Bitcoin: examining the crypto-currency. Mathematics of the Bitcoin.

Authors:
Anton Okhrimchuk
Jonggoo Kang
Sungil Kim
Yitong Zhong

Introduction

Long before coins and banknotes were invented, there was a moment when people executed transactions with shells and by barter in the past. Then the technology and cultures developed; flocculents like silk had flourishingly been a standard currency. Likewise, currency has been changed in a wide variety of ways. Nowadays, people utilize paper and coin currency, which could be exchanged to another form of paper or coin currency, depending on the country. What if currency could be stored electronically, allowing people to have any amount at any point in time without worrying about space? This ambiguous thought became the reality when Bitcoin was created. Bitcoin is shapeless online virtual currency that is different from tangible coins and banknotes. It is known that Satoshi Nakamoto, an unidentified developer, made this payment system in 2009. Bitcoin was born to make repulsion of monopolizing and carrying out arbitrary monetary policies by central banks in the world. Bitcoin started to attract attention as an alternative currency from concerns about dollar depreciation by quantitative easing and Fed supplying huge amount of dollar to the world in 2009. Users can use Wallet software to send and receive Bitcoins electronically. Also, the users can convert to or from Bitcoins when they want. Therefore, a lot of users in North America, Europe and Far East Asia gradually use Bitcoins more frequently. Thus, Bitcoin exerts its strong influence over the world. With this reason, our group started to get interested in Bitcoins.

In essence, unlike the other currencies, Bitcoin does not have a central bank of any kind, but rather operates with the help of the peer-to-peer network. Because there is no unified institution to regulate the supply of the Bitcoin, some precise rules have to be followed within the peer-to-peer network to ensure the successful operations of the overall system. Unlike a typical currency, which relate on the central bank, the decentralized Bitcoin currency relies on what can be described as a global ledger. Every transaction involving Bitcoin is recorded. For the system to operate successfully, each transaction should be recorded precisely following some rules. To understand how transactions are recorded, we should first examine how the transaction flows.

Transaction Processes

Transaction is one of the most important parts in Bitcoin system. The smallest fraction of a Bitcoin allowed to be transferred is 0.00000001 BTC, which is 1/100-millionth of a Bitcoin. This fraction is named after the inventor of Bitcoin, a satoshi. In order to have a better understanding of the transaction processes, let us briefly talk about two concepts, SHA-256 and ECDSA (Elliptic Curve Digital Signature Algorithm), first.

SHA-256 in a nutshell is a function, typically referred to as a hash function. This hash function transforms any arbitrary length input string of data will generate an output (a digest) of a unique hash number, which will be comprised of numbers and English letters. There are some properties a function has to possess in order to be considered a strong and reliable has function. Firstly, the hash function has to be computationally efficient. In other words, it has to be relatively easy to compute the digest of a certain input. One of the main properties is collision resistance, which means that the hash function should be constructed in such a way that it will be hard to find two distinct inputs that have the same digest. The digest should be appear fairly random and generated in such a way that nothing can be inferred about the input efficient in a sense that it should not be able to infer anything about the input from the output. It should look random.

ECDSA is a public key cipher. The message receiver has a private key, which he keeps secret. The encryption function (public key) is public. Therefore, everyone can encrypt his message using the public key, but only the receiver can decrypt it. Meanwhile, if the receiver uses his private key to encrypt a plaintext and send the message to another person, then the public key can be used to verify this message.

After knowing those two concepts, we can continue our journey of the Bitcoin transaction.

Simply put, a Bitcoin transaction can be divided into the following steps:

Signature:

1. Payer A first makes a transaction sheet T_2. T_2 should include Payee B's public key, and the amount A wants to pay.
2. Next, A gets data from last transaction sheet T_1 which contains the source of the Bitcoins that A wants to pay B. T_1 represents the source of the money because the money must be paid to A by someone else before via T_1 (or mining).
3. Then A combines the data from T_1 and B's public key, calculates a hash number X.
4. Next, via ECDSA (or some other digital signature system), A encrypts X by using his private key and obtains the signature S.
5. Finally, A attaches S to T_2 and sends it to B.

Validation:

B wants to confirm that this signature is indeed comes from A, he needs to perform the following steps:

1. B first obtains player A's public key, which is the key to decipher the signature S. Payer A's public key is included in last transaction sheet T_1 because T_1 is the transaction sheet that someone else paid the money to A.
2. Secondly, B deciphers the signature S on T_2 by using A's public key and obtains the plaintext X.
3. Then B combines the data from T_1 and his own public key, calculates a hash number Y.
4. Finally, B compares X and Y. If X equals to Y, then the transaction sheet T_2 is valid. T_2 must be issued by A, because only A can encrypt X to S by his private key. Moreover, the source of the amount is clear because all the information is included in T_1.

Although these steps are complex, it avoids some of the problems that may arise from the transaction processes. First of all, it guarantees no one else can imitate the payer's signature as long as he keeps the private key secret. From this point of view, it is more secure than our signature on the check. Secondly, it ensures that the payer has enough money in his account to make the payment. Because each transaction sheet contains the source of the money, nobody can create any Bitcoin out of thin air. However, there is still a problem: How to guarantee money is not used twice? This is explained in the next section.

Bitcoin Mining

Bitcoin was receiving attention when the global financial crisis occurred around 2009, with the price as low as five cents per one Bitcoin. However, the price of Bitcoin has been rising substantially since then, costing about 380 USD recently. Because the value of Bitcoin has been escalating, abundant number of individuals attempts to "mine" Bitcoin. The term "mining" conveys the process of attaching transaction records to preceding transactions. A "block" in mining is a record of the most recent Bitcoin transactions that have not been already recorded in the past blocks. The combination of these blocks is called the "block chain," which is a chain of previous transactions. The first transaction block in the history of Bitcoin is called a Genesis block.

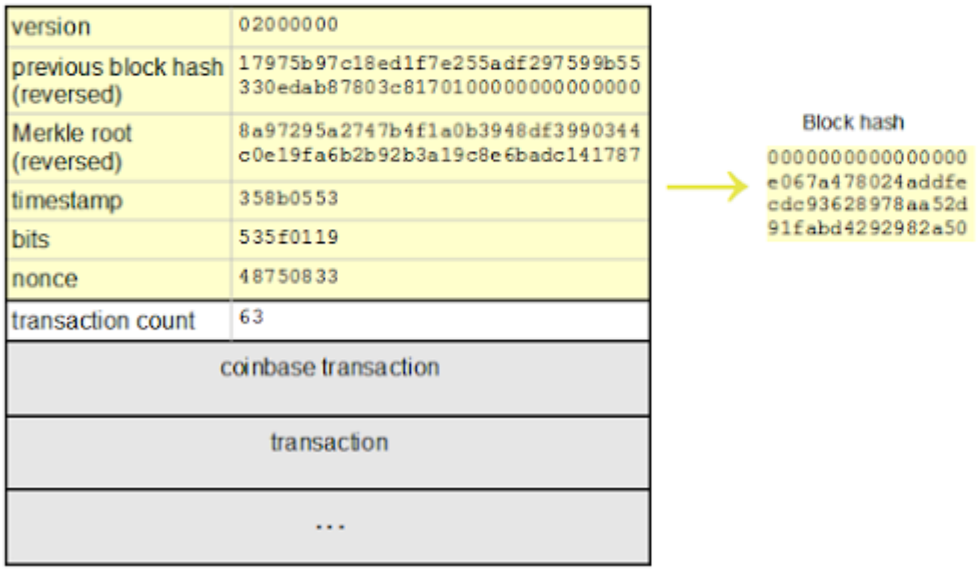
Most people who are not familiar with specifics of how Bitcoin works or in depth about how Bitcoin mining works would think that the term "mining" is just an action that creates a certain amount of Bitcoin out of the blue, just like mining for gold. Although generating a small amount of Bitcoin is true about mining, there are far more important roles of Bitcoin mining that makes Bitcoin more peculiar than other systems. The primary purpose of Bitcoin mining is to ensure the uniform vision of Bitcoin data. Since Bitcoin is a distributed peer-to-peer system, the central database that stores all the data used in transactions and such does not exist; however, all the tracks and logs of them are distributed throughout the network.

Although Bitcoin mining requires strenuous tasks to be performed, it is extremely easy to verify the results on the other hand. Bitcoin mining utilizes double SHA-256 cryptography system. Since it is not possible to straightforwardly solve for the hash value needed, the miners attempt countless different inputs, trying to guess the right input for the required output. Once the necessary hash value is found, it is easy to verify the hash. Due to these aspects of SHA-256 cryptographic system, proof-of-work of Bitcoin is implemented.

With more specification, the first step of mining a block involves collecting the new transactions into a block. Hashing the block to create a 256-bit hash value is the next step; various hash values get input, which they get modified slightly for abundant attempts. Hash beginning with enough zeros represents success of mining that block. When mining the block is completed, the block gets transmitted to the Bitcoin network and the hash becomes the identifier for the block. This process is called the Computationally-Difficult (Mining) Problem.

The chart below demonstrates the structure of a specific block and the process of getting hashed. The yellow highlighted part of the chart is the block header and it is followed by the block transactions.

The hash 0000000000000000e067a478024addfecdc93628978aa52d91fabd4292982a50 is successful and becomes #286819 block in the blockchain. The Python code to mine the given block is provided in the Appendix A.



version	02000000
previous block hash (reversed)	17975b97c18ed1f7e255adf297599b55 330edab87803c8170100000000000000
Merkle root (reversed)	8a97295a2747b4f1a0b3948df3990344 c0e19fa6b2b92b3a19c8e6badc141787
timestamp	358b0553
bits	535f0119
nonce	48750833
transaction count	63
coinbase transaction	
transaction	
...	

The block header consists of few pieces of descriptions of the block. The first part, the protocol version, is followed by the previous block hash, which ensures the entire blocks to structure an unbroken sequence in the blockchain. The Merkle root is a distinctive hash of all the transactions in the block. This is part of the Bitcoin's security that prevents others from tweaking the transactions after the transactions become a part of a block. Then there are timestamp and the mining difficulty level in bits. The final part is the nonce, an arbitrary value incremented on each hash attempt to provide a new hash value. Nonce is the problematic factor in mining, as it is challenging to discover a nonce that works.

It can be already seen that mining Bitcoins, i.e. coming up with a transaction block is very computationally complex and requires a lot of computing power. This complexity contributes to the security of the overall system, incentivizing the peers to be honest. To see how this system works, let us look at an example.

Assume some person wants to cheat the system and spend the same Bitcoins twice. This issue is referred to as "double spending". Because the network accepts the longer block chain (i.e. the one that has more work in it) if any two block chain conflict, all that a person has to do is come up with a longer block chain. Although, it sounds easy, it is not quite so. We have seen earlier that enormous amounts of computational power is needed to successfully generate a proof of work. In addition, the cheating person will be starting at a handicap, so they will require more CPU than the rest of the peer network has. If the person does not have that much CPU at their disposal, it will be virtually impossible to outrun the combined efforts of the honest nods in the system. If the person has enough computing power, it will be to their advantage to start mining Bitcoins instead, since it will provide a consistent revenue stream. Therefore, there is little incentive to try to cheat the system and the Bitcoin network thus promotes an honest environment.

This security protocol is enforced one step further by what is called a trusted timestamp. A trusted timestamp allows one to prove, that he possessed a document at certain point of time, in a way that cannot be forged. In the Bitcoin network, each block in the transaction block chain has a Unix time stamp. This part of the block serves two functions: it makes the block chain harder to manipulate and it provides additional variation for the block hash. In the Bitcoin system, a timestamp is accepted only if it is less than (network adjusted time + 2 hours) and greater than then median timestamp of the previous 11 blocks. These restrictions prevent and adversary from efficiently manipulating the block chains to his benefit.

Mathematics of the Bitcoin

Let us explain the concepts used above more rigorously from the mathematical point of view. Bitcoin network by itself is a system, which requires complex mathematical computations throughout the process of recording the transactions. One of the key roles is played by the hash function. Mathematically, an n-bit hash function is a map from arbitrary length messages to n-bit hash values. A hash function is a one-way and a collision-resistant function. *One-way* – it requires a work of approximately 2^n hash computations to find the hashed message. *Collision* – two distinct messages that hash to the same digest (usually requires about $2^{n/2}$ hash computations). Hashes are an important tool used for such things as digital signatures and password protection. In the scope of our paper, hash functions are used to compress the records about transactions into n-bit length strings. In addition, hash functions are also used by the participants in the peer-to-peer network as a part of the digital signature on the transactions.

There are many hash functions existing, however, one of the most widely used is the Secure Hash Algorithm (SHA). *SHA-2* is a set of cryptographic hash functions, designed by the NSA. The hash function that is used in the Bitcoin framework, *SHA-256* belongs to the set SHA-2. SHA-256 has received its name for outputting the digest of 256-bit length. SHA-256 provides 128 bits of security against collision attacks.

With the SHA-256, the input goes through some steps before the actual digest is output. First, the message is padded with its length so that the result is a multiple of 512 bits long. After this step is performed, the input is parsed into 512-bit message blocks $M(1), M(2), \ldots, M(N)$. At this point, the input is processed block-wise, one block at a time. For each block, the function initializes the hash value $H(0)$. Then, it initializes registers a, b, c, d, e, f, g, h with the $(i - 1)^{st}$ intermediate hash value. When this step is finished, the final hash value is computed by the following formula:

$$H(i) = H(i-1) + C \cdot M(i)(H(i-1))$$

where C is the SHA-256 compression function. This is repeated for every block until block $M(N)$ is reached.

The purpose that the hash serves is the verification that the encrypted message has not been altered during the transmission. When a person sends an encrypted message to the owner of the private key along with the hash, the owner can check whether the hash he computes and the one he receives match. This is possible because hash functions are collision-resistant.

Another important concept, which is used in the Bitcoin network widely, is the digital signature mechanism. Once again, there exist many protocols for signatures. However, there are some protocols that are typically preferred to the other. One of such protocols is the Digital signature standard (DSS), which outlines the Digital Signature Algorithm (DSA). The described method has been fully described in the document published by the U.S. Department of Commerce. To successfully sign something digitally, some parameters have to be specified and computed. Firstly, the prime modulus p and the prime divisor of $(p - 1)$, q, should be specified. The prime modulus p of bit-length L should be in the range $(2^{L-1}, 2^{L})$ and the prime divisor q of bit-length should be in the range $(2^{N-1}, 2^{N})$. A generator of a subgroup of order q, denoted by g, is also required by the DSA. The signer should have his own secret private key, x, which should be in the range $[1, q - 1]$. This key will be used as an encryption function to encode the message.

The signing person should also have a public key, *y*, which is related to the private key in the following manner: $y = g^x \bmod p$. A per-message secret number k in the range [1, *q* - 1] should be generated every time. An important security feature is that k should be randomly or pseudo-randomly generated every time a new message is being signed. This ensures that the computation of the private key remains computationally complex for someone who only knows the input text and the public key. The parameters that vary every time are the data to be signed, and the hash function digest. It is important to note that the Digital Signature Standard requires that only the approved hash functions should be used.

The DSS also specifies a list of the values for *L* and *N*. According to the document, the *L* and *N* lengths of *p* and *q* respectively should be one of the following pairs:

$L = 1024, N = 160$ **or** $L = 2048, N = 224$ **or** $L = 2048, N = 256$ **or** $L = 3072, N = 256$

The signature verification process involves a few computation parts, which help ensure that no portion of the message has been altered and that the signature truly belongs to the owner of the private key. For demonstration purposes, assume that *M', r'*, and *s'* are the received versions of *M, r*, and *s*, where *M* is the message, and *r* and *s* are computed in the following manner:

$r = (g^k \bmod p) \bmod q$

z = the leftmost **min**(*N*, *outlen*) bits of **Hash**(*M*)

$s = (k^{-1}(z + xr)) \bmod q$

The *outlen* number is the bit length of the digest of the hash function. It is worth noting that *z* is therefore a string, and thus should be converted into an integer before *s* can be computed. A string *z* of a kind $\{z_1, z_2, \ldots, z_n\}$ is mapped to ℝ in the following way:

$$\{z_1, z_2, \ldots, z_n\} \mapsto (z_1 \cdot 2^{n-1}) + (z_2 \cdot 2^{n-2}) + \cdots + (z_{n-1} \cdot 2) + z_n$$

Now that we understand what the terms *M, r* and *s* stand for, the verification algorithm can be described. First, the algorithm checks whether $0 < r' < q$ <u>and</u> $0 < s' < q$. If either is violated, the algorithm will terminate and reject the signature. After this step, the following values are computed:

$w = (s') - 1 \bmod q$

$z =$ the leftmost $\mathbf{min}(N, outlen)$ bits of $\mathbf{Hash}(M')$

At this point, the string *z* shall be converted to an integer using the mapping above.

$u1 = (zw) \; mod \; q$

$u2 = ((r')w) \; mod \; q$

$v = (((g)^{u1}(y)^{u2}) \bmod p) \bmod q$

Once the values are computed, it takes only one step to check whether the signature should be rejected or accepted. If $v = r'$, the signature is verified, and if $v \neq r'$, the signature is considered to be invalid and is rejected.

Unlike most other currencies in the world, the Bitcoin currency has an upper bound to how many Bitcoins can be circulating in the whole network. The number is 21,000,000 (21 million) BTC and it is estimated that the upper bound will be reached in 2140, after which new Bitcoins will not be generated. However, this seems like the system will crash at that point, most believe that Bitcoin will be so widespread by then that the transaction fees (which are part of the pay of the miners) will be sufficient enough to keep the incentive to mine new transaction blocks.

To ensure that the maximum supply is not reached too soon, a few measures are currently implemented in the system. Firstly, the amount of Bitcoins that is allowed to be generated by the miner decreases over time. To be more precise, the reward decreases by a half every 210,000 (210 thousand blocks). This is approximately equal to four year (210,000/6 blocks per hour/24 hours in a day/365 days in a year = 3.995 years ≈ 4 years). Furthermore, every 2,016 blocks (two weeks), the system approximates how long it took to generate those blocks. If the amount of time required deviates significantly, the necessary adjustments are made.

Conclusion

Bitcoin is one of the most popular virtual currencies nowadays. The decentralized system of peer-to-peer interactions assures that no single institution can affect the natural flow of the currency. The currency itself is generated as a reward for the computational power used. As we have seen, Bitcoin system is a fairly complex one both in terms of computations and in terms of computations. Therefore, individuals who agree to contribute to maintenance of the system shall be rewarded accordingly both for their time and the resources spent (electricity, CPU, etc.). The Bitcoin system uses a wide range of cryptological and mathematical tools, most of which are standardized (like SHA-256 or DSA). Bitcoin network is close to impossible to breach and is protected against untruthful behavior through the collective power of every peer participating in the network.

Bibliography

"Descriptions of SHA-256, SHA-384, and SHA-512." (2012). National Institute of Standards and Technology. Information Technology Laboratory. National Institute of Standards and Technology. Web. 20 Nov. 2014. <http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf>.

"Digital Signature Standard (DSS)." *Federal Information Processing Standards Publication* 186.4 (2013). Print.

Shirriff, Ken. "Bitcoin Mining the Hard Way: The Algorithms, Protocols, and Bytes." *Ken Shirriff's Blog*. 23 Feb. 2014. Web. 20 Nov. 2014. <http://www.righto.com/2014/02/bitcoin-mining-hard-way-algorithms.html>.

Appendix A. Python code to mine the block

```
import hashlib, struct
ver = 2
prev_block = "000000000000000117c80378b8da0e33559b5997f2ad55e2f7d18ec1975b9717"
mrkl_root = "871714dcbae6c8193a2bb9b2a69fe1c0440399f38d94b3a0f1b447275a29978a"
time_ = 0x53058b35 # 2014-02-20 04:57:25
bits = 0x19015f53

# https://en.bitcoin.it/wiki/Difficulty
exp = bits >> 24
mant = bits & 0xffffff
target_hexstr = '%064x' % (mant * (1<<(8*(exp - 3))))
target_str = target_hexstr.decode('hex')
nonce = 0
while nonce < 0x100000000:
    header = ( struct.pack("<L", ver) + prev_block.decode('hex')[::-1] +
    mrkl_root.decode('hex')[::-1] + struct.pack("<LLL", time_, bits, nonce))
    hash = hashlib.sha256(hashlib.sha256(header).digest()).digest()
    print nonce, hash[::-1].encode('hex')
    if hash[::-1] < target_str:
    print 'success'
    break
    nonce += 1;
```