Zach von Naumann Follow
Crypto 24/7
Oct 19 · 13 min read

The Evolution of the Cryptographic Hash Function in Blockchains

You've heard people talking about blockchains and cryptocurrencies for some time now, and you understand that blockchains are distributed ledgers, but what exactly goes into a block? Blocks make blockchains, and transactions go into blocks, but what is the glue that keeps it all together? Well it turns out blocks are linked to one another through a process called *hashing*.

A *hash function* takes any input (i.e. numbers, words, etc.), and through the use of an algorithm, produces an output of a specific length. The process of applying a hash function to some data is called hashing. A proper cryptographic hash function has two main qualities:

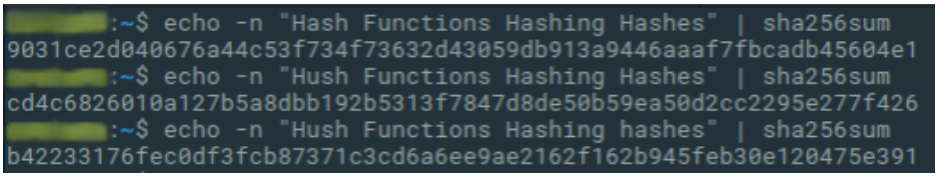
1. Pre-image resistance: The hash function works in only one direction, meaning you cannot deduce the input from the output.

Consequently, for two sets of inputs, even if the inputs only differ by the smallest detail, the outputs should be wildly different and not resemble one another.

2. Collision resistance: When a hash function produces the same output for two different inputs, this is called a *collision*. It is imperative that collisions are avoided in order to guarantee data integrity. If two pieces of data produce the same hash, then one can be interchanged with the other, leading to a breakdown of continuity.

Bitcoin uses hash functions for creating addresses out of public keys, and to add blocks of transactions into the blockchain. Public keys are generated through another set of mathematics involving random number generation and Elliptic Curve encryption (a detailed explanation of this process is outside of the scope of this article). For our purposes, all we need to know is that after we generate our public key, a series of hash functions are applied to it, which results in the creation of a unique address. The underlying hash function used in bitcoin is *SHA256*. SHA256 is a function that results in a 256-bit, or 64 character, output. In other words, any data set inputted into the SHA256 function will result in a 64 hexadecimal string of letters and numbers. In practice, Bitcoin actually uses a double SHA265 hash function in order to minimize the chances of a collision.

Time to watch the SHA256 algorithm work its magic in real time by entering a simple command into our Linux terminal:



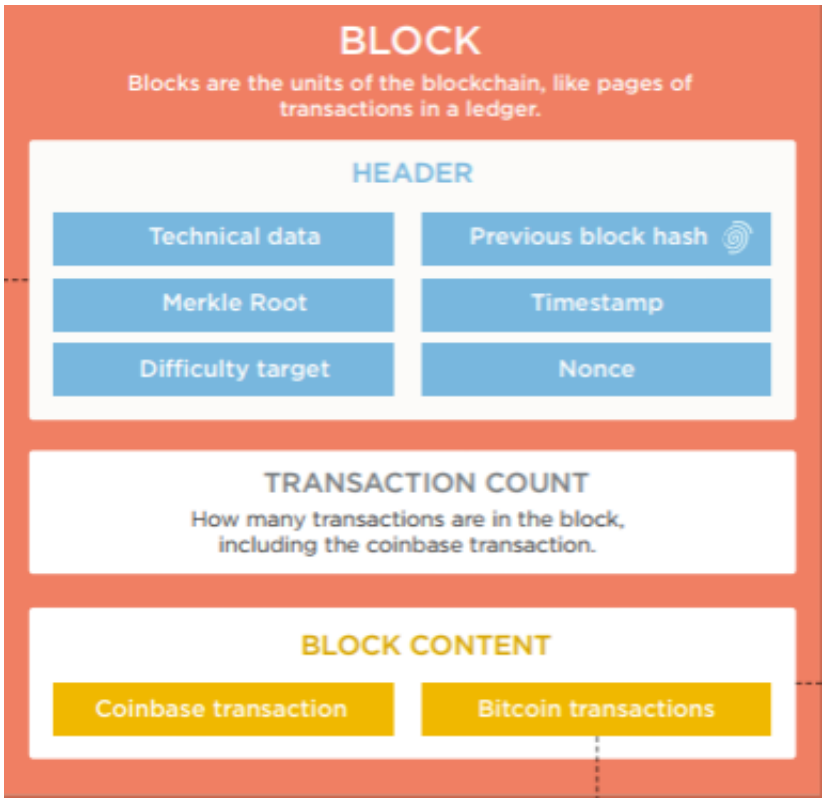
As you can see, altering one character of the input, by changing the first ‘a’ into ‘u,’ or not capitalizing an ‘h,’ resulted in entirely different outputs.

Now how does this relate to adding blocks to a blockchain? Well, if we recap what goes into a block, then we should have a better idea of the

role hashing plays in maintaining a blockchain. A bitcoin block contains the header, transaction count, and block content. The block content section contains all of the data regarding the individual transactions, and the transaction count is the number of transactions in the block. The block header is what links one block to another and contains the following elements:

1. The block version number (i.e. the software version)
2. The 256-bit hash of the previous block header
3. The 256-bit hash derived from all of the transactions in the block (this involves the Merkle Root, something which will be discussed later)
4. A timestamp
5. The current difficulty target (a 256-bit number, usually starting with a string of leading zeros). The target is adjusted based on the computational power of the network in order to guarantee a set number of blocks per timeframe. The desired rate for bitcoin is one block every 10 minutes, and the difficulty is set to adjust every 2016 blocks, or approximately two weeks.
6. A *Nonce (*a random 32-bit number). Miners change the value of the nonce in order to find a hash that is below the currently difficulty target.

Here are a couple charts that help illustrate the contents of a block and the fields in the block header:



Source

Block header:

Field	Purpose	Updated when...	Size (Bytes)
Version	Block version number	You upgrade the software and it specifies a new version	4
hashPrevBlock	256-bit hash of the previous block header	A new block comes in	32
hashMerkleRoot	256-bit hash based on all of the transactions in the block	A transaction is accepted	32
Time	Current timestamp as seconds since 1970-01-01T00:00 UTC	Every few seconds	4
Bits	Current target in compact format	The difficulty is adjusted	4
Nonce	32-bit number (starts at 0)	A hash is tried (increments)	4

Source

After looking at the components, we begin to understand that bitcoin mining is just a competition for finding a value for SHA256(SHA256(Block_Header)), which is lower than the set difficulty for that period. Recall that the nonce is a random 32-bit number within the string of values in the block header, which miners are allowed to change in order to find a hash that satisfies the difficulty requirement. Therefore, when mining bitcoin, all you are doing is altering the nonce value within the block header, over and over, until you find an

appropriate hash. Once you find a hash that 'fits,' the block is broadcast, the network performs verification, and it is added to the blockchain.The use of hashing ensures no piece of data can be altered on the blockchain without everyone being able to spot the inconsistency. Hashing is the reason people can label bitcoin an 'immutable' ledger.

Today, we have over 1000 different cryptocurrencies, and almost as many hash functions. Most have arisen claiming to address some of the perceived shortcomings of double SHA256. Assessing these new algorithms shouldn't be a problem now that have a working understanding of the role hash functions play in a blockchain. In the following section, I provide a short summary of three of the most popular, newer algorithms, Scrypt, X11, and Cryptonight. Afterwards, I will cover an invention that has the capacity to completely revolutionize cryptocurrency hash functions by 2018.

Scrypt:

Scrypt is a key derivation function (KDF) developed by Colin Percival, in 2009, as a response to the spread of specialized ASICs (Application Specific Integrated Circuits). The hash function in bitcoin is not especially resource intensive; therefore, specialized units to perform mining started to takeover the mining market aroun 2013. This resulted in the exponential growth of network hashing power, which led to a proportional rise in difficulty. Ultimately, to profitably mine bitcoin, massive mining farms full of specialized hardware became necessary. The large amounts of physical space, electricity, and hardware, needed to effectively mine bitcoin alarmed many members of the crypto community. If mining is performed only by those with access to specialized resources, aren't we predisposing our 'decentralized' ledger to centralization? The solution provided by the Scrypt function and its derivatives, relies on being memory intensive in order to raise the costs needed to monopolize hashing power.

Here is an excerpt from the Internet Engineering Task Force (IETF) paper on the topic, which shows the inputs of the Scrypt Algorithm:

The scrypt Algorithm

The PBKDF2-HMAC-SHA-256 function used below denotes the PBKDF2 algorithm [RFC2898] used with HMAC-SHA-256 [RFC6234] as the Pseudorandom Function (PRF). The HMAC-SHA-256 function generates 32-octet outputs.

Algorithm scrypt

Input:

P	Passphrase, an octet string.
S	Salt, an octet string.
N	CPU/Memory cost parameter, must be larger than 1, a power of 2, and less than 2^(128 * r / 8).
r	Block size parameter.
p	Parallelization parameter, a positive integer less than or equal to ((2^32-1) * hLen) / MFLen where hLen is 32 and MFlen is 128 * r.
dkLen	Intended output length in octets of the derived key; a positive integer less than or equal to (2^32 - 1) * hLen where hLen is 32.

Output:

DK	Derived key, of length dkLen octets.

This algorithm is only a piece of the overall computations conducted in the Scrypt function; however, it plays a foundational role and is useful to help visualize some underlying facts about the system. The IETF paper states, “Users of scrypt can tune the parameters N, r, and p according to the amount of memory and computing power available, the latency-bandwidth product of the memory subsystem, and the amount of parallelism desired.” Furthermore, you can also set the hash size of the output to a smaller number, which allows for quicker block times. The adaptability of Scrypt, coupled with adjustable memory intensiveness, presents a viable structure for an ASIC-resistant function.

All of the different variables of the Scrypt algorithm are designed to prevent the kind of centralization and ASIC development we have seen with bitcoin. Unfortunately, making the function extremely memory intensive is not efficient when it comes to the verification of computed proofs in blockchain systems. Therefore, cryptocurrencies like Litecoin, have simplified their Scrypt implementations to maintain a lower rate of memory usage. Due to this simplification, Scrypt mining ASICs entered the picture, and now we are back to where we started with centralization and hash power monopolization.

X11:

In 2014, we find X11, the invention of the creator of Dash, Evan Duffield. X11 uses eleven different hashing functions, chained together, to calculate the block header. The eleven algorithms are blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, and echo. One advantage of the system, is its protection against Single Point Failures, or a scenario where an algorithm is cracked. In the event of an attack, all eleven of the hash functions would need to be compromised, making X11 extremely secure. X11 is also more power efficient, and allows for quieter and cooler mining operations than SHA256. The increased memory requirements resulting from the use eleven different hashing algorithms, temporarily prevented the use of ASICs. Eventually, of course, ASIC manufacturers caught wind of the profitability of the project and developed miners for this algorithm as well.

Currently there are several copies of X11, from X13 to X17, which are derived by adding even more algorithms to the chain of hashing functions in X11. Many will point out that this is overkill and does not add any real functional value to a project. Ultimately, X11 does little to address the main underlying concerns with currently available hash functions.

Cryptonight:

Cryptonight was originally developed in 2013, as the hash function of Cryptonote, and now serves the same role in the popular cryptocurrency Monero. Cryptonight is defined as, “a memory-hard hash function. It is designed to be inefficiently computable on GPU, FPGA and ASIC architectures. The CryptoNight algorithm’s first step is initializing large scratchpad with pseudo-random data. The next step is numerous read/write
operations at pseudo-random addresses contained in the scratchpad. The final step is hashing the entire scratchpad to produce the resulting value.” In simple terms, this means Cryptonight was designed with the objective of being inefficient for ASICs and GPUs. This is achieved by requiring at least 2MB of readily accessible memory (the ‘scratchpad’) for performing the necessary hashing operations. In

computational terms, 2MB of memory per instance translates into a limited number of possible parallelized hashing attempts. Modern ASICs excel at mining SHA256 because within a relatively small package, you can run multiple iterations of hashing attempts in parallel using a collection of specialized processors. Once you are required to possess 2MB of memory per iteration, the use of specialized processors and even GPUs no longer afford the same advantages over the traditional CPUs used in your average home computer.

Cryptonight uses several hashing functions, including Keccak, also know as SHA3, the chosen replacement to SHA2, along with AES encryption at several stages. Both the hash functions and encryption standard are open source developed, which reflects the obsession of the Cryptonight team with preserving privacy and the highest levels of security. To date, there are no ASICs that can mine the Cryptonight protocol and the majority of mining takes place using traditional CPUs. Some concerns with this algorithm involve the fact that a typical Monero tx is about 13 kb, which is approx 25 times the size of a bitcoin tx. Another issue that has arisen recently is the use of botnets, which due to Cryptonight's optimization for CPU mining, have been spread using malware into unsuspecting computers and then used to mine Monero. This recent discovery unveils the risk of a single individual in control of a botnet gaining a large proportion of hash power.

Despite these concerns with the Cryptonight algorithm, I believe it is the closest we have gotten to a hash function that works to counter mining centralization. However, only time will tell if it can prove sustainable and scalable.

MTP

Now we arrive at the end of 2017, in our journey through time, exploring the different hash functions that have evolved in the world of blockchain technology since 2008. Here we find an interesting concept called *Merkle Tree Proof* mining algorithms (MTP). Alex Biryukov and Dmitry Khovratovich published a paper entitled, *Egalitarian Computing*, in June, 2016, which proposes a solution to "…several contexts where

an adversary has an upper hand over the defender by using special hardware in an attack." They continue to suggest, "… memory-hard computing as a generic paradigm, where every task is amalgamated with a certain procedure requiring intensive access to RAM both in terms of size and (very importantly) bandwidth, so that transferring the computation to GPU, FPGA, and even ASIC brings little or no cost reduction. Cryptographic schemes that run in this framework become egalitarian in the sense that both users and attackers are equal in the price-performance ratio conditions." At this point, this all looks pretty similar to every other serious proposition to tackle mining inequalities. We have seen that the only realistic way to stop the spread of ASIC or GPU dominance, is to increase the amount of memory our hash functions require. Let's see what Alex and Dmitry plan on doing differently from Scrypt and Cryptonight.

In technical terms, the authors, "…suggest a cryptocurrency proof-of-work based on Argon2d with 4 parallel lanes. We aim to make this PoW unattractive for botnets, so we suggest using 2 GB of RAM, which is very noticeable (and thus would likely alarm the user)…On our 1.8 GHz machine a single call to 2-GB Argon2d runs in 0.5 seconds, but the Merkle tree computation is more expensive, as we have to hash 2 GB of data splitted into 1 KB blocks. We suggest using Blake2b for H, as it is already used in Argon2d, but restrict to 128-bit output, so that the total running time is about 3 seconds." Now that we have our blueprint, let's dissect the terms and see how this proposition solves some of our fundamental problems.

1. Argon2: Winner of the 2015 Password Hashing Competition, designed to "….maximize the cost of password cracking on ASICs…" The function is, "…optimized for the x86 architecture and exploits the cache and memory organization of the recent Intel and AMD processors." The Argon2d version is optimized for speed and use in cryptocurrencies.

2. The use of 2GB of RAM makes the infection of computers with botnet malware easy to detect, and amplifies ASIC resistance.

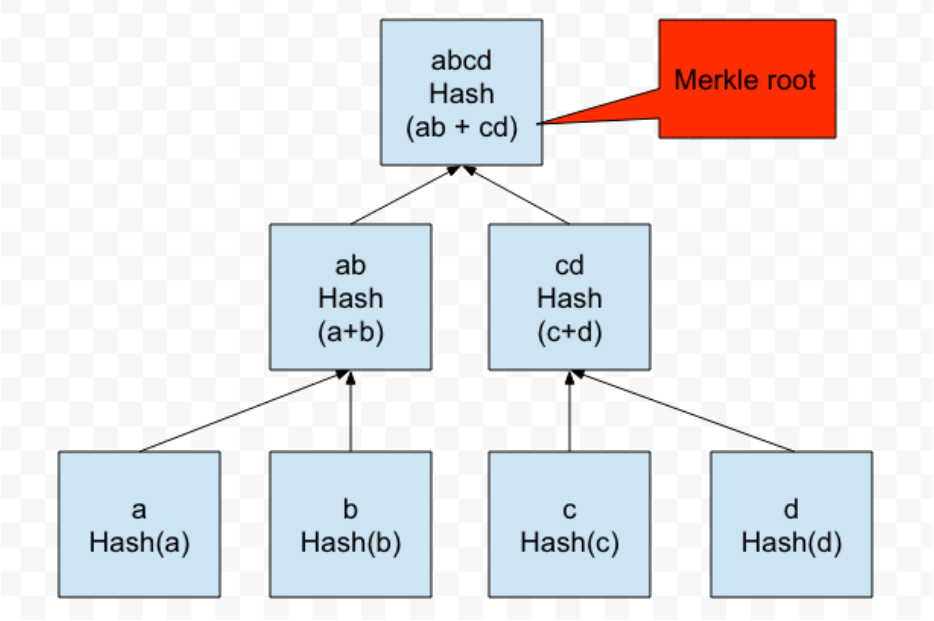
3. Blake2b: A hash function faster than MD5, SHA-1, SHA-2, and SHA-3, yet is at least as secure as the latest standard SHA-3.

4. Merkle Tree Computation: Most blockchains use a Merkle Tree to store information on all of the transactions within a block without having to include every transaction hash in the block header.

From this quick definition of terms we see that Argon2b would be the primary hash function in MTP, due to its built in resistance to ASICs and GPUs, and Blake2b would be used in the creation of Merkle Trees, due to its speed at hashing large amounts of data. The use of 2GB of RAM, serves to both complicate attempts at using ASICs and to make any implementation of botnets wildly apparent to the user. At this point, theoretically, we have surpassed the qualities of Cryptonight by embedding a method for preventing botnets; however, how will MTP address the shortcomings, as shown with Scrypt, of using large amounts of RAM when it comes to proof verification? In order to explain their creative solution to this problem, we must delve a little deeper into the concept of Merkle Trees and Proofs.

In the words of Ethereum creator, Vitalik Buterin, "A Merkle tree…is a way of hashing a large number of "chunks" of data together which relies on splitting the chunks into buckets, where each bucket contains only a few chunks, then taking the hash of each bucket and repeating the same process, continuing to do so until the total number of hashes remaining becomes only one: the root hash."

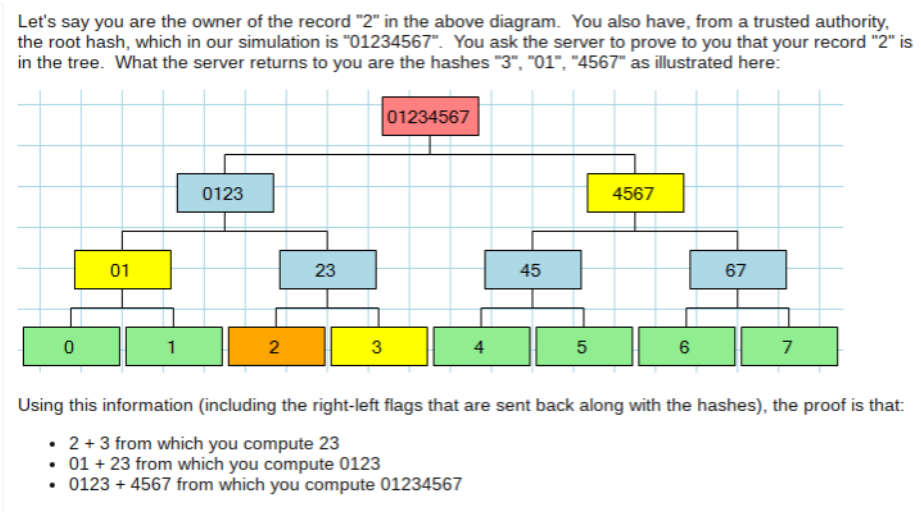
Here is an example of a Merkle Tree:



Source

On a blockchain, once you have a Merkle Tree, you can perform a Merkle Proof. A Merkle Proof allows you to, "…securely verify that a transaction has been accepted by the network (and get the number of confirmations) by downloading just the small block headers and Merkle tree—downloading the entire block chain is unnecessary." To verify a transaction, I don't need to know what every transaction in the blockchain looks like, I only need to know, "… our transaction and its sibling (if it has one) and calculate the hash of those two and verify that it matches the expected value. From that we can ask for the sibling branch of that and calculate the hash of that and verify it. And continue with this process, up the tree. Which only takes ten verifications for 868 transactions." In bitcoin, Merkle Proofs were designed to enable simplified payment verification for light clients.

Here is visualization of the Merkle Proof process:



Source

In the MTP mining algorithm, as the name suggests, *Merkle Tree Proofs* play an invaluable role in transaction verification. Within a traditional blockchain, once a block hash value is found that satisfies the difficulty requirement, it must be broadcast to the network and its contents validated by the other nodes. In the case of a memory intensive hash function, if every node attempted to verify every transaction in the newly mined block, the delay caused my the memory intensiveness of the hash function would open the door to numerous possible attack vectors. This is why, in the case of Scrypt, we see blockchain implementations limit the size of memory usage. MTP plans on solving this problem by no longer requiring every node to validate every transaction in a newly mined block. MTP only requires the validation of the Merkle Proof associated with the transaction; thereby, cutting down the validation time and preventing any associated attacks.

Currently, Alex Biryukov and Dmitry Khovratovich are working on a revised version of their paper, which addresses some issues found in their documentation through an audit campaign launched by Zcoin. Zcoin, founded in 2016, by Poramin Insom, aims to be the first cryptocurrency to implement the Argon2b-MTP mining algorithm. MTP has been running on their testnet, and is slated for mainnet release sometime this year or the beginning of next year. I am excited to see how MTP develops! This could finally be the solution to our hash function problems, and could usher in a new era of decentralization and egalitarian blockchain technology.

The short video below is from the Zcoin website and explains MTP with animations:

Stay tuned for updates on our products and more articles on the cutting edge technology in blockchains. We at Shokone believe promoting our products and providing educational material on blockchain tech, walk hand-in-hand on the road to crypto-fueled success!

If you enjoyed this article, or think someone else might, please share. If you want to talk, or debate my points, leave a comment below, or contact me on linkedin