, like the other examples. Thus, in Appendix *A*, we share our own ".conf" file.

7 Building the Wallet

Now that the nodes and ".conf" file have been successfully created, the source code can finally be recompiled and built with wallet functionality. Enabling wallet functionality will compile the source code with a functional user interface that will allow the users of the cryptocurrency to both mine and exchange currency between one another. When using the digital wallet that will be compiled by Litecoin's source code, each user (wallet) will receive a set of "keys" that will allow users to interact with each other's wallets. The user's "private key" is to sign and protect the information of the user's wallet [7, 8]. If a user has the private key to an address (wallet), then that user can use that key to access the currency associated with that address from any Internet-connected computer [2, 7]. Litecoin's source code includes the tools necessary for wallet functionality through the use of "QT," an application designed for developing user interfaces. Compiling the new source code with the QT application provided by Litecoin produces a new executable file that will run the new cryptocurrency wallet. If the various user-interface assets are not updated to reflect the new cryptocurrency, they may still refer to Litecoin on the user interface. However, all transactions that take place in the executable file will still use the new cryptocurrency, so changing the names of the assets is not necessary for deployment. If the new cryptocurrency is intended for public use however, it's recommended that the assets are updated to reflect the names and abbreviation of the new currency.

To build the wallet, run the "autogen.sh" and the "configure" scripts. This creates an executable file named "Litecoin.qt." This file should be run to access the user interface of your wallet as shown in Fig. 10. It is also possible that the executable filename may also be named after your cryptocurrency name—and it is also possible that it could be misplaced in one of the source code subfolders upon compilation. If you cannot find the ".qt" file, the code should be recompiled.

If the Litecoin QT application successfully connects to the nodes, it is possible to locally mine the new cryptocurrency on the developer's computer. Generating currency can also allow the developer to test transactions between users once enough currency has been generated. To begin mining currency, a simple executable file needs to be created with the code shown in Fig. 11. The developer should change the instance of "Litecoin" in the code to reflect the name of the new currency.

While there are no necessary steps left to take for producing a privately distributed build for testing purposes, there are several additional steps online that the developer may wish to follow to make managing the blockchain easier, as well as prevent potential security breaches. It is recommended that the developer research these additional steps if they wish to make their cryptocurrency available to the public.

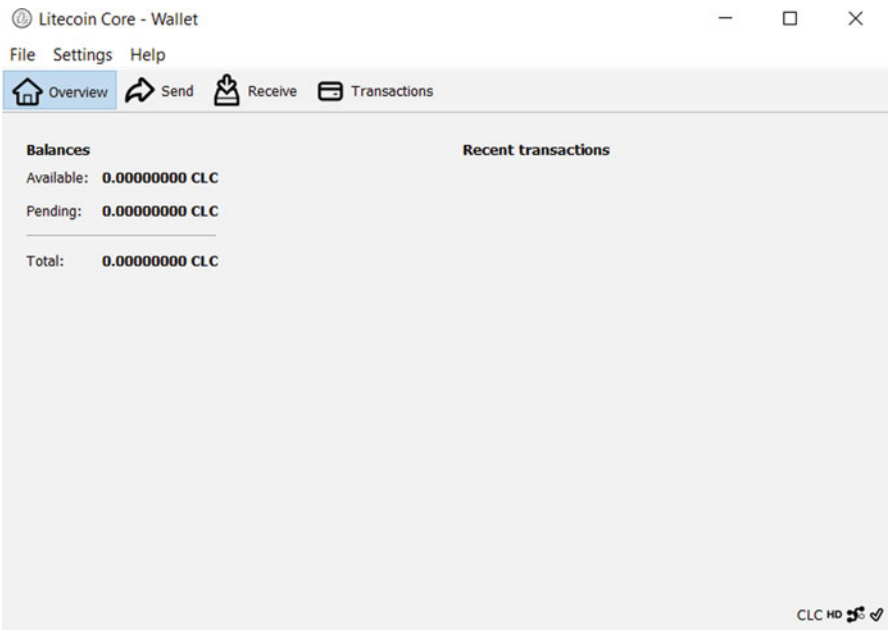

Fig. 10 Litecoin's built-in QT wallet application. While all of the assets present in the user interface were not changed (such as the logo used for Litecoin and the header of the executable file saying "Litecoin Core"), the assets referring to the type of currency being used were updated to reflect the new currency, as referenced by the "CLC" abbreviation next to the balance. Normally, the abbreviation would be "LTC" to refer to Litecoin, whereas in this executable file, it was changed to "CLC" to reflect "CloudCoin," an imaginary currency used for the sake of this study

```
#!/bin/bash
echo "Generating currency! (CTRL+Z to stop)"
while :
do
  litecoin-cli generate 1
done
```

Fig. 11 The code used to mine the newly compiled cryptocurrency

8 Results and Discussion

Scalability and security are two of the most important aspects of cryptocurrencies. With public interest of cryptocurrency rising, the possibility of encountering scalability issues has unfortunately become inevitable. In short, this problem refers to the capability of a single node on a blockchain network to handle a growing amount of transactions per second and thus be enlarged to accommodate that growth [9]. While there have been various attempts to combat this issue such as decreasing the block size or increasing the number of nodes operating with the blockchain, many of the potential solutions are expensive and potentially cost-ineffective.

Because of scalability issues, many cryptocurrencies are often limited to how many transactions they can handle at one time. There are several other factors that contribute to this limitation as well. One of the other factors that could be considered is the speed at which the transactions are completed and placed on the chain. Typically, this is determined by the amount of network activity taking place on the blockchain, alongside the transaction fees associated with the exchanging of the currency itself. When comparing against similar cryptocurrencies such as Bitcoin, Litecoin completes transactions around 4 times faster than Bitcoin. On average, Litecoin takes approximately 2.5 minutes to complete a single exchange, while Bitcoin takes approximately 10 minutes to complete the same task. Additionally, Litecoin can handle about 56 transactions per second as opposed to Bitcoin, which can only handle around 7 [10]. Litecoin is usually seen as a faster alternative to Bitcoin when it comes to exchanges on the blockchain, so this report decided to use the Litecoin codebase to provide an alternative solution for rapid development and experimenting that can support larger numbers of transactions at a time than Bitcoin.

Aside from transaction speed, the ability to provide a strong, secure method for users to interact with the blockchain can also be considered an extremely important aspect of development. In other words, the security of the blockchain is a major concern—and typically involves the confidentiality, integrity, and availability of the technology itself [9]. To satisfy these security concerns, both Bitcoin and Litecoin utilize what are called "proof-of-work" (PoW) algorithms to cryptographically seal the transactions in a block of the blockchain. In short, these algorithms prevent others from tampering with information in a block, providing a secure way of storing transactions in a block. While it is easy to verify the validity of the block or the entire blockchain, it is infeasible to modify a transaction without rerunning the PoW algorithm for each block in the chain!

While there are several different types of PoW algorithms that are used with various cryptocurrencies, the most immediate example would be Bitcoin, with its use of the SHA-256 hash algorithm. Litecoin's source code holds many similarities with Bitcoin. However, one of its key differences includes Litecoin's decision to use a PoW hash algorithm, scrypt, instead of SHA-256. Both algorithms aim to compute hashes of data present on the blockchain, as well as authenticate the transaction data that is stored in each block.

Both SHA-256 and scrypt hash functions are computationally inexpensive to run. However, there is no known way of generating a specific hash value based on some input. Miners try different combinations of nonce and rerun these hashing algorithms, and when a desired hash value is computed, they are awarded, and the block can be added in the blockchain. What makes it even harder is that scrypt is also memory intensive because the generated hashes are stored in memory, and then they need to be accessed before submitting a solution. This makes scrypt appealing since miners cannot use Application-Specific Integrated Circuits (ASICs) to mine hashes fast. Scrypt provides users with less-devoted hardware to be able to mine currency from the blockchain, as opposed to SHA-256 which requires users to join "mining pools" to cooperate in mining currency. This does not mean SHA-256's methods

are completely safe, however. As stated by Chang, blockchain mining pools are also vulnerable to attacks in which the miner in a compromised pool withholds and delays blocks while submitting shares, effectively taking all of the rewards from the mining pool [11]. A precise definition of this occurrence would be what is called a "block withholding attack." According to Kamhoua, these attacks are defined as the situations in which a miner decreases the expected revenue of a mining pool by withholding authenticated blocks—but also increases their own reward by submitting as many shares as possible to the pool [12]. The choice to use Litecoin for this study allows for a better testing environment upon initial deployment—but like working with any codebase, it will require improvements to security protocol and maintenance of several blockchain components if there are any attempts in making the cryptocurrency commercially viable.

Appendix A

An example ".conf" file created for clients of the blockchain to know how and where to receive updates and submit transactions. The file provides details concerning the proper nodes and ports to connect to the blockchain. The ".conf" file should be added to the root directory of the project.

```
#cloudcoin.conf configuration file.
# Network-related settings:
# Run on the test network instead of the real cloudcoin network.
#testnet=0
# Connect via a socks4 proxy
#proxy=127.0.0.1:9050
# Use addnode= settings to connect to specific peers
addnode=NODE1.eastus.cloudapp.azure.com:9444
addnode=NODE2.eastus.cloudapp.azure.com:9444
# Use connect= settings as you like to connect ONLY to
    specific peers:
#connect=localhost:9444
# Do not use Internet Relay Chat (irc.lfnet.org #cloudcoin
    channel) to find other peers.
#noirc=0
# Maximum number of inbound+outbound connections.
#maxconnections=
# JSON-RPC options (for controlling a running cloudcoin/
    cloudcoind process)
# server=1 tells cloudcoin-QT to accept JSON-RPC commands.
server=1
# You must set rpcuser and rpcpassword to secure the JSON-RPC
    api
rpcuser=username123
rpcpassword=password123
# How many seconds cloudcoin will wait for a complete RPC HTTP
    request after the
# HTTP connection is established.
```

```
rpctimeout=30
# By default, only RPC connections from localhost are allowed.
    Specify as many rpcallowip= settings
# as you like to allow connections from other hosts (and you may
    use * as a wildcard character):
#examples:   rpcallowip=10.1.1.34     rpcallowip=192.168.*.*
    rpcallowip=1.2.3.4/255.255.255.0
rpcallowip=127.0.0.1
# Listen for RPC connections on this TCP port:
#rpcport=9432
# You can use cloudcoin or cloudcoind to send commands to
    cloudcoin/cloudcoind
# running on another host using this option:
#rpcconnect=192.168.2.29
# Use Secure Sockets Layer (also known as TLS or HTTPS)
    to communicate with
# cloudcoin -server or cloudcoind
#rpcssl=1
# OpenSSL settings used when rpcssl=1
#rpcsslciphers=TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!AH:!3DES:
    @STRENGTH
#rpcsslcertificatechainfile=server.cert
#rpcsslprivatekeyfile=server.pem
# Miscellaneous options. Set gen=1 to attempt to generate
    cloudcoins
gen=1
# Use SSE instructions to try to generate cloudcoins faster.
4way=1
# Pre-generate this many public/private key pairs,
    so wallet backups will be valid for both prior
# transactions and several dozen future transactions.
#keypool=100
# Pay an optional transaction fee every time you send
    cloudcoins.  Transactions with fees are more likely
# than free transactions to be included in
    generated blocks, so may be validated sooner.
paytxfee=0.001
# Allow direct connections for the 'pay via IP address' feature.
#allowreceivebyip=1
# User interface options
# Start cloudcoin minimized
#min=1
# Minimize to the system tray
#minimizetotray=1
#THIS IS THE END OF THE FILE.
```

References

1. Bitcoin Wiki: Difficulty https://en.bitcoin.it/wiki/Difficulty. Retrieved 2 Feb 2020
2. M. Swan, *Blockchain – Blueprint for a New Economy* (O'Reilly Media Inc., 2015) ISBN-13: 978-1491920497

3. D. Puthal, N. Malik, S.P. Mohanty, E. Kougianos, C. Yang, The Blockchain as a Decentralized Security Framework. IEEE Consumer Electronics Magazine, 18–21 (2018)
4. I. Giechaskiel, C. Cremers, K.B. Rasmussen, When the Crypto in Cryptocurrencies Breaks: Bitcoin Security under Broken Primitives. IEEE Computer and Reliability Societies (2018)
5. A.M. Antonopoulos, *Mastering Bitcoin, 2nd Edition*". ISBN: 9781491954386 (O'Reilly Media Inc., 2017)
6. S. Nakamoto, Bitcoin: "A PeertoPeer Electronic Cash System". https://bitcoin.org/bitcoin.pdf. Retreaved 3 Feb 2020
7. K.A. Taher, T. Nahar, S.A. Hossain, Enhanced Cryptocurrency Security by Time-Based Token Multi-Factor Authentication Algorithm. International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST) (2019)
8. J. Song, *Programming Bitcoin. Learn How to Program Bitcoin from Scratch* (O-Reilly Media Inc., 2019) ISBN-13: 978-1492031499, 2017
9. G. Sargsyan, N. Castellon, R. Binnendijk, P. Cozijnsen, Blockchain Security by Design Framework for Trust and Adoption in IoT Environment. IEEE World Congress on Services (SERVICES) (2019)
10. Which Cryptocurrencies Have The Fastest Transaction Speeds? International Business Times [U.S. ed.], 2018. Gale Academic OneFile, https://link.gale.com/apps/doc/A523776350/AONE?u=wit_main&sid=AONE&xid=ea2a13f9
11. S-Y. Chang, Y. Park, Silent Timestamping for Blockchain Mining Pool Security. 2019 Workshop on Computing, Networking and Communications (CNC)
12. D.K. Tosh, S. Shetty, X. Liang, C.A. Kamhoua, K.A. Kwiat, L. Njilla, Security Implications of Blockchain Cloud with Analysis of Block Withholding Attack. 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2017)