Proof-of-Work and Cryptographic Hashing Function in Blockchain

Blockchain technology was developed to create a decentralised, distributed collection of verifiable transactions using a Peer-to-Peer (P2P) network. In this sense it does not describe a monetary transaction - the original application during conception by an individual or group known as Satoshi Nakamoto in 2008 (2) - but rather the transferring of data between parties. Essentially, “blockchain is a giant ledger”(1) of transaction history held by every ‘node’ or machine on the blockchain network. These nodes agree on the ‘true’ block to work from using a consensus mechanism and add incoming transactions to this block. Arguably, the most prolific use of blockchain technology is in the digital currency known as Bitcoin. Bitcoin was developed to solve the double spending problem that plagued previous digital currencies (2). Due to the intangible nature of digital currency, it was possible to spend the same ‘coin’ in multiple locations if each transaction was processed before the central network had deducted the transaction amount from the consumer's account. Bitcoin proposed the use of a proof-of-work (PoW) protocol that would decentralise the currency and protect it from double spending. It is possible to see the benefits of blockchains on modern day society. Blockchain has been described as “The God Protocol”(4) by leading academics and authors. The reason many argue this is an accurate portrayal of the technology, is due to its transparent nature in regards to truth whilst maintaining privacy for individuals. Tapscott et.al describes blockchain as “...a platform for everyone to know what is true - at least in regards to structured recorded information.”(4).

Cryptographic hash functions and digital signatures are the cornerstones of blockchain privacy and security. They are mathematical transformation algorithms with specific features. Firstly, they must be able to take an input of arbitrary size - the message you wish to hash. Secondly, they must output a hash, also known as a digest, of fixed size. In the case of the Secure Hash Algorithm 256 (SHA-256), this outputs a digest of 256 bits regardless of the input message size (7). Hash algorithms must also aim to be collision free and must hide any data - or metadata - that was intended to be hashed. One possible definition of a hashing algorithm is function that produces a fixed size unique identifier for a selection of data. For example, if a message is hashed using SHA-256, a 256 bit - apparently random number - must be generated that could only be generated by hashing this exact message with SHA-256. If a different message was to be hashed using SHA-256 and the same digest was output, then a collision has occurred. This is often represented as $x \neq y \land H(x) = H(y)$ where x and y are inputs and H is the hashing function. Collisions are possible however, because an extremely large variation of inputs must be mapped to a 256 bit number - meaning a seemingly infinite amount of inputs must be represented by 2^{256} possible outputs. However, the likelihood of finding such a collision should be very low. Narayanan argues that if any hashing algorithm is given 2^{130} different inputs then the probability of finding a single collision would be 99.8% (5). However, such a high number of inputs would need to be generated and hashed that the ability to do this is currently computationally impossible (5). To meet the condition stated earlier in which the information should be hidden - the hash output should not be explanatory of the input - hashing functions also take a random key as an input, which when concatenated with the message, returns a seemingly random output. The final outcome of this process is - again, using SHA-256 - a 256 bit message digest that represents the message. Others can then use SHA-256 on the

concatenated random key and the real message and obtain a hash. This hash should always be the same, given the same inputs - known as a deterministic function - meaning others can compare digests to verify that the message has not been changed. Blockchain uses cryptographic hash functions alongside public and private key generation to hash information and verify it has not been changed. To understand the overall use of digital signatures and cryptographic hashing functions in relation to blockchain, a basic understanding of pointers - used in languages like C - is assumed. Blockchain acts like a linked list contained in a binary tree, collectively known as a merkle tree (9) where each element of the list holds data and a pointer to the previous element of the list. These pointers hold the memory address of the previous element and hold the hash of that previous elements data. Therefore, no data that exists in the blockchain can be changed because the next elements hash pointer will no longer match the hash of that data. This allows data integrity and a 'tamper-evident log'(6). (Shown in in *appendix 1).* This allows blockchain to entrench privacy and security into its protocol. At this point in the process the transaction is sent to all the nodes on the network. In order to prevent fraudulent blocks from being added, consensus on the correct block must be found by all nodes.

Consensus is needed in the blockchain protocol to avoid double spending, an example is shown in *appendix 2.* In order to solve this problem, blockchains often use consensus protocols such as proof-of-work or proof-of-stake (8).

Proof-of-work is a protocol that asks a user to solve a challenge in order to complete or proceed with an action. The fundamental quality of a PoW protocol is that it must take a much greater amount of effort to solve the problem than it does to check the validity of the solution. PoW gives the challenger a 'challenge string' - represented by C and asks the user to find an appropriate proof - or nonce - such that when concatenated with C and then cryptographically hashed, the output has a number of leading zeros - decided by the individual implementation of the PoW protocol. This forces the user to 'brute force' or repeatedly try multiple proof strings. The number of leading zero's determines the average number of attempts that need to be tried before the challenger finds the proof string. PoW protocols have been used to deter email spam and Denial-of-Service (DoS) attacks (12). PoW forces the sender - in the case of email - or client - in the case of DoS attacks - to pay a micro cost to engage in the activity. By asking the user to come up with a nonce by concatenating and hashing the nonce with the challenge string, a hash is produced with *n* number of leading zeros. The user must expend CPU cycles trying to find the nonce, increasing the time and literal cost - in the form of electricity - to engage in the action. For legitimate users who wish to access a web resource a small number of times, or send an email, this cost is so minute that it should not act as a barrier. However, for those wishing to send thousands of emails or generate millions of requests to a web server, having to solve the PoW challenge for each request or email could be expensive.

As of August 2015, the average number of attempts that needed to be made to find the nonce in Bitcoins implementation of the PoW protocol, was 10^{50}(8). This is an astronomically high number for an individual to compute but with multiple nodes on the network attempting to find the nonce, it is possible. Once a node has found the nonce, this is announced to the whole network, who can verify the PoW quickly. To solve the problem stated in *appendix 2* where **X** transacted the same resource with **Y** and **X'**, **Y** should not assume the transaction is complete until the transaction between **Y** and **X** is sufficiently buried in the blockchain. Thus, the

transaction can be called ‘complete’ once the transaction between **Y** and **X** has been added to the blockchain and *n* number of other transactions in the form of blocks have been added to the block containing the transaction between **X** and **Y.** The reason for this comes in the form of an exponentially diminishing probability (2).

If two competing blocks with legitimate nonces are broadcast to the node network, both have a chance of being accepted as the next block and being built upon as shown in figure 2 where Block Y represents the block containing the legitimate transaction **X->Y** and Block X’ contains the double spending attempt **X->X’**.

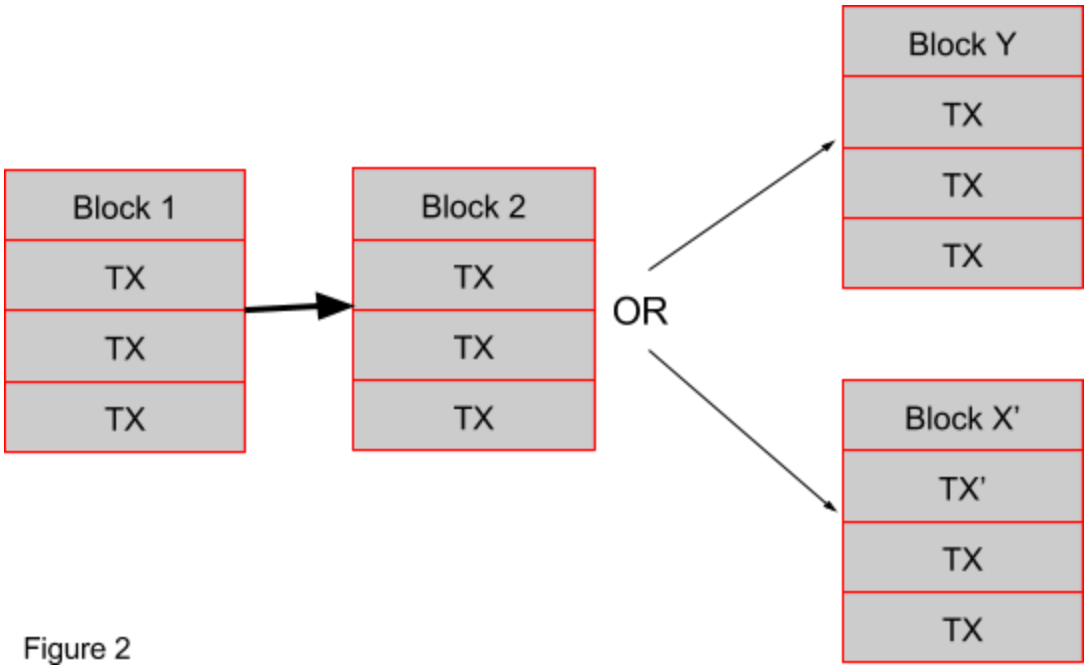

Figure 2

However, if block Y was built upon and double spending was attempted as demonstrated in figure 3, as more blocks were added to the chain containing block Y, block X’ would have to create a competing block chain and work to catch up with block Y and then surpass it. If block X’ could do this, it might be accepted as the real blockchain and proceed to be built upon.

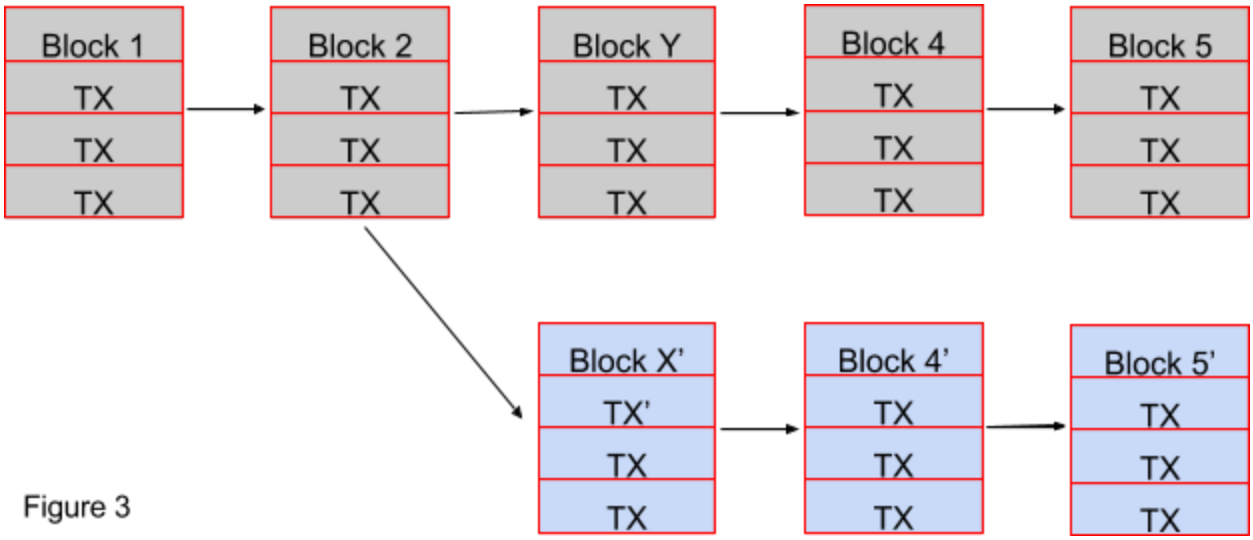

Figure 3

However, as Satoshi Nakamoto stated, the probability that any individual will find the next block depends on the likelihood that the attacker can find the nonce for the PoW challenge. This can be calculated as users fraction of hashing power in relation to total hashing power on the network. Therefore, if an individual had 1% of the hashing power in the node network, they have a 1% chance of finding the nonce. As multiple nodes will be working to extend the current branch that contains **X->Y,** the attacker will only be able to catch up with the current position of the branch if they control 51% or more of the hashing on the network - known as a 51% attack. Therefore, as each block is added to the blockchain, the attacker has diminishing probability of catching up to and surpassing the true blockchain.

Recently, criticisms of the PoW protocol have emerged. Many claim, that proof-of-work protocols as a means to find consensus on a blockchain in a distributed environment, is a "waste of energy…" (10). One reason for this is the way that Bitcoin specifically implements PoW. As explained, the probability of finding the nonce in a distributed network is directly connected to the hashing power each node controls. As computational power is reliant on investing money into powerful hardware, hashing power is expensive. To encourage nodes to find the nonce, Bitcoin rewards those that find the nonce with bitcoins. This had lead to an 'arms race' where nodes invest in more hashing power to increase their probability of finding a nonce. It may be beneficial short term to increase hashing power, but in the long run it turns into a zero sum game where although total power on the network increases, each node's power as a percentage of total power stays the same. Many argue this is resulting in huge, unnecessary energy expenditure.. For this reason, Proof-Of-Stake (PoS) has been proposed as an alternative consensus protocol. PoS randomly chooses the next node that can create the next block. However, seeding this pseudo random choice is the amount of currency each node holds, skewing the choice towards those who have a larger stake in the currency. The general principle is that although each node proposing the next block is not being financially rewarded for proposing the next block, the nodes with more currency are more likely to want to validate blocks for the health of the blockchain (11). This does not mean that PoS protocols can only be used within blockchains that focus on digital currency.

To summarise, cryptographic hashing functions and consensus mechanisms are strongly linked in the blockchain protocol. Cryptographic hashing functions in the case of PoW protocols allow users to verify the validity of transactions whilst creating an exponentially diminishing probability that an attacker can change data in the blockchain as it gets longer. However, this protocol is open to 51% attacks and is extremely inefficient from an energy conservation perspective. It is clear that as blockchain evolves, new consensus mechanisms, such as PoS protocols will be developed to solve these problems.

Appendix:

1. In an example where the previous owner of a property deed - Person **A** - has already transferred ownership of the property via blockchain to Person **B**, who in turn is now transferring the property rights again to someone else - Person C, given than the current transaction is from **B -> C**, the original transaction of **A->B** would be digitally signed. This digital signature, along with the public verification key of the receiver (**VK_c**) and the hash representing the property deed - **H(PD)** - would be 'collected' together and the collection itself would be hashed to sign the 'collection' or transaction - **H(Tx)** - resulting in the follow:

   $$Tx=\{H(A->B)\ ,\ VK_C\ ,\ H(PD)\ ,\ H(Tx)\}$$

   This represents the total transaction as seen in the blockchain. This transaction allows any node on the blockchain network to verify that **B** owns the **PD** (as shown in **H(A->B)**), the person who should receive the **PD** (represented by **VK_c**), the property deed itself - represented by **H(PD)** - and a digital signature stating that the information inside the **Tx** has been signed by **B** - **H(Tx)**. Furthermore, the fact that the information is cryptographically hashed means that users can verify the information, without seeing the content or metadata such as the location of the property, the property name, the name of the recipient or the name of the sender.

2. Arguably, double spending could be described as transferring data - ***a*** in this example - to multiple different nodes on the network - X, Y and Z. This could be represented as:

   $$(\ X_a\ ->\ Y\ =\ Y_a\ )\ \wedge\ (\ X_a\ ->\ X' = X'_a\ )$$

   In this example, **X** transfers **a** to **Y** and **X** transfers **a** to **X'**. **X'** could be a separate node on the network controlled by **X**. This is possible due to the pseudo anonymity that blockchain provides by only referencing individual nodes by public key. If **Y** were to transfer goods, information or other tangible or intangible commodities to **X** in the real world - thinking that **X** had met the conditions of the trade by making a transaction to **Y** on the blockchain - it is possible that **Y** could be the victim of fraud, as **X** could also have

made the same blockchain transaction to **X'**, essentially double spending, and it is now possible **X'** will receive the funds instead of **Y**.

References

1. International banking: Blockchain The next big thing. Economist.com. 2015. Available from: http://www.economist.com/news/special-report/21650295-or-it-next-big-thing. [cited 16 March 2017]

2. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. 1st ed. 2008. Available from: https://bitcoin.org/bitcoin.pdf. [cited 16 March 2017]

3. Ethereum Project. Ethereum.org. 2016. Available from: https://www.ethereum.org/. [cited 16 March 2017]

4. Tapscott D, Tapscott A. Blockchain revolution. 1st ed. London: Portfolio Penguin; 2016.

5. Princeton University. Cryptocurrency. 2015. Available from: https://www.coursera.org/learn/cryptocurrency/lecture/gFEJL/cryptographic-hash-functions. [cited 17 March 2017]

6. Princeton University. Hash Pointers and Data Structures. 2015. Available from: https://www.coursera.org/learn/cryptocurrency/lecture/EYEAo/hash-pointers-and-data-structures. [cited 17 March 2017]

7. Descriptions of SHA-256, SHA-384, and SHA-512. 1st ed. IWS; 2001. Available from: http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf. [cited 17 March 2017]

8. Princeton University. Consensus without Identity: the Block Chain. 2015. Available from: https://www.coursera.org/learn/cryptocurrency/lecture/71F3l/consensus-without-identity-the-block-chain. [cited 18 March 2017]

9. Merkle Tree - Bitcoin Glossary [Internet]. Bitcoin.org. 2017. Available from: https://bitcoin.org/en/glossary/merkle-tree. [cited 19 March 2017]

10. Alternatives for Proof of Work, Part 1: Proof Of Stake — Bytecoin Blog. Bytecoin: private secure financial system. 2015. Available from: https://bytecoin.org/blog/proof-of-stake-proof-of-work-comparison/. [cited 19 March 2017]

11. A Proof-of-Stake lecture at Oxford university [Internet]. Input Output. 2017. Available from: https://iohk.io/blog/a-proof-of-stake-lecture-at-oxford-university/ [cited 19 March 2017]

12. Hashcash.org. 2003. Available from: http://www.hashcash.org/ [cited 21 March 2017]