Analyzing the Security and Privacy of Bitcoin

Findings on the Efficacy of Digital Currencies like Bitcoin

Max Grice
University of Oregon
mgrice@uoregon.edu

ABSTRACT

In the past, online transactions have relied heavily on trusted third party systems. Whether it is checking a bank account or transferring funds, financial institutions are often held responsible for ensuring the reliability, security, and privacy of such transactions. Recently however, a growing lack of trust in financial institutions has led to the creation of cryptocurrencies like Bitcoin, which are decentralized and function without third parties. Launched in 2009 by Satoshi Nakamoto, Bitcoin in particular has has established itself as a reputable method of exchange. Some even argue that Bitcoin is superior to third party systems like PayPal and online banking systems. This poses the logical question of whether blockchain technologies like Bitcoin are a sustainable solution to the vulnerabilities existing within third party systems of currency. To address this question, one must both explain and analyze the critical features which allow Bitcoin to function in a secure, private, and reliable manner. These features include concepts like proof of work, blockchain, mining, consensus protocols, and networking infrastructure. Though vulnerabilities do exist within these features and may threaten the scalability of Bitcoin, the solutions which are either partially implemented or have been proposed support the belief that Bitcoin will become a sustainable solution to our current system of digital exchange and storage. Overall, the purpose of this paper aims to not only explain the specific features which allow cryptocurrencies to function but also discuss the extent to which these features provide enough security, privacy, and reliability for Bitcoin to function as an alternative method of digital currency.

Keywords: Bitcoin, Blockchain, Cryptocurrency, Proof of Work, Mining, Majority Attack, Node Network, Double Spending, Distributed Consensus Model

I. INTRODUCTION

Bitcoin is open source software that operates over a peer to peer network, using blockchain technology to validate transactions without any need for a trusted third party (1). Since its release in 2009, Bitcoin has launched us into an age of decentralized digital currency, controversial at times due to its popularity in the black market. Despite its appeal to malicious users, Bitcoin continues to be the most successful cryptocurrency to date, operating with a market capitalization of over 131,000,000,000 and more than 311,000 transactions per day (December, 2019). The appeal and success of bitcoin as a digital currency makes it particularly vulnerable to attackers seeking to exploit the use of this new technology. These malicious attempts to thwart the system consist mainly of double spending, net-split, transaction malleability, networking attacks, and mining attacks. As Bitcoin inspired blockchain technologies continue to gain mainstream attention and grow into other fields like healthcare and government, it is critical to ensure that the security and privacy of such technologies are kept in check. After giving an overview of the major features that have contributed to Bitcoin's success, vulnerabilities will be discussed in more detail to analyze overall security, reliability, and privacy.

II. RELATED WORK

There are many other sources which have been written to investigate the functionalities of bitcoin. These papers are more technical in their explanations of how Bitcoin is implemented functions in a decentralized node network and the major attacks which can be implemented. One such paper included "A Survey on Security and Privacy Issues of Bitcoin," which not only covered Bitcoin's major features, but also discussed potential attack scenarios as well as overall strengths, weaknesses, and solutions. However, the paper was slightly old and consequently did not contain some of current solutions being use to combat issues in scaling Bitcoin's. It also went into more detailed analysis of protocols and proofs rather than focusing on overview and analysis of features. Consequently, this paper differs from related work in that it provides a more easily digestible general overview and discussion and also covers more currently implemented solutions in system vulnerabilities.

Another distinct difference between related work and this report was the basic implementation of a block chain that was used to demonstrate strengths and weaknesses of critical Bitcoin features. The design and execution of the implementation not only provided a more comprehensive understanding of Bitcoin features but was also tested and used as evidence to back claims of Bitcoin's resilience against specific kinds of attacks.

Overview

In Bitcoin, transactions are used to make payments. Transactions contain information on the coins to be transferred (the inputs) and also the destinations of these transfers (the outputs). Outputs contain the destination addresses of users which are generated by performing irreversible cryptographic hash functions on that user's public key. In order to spend bitcoins, the user must use their private key to sign a desired transaction. Once signed, the transactions are hashed into blocks and added to the blockchain via a verification process known as "mining." Miners are other nodes in the network who use computational power to bundle and verify blocks in a process known as "proof of work." Once the proof of work is complete, the miner broadcasts the block to the rest of the network to be accepted into the blockchain, after which they may receive a small mining reward (specified in the transaction fee).

Validating Transactions via Proof of Work

Transactions are the fundamental unit of bitcoin exchange, containing key fields that are hashed together and include things like a lock-time, input/s, and output/s (1). Every input belongs to a user and is identified by a hash pointer to a previous transaction (ie the output that is now being used as an input). It also includes the owners public key and a digital signature. The outputs contain the addresses of recipient users (note that a sender can generate an address and send part of the output to themselves, working as a kind of change). Because this information is stored in the public blockchain ledger, anyone has the power to verify whether the user transferring coins is non malicious by tracking past transactional history and verifying hashes.

Understanding transactions is critical in order to analyze the potential for Double Spending, an attack in which a malicious user generates two different transactions simultaneously to send the same set of coins to two different receivers. As a solution to prevent this kind of attack, Bitcoin uses a Distributed Consensus Model, a process relying on majority agreement among miners if the blockchain goes into an inconsistent state. This would however still leave the system vulnerable to Sybil attacks where miners could create virtual nodes and use these nodes to gain the majority and vote for a malicious transaction (4). With Proof of Work (PoW), this vulnerability is also accounted for because it forces miners to perform a computational task to verify a transaction which is much harder for virtual nodes to do. The task consists of scanning for a value known as a nonce which begins with a certain number of zeros when hashed using SHA-256. This kind of crypto-puzzle is very difficult to fake due to the heavy computational resources that are needed to solve it. Once a miner calculates the correct hash for a given block, they broadcast it to the entire network where other miners can easily verify the hash by comparing it to the target value (after which they can then update their own local blockchain). Miners are motivated by a payment rewards which contains both a predetermined value (halved every 210,000 blocks) and also a transaction fee that is specified by the sender of the transaction (note that one weakness here in specifying a smaller fee is that it may leave the transaction vulnerable to starvation).

Blockchain

As mentioned before, Bitcoin's decentralized authentication works via the use of blockchain, a shared public ledger and stores all validated transactions in hashed units called blocks. Within each block, transactions are stored in a Merkel Tree, containing a time stamp and a hash to the previous block. This process protects blocks from modification since any changes would change the hash of that block, making it inconsistent with the old hash contained in the block after it.

Because blocks are visible to the entire node network, users have the power to check and authenticate transactions, protecting them against double spending attacks and falsified bitcoin ownership. The organizational structure of the blockchain is maintained via cryptographic hash functions which help to condense the transactional information stored in each block. In order to authenticate transactions within blocks, users sign transactions with private keys which can be later validated by other nodes via a public key. Once transactions are collected in a block, they are broadcasted to the network and added to the blockchain in the mining process, thus sequentially adding the blocks to the chain. The consensus protocol is then used to confirm pending transactions and includes them in the chain, enforcing order through mutual agreement among nodes in the network.

In summary, the process of miners adding new blocks consists of 1) determining a valid hash value, 2) adding the block to the local blockchain of the miner and broadcasting this solution to the entire network, and 3) having other nodes check the validity of the broadcasted block by recomputing the hash. This leads us to consider the hypothetical situation where two valid solutions are found and broadcasted simultaneously, resulting in a blockchain fork. These forks present vulnerabilities since they leave the chain in an inconsistent state. This weakness however is countered by the miners who continue to mine on top of an arbitrary chain. Eventually, one chain in the fork will be longer which will lead miners to discard the shorter chain and accept the longest. To further guard against the ambiguities presented by forking, it is important to wait for a block to be verified by as many nodes as possible before accepting the associated transactions. The more a block is validated by miners, the more likely it will be linked to the longest chain and accepted by the rest of the network. Though this creates longer transaction validation times, it maximizes security and reliability by ensuring transactions have been validated by many separate parties.

III. IMPLEMENTATION

A simple implementation of a partially functioning block chain was designed and tested in order to demonstrate the above features. The functionalities of this blockchain is based on the following requirements:

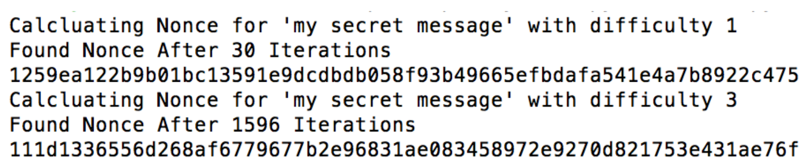
To make a successful transaction, the sender will use a private key and needs the public key of their recipient. The user must also digitally sign the transaction to prove that they own the coins and then add the transaction to a block. Other user wallets will be responsible for validating signatures and solving crypto-puzzles by computing a hash starting with the correct number of zeros. Once users successfully mine a block, it can be accepted as a valid block in the chain.

While most of the above requirements were implemented, it was difficult to find a way to effectively simulate a decentralized node network. While new wallets could be manually created and selected to mine a block, the control of this kind of implementation became one feature that limited the authenticity of this implementation.

The implementation itself consists of wallet, transaction, and block classes. The following section illustrates the reasoning behind the design process and gives examples that demonstrate the resulting program functionalities.

mine.py

The first program created consisted of two functions to aid in the encryption and mining of blocks in the chain. The first, hash(message) returned a hash of a given message string (with arbitrary length) and the second, mine(message, difficulty), could be used to calculate and return the nonce of a given message and difficulty index. As the difficulty index increased in number, it would take longer and longer for the nonce to be found. This parallels the way in which miners use their computational power to find the solutions to crypto puzzles in order to validate the blockchain (often working in mining pools because problems have become so difficult). An example output of the program was run and is shown below:

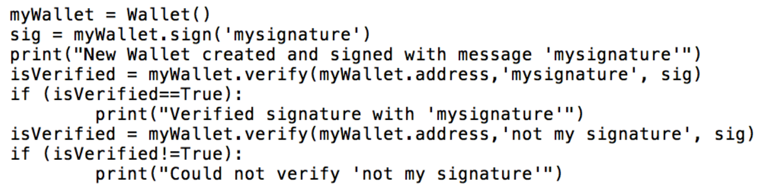


```
Calcluating Nonce for 'my secret message' with difficulty 1
Found Nonce After 30 Iterations
1259ea122b9b01bc13591e9dcdbdb058f93b49665efbdafa541e4a7b8922c475
Calcluating Nonce for 'my secret message' with difficulty 3
Found Nonce After 1596 Iterations
111d1336556d268af6779677b2e96831ae083458972e9270d821753e431ae76f
```

In the example above, one can see that the nonce calculated begins with a number of 1s and is directly related to the level of difficulty specified. As the difficulty increases, the significant increase in the number of iterations through the loop illustrates how the time and power needed to find this value is much greater. The purpose in this step of the design was thus to give a basic demonstration on the process of mining. Specifically, the process showed how, given an arbitrary string *message*, a nonce could be found such that hash(message+nonce) produced a hash starting with a specified number of leading 1s (with more 1s proportional to a greater difficulty level). While this is a simplified version when compared to the number of hashes computed in Bitcoin, it does a good job of showing the basic functioning and purpose of mining.

wallet.py

After using the previous program to illustrate the process of mining, the next logical step in the design process was to generate the private and public keys which could be used to sign transactions. These features were organized in a Wallet class which contained a signer attribute and also private and public keys. For simplification, the public key became the address of a given wallet, differing slightly from Bitcoin which generates addresses that are not the same as the public key (for security and privacy purposes). To verify the functionality of the program, a wallet could be created, signed, and checked for valid and invalid signatures:



```
myWallet = Wallet()
sig = myWallet.sign('mysignature')
print("New Wallet created and signed with message 'mysignature'")
isVerified = myWallet.verify(myWallet.address,'mysignature', sig)
if (isVerified==True):
        print("Verified signature with 'mysignature'")
isVerified = myWallet.verify(myWallet.address,'not my signature', sig)
if (isVerified!=True):
        print("Could not verify 'not my signature'")
```

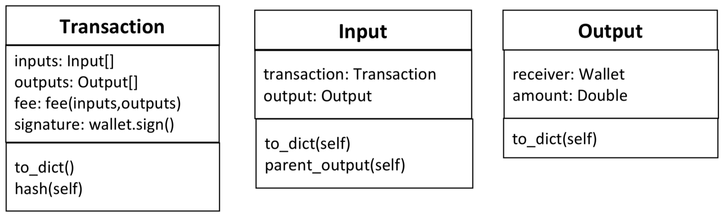
When this code is run in the terminal, the functions were tested to be correct because the first signature is validated while the second is not:

```
New Wallet created and signed with message 'mysignature'
Verified signature with 'mysignature'
Could not verify 'not my signature'
```

This functionality is incredibly important in cryptocurrencies like Bitcoin because it prevents malicious users from forging signatures and participating in transactions that are not their own.

transaction.py

In order to transfer money between people, a transaction class was designed and implemented to contain a given number of inputs containing the address and signature of the sender and outputs to specify how much money should be send to which recipient. A transaction fee would also be included to mimic the mining reward in Bitcoin and serve as motivation for miners to validate transactions (by finding the correct hash) on the block. Since there could be multiple inputs and outputs involved in the transactions, there would also need to be sub classes for input and output. While the input would contain a pointer to its outputs and also to the next transaction, and output would need to specify a recipient and an amount. For all classes, a dictionary representation was included in order to hash the representation into easily digestible blocks. Since the first transaction in the blockchain would contain no inputs, this would need to be a separate class as well (though it would remain as a type of transaction). Diagrams of the described classes are given below for further clarity of design:



In order to test the functionality of these classes, wallets were created to represent three nodes in the network: User1, User2, and User3:

```
user1 = Wallet()
user2 = Wallet()
user3 = Wallet()
```

After creating these wallets, a series of transactions were initiated and validation tests were run in order to check if the code was functioning properly, the comments above each line explain the transaction and expected output:

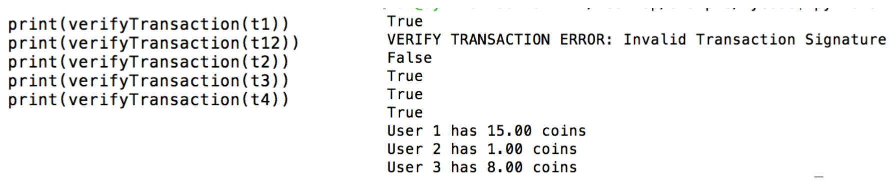
```
# First Transaction: 25 coins to user1
t1 = firstTransaction(user1.address)
# INVALID TRANSACTION: user2 tries to spend money from user1
t12 = Transaction(user2, [Input(t1,0)], [Output(user3.address,10.0)])
# user1 gives 5/25 to user2, 5/25 to user3, and keeps 15/25
t2 = Transaction(user1,[Input(t1,0)], [Output(user2.address,5.0), Output(user1.address,15.0), Output(user3.address,5.0)])
# User3 gives 5/5 coins to user2
t3 = Transaction(user3,[Input(t2,2)], [Output(user2.address,5.0)])
# User2 gives 8/10 coins to user3, and uses 1/10 as fee, leaving 1 remaining
t4 = Transaction(user2, [Input(t2,0), Input(t3,0)], [Output(user3.address,8.0), Output(user2.address,1.0)])
transactions = [t1,t2,t3,t4]
```

The first transaction is its own class which inherits from transaction but contains no inputs because it is the first transaction. It also specifies one Output with a receiver and a default amount (which in this case is set to default=25). In the case above, the first transaction gives 25 coins to the address of user 1. In order to verify the transactions (in a function called verifyTransaction(transaction)), two points needed to be ensured:

1) Only the owner of the wallet was spending the money

2) The owner was not spending more money than they had

While the first point could be checked by validating the signature as implemented in the wallet class, the second required a calculation to determine the difference between inputs and outputs (done in a function called fee(inputs, outputs). If the outputs were greater than the inputs, than the user was spending more money than they had and the transaction would be invalid. The first point is tested in the screen shot above during the transaction t12 when user 2 attempts to spend the 25 coins that were given to user 1 in the first transaction. The output based on this code (shown below) is therefore correct and suggests that the code is able to correctly evaluate each transaction in terms of its validity.



```
print(verifyTransaction(t1))
print(verifyTransaction(t12))
print(verifyTransaction(t2))
print(verifyTransaction(t3))
print(verifyTransaction(t4))
```

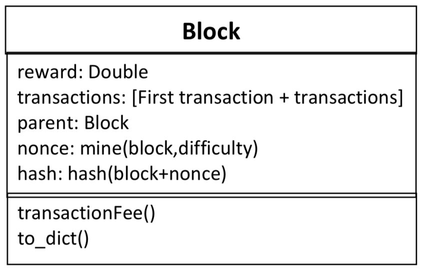
```
True
VERIFY TRANSACTION ERROR: Invalid Transaction Signature
False
True
True
True
User 1 has 15.00 coins
User 2 has 1.00 coins
User 3 has 8.00 coins
```

This program was important because it provided the error checking and hashing of input and output transactions needed to form blocks in a blockchain.

block.py

After implementing 1) A definition of a wallet, 2) A definition of a transaction between wallets, and 3) A way to verify transactions, the next logical step would be to design a way to mine each block and it to a growing blockchain. This would require 2 additional steps, the first being a way to check for valid transactions within a whole block and the second being to find the correct nonce (part of which was already been implemented in the mine function). In order to implement this, a block class was needed to contain both a series of transactions, a parent pointer block (because the blockchain is sequential), a mining incentive, a nonce, and a hashed representation of itself. The first block in the chain would inherit from the block class but would differ in that it would contain no transactions or parent block. The attributes of blocks are illustrated visually in the class diagram below:

In order to verify each block, there are several factors listed below that would need to be checked:



• Block hash begins with the correct number of leading 1s (specified in a set difficulty level)
• All transactions in block must be valid (eg same transaction output cannot be used more than once)
• First transaction in the block must be of type first transaction (ie with no previous inputs and a fee)
• Transaction outputs cannot be used more than once

These factors were checked in a function called verifyBlock() which, given a block and the first block in the sequence, would verify that chain starting from the first block and ending at the specified block. Additional helper functions were made to assist in error checking and included getTransactions(), which listed all the transactions from the first block up until a specified block, and chainLength(), which returned the length of a given blockchain.

In order to test these functionalities in a simple block-chain simulation, I created three users like before and then created a start block mined by user 1 (thus by default giving them the set value of 25 coins). This became the first transaction in the block, followed by transactions 2, 3, and 4 which are listed on the following page:

```
t1 = startB.transactions[0] # transaction 2: 25 coins are mined by user1 - 5
given to user2, 5 to user3, and 15 back to user1

# user1=15, user2=5, user3=5
t2 = Transaction(user1, [Input(t1,0)], [Output(user2.address,5.0),
Output(user1.address,15.0), Output(user3.address, 5.0)])
# transaction 3: user3 gives 5 to user2 leaving them with 0 and user2 with 10

# user1=15, user2=10, user3=0
t3 = Transaction(user3,[Input(t2,2)], [Output(user2.address,5.0)])

# transaction 4: user2 gives 8 to user3, and uses 1 for a fee
# user1=15, user2=1, user3=8
t4 = Transaction(user2, [Input(t2,0), Input(t3,0)],
[Output(user3.address,8.0), Output(user2.address,1.0)])
```

After creating the four transactions, they were arbitrarily mined and stored in blocks. For both cases user 2 was selected as the miner for the two blocks, giving them the 25 coin reward twice:

```
block1 = Block([t2], parent=startB, minerAddress=user3.address)

block2 = Block([t3,t4], parent=block1, minerAddress=user3.address)
```

Its important to note here the limitations of this program as the miner must be manually selected. With more time and resources, more effort would have been spent on finding a way to simulate a node network so that the miners selected would be random and more generally distributed like the original decentralized model. When validating these blocks, the output below correctly indicates that all blocks are valid and the users received the correct amounts of money:

```
MAIN: First Block Hash:
113ca90de90bf2145e10505f89458a639b1727fd3ee816ff083c5b04f2760f8f with
Fee: 0

MAIN: Block 1:
117ec62680c79b32f7588bc898b31e7de8d27820dec26d4925d8535d9e89d056 with
Fee: 0.0

block 2 stores transactions 3 and 4
MAIN: Block 2:
1196dd759c2a4573577cac5c7261cf1c3846be4e675cff7cebf4779d4aa54e2fFee: 1.0
MAIN: Block verification until block1 is True
MAIN: Block verification until block2 is True
MAIN: User 1 Balance: 15.00
MAIN: User 2 Balance: 1.00
MAIN: User 3 Balance: 59.00
```

IV. ANALYSIS

Based on the functionalities outlined above, there are some key features which contribute to overall reliability, security, and privacy. After identifying strengths of previously described features, vulnerabilities will also be addressed along with potential and already implemented solutions and best practices.

Asymmetric Cryptography
While credit cards can be stolen, bitcoins can only be taken if the attacker has access to the users private key. As long as private keys are kept offline and in a secure location, it is much harder to steal Bitcoin. Furthermore, private keys are encrypted during bitcoin transfers which makes it impossible for malicious users to intercept sensitive information like they might be able to do with third parties. Since the transaction cannot be reversed once it has been validated, the system is also protected from charge backs.

Though heavy encryption and hashing makes it difficult to intercept and modify transactions, it is possible to identify users and track their transactions with their public key. This could become a potential vulnerability if the user wishes to remain anonymous when participating in transactions. Furthermore, users often highlight poor usability due to key recovery and storage. Keys are generated as long strings of characters which are difficult to remember and must be recorded in a secure location. While users can easily recover lost private information in third party systems by contacting the third party (eg calling the bank), there is no way to recover lost private keys. With around 22.5% of participants reporting loosing bitcoins due to poor key management, it is important to highlight the ways to maximize effective key management (3).

One solution to preserve the anonymity of users is simply to generate a new public address every time a transaction is initiated. This can be done through the use of deterministic wallet software which derives keys from a single starting point known as a seed. With many different addresses, it is difficult to identify and track users. This allows for transactions to be much more private, especially when compared to third party systems which often have access to the personal data in order to validate transactions. In order to protect the system against poor key management and recovery, it is necessary to not only protect, but to track all private keys associated with Bitcoin wallets. This problem can be partially solved through the use of BIP39, a mnemonic code used to generate seed words. Rather than generating long strings of random digits as public and private keys, BIP39 defines a way in which a user only needs to store and remember a single string of words. This is done by an algorithm which generates entropy in a multiple of 32 bits from the SHA256 hash. After a few more calculations, it uses encoding numbers as an index into a word list in order to form the seed words (6). Since a word list can be created for any language, this makes it much easier to remember a given string of words rather than long strings of digits. While key loss is still possible and impossible to recover, having seed words minimizes chances of loosing sensitive information.

Another promising solution to the issues associated with private key security would be Shamir's Secret, a cryptographic method used to divide a seed word phrase into multiple secrets (9). Once these secrets are generated, they can be distributed to a number of people whom the user trusts. The user may then specify how many people out of the total number of holders must come together in order to reform the original secret. Once this certain number of people come together to combine their secrets, they may have access to the users Bitcoin wallet. This method provides a secure backup where the user can partially entrust people to their wallet without giving them full authorized access.

Decentralization:
Bitcoin's decentralized network makes it impossible for any central authority or government to manipulate and seize coins within the system. This is incredibly appealing to people in countries that struggle with governmental corruption and inflation, making them in need of a reliable source of

currency. Because each user maintains their own bitcoin via the use of a private key and wallet software, there is no way for governments to confiscate funds or arbitrarily change values.

Though decentralization protects coins against malicious manipulations by third parties, it simultaneously requires a large amount of computing resources which are needed to validate transactions in proof of work protocols (2). Though the consensus algorithm facilitates high scalability in terms of decentralized participating nodes, it often wastes large amounts of energy in hash computations throughout the mining process.

To solve this issue, there have been many different proof of work protocols proposed in order to validate transactions. Although these protocols may exhibit better energy efficiency, many are also reliant on a third party which poses new issues in authentication and verification standards. There are however a few which satisfy the condition of decentralization, one of which being Proof of Search (PoS). In this protocol, users submit jobs for finding solutions to optimization problems (10). While this protocol requires further investigation, it indicates a critical area for future development when scaling Bitcoin to fit the needs of a growing population of users.

Transaction Times

When transferring or withdrawing large sums of money between banks, it often takes a significant amount of time to process and validate the transaction. Because banks do not store large amounts of money, the process is slow going and disadvantageous to the bank who wants to keep customers and will often try to postpone for as long as possible. With Bitcoin however, it is incredibly easy to transfer large sums of money, even internationally. Since there is no third party verification, the transaction is simply treated as any other transaction and validated along with the rest of the block. Though large transactions and money transfers may be more efficient with Bitcoin, problems tend to occur in the network when it is overwhelmed by many smaller transactions between parties (eg day to day spending). The shortcomings associated with these kinds of transactions are described in further detail below.

As the network continues to grow, the latency between the validation of a block and its receipt by other nodes increases the potential for forking. Although the PoW model ensures eventual consistency in the chain, the forking will still lead to longer transaction conformation times. This can be seen today as the the Bitcoin network is restricted to a sustained rate of 7 transactions per section due to the restricted block size of 1MB (5). When comparing this to the millions of transactions processed by MasterCard or VISAs, this rate is incredibly slow and presents a significant scaling challenges for Bitcoin.

As one partial solution, miners have begun to congregate in groups known as *mining pools.* Mining pools allow miners to combine resources so that problems can be solved in shorter periods of time and the reward can be split among the participants in the pool. Another promising solution would be the growing popularity of the Lightning Network, a second layer technology that uses micropayment channels to decongest and reduce transaction fees. In this system, two parties may create a ledger entry on the blockchain by signing off. They may then choose to create transactions on the ledge entry without broadcasting them to the blockchain, creating a smaller chain of personalized transactions with the last entry as the most valid output. These channels can be used by other parties searching for a path across the networks, thus helping to increase the speed of the transactions (7). As a result, transactions are able to be made securely off the chain which allows for faster and equally secure transactions.

Net-Split

A netplit attack occurs when an attacker creates an inconsistent view of the network (and blockchain) at the attacked node. In order to minimize the potential for this attack, Bitcoin has made the selection of network peers random and selects a new set of peers (maintaining a minimum of 8) after a certain amount of time (5). Because this solution has already been implemented within the system, it has posses little threat to the security and integrity of the Bitcoin system.

Double Spending
Though the decentralized network makes bitcoin distribution more reliable and private, many users who are first introduced to Bitcoin worry about the possibility of double spending. Double spending is an attack occurring when an attacker sends two conflicting transactions in rapid succession. This would hypothetically allow an attacker to spend the same set of coins twice in two separate transactions, creating new coins which would undermine the stability of Bitcoin as a digital currency. However, while this is occurs in digital cash schemes where tokens or files can be falsified, it is actually quite difficult to double spend in Bitcoin (8). This is because, as described previously, Bitcoin verifies each transaction by making sure that the output is not spend more than once. This is further supported by the blockchain implementation user 2's attempt to send coins that were not their own was rendered invalid. As a result, I can conclude that, because Bitcoin protects against double spending with PoW and miner verification process, it is arguably more secure than other digital cash systems.

Majority (51%) Attack
Though it may be impossible to double spend coins and bitcoin after one of the two transactions is validated (since the coins now belong to someone else and can be verified with a digital signature), it is hypothetically possible to double spend with a 51% majority. 51% Attacks are a type of vulnerability where malicious miners can write blocks and fork the blockchain if they control 51% of the mining power. This is due to the fact that the verification process is done by the entire node network and, if a majority of the node network accepts a malicious block, then that block will be added to the blockchain regardless of its integrity.

This system vulnerability makes it essential that honest nodes in the network control a majority of the computing resources. With this assumption, Bitcoin makes the probability of miners solving the hash value problem (known as a crypto-puzzle) proportional to a number of computing resources used. This idea has been formalized by the PoW based-consensus algorithm which requires no authentication for joining participants of the network. Once malicious users have the power to write their own blocks, an array of other attacks become possible, including double spending, eclipse, and denial of service. This vulnerability could be exploited in the case where honest miners are incentivized by malicious users to join the attacking coalition.

In order to assess the overall threat of this vulnerability and gain a more comprehensive understanding, a simplified version of the attack was tested in the basic blockchain implementation. In this attack, one of the previously created user wallets, User 2, sends 8 coins to User 3 and keeps one:

```
t6 = Transaction(user2,[Input(t2,0), Input(t3,0)],
[Output(user3.address,8),Output(user2.address,1.0)])
```

In order to successfully launch the attack, User 2 would then need to rewrite the transaction by mining a new malicious block where the 8 coins are given back:

```
badt4 = Transaction(user2,[Input(t2,0),Input(t3,0)],
[Output(user2.address,8.0),Output(user2.address,1.0)])
badBlock2 = Block([t3,badt4],parent=block1,minerAddress=user2.address)
```

When user 2 mines the block, a fork is created with two chains of equal length. However, if user 2 were to mine another block immediately after, the malicious chain would become longer than the honest chain:

```
badBlock3 = Block([t_after], parent=badBlock2, minerAddress=user2.address)
```

Once user 2 mines this second block, the print messages indicate that the bad block chain is now the longest in the fork. Using Bitcoin standards, this would mean that the honest chain would be discarded and replaced with the malicious one, thus allowing user two to regain the eight coins.

While the attack was successful, it is important to note here that the program specifies in the block definition the name of the successful miner. Rather than using random chance and computational power to mine blocks, this manual setting made it easy to launch the attack and double spend. Under normal circumstances, it would have been highly unlikely for user 2 to mine two blocks in a row. This is due to both the heavy computational power needed and also the growing traffic of miners and incoming transactions. In this particular case, the congestion of Bitcoin's network actually serves as a strength because it makes it much more difficult to establish a system majority. While it is hypothetically possible and tested via the blockchain implementation, the growing number of network nodes and honest miners makes this vulnerability highly unlikely in any normal scenario.

V. DISCUSSION

Though key strengths, weaknesses, and solutions were identified, there are other factors which remain unexplored but none the less require further discussion when examining the future success and scalability of Bitcoin. As one example, it would have been interesting to implement a python version of a deterministic wallet complete with the BIP39 generation of easy to remember seed words. This would provide better understanding into the strengths and weaknesses present within the solution of deterministic wallet software. Using the test network in Bitcoin to access the lightning network would be another relevant extension to this project because it would provide insight into the way in which the network works and is different from the general bitcoin network.

Overall, this report became less about implementing code because it relied so heavily on blockchain research and general explanations. Since I had no prior experience with Bitcoin and blockchain, the majority of my time was spent reading reports on security and privacy standards and compiling the most important and relevant information for this report. The coding implementation was helpful because it provided a way to test basic functionalities existing within Bitcoin in order to better understand how these features could work in a larger network.

With more time, the code would have been developed much further so that users could interact with and type in values for transactions that were not set at default or programed into the main function. I would have also continued to test my implementation of a majority attack to see if more validations would render the malicious chain to be shorter and thus to longer accepted by the node network. This would have required a simulation of a node network which would be necessary to illustrate the randomization of the mining validation process by nodes.

Some other major differences between the basic implementation and Bitcoin have to do differences in the level of security checks (due to more hashes and different encryption algorithms). As one example, Bitcoin uses more complex addresses rather than simply the users public key. Bitcoin also uses two rounds of SHA256, Merkel Trees for storage, and a scripting language for transactions rather than simply transactions with one amount for every output. As mentioned before, the lack of a node network also made it difficult to implement a majority attack since miners were manually chosen which is inconsistent with the Bitcoin model. In further iterations, it would be critical to implement a way to broadcast blocks to the network in order to effectively test blockchain mining and validation.

VI. CONCLUSIONS

There are a number of conclusions which can be summarized from analysis, implementation, and discussion of Bitcoin as a solution to digital currency and storage. For organizational purposes, these findings were categorized into three paragraphs, touching on critical strengths, weaknesses, and solutions to vulnerabilities.

In terms of strengths, Bitcoin has a number of features which have solved current problems existing within third party transaction schemes. For one, Bitcoins use of asymmetric cryptography in private and public keys ensures secure transactions and allows for features like digital signatures and a decentralized form of authentication (using private-public key verification of encrypted hashes). In addition to the security that is ensured through cryptography, Bitcoins decentralized system is supported by a distributed consensus model of network nodes. Rather than one party having the power to inflate the currency or create malicious transactions, miners force the network to remain honest by preforming Proof of Work verifications of each bundled block of transactions. Subjugated to computational tasks over flawed human decisions makes Bitcoin a much more stable and reliable source of value. It is this protection from human manipulation which makes Bitcoin a great source for transferring large sums of money. While banks are often reluctant to transfer large sums and require further authentication and time to process, Bitcoin views the transaction as any other, bundling it along to be validated via proof of work. To conclude, it is clear that these specific features make Bitcoin a great tool for storing and transferring large amounts of money, (assuming the seed words are kept offline).

Despite these strengths, there are still vulnerabilities which have yet to be fully addressed and leave the system vulnerable to attacks. One issue discussed is the computational power that is needed for the PoW protocol, creating computational waste. This has become especially apparent as Bitcoin grows and becomes more and more overwhelmed by transactions waiting to be validated via proof of work, leading to slow validation times. In addition to the overwhelmed miners searching for target hash values, transaction fees are also increased as a consequence of increased demand. While there have been a number of alternative protocols presented, developers of Bitcoin are still working to confront this issue fully. Consequently, Bitcoin is not yet scaled to handle many small day to day transactions. While working as a secure and reliable source for larger transactions and storage, Bitcoin needs more implemented solutions to make it useable for day to day transactions. In addition to scalability issues, security is based on the assumption that users are responsible for keeping track of private and public keys which can lead to issues in key recovery. As a solution, it is important to store wallets with sensitive information offline, a feature which is not always implemented in wallet software but should be taken into consideration. Finally, there is the issue of attacks such as the majority attack which can lead to double spending. While rare, these kinds of attacks are possible and appropriate precautions should be taken accordingly. A partial solution is simply to wait for verification by multiple nodes before accepting a series of transactions in the blockchain. Software which tracks the number of verifications and monitors forks might be helpful in supporting this function.

While requiring further development, there are solutions that currently exist to confront the above problems in order to maximize the efficacy of transactions in Bitcoin. The first includes the lightning network decongests pending transactions with micropayment channels. As described previously, these channels provide a more efficient kind of routing promises faster transactions and lower fees. The BIP39 algorithm was another solution that I explored and has been fully implemented in many deterministic wallets. This has been widely successful in providing users with a way to not only generate multiple addresses from a single string, but also creating a string that is much easier to recall and thus recover than the long strings of characters in private and public keys. Another identified solution is Shamir's secret since its cryptographic methods allow one to share a secret between different trusted parties without fully compromising access to funds. While this method is still under development due to issues associated with parties forming the secret key without the users knowledge, it is still a partial solution to the problem of backing up knowledge of seed words.

Lessons Learned
Based on these strengths, weaknesses, and solutions, I have learned that, despite the further development needed to make Bitcoin more user friendly, it is still able to effectively function as a solution to third party digital currency schemes. Bitcoin's novel approach to authentication and validation through a decentralized network eliminates many of the problems associated with human error and general institutional corruption in online banking systems.

VII. REFERENCES

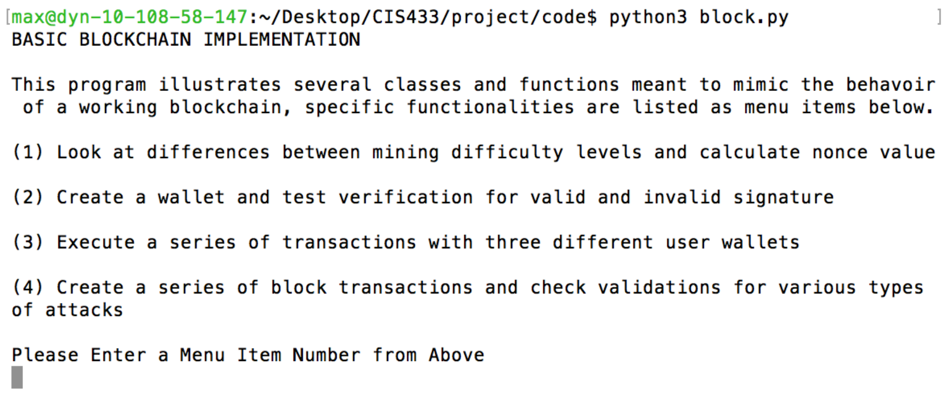
1) S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” Available: http:// bitcoin.org/ bitcoin.pdf , 2008.
2) P. Fairley, “Blockchain world - feeding the blockchain beast if bitcoin ever does go mainstream, the electricity needed to sustain it will be enormous,” IEEE Spectrum, vol. 54, no. 10, pp. 36–59, October 2017.
3) K. Krombholz, A. Judmayer, M. Gusenbauer, and E. Weippl, “The other side of the coin: User experiences with bitcoin security and privacy,” in Financial Cryptography and Data Security: 20th Inter- national Conference, FC 2016, Christ Church, Barbados. Springer Berlin Heidelberg, 2017, pp. 555–580
4) J. R. Douceur, “The sybil attack,” in the First International Workshop on Peer-to-Peer Systems, ser. IPTPS ’01. London, UK: Springer- Verlag, 2002, pp. 251–260.
5) Conti M, Kumar S, Lal C, Ruj S, “A Survey on Security and Privacy Issues of Bitcoin,” 2017, Available https://arxiv.org/pdf/1706.00916.pdf
6) Ian Coleman, “BIP39 Mnemonic Code Converter,” Available https://iancoleman.io/bip39/
7) Lightning Network, “Scalable, Instant, Bitcoin/Blockchain Transactions, Available https:// lightning.network
8) BitcoinWiki, “Double-Spending," Available https://en.bitcoinwiki.org/wiki/Double-spending
9) J. Redman, “Shamir's Secret Explained: Distributing a Seed Phrase into Multiple Parts,” *Bitcoin Insider*, 2019, Available https://www.bitcoininsider.org/article/56719/shamirs-secret-explained-distributing-seed-phrase-multiple-parts
10) N. Shibata, “Proof-of-Search: Combining Blockchain Consensus Formation with Solving Optimization Problems,” Available https://arxiv.org/pdf/1908.01915.pdf

VIII. APPENDIX

The software I developed is quite basic and something which I would have liked to have more time to work on. However, I currently have all the classes discussed in my implementation section working and have entered in arbitrary values to demonstrate major functionalities. In the end, the coding portion of this project became more of a way to validate the research I was doing by demonstrating some of the basic concepts.

To run the code, simply cd into the folder containing the code using the terminal and run using python 3:
>> python3 block.py

Once this command is invoked, the user will be sent to the main page which presents and describes 4 menu options. These four menu options correspond in order to the file examples described in the implementation section of this report.



```
max@dyn-10-108-58-147:~/Desktop/CIS433/project/code$ python3 block.py
 BASIC BLOCKCHAIN IMPLEMENTATION

 This program illustrates several classes and functions meant to mimic the behavoir
  of a working blockchain, specific functionalities are listed as menu items below.

 (1) Look at differences between mining difficulty levels and calculate nonce value

 (2) Create a wallet and test verification for valid and invalid signature

 (3) Execute a series of transactions with three different user wallets

 (4) Create a series of block transactions and check validations for various types
 of attacks

 Please Enter a Menu Item Number from Above
```

The user must then type in and enter the menu number they would like to access via the terminal. Once the menu number is chosen, the code will run and describe the test and outputs given in the terminal. The user may then choose to either return to the main menu and choose a different menu option (press 1) or to quit the program (press 0).

When the user chooses a menu option, the default values in main are invoked and explained in the output through print statements. As an example, option 3 creates a series of transactions and prints the amounts that the users transfer. It then prints the final balance of all user wallets and verifies each transaction.