Ken Shirriff's blog

Computer history, restoring vintage computers, IC reverse engineering, and whatever

Bitcoin mining the hard way: the algorithms, protocols, and bytes

This article explains Bitcoin mining in details, right down to the hex data and network traffic. If you've ever wondered what really happens in Bitcoin mining, you've come to the right place. My previous article, Bitcoins the hard way described how I manually created a Bitcoin transaction and sent it into the system. In this article, I show what happens next: how a transaction gets mined into a block.

The purpose of mining

Bitcoin mining is often thought of as the way to create new bitcoins. But that's really just a secondary purpose. The primary importance of mining is to ensure that all participants have a consistent view of the Bitcoin data. Because Bitcoin is a distributed peer-to-peer system, there is no central database that keeps track of who owns bitcoins. Instead, the log of all transactions is distributed across the network.

The main problem with a distributed transaction log is how to avoid inconsistencies that could allow someone to spend the same bitcoins twice. The solution in Bitcoin is to *mine* the outstanding transactions into a block of transactions approximately every 10 minutes, which makes them official. Conflicting or invalid transactions aren't allowed into a block, so the double spend problem is avoided.

Although mining transactions into blocks avoid double-spending, it raises new problems: What stops people from randomly mining blocks? How do you decide who gets to mine a block? How does the network agree on which blocks are valid? Solving those problems is the key innovation of Bitcoin: mining is made very, very

difficult, a technique called proof-of-work. It takes an insanely huge amount of computational effort to mine a block, but it is easy for peers on the network to verify that a block has been successfully mined. [1]

Each mined block references the previous block, forming an unbroken chain back to the first Bitcoin block. This blockchain ensures that everyone agrees on the transaction record. It also ensures that nobody can tamper with blocks in the chain since re-mining all the following blocks would be computationally infeasible. [2] As long as nobody has more than half the computational resources, mining remains competitive and nobody can control the blockchain.

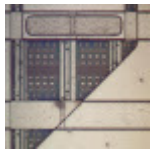
As a side-effect, mining adds new bitcoins to the system. For each block mined, miners currently get 25 new bitcoins (currently worth about $15,000), which encourages miners to do the hard work of mining blocks. With the possibility of receiving $15,000 every 10 minutes, there is a lot of money in mining.

How mining works

Mining requires a task that is very difficult to perform, but easy to verify. Bitcoin mining uses cryptography, with a hash function called double SHA-256. A hash takes a chunk of data as input and shrinks it down into a smaller hash value (in this case 256 bits). With a cryptographic hash, there's no way to get a hash value you want without trying a whole lot of inputs. But once you find an input that gives the value you want, it's easy for anyone to verify the hash. Thus, cryptographic hashing becomes a good way to implement the Bitcoin "proof-of-work".

In more detail, to mine a block, you first collect the new transactions into a block. Then you hash the block to form a 256-bit block hash value. If the hash starts with enough zeros [3], the block has been successfully mined and is sent into the Bitcoin network and the hash becomes the identifier for the block. Most of the time the hash isn't successful, so you modify the block slightly and try again, over and over billions of times. About every 10 minutes someone will successfully mine a block, and the process starts over.

The diagram below shows the structure of a specific block, and how it is hashed. The yellow part is the block header, and it is followed by the transactions that go into the



Teardown of a logic chip from a vintage IBM ES/9000 mainframe

A Multi-Protocol Infrared Remote Library for the Arduino

Bitcoin mining the hard way: the algorithms, protocols, and bytes

Apple iPhone charger teardown: quality in a tiny expensive package

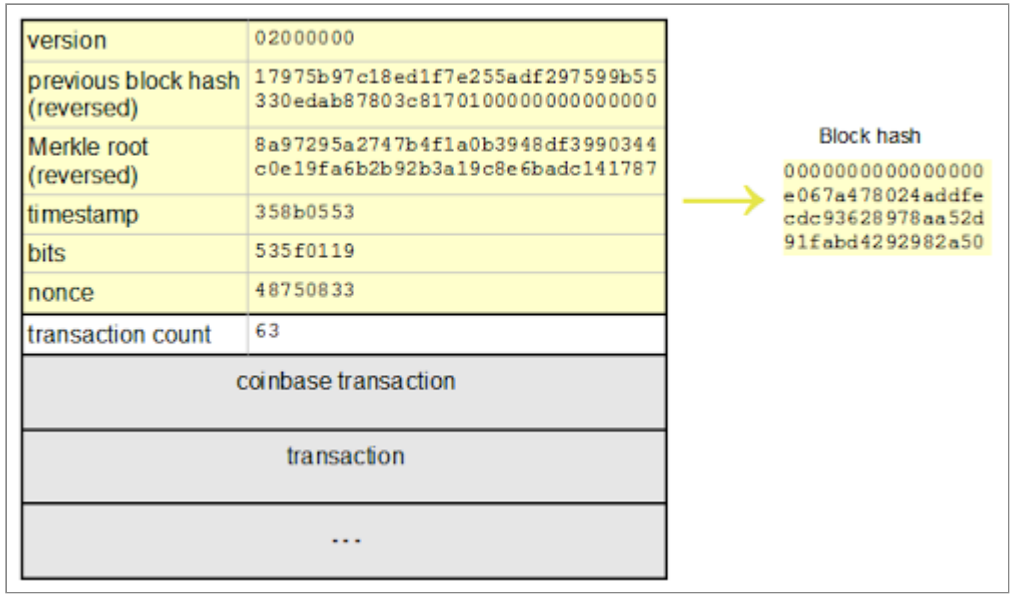
A dozen USB chargers in the lab: Apple is very good, but not quite the best

Search This Blog

Search

Labels

block. The first transaction is the special *coinbase* transaction that grants the mining reward to the miner. The remaining transactions are standard Bitcoin transactions moving bitcoins around. If the hash of the header starts with enough zeros[3], the block is successfully mined. For the block below, the hash is successful: `0000000000000000e067a478024addfecdc93628978aa52d91fabd4292982a50` and the block became block #286819 in the blockchain.



Structure of a Bitcoin block

The block header contains a handful of fields that describe the block. The first field in the block is the protocol version. It is followed by the hash of the previous block in the blockchain, which ensures all the blocks form an unbroken sequence in the blockchain. (Inconveniently, the hash is reversed in the header.) The next field is the *Merkle root*,[4] a special hash of all the transactions in the block. This is also a key part of Bitcoin security, since it ensures that transactions cannot be changed once they are part of a block.[5] Next is a (moderately accurate) timestamp of the block, followed by the mining difficulty value *bits*.[3] Finally, the *nonce* is an arbitrary value that is incremented on each hash attempt to provide a new hash value. The tricky part of mining is finding a nonce that works.

6502 8008 8085 8086 8087 alto analog Apollo apple arc arduino arm beaglebone bitcoin c# calculator chips css electronics f# fpga fractals genome haskell html5 ibm ibm1401 intel ipv6 ir java javascript math oscilloscope photo power supply random reverse-engineering sheevaplug snark space spanish teardown theory unicode Z-80

Blog Archive

► 2021 (11)
► 2020 (33)
► 2019 (18)
► 2018 (17)
► 2017 (21)
► 2016 (34)
► 2015 (12)
▼ 2014 (13)
  ► December (1)
  ► October (1)
  ► September (3)
  ► May (2)
  ► March (1)

ASIC Bitcoin Miner
Photo by Mirko Tobias Schaefer, (CC BY 2.0)

A short program to mine a block

I wrote a Python program that mines the above block. The program itself is pretty simple - the hardest part of the code is computing the difficulty target from *bits*.[3] Otherwise it's just a loop over different nonce values. Each iteration puts the data into a structure, hashes it, and tests the result.

```
18            mrkl_root.decode('hex')[::-1] + struct.pack("<LLL", time_, bits, nonc
19        hash = hashlib.sha256(hashlib.sha256(header).digest()).digest()
20        print nonce, hash[::-1].encode('hex')
21        if hash[::-1] < target_str:
22            print 'success'
23            break
24        nonce += 1
```

mine.py hosted with ♥ by GitHub view raw

The following table shows the hash obtained for selected nonce values. The key point is that each nonce generates a basically-random hash value. Every so often a "lucky" nonce will generate a hash starting with some zeroes. To get a lot of zeroes,

you need to try an exponentially large number of nonces. For this block, the "winning" nonce is 856192328.

nonce	hash
0	5c56c2883435b38aeba0e69fb2e0e3db3b22448d3e17b903d774dd5650796f76
1	28902a23a194dee94141d1b70102accd85fc2c1ead0901ba0e41ade90d38a08e
2	729577af82250aaf9e44f70a72814cf56c16d430a878bf52fdaceeb7b4bd37f4
3	8491452381016cf80562ff489e492e00331de3553178c73c5169574000f1ed1c
39	03fd5ff1048668cd3cde4f3fb5bde1ff306d26a4630f420c78df1e504e24f3c7
990	0001e3a4583f4c6d81251e8d9901dbe0df74d7144300d7c03cab15eca04bd4bb
52117	0000642411733cd63264d3bedc046a5364ff3c77d2b37ca298ad8f1b5a9f05ba
1813152	00000c94a85b5c06c9b06ace1ba7c7f759e795715f399c9c1b1b7f5d387a319f
19745650	000000cdccf49f13f5c3f14a2c12a56ae60e900c5e65bfe1cc24f038f0668a6c
243989801	0000000ce99e2a00633ca958a16e17f30085a54f04667a5492db49bcae15d190
856192328	0000000000000000e067a478024addfecdc93628978aa52d91fabd4292982a50

I should point out that I cheated by starting with a block that could be successfully mined. Most of the attempts to mine a block will fail entirely - none of the nonce values will succeed. In that case, you need to modify the block slightly and try again. The timestamp can be adjusted (which is why the timestamp in mined blocks is often wrong). New transactions can be added to the block, changing the Merkle hash. The coinbase transaction can be modified - this turns out to be very important for mining pools. Any of these changes will result in totally different hashes, so the nonce values can be tried again.

My Python program does about 42,000 hashes per second, which is a million times slower than the hardware used by real miners. My program would take about 11 million years on average to mine a block from scratch.

Mining is very hard

The difficulty of mining a block is astounding. At the current difficulty, the chance of a hash succeeding is a bit less than one in 10^{19}. Finding a successful hash is harder than finding a particular grain of sand from all the grains of sand on Earth. To find a hash every ten minutes, the Bitcoin hash rate needs to be insanely large. Currently,

the miners on the Bitcoin network are doing about 25 million gigahashes per second. That is, every second about 25,000,000,000,000,000 blocks gets hashed. I estimate (very roughly) that the total hardware used for Bitcoin mining cost tens of millions of dollars and uses as much power as the country of Cambodia.[6]

Note that finding a successful hash is an entirely arbitrary task that doesn't accomplish anything useful in itself. The only purpose of finding a small hash is to make mining difficult, which is fundamental to Bitcoin security. It seems to me that the effort put into Bitcoin mining has gone off the rails recently.

Mining is funded mostly by the 25 bitcoin reward per block, and slightly by the transaction fees (about 0.1 bitcoin per block). Since the mining reward currently works out to about $15,000 per block, that pays for a lot of hardware. Per transaction, miners are getting about $34 in mining reward and $0.10 in fees (stats).

15 GH/s FPGA Bitcoin mining configuration with 41 Icarus. Photo by permission of Xiangfu Liu

Mining with a pool

Because mining is so difficult, it is typically done in mining pools, where a bunch of miners share the work and share the rewards. If you mine by yourself, you might

successfully mine a block and get 25 bitcoin every few years. By mining as part of a pool, you could get a fraction of a bitcoin every day instead, which for most people is preferable.

Mining pools use an interesting technique to see how much work miners are doing. They send out a block to be mined, and get updates from a miner whenever a miner gets a partial solution. Each partial solution proves the miner is working hard on the problem and gives the miner a share in the final reward when someone succeeds in mining the block.

For instance, if Bitcoin mining requires a hash starting with 15 zeroes, the mining pool can ask for hashes starting with 10 zeroes, which is a million times easier. Depending on the power of their hardware, a miner might find such a solution every few seconds or a few times an hour. Eventually one of these solutions will start with not just 10 zeroes but 15 zeroes, successfully mining the block and winning the reward for the pool.[7] The reward is then split based on each miner's count of shares as a fraction of the total, and the pool operator takes a small percentage for overhead.[8]

Most of the time someone outside the pool will mine a block first. In that case, the pool operator sends out new data and the miners just start mining the new block. People in a pool can get edgy if a long time goes without a payout because of bad luck in mining.

Stratum: The communication between a pool and the miners

Next I'll look in detail at the communication between a miner and the mining pool. The communication between the pool and the miners is interesting. The pool must efficiently provide work to the miners and collect their results quickly. The pool must make sure miners aren't duplicating work. And the pool must make sure miners don't waste time working on a block that has already been mined.

An important issue for mining pools is how to support fast miners. The nonce field in the header is too small for fast miners since they will run through all the possible values faster than the pool can send blocks. The solution is to allow miners to update

the coinbase transaction so they can put additional nonces there. This makes mining more complicated since after building the coinbase transaction the miner must recompute the Merkle hash tree and then try mining the block.

I'm going to look at the Stratum mining pool protocol that is used by many pools. (Some alternative protocols are the Getwork and Getblocktemplate protocols.) The following Python program uses the Stratum protocol to make a mining request to the GHash.IO mining pool and displays the results. (This program is a minimal demonstration; don't use this code for real mining.)

```
1  import socket
2
3  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4  sock.connect(("us1.ghash.io", 3333))
5
6  sock.send("""{"id": 1, "method": "mining.subscribe", "params": []}\n""")
7  print sock.recv(4000)
```

stratum.py hosted with ♥ by GitHub view raw

The information below is what the mining pool sends back over the network in response to the program above. Since the Stratum protocol uses JSON-RPC the results are readable ASCII rather than the binary packets used by most of Bitcoin. This provides all the data needed to start mining as part of the pool:

```
{"id":1,"result":
[[["mining.set_difficulty","b4b6693b72a50c7116db18d6497cac52"],
["mining.notify","ae6812eb4cd7735a302a8a9dd95cf71f"]],"4bc6af58",4],
"error":null}

{"id":null,"params":[16],"method":"mining.set_difficulty"}

{"id":null,"params":
["58af8d8c","975b9717f7d18ec1f2ad55e2559b5997b8da0e3317c803780000000
100000000","01000000010000000000000000000000000000000000000000000000
00000000000000000ffffffff4803636004062f503253482f04428b055308","2e5
```

```
22cfabe6d6da0bd01f57abe963d25879583eea5ea6f08f83e3327eba9806b1411971
8cbb1cf040000000000000000000001fb6734950000000001976a91480ad90d4035
81fa3bf46086a91b2d9d4125db6c188ac00000000",
["ea9da84d55ebf07f47def6b9b35ab30fc18b6e980fc618f262724388f2e9c591",
"f8578e6b5900de614aabe563c9622a8f514e11d368caa78890ac2ed615a2300c","
1632f2b53febb0a999784c4feb1655144793c4e662226aff64b71c6837430791","a
d4328979dba3e30f11c2d94445731f461a25842523fcbfa53cd42b585e63fcd","a9
04a9a41d1c8f9e860ba2b07ba13187b41aa7246f341489a730c6dc6fb42701","dd7
e026ac1fff0feac6bed6872b6964f5ea00bd8913a956e6b2eb7e22363dc5c","2c3b
18d8edff29c013394c28888c6b50ed8733760a3d4d9082c3f1f5a43afa64"],"0000
0002","19015f53","53058b41",false],"method":"mining.notify"
{"id":2,"result":true,"error":null}
{"id":null,"params":[16],"method":"mining.set_difficulty"}
```

The first line is a response from the pool server with the subscription details. The first values are not too important. The value `4bc6af58` is the value *extranonce1* that is used when building the block. Each client gets a unique value to ensure that all the mining clients generate unique blocks and don't duplicate work. The following value (`4` bytes) is the length of the *extranonce2_size* value that the miner puts in the coinbase while mining.

The second line is a *mining.set_difficulty* message to our client. With a difficulty of 16, I can get a share every hour or two on my PC. In comparison, the Bitcoin mining difficulty is 3,129,573,174.52 [3] - thus it's about 200 million times easier to get a share in this pool than to successfully mine a block independently. That's why people join pools.

The third line is a *mining.notify* notification to our client. This message defines that block for us to mine. There's a lot of data returned under "params", so I'll explain it field by field.

job_id	58af8d8c
prevhash	975b9717f7d18ec1f2ad55e2559b5997b8da0e3317c803780000000100000000
coinb1	01000000010000000000000000000000000000000000000000000000000000000000000000 0000000000ffffffff4803636004062f503253482f04428b055308
coinb2	2e522cfabe6d6da0bd01f57abe963d25879583eea5ea6f08f83e3327eba9806b

	14119718cbb1cf0400000000000000000000000001fb673495000000001976a914 80ad90d403581fa3bf46086a91b2d9d4125db6c188ac00000000
merkle_branch	["ea9da84d55ebf07f47def6b9b35ab30fc18b6e980fc618f262724388f2e9c591", ...]
version	00000002
nbits	19015f53
ntime	53058b41
clean_jobs	false

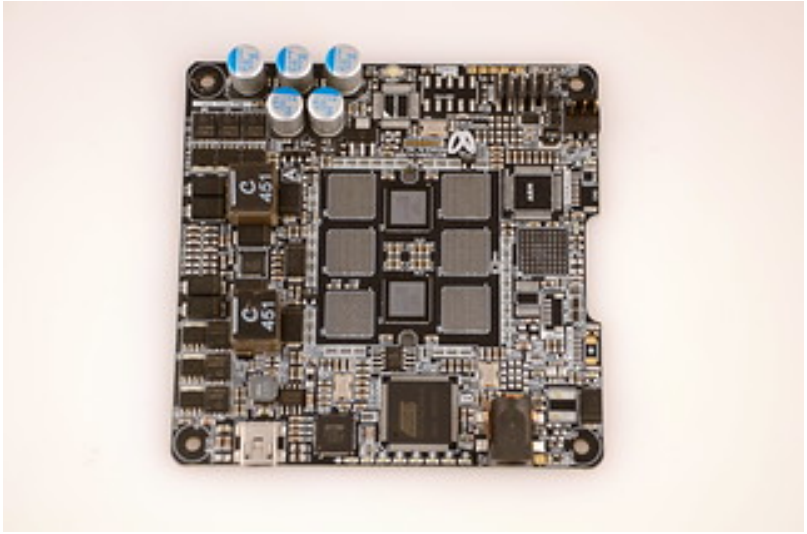
The *job_id* is used to identify this mining task if the miner reports back success.

Most of the fields are used in the block header. The *prevhash* is the hash of the previous block. Apparently mixing big-ending and little-endian isn't confusing enough so this hash value also has every block of 4 bytes reversed. The *version* is the block protocol version. The *nbits* indicates the difficulty [3] of the block. The timestamp *ntime* is not necessarily accurate.

The *coinb1* and *coinb2* fields allow the miner to build the coinbase transaction for the block. This transaction is formed by concatenating *coinb1*, the *extranonce1* value obtained at the start, the *extranonce2* that the miner has generated, and *coinb2*. The result is a transaction in Bitcoin protocol. The *merkle_branch* hash list lets the miner efficiently recompute the Merkle hash with the new coinbase transaction.

clean_jobs is used if the miner needs to restart the mining jobs.

After receiving this data, the miner can start generating coinbase transactions and mining blocks.



Butterfly Labs Jalapeño ASIC miner, 7+ GH/s, by 0xF2,

Creating a block for a pool

Once the miner has received the information from the pool, it is straightforward to form the coinbase transaction by joining the *coinb1*, *extranonce1*, *extranonce2*, and *coinb2* to form a coinbase transaction. The diagram below shows how the combination of these four values forms a complete transaction, with the nonces in the middle of the coinbase script. (The block below is slightly different from the one described earlier.)

version	01000000
input count	01
previous hash	0000000000000000000000000000000000000000000000000000000000000000
index	ffffffff
scriptlen	60
script	03636004062f503253482f04358b0553084404f253000017e446522cfabe6d6d690688fb886c0df0c87cbc7ea4f7f1b5c0050bd0ac3751cfc997d9d6971328de04000000000000004861707079204e592120596f757273204748617368e2e494f
sequence	00000000
output count	01
value	cb81319500000000
scriptlen	19
script	76a91480ad90d403581fa3bf46086a91b2d9d4125db6c188ac
lock time	00000000

A coinbase transaction generated by the GHash.io mining pool

The structure of the coinbase transaction is similar to a regular transaction, but there are a few important differences. A normal transaction transfers bitcoins from *inputs* (usually source addresses) to *outputs* (usually destination addresses). A coinbase transaction is generating new bitcoins out of thin air, rather than doing a transfer, so the transaction is slightly different. The previous output hash and index are irrelevant for the coinbase transaction. the first script is the *scriptSig* which signs the transaction to prove ownership of the incoming bitcoins. In a coinbase transaction, this is irrelevant, so instead the field is called the *coinbase* and is mostly arbitrary data.[9] (Many miners hide messages in there.) The *value* field in the coinbase transaction is the 25 bitcoin mining reward plus any bitcoins left over from the other transactions (the left over bitcoins are treated as mining fees). Finally, both regular transactions and the coinbase transaction use the second script (*scriptPubKey*) to specify the recipients of the bitcoins.[10] For details on transactions, see my my previous article.

Once the coinbase transaction is created, the hash for this coinbase transaction is combined with the *merkle_branch* data from the pool to generate the Merkle hash [4] for the entire set of transactions. Because of the structure of the Merkle hash (explained below), this allows the hash for the entire set of transactions to be recomputed easily.

Finally, the block header is built from the new Merkle hash and the data provided by the pool, and the hash algorithm can iterate over the nonce values in the header, just like the Python program earlier. Once all the nonce values have been tried, the miner increments the extranonce2, generates a new coinbase transaction and continues.

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

A Bitcoin block header

Informing the mining pool of success

The difficulty [3] for a mining pool is set much lower than the Bitcoin mining difficulty (fewer leading zeros required), so it's much easier to get a share. When a block is hashed to the pool's difficulty, you send a simple JSON message to the mining pool to submit it:

```
{"method": "mining.submit", "params": ["kens.worker1", "58af8db7",
"00000000", "53058d7b", "e8832204"], "id":4}
```

The parameters are the worker name, job id, extranonce2, time, and header nonce. This information is sufficient for the pool to build the matching coinbase transaction and header, and verify the block. If the hash meets the pool difficulty, you get a share. If the hash also meets the much, much harder Bitcoin difficulty, the block has

been successfully mined. In this case the pool submits the block to the Bitcoin network and everyone with shares gets paid accordingly.

Mining for fun and profit

If you're curious about mining, it's surprisingly easy to try out mining yourself, although you'll be lucky to earn even a penny. Just create an account at a mining pool such as BTC Guild, download mining software such as cpuminer (minerd.exe), and run the software to start mining. For a pool with low difficulty, you should get shares in a few minutes; in a pool with a higher difficulty (such as GHash.IO), it may take you an hour or two to get a share, which is more frustrating. [3]

Unprofitable Bitcoin CPU mining on my PC

The screenshot above shows what mining looks like as you get shares and blocks get mined. I got lucky and it only took me a minute to successfully mine a share. A

minute later someone successfully mined a block, so the pool tells everyone to start over. Another block was mined less than a minute after that - although blocks are 10 minutes apart on average, the times can vary widely. It took 12 minutes for my next share to be generated. After running for a while, I earned 0.00000043 BTC, which is a tiny fraction of a cent.

Bitcoin mining is an "arms race". Originally people could mine with the CPU on a regular PC, but that hasn't been practical for a while. Next mining was offloaded to GPUs. Now, mining is done with special-purpose ASIC hardware, which is rapidly increasing in speed. For-profit mining is very competitive, and you'll need to look elsewhere for information.

If you want to try out mining just for fun, you may prefer to mine a currency such as Dogecoin rather than Bitcoin. First, Dogecoin uses a different hash algorithm which doesn't work well with ASIC hardware, so you're not as disadvantaged compared to professional miners. Second, because dogecoins are worth much less than bitcoins, you'll end up with a much larger number of dogecoins, which seems more rewarding. For Dogecoin mining, I used the dogepool.pw pool somewhat arbitrarily. The process is almost the same as Bitcoin mining, except you use the *scrypt* algorithm instead of *sha256d*. There are many other alternative cryptocurrencies to choose from.

Notes and references

[1] Bitcoin mining seems like a NP (nondeterministic polynomial) problem since a solution can be quickly verified. However, there are a couple of issues with making this rigorous. First, since hashes are a fixed size, mining can be done in constant time (but with a very large constant of 2^256). Thus, you'd need to consider an extended mining scheme where the difficulty can go to infinity. Second, mining would need to be turned into a decision problem - e.g. instead of finding a nonce, the problem would be "Is there a successful nonce less than k". (Note that if you can solve that problem, you can rapidly find the nonce with binary search.)

With these changes, the mining problem is in NP. The next question is if it is NP-complete. That is, can an arbitrary NP-complete problem be turned into a mining problem? I believe that is currently unknown.

[2] You might wonder what happens if two miners succeed in mining a block at approximately the same time. Has the problem of conflicting transactions has just been replaced by the problem of conflicting blocks?) The rule is that only the longest chain of valid blocks is used, and the other branch is ignored. Thus, when a miner extends the chain with one of the two parallel blocks, the other block becomes an orphan block and is ignored.

Orphan blocks are fairly common, roughly one a day. For this reason, the (somewhat arbitrary) recommendation is to wait for six confirmations (about one hour) before considering a transaction solidly confirmed.

[3] I've been describing a successful hash as starting with enough zeros, but there's an official definition of difficulty. A valid block must have a hash below a target value. (Since the target starts with a bunch of zeros, so will the valid hash.)

There are two different hard-to-understand ways of representing the target. The first, *bits* is a mantissa/exponent representation of the target in 32 bits. The second, *difficulty* is the ratio between a base target and the current target. A difficulty of *N* is *N* times as difficult as this base target. The base target is `0x00000000FFFF0000000000000000000000000000000000000000000000000000`, which corresponds to approximately 1 in 2^{32} or 1 in 4.2 billion hashes succeeding.

Difficulty changes approximately every two weeks to keep the block hash rate around 1 every 10 minutes. The https://blockchain.info/stats difficulty value is 3,129,573,174.52, corresponding to a target of `00000000000000015f530000000000000000000000000000000000000000000`. Multiplying my PC's performance by the current difficulty shows it would take my PC about 35,000 years to mine a block.

The pool difficulty is important when using a mining pool. My PC can do about 12 million hashes/sec running cpuminer, so at a difficulty of 1 my PC could find a block every 6 minutes. The BTC Guild pool uses a difficulty of 2, so I get a share about every 12 minutes. GHash.IO has a minimum difficulty of 16 on the other hand, so I

only get a share every hour or two on the average. (My overall earnings would be similar either way, since the shares per block scale inversely with the difficulty.)

[4] Instead of hashing all the transactions into the block directly, the transactions are first hashed together to yield a *Merkle root*. The Merkle root is the root of a binary Merkle tree. The idea is to start with all the transaction hashes. Pairs of hashes are hashed together to yield new hashes. The process is repeated on the new list of hashes and continues recursively until a single hash is obtained. This final root hash is the value used when computing the block. (See Wikipedia for more details.)

In the Merkle tree, each transaction is hashed. Then pairs of hashes are hashed together. Then pairs of the new hashes are hashed together, and so on, until a single hash remains. This allows the hash of a single transaction to be verified efficiently without recomputing all the hashes. One place this comes in useful is generating a new coinbase transaction for a mining pool.

The (patented) idea of a Merkle tree is if you need to modify or verify a single transaction, you don't need to recompute everything, but can just recompute the affected pairs. Personally, I think the Merkle tree is a pointless optimization for Bitcoin and for reasonable transaction numbers it would be faster to do a single large hash, rather than multiple hashes up the Merkle tree.

Here's some demonstration code to compute the Merkle root for the block I'm discussing. The 99 transaction hashes are hard-coded for convenience. The resulting Merkle root is `871714dcbae6c8193a2bb9b2a69fe1c0440399f38d94b3a0f1b447275a29978a`

```
import hashlib

# Hash pairs of items recursively until a single value is obtained
def merkle(hashList):
    if len(hashList) == 1:
        return hashList[0]
    newHashList = []
```

merkle.py hosted with ♥ by GitHub view raw

[5] There are a few ways that third parties can modify transactions without invalidating the signature on the transaction. This is known as transaction malleability. These modifications change the hash of the transaction. Since the hash is part of the block, a transaction has a fixed hash and cannot be modified by malleability once it has been mined into a block. (Unless the whole block is orphaned, of course.)

[6] It's hard to estimate the cost of mining because the hardware is changing so rapidly and it's unclear what is actually in use, but I'll do a rough calculation. Looking at the Bitcoin mining hardware and Mining hardware comparison pages, the HashBlaster looks like the most efficient currently available at 375 MH/s/$ and 1818 MH/s/W. The Bitcoin network is 25 billion MH/s, which works out to about $70 million hardware cost and 15 MW. (This is about the total power consumption of Cambodia.) At $0.15/kWH, that would be about $50,000/day on electricity ($300 per block or $0.70 per transaction). Since mining generates about $140,000 per day, spending $50,000 per day on electricity seems like the right ballpark. Other estimates are at Hacker News.

[7] You might wonder why a miner doesn't cheat. If they successfully mine a block, why not submit it themselves so they can claim the full mining reward, rather than splitting it? The main reason is the coinbase transaction has the pool's address, not the miner's address. If the miner submits the block bypassing the pool, the reward still goes to the pool. And if the miner changes the address, the hash is no longer valid.

[8] There are several different reward systems used by mining pools. For instance, a pool can pay out the exact amount earned from a block or an average amount. Or a pool can pay a fixed amount per share. A pool can weight shares by time to avoid miners switching between pools mid-block. These different systems can balance risk between the miners and the pool operator and adjust the variance of payments. For details, see the Bitcoin wiki here or here.

[9] I've figured out a lot of the structure of the coinbase script above. First it contains the block height (0x046063 or 286819), which is required for version 2). Next is the string '/P2SH/' which indicates the miner supports Pay To Script Hash). This is followed by a timestamp. Next is 8 bytes of the two nonces. This is followed by apparently-random data and then the text "Happy NY! Yours GHash.IO".

[10] The typical coinbase script format has changed over time. Originally, the output scripts were all pay-to-pubkey, with the script: *public_key* OP_CHECKSIG. This script puts the public key itself in the script. However, now about 95% of coinbase transactions use the standard pay-to-pubkey-hash script: OP_DUP OP_HASH160 *addr* OP_EQUALVERIFY OP_CHECKSIG. This script only includes the public key hash (the address) and requires the redeemer to provide the public key. To see the difference, compare the output scripts in this transaction and this transaction.

Labels: bitcoin

66 comments:

Peter Todd said...

> Personally, I think the Merkle tree is a pointless optimization for Bitcoin and for reasonable transaction numbers it would be faster to do a single large hash, rather than multiple hashes up the Merkle tree.

The merkle tree is a *critical* optimization for Bitcoin - it's what makes SPV wallets like Multibit possible. In fact the among the experts there's consensus that the merkle tree should have extended into transactions themselves, so that all the inputs and outputs of a transaction would be committed to via a merkle tree. In the future this will probably be done, and is needed for things like fraud proofs.

Incidentally, here's a fairly complete and "pythonistic" Python library for Bitcoin: https://github.com/petertodd/python-bitcoinlib It's a "ground up" library that re-implements all the Satoshi bitcoin functionality, and is focused on making low-level code easy to write in Python. Network code is still in flux, but there exists an RPC

module for use with a local bitcoin node. A simple example of that type of use is in my dust-b-gone.

February 23, 2014 at 5:22 PM

Peter Todd said...

As for the overhead of using a merkle tree to hash some data verses hashing it in one go, it's roughly speaking double the work. It's easy to see why if you remember your sum-of-series stuff from highschool: So the first pass consumes n 64-byte blocks, producing n/2 digests. The second takes those n/2 digests, and produces the next level, n/4 digests. Summing that up you get n + n/2 + n/4 + ... = 2*n

A constant factor of two for something as fast as hashing is basically irrelevant.

February 23, 2014 at 5:31 PM

Ken Shirriff said...

Thanks for the detailed comments, Peter. The funny thing is I realized when looking at mining pool computations that the Merkle hash was in fact useful. I thought I had removed the part about it being pointless, but I guess not :-)

February 23, 2014 at 7:55 PM

Anonymous said...

A typo?

>make a mining request to the GHash.IO mining pool ...
sock.connect(("stratum.btcguild.com",

February 23, 2014 at 11:04 PM

Ken Shirriff said...

Thanks Anonymous - I've fixed that.

February 24, 2014 at 10:25 PM

Peter Todd said...

No problem! There's a lot of stuff in Bitcoin that at first glance doesn't look useful, and then only later do you realize why it's so important.

On the other hand, there's also a lot of stuff that makes you wonder WTF was Satoshi smoking... but that's hindsight for you.

February 25, 2014 at 2:05 AM

KovaaK said...

> If the hash meets the pool difficulty, you get a share. If the hash also meets the much, much harder Bitcoin difficulty, the block has been successfully mined. In this case the pool submits the block to the Bitcoin network and everyone with shares gets paid accordingly.

I'm still quite new to the mechanics of Bitcoins and pools, but hypothetically, could a malicious pool participant be programmed to send hashes that met the pool difficulty to the pool, but keep the Bitcoin difficulty-level hashes for himself and submit them directly to the Bitcoin network? Are there any checks that could be implemented on the pool software to make sure malicious clients aren't stealing successes?

February 25, 2014 at 9:53 AM

Stephen Cagle said...

Thanks for this article, especially footnote number 7. In reading about this mining pool stuff, I had always wondered why people didn't just cheat and not submit the winning share to their pool if they happened to find it. Thanks to your article I finally (duh) figured it out.

February 25, 2014 at 12:33 PM

Unknown said...

Ken, this is great stuff, I find myself already looking forward to your next post, and cutting and pasting the code to play with it. One question: your python code for the merkle root ignores the last transaction if it is odd...correct? I was under the impression that the odd one out was hashed with itself and that hash was then included as a branch. Thanks again. James

February 28, 2014 at 6:04 AM

Unknown said...

Ahhh. As soon as I posted I saw the line of code that does it. Silly me. So yes you do hash the odd transaction at the end to itself. Thanks again for a great post, with code!!

February 28, 2014 at 6:20 AM

Anonymous said...

I'm a little confused if the pool informs the clients of all of the transactions it wants to include in a block (if one is found.

From the wiki (https://en.bitcoin.it/wiki/Block_hashing_algorithm)
You hash the entire header
The header includes the merkle root
The merkle root is a hash of all of the transactions including coinbase

Wouldn't that mean the pool has to constantly update miners about which transactions need to be included in their merkle root calculation?

March 5, 2014 at 5:22 PM

Anonymous said...

Still confused. The transaction, with each scriptPubKey copied into scriptSig, is double SHA256 hashed... is that the hash that becomes the txid? Is that the hash that is used to sign each input?

I'm trying to get my head around inputs that are signed by different owners (and at different times). Is the above double hash used to sign each input (no matter who the owner), or are different hashes signed?

Thx

March 25, 2014 at 9:53 AM

Unknown said...

I rewrote your sample Python hashing script for merkle trees in PHP if anyone is interested.

http://pastebin.com/JbCVyRiu

April 16, 2014 at 12:44 PM

Unknown said...

Hello Ken,
excellent deep article.

I wonder how you get the figure of 11 million years on average ?

"My Python program does about 42,000 hashes per second, which is a million times slower than the hardware used by real miners. My program would take about 11 million years on average to mine a block from scratch."

April 25, 2014 at 8:37 AM

Michael said...

Very detailed article for those wanting to know how mining actually works. Will point readers here when they ask how mining results in BTC. Very good.

August 13, 2014 at 12:10 PM

عبدالله said...

I ask for your permission to copy the article and translate it into another language .

September 28, 2014 at 11:39 AM

Anonymous said...

Ken, how is it your articles on Bitcoin are always the only ones that provide answers to the questions I have in my head? Uncanny. Thank you.

October 17, 2014 at 12:42 AM

Anonymous said...

The [2] need more precision. The length mean « The 'length' of the entire block chain refers to the chain with the most combined difficulty, not the one with the most blocks »

As stated here: https://en.bitcoin.it/wiki/Blocks

December 1, 2014 at 9:58 AM

__ said...

Great post. Helped me understand the details. I trained a machine learning classifier to answer the question "Is the nonce greater than k", similar but not the same as your Notes and references point 1. Posted here. Thanks again for a great post.

March 19, 2015 at 8:37 PM

Anonymous said...

The statement that the Merkle tree idea is patented is rather pointless, because the patent expired more than 10 years ago (it was issued in 1982 and patents last for 20 years).

October 31, 2015 at 10:48 AM

Ruslan said...

Ken, can you please explain more about "Creating a block for a pool",
coinb1 and extranonce1 is ok, but from where you get - "000017e4" and continue "46522cfa...." of "extranonce2, and coinb2" are difference from first block?

I try to write it on VB.net

Thank you ~

November 1, 2015 at 6:39 AM

Gilberto said...

Good night, excuse me my poor english, My name is **Gilberto**, I'm Mexican and I'm Master student, and I'm trying to investigate a new form to do mining of Bitcoins, but I have a lot of questions about How is the manual procedure to do mining of Bitcoins?, How I use the algorithms SHA-256, RIPEMD160 and Base58 in Bitcoins?, You have libraries about these? I want to do a embedded system in a 7.E64G401 Ephifany 64-core card and use computing paralell and I not have idea How begin?, **please help me.**

December 12, 2015 at 8:06 PM

Anonymous said...

Gilberto, first lean how to hash SHA1 manually from this guy and then go from there.
https://www.youtube.com/watch?v=5q8q4PhN0cw

March 22, 2016 at 8:17 PM

benwest said...

Hi ken;
Question1:
do the miners start all from the same nonce (nonce=1 for example) and every miner increase its nonce depending of the speed of the hardware where and

software he use for mining ?
otherwise could someone enter manually the beginning nonce0 and by luck he find the right nonce to get a valid hash for a new block ?

Question2:
a part the coinbase rewarding transaction; do the miners receive the same transactions in the block to mine ?
Question3:
we suppose 2 miners find a valid blocks block1 and block2 at the same time.
do block1 and block2 contain the same number of transactions?

Thanks.

January 8, 2017 at 11:24 AM

Ken Shirriff said...

benwest: Q1: Miners normally all start with the same nonce value and then count through as fast as they can. But other parts of the block will be different, so they're trying different blocks.

Q2: In a mining pool, miners probably get the same transactions but a different extranonce1, to avoid different miners duplicating work. But different miners could get different transactions, if the pool operator updates the block as time goes on.

Q3: Two blocks mined at the same time could have the same number of transactions, or could be totally different. It's possible they have no transactions in common. Or they could have identical transactions.

January 9, 2017 at 2:47 PM

benwest said...

Q4: does a single miner, that he is not a part of a pool, has any controle over the block he work on ?
for example: work only the block containing only the coinbase rewarding transaction and another transaction?

Q5: could someone who has a very slow pc (old pc) find a block before other miners by luck (like a lotto game) ?

Thanks.

January 10, 2017 at 7:36 AM

Ken Shirriff said...

benwest: Q4: Yes, a single miner can pick the transactions that go into the block the miner is working on. Normally the miner would pick a bunch of transactions (to get more fees) but they don't need to. Q5: Yes, it's like a lotto. Someone with a slow PC could get lucky and mine a block (or even someone mining on an old punchcard computer. It's just very unlikely since fast hardware gives you many more chances to "win"

January 10, 2017 at 11:31 PM

Anonymous said...

Great article and given that you're still answering questions 3 years later I thought I'd repeat an unanswered question from earlier that piqued my curiosity.

"... could a malicious pool participant be programmed to send hashes that met the pool difficulty to the pool, but keep the Bitcoin difficulty-level hashes for himself and submit them directly to the Bitcoin network? Are there any checks that could be implemented on the pool software to make sure malicious clients aren't stealing successes?"

January 12, 2017 at 6:31 AM

Ken Shirriff said...

Anonymous: can a pool participant steal the successful hashes? That's a good question, but no. The coinbase transaction contains the pool owner's scriptPubKey, so the pool owner is the only one who can access the reward. If the miner changes the scriptPubKey, the hash is no longer valid. There's a theoretical attack where the miner throws away a fully-successful hash so nobody collects.

Then the miner gets paid for the partially successful hashes but the pool owner doesn't get the reward payout. This is known as the withholding attack. Since it doesn't benefit the miner, it's not too useful as an attack.

January 12, 2017 at 3:41 PM

benwest said...

a mining-pool is seen by the network as a single node (as a solo miner).
the nonce run from 1 to n-1 to find the good bock's hash.
how to avoid miners not doing the same work by testing the same values of the nonce ?
does the pool give every miner a part of the set containing all possible nonce value ?
thank you.

January 15, 2017 at 12:45 PM

Ken Shirriff said...

benwest: I believe the mining pool gives each miner different extranonce1 values. A miner can then run through all the nonce and extranonce2 values without duplicating work. Because of the different values for extranonce1, each miner can work with the same transactions but will still be generating unique blocks.

January 16, 2017 at 11:17 PM

benwest said...

what I believe from the following comment I found in another blog is:
we can change manually the nonce.
for example begin with nonce = 2677 or even try to predict the right nonce that would give the valid block hash.

This comment is extract from another blog:

There is no precise nonce finding protocol.

The miner can arbitrarily choose a nonce c to perform the hashing operation.

Mining is a mathematical game where the goal is to make the result of the hash function smaller than a given number (this is what "a result starting with x zeros" is looking for). The number is directly based on the current Bitcoin network difficulty and changes every two weeks to keep average block finding time at 600 seconds.

Most nonce generators just increment by 1 but the key is where they start. If you are solo mining, you can pick a random number. If you are mining with multiple devices or you are a pool administrator, you have to divide the work to avoid calculating the same hash twice (make sure they never use the same c).

January 19, 2017 at 2:04 PM

Andreza said...

Hi! I'm a big fan of your articles. Can I translate some of them to my blog in Portuguese? Thank you in advance.

January 30, 2017 at 7:14 AM

Ken Shirriff said...

Andreza: go ahead and translate. Thanks for asking. If you send me a link to your translation, I can add it to this page.

January 30, 2017 at 10:21 AM

Jonathan Annett said...

about this:

http://static.righto.com/images/bitcoin/coinbase_txn2-s400.png
(it's a table with data from your page)

are we sure about the contents?

for example the last line has 9 zeros.

i'm trying to code a parser to break apart statum messages and this does not seem to line up to the earlier data.

May 9, 2017 at 6:41 AM

Ken Shirriff said...

Jonathan: the block in that figure is different from the block discussed earlier, so you can't expect them to match up. (The text above the figure should mention this.)

May 9, 2017 at 1:36 PM

Jonathan Annett said...

it does mention that.

what it does not mention is how a 32 bit lock time is expressed using 9 hex digits. what black magic is this?

all good, i managed to glean enough to parse stratum messages in node js, dumping the relevant parts of the coinbase snippets to the console log - verifying that a solo pool does in fact split the coinbase transaction 2 ways.

the trick was using a jsonrpc call to bitcoind (happpens to be running on same server) to decode the second script segment revealing the transaction containing my wallet address. i didn't feel like deep diving into the pecularities of the internal script format when i don't really need to get that concerned with that side of the proticol.

i've coded a pass through proxy that dumps each stratum message to a log, while saving pertinent parts of the messages to memory.

i am now going throufh the academic exercise of hashing a block alongside the miner (fairly pointless, but i want to understand the protocol completely, so i'm inventing a separete extranonce2 of the appropriate length, collating the coinbase, hashing it,

building a merkle root from the coinbase tx hash and supplied merkle branch array, forming a block header, hashing it and .... well i suppose if the proxy gets lucky it could go ahead and submit it on behalf of the miner!

i'm pretty sure once i've completed that, your captcha ("i am not a robot") feature will reject my post, as tbh, you need to think like one to code this stuff.

May 18, 2017 at 3:27 PM

Ken Shirriff said...

Jonathan, you're right - there's an extra digit for the lock time in the diagram. You get points for studying the diagram more carefully than anyone else ;-) Your post got through fine, by the way (along with a lot of spam I'm constantly removing).

May 18, 2017 at 7:04 PM

Unknown said...

Hi

first of all thanks for this really fantastic series,this really helpful.

but i really wondering about solo mining ,i read an ebook(mastering bitcoin) i was good but i don't understand completely about solo mining ,how we select transaction and make Merkle Root and even timespan (because for example when check in blockchain.info and see last block all transaction is selected and merkle rook is made so we have to solve this hash or we can select transaction that we want and then solve the hash?)
and after that how we can submit the hush to blockchain ?

could you please write another article about solo mining and such this problem?

thanks.

May 25, 2017 at 3:14 AM

Ken Shirriff said...

Pooriya: solo mining is pretty much the same except you have the choice of what goes in the block. You pick which valid transactions you want to put in the block. You pick a roughly accurate value for the timestamp. The Merkle root is formed by hashing pairs of transactions and then hashing pairs of hashes until you have a single value (see footnote 4 for details). Then you try to hash the resulting block with different nonces, hoping to find a successful block

If you succeed in mining, you send the block to the Bitcoin network. Since the network is peer-to-peer, you send your successfully mined block to other computers (peers) in the Bitcoin network, who send it to other computers, until everyone has received it in a few seconds. Peers are always sharing blocks, which is how they get passed around the network, and there is nothing special about you sharing a block that you just mined versus a block that you received from someone else. Miners will then start using your block as the previous block for their mining, which is how your mined block becomes part of the blockchain.

May 25, 2017 at 7:39 AM

Unknown said...

Hi

thanks for reply

your explanation was helpful but there is 2 point that steel unclear for me.

for example at this time the block no #468323 is found (by F2Pool) and we want to find next block so we have to make block header and search

1) the most unclear part for me is how making coinbase (for example for block no #468323 is 036a6f397243382eae70626e631469c74bdf40d5e3db6539ca41fe928cc857aa and Of course it is related to F2Pool) how should i make mine?

2) what value i have to use for Difficulty (for block no #468323 is 402774100 but it is for Previous, is it specified for next block that we want to find (#468324)?

May 26, 2017 at 10:13 PM

..i.. said...

No, because the address to grant the reward is the pools address. If you put your own address in, then you are not getting pools rewards. You could throw away the block chain difficulty reward as a denial of money but that does nobody any good.

July 10, 2017 at 3:32 PM

Dilli Raj Maharjan said...

Hi Ken,
Thank you for the detailed post on Bitcoin mining. I have setup the BTC Full Node. I have six cores of CPU and around 16GB. I have installed cpuminer to test the mining process. Can you share some information on how to setup mining process just to mine own transaction. I want to speed up my low fee transaction with my own mining server. Can we do it with CPU Mining or we need GPU,FRGA Mining and ASIC Mining.

August 1, 2017 at 11:14 PM

Unknown said...

This comment has been removed by the author.

August 8, 2017 at 5:13 AM

Unknown said...

Please, can you send me the already compiled program for bulkhead nonce for Windows? The one that you have in the example. Thank you

August 8, 2017 at 5:24 AM

NGR TV said...

how to create mining software for
Alt coins . which are on coinmarket cap . and source code available on github.

September 1, 2017 at 7:35 AM

Anonymous said...

I've not yet seen any discussion of how we can be certain that any/every block can be mined. Is it not possible one day, to encounter a block that simply never meets the criteria, that can never meet the criteria?

September 24, 2017 at 6:51 AM

Ken Shirriff said...

Anonymous: you ask if we know every block can be mined. Given current difficulty it is very likely that a specific block can't be mined - there's no nonce that works. In that case, miners simply change the block slightly (e.g. change the time, use different transactions) and try again. Eventually some block will work. In other words, there's not one specific block that has to be the next block, but zillions of possibilities and miners just need to find one that works.

September 24, 2017 at 9:45 AM

benwest said...

who is the authority who protect the bitcoin core code or protocol or software against attacks or hacks?

October 3, 2017 at 9:41 AM

Stuart Cracraft said...

benwest --> https://github.com/bitcoin/bitcoin

October 4, 2017 at 8:07 PM

Anonymous said...

Hi

Great post. Let's say we have 1000 computers and would like to mine. But these

computers are slow. That's why they can not mine strong diffs. So is it possible to divide the job for them? and give them a less diff?

Here is the example:

A standard pc can do 42.000 hashes /s. But my pcs can do only 10.000 hashes. Because i already have 1.000 pcs which means 10.000.000 hashes /s. is it possible to merge their powers?

Andy

December 19, 2017 at 4:44 PM

Ken Shirriff said...

Andy: yes, computers can mine in parallel. Finding the hash is a matter of trying lots and lots of hashes until you find a good one, so it's easy to have many machines working in parallel. The only trick is to make sure all the machines are doing different hashes and not duplicating work, which is straightforward. This is basically what mining pools do; they also spread the winnings across all the participants.

December 19, 2017 at 4:48 PM

Antonio said...

Hi Ken, I saw you are answering interesting questions about how BTC mining works, I have a couple of questions I have not found the proper answer: Q1: How can I try to solve a block (educational purposes) after getting the GBT information from my Full node fully synced. Q2: How do I parse that with python program to obtain the merkle root with my coinbase transaction and Q3: How do I antually submit work once nonce found.

Thanks in advance.

January 11, 2018 at 3:10 PM

James said...

Hi Ken - in regards to Anonymous' question above: "A standard pc can do 42.000 hashes /s. But my pcs can do only 10.000 hashes. Because i already have 1.000 pcs which means 10.000.000 hashes /s. is it possible to merge their powers?"

This is what a stratum proxy would do, correct? Split the nonces etc an issue proxy-local 'jobs' between the 1000 'slow' computers? I'd be interested in just how a proxy would 'split' the work, to ensure each of the 1000 computers wasn't duplicating work.

January 26, 2018 at 12:43 PM

---

paul said...

hey man, cool blog. I really appreciate the clarity concerning the algo. Excellent work.

Really interesting stuff about the alto and the ibm mainframes and stuff also. thank you.

my 3rd gen i5 laptop ran your python code at around 42 kH/s. Next i'll be trying to port it to perl.

good reading

January 27, 2018 at 8:23 PM

Unknown said...

some basic questions that seem to be unaddressed beyond anything than vague generalities within the hype.
what data is being mined ? what is the purpose of it ?
who is paying for the number-crunching results ?

February 17, 2018 at 7:43 PM

---

wheezil said...

I'm having a hard time grasping the "degrees of freedom" available to the miner. Clearly, the nonce comes first (32 bits) and the timestamp can be adjusted (but by how much?). Also there is some freedom in choosing which transactions to include (although it would seem you always want to include the most transactions possible so as to be the winner in case of a fork)? And finally, "the coinbase transaction can be modified". I *think* this refers to the "coinbase parameter" of the coinbase block, which in the case of the genesis block had the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". That being the case, the coinbase parameter itself would seem to offer unlimited degrees of freedom.

February 21, 2018 at 6:22 AM

tanglewood said...

Hi Ken; I am not sure if this blog is still active but in the paragraph 'How Mining Works' you say 'In more detail, to mine a block, you first collect the new transactions into a block. Then you hash the block to form a 256-bit block hash value. If the hash starts with enough zeros[3], the block has been successfully mined' You didn't mention the nonce, also, I thought that in addition to the leading zeroes the hash has to be lower than the 'target hash'. If that's true, can you explain what the target hash is? Is it possible to reverse engineer and wind up with a valid nonce?

April 29, 2018 at 5:54 PM

Ken Shirriff said...

wheezil: you seem to have a good grasp on the degrees of freedom, i.e. how much miners can change the block. Also note that the order of transactions gives you N factorial choices. As far as modifying the coinbase transaction, typically this is a combination of values from the mining pool (to ensure miners aren't doing the same mining) and "extranonce" values from the miner. Overall, there's enough degrees of freedom that mining isn't going to get "stuck".

tanglewood: the transactions are collected into a block, along a few other fields including the nonce. The diagram and text describe these fields in more detail. And yes, the hash has to be lower than the target hash; "enough zeros" is a slight

simplification. (The "last zero" could be a small digit, small enough that the hash is still under the target value.) The target hash is a number generated by an algorithm every 2016 blocks to ensure that Bitcoin blocks are generated on the average every 10 minutes (details). As far as reverse engineering, the idea of SHA-256 is there's no way to go backwards to find a valid nonce; you need to use brute force. (It's possible that a sufficiently smart mathematician could break SHA-256 but based on the state of cryptography research this seems very unlikely.)

April 29, 2018 at 7:21 PM

Anonymous said...

This is a most elucidating blog.
It helped me so much when implementing a C++ miner from scratch.
From the general discussion down to the byte layout.

May 15, 2018 at 1:39 PM

Anonymous said...

Fantastic work a great read well done!

May 21, 2018 at 3:24 PM

Anonymous said...

I am new to cryptocurrency. I have a doubt: when you implement these cryptocurrency algorithms (in python or some other programming languages), how do you tell machine to use GPU and not CPU in order to do math calculations? or it is always used GPU by default automatically and not need to explicitly tell it? In that case, how do you ensure that GPU is being used instead of CPU?

August 25, 2018 at 4:30 AM

Anonymous said...

Amazing! Thanks for great stuff :)

October 6, 2018 at 7:49 AM

Anonymous said...

Thanks to Mr josehp who help me become a millionaire by connecting my wallet to his mining stream i am so happy today i can get what i want i just got a new house,He make me earn 0.6 BTC daily, you can earn BTC also by email him and request for his help and he will also connect you to his stream right Away 24hoursbitcoinminer@gmail.com

November 21, 2018 at 8:03 AM

Marcelo Maico said...

Hi Ken. Wonderful article, congratulation!

I have a question regarding competition between the pools. All pool will try to put the next node, but the pool with less power will have to start the job because the previous block was change. That is, the computer with greater computational power will always win, but this is not what happens in practice. How does it work?

January 18, 2019 at 3:32 AM

Unknown said...

Here we are 7+ years after this article was posted and it is still relevant and more informative than any single article I've seen. Thanks Ken for such a great contribution!

I was on the edge of my seat between the part explaining the nonce and the structure of the pool header. Today's miners can rip through the 2^32 hash variations provided by nonce variations thousands of times in one second so I was anxious to see how the throughput problem that creates got solved.

One thing I'm curious about: what happens to transactions in an orphaned block that were not also in the competing block that made it into an orphan? Will they go back into the queue and get picked up in a subsequent block or do they just become invalid and have to be reissued?

Rodney

March 27, 2021 at 7:58 PM

Post a Comment

Newer Post Home Older Post