Blockchain Technology and Applications.

by

Andrew Gallagher

z5118026

Comp4121: Advanced and Parallel Algorithms – Project Component.

Abstract

This paper delves into the core components of Blockchain Technology and a variety of its useful applications. The paper begins by discussing Blockchains overall, and the purpose that it has. From here, a more in-depth analysis and explanation is given on key Computer Science and Cryptography based features that Blockchain is composed of; these being Merkle Tree and Cryptographic Hashing applications within Blocks of a Blockchain. Following this, the paper examines relevant applications of Blockchain Technology in regard to its previous successes, and potential issues it has for further applications to different fields.

Table of Contents

Introduction

Due to the nature of centralised currencies, demand has always existed for a conceptual currency that has the features of what is seen in modern decentralised cryptocurrencies; the main of these being anonymity, transparency, security, and verifiability among various other features.[blockchain03.pdf].

With the introduction of Bitcoin, the first successful attempt at a decentralised cryptocurrency with such features following its release in 2009 [bitcoin.pdf], there has been an unprecedented rise in the popularity of Bitcoin, and as a result cryptocurrencies in general from both an industry and academic perspective. Subsequently, there has been a similar growth in interest in the underlying technologies that gave Bitcoin its success; Blockchain Technology.

Blockchain Technology is one of the core technologies that Bitcoin introduced and heavily influenced its success. Blockchains for short can be seen as a distributed ledger of transactions. Simply put, blockchains can be represented via a distributed network, giving all parties involved access to the same up-to-date copy of such ledger, regardless of where it's accessed from.

For Blockchain Technology to successfully exist in this manner, it relies on the use of key Computer Science and Cryptography concepts; the focus of these being Merkle Trees and Cryptographic Hashing within its Block structures. The use of these concepts through various methods including "proof of work" gives Blockchain Technology such features that an end user wants in a decentralised currency.

Due to the successes of Bitcoin in making plenty of people rich, and thus, just as many people equally as poor, many have to tried to capitalise on such success through the boom of cryptocurrency and subsequent CFD markets. Because of this, blockchain has its most lucrative application through its use in countless number of cryptocurrencies that make up such markets. More recently, there has been development into whether blockchain technology can be applied in other uses with a similar degree of success.

Background

With the publicity surrounding Blockchain Technologies, approaches to the subject can be taken from both an industry and academic viewpoint. With the aim of giving Computer Scientists a lower level understanding of this topic, this background has higher focus on the low levels of Blockchains, but still may accessible to those with backgrounds in different fields.

2. Blockchain Technology

Blockchains are structures used to store transactions for its underlying 'currency'. The motivation driving this is quite simple; For a given currency, by having a record of its transactions, the distribution[1] of such currency can be easily traced with transparency, security and verifiability. Blockchains achieve this through storing a ledger with complete transactional history within a distributed network.

2.1. Network Structure of a Blockchain

The network choice is chosen due to the requirement of blockchains to be transparent. Networks may exist in the form of one of three underlying structures: centralised, decentralised, and distributed.

[1] With which party such currency is distributed to is not necessarily defined. Currency can be traced to an *anonymous* address, or to a specified party.

For a centralised network (Fig. 1.1), there exists a parent node, containing a full set of data which it is updated via this node. For any other node, they exist as a child node to such parent node, in which they receive updates from.

In terms of a decentralised network (Fig. 1.2), for a given set of nodes, there exists a subset of clusters formed from these nodes. For each cluster, there exists a link between such cluster, and another cluster in the network. Each cluster may store a subset of a complete set of data. Thus, connection of such clusters creates a decentralised network with such complete set of data.

For distributed networks (Fig. 1.3), of which blockchains are built upon, given a set of nodes and a complete set of data, for each node a copy of such set of data is stored within it. Each node may be connected at least one other node, such that all nodes are form a connected network. For a change in the set of data stored in one node, this change is then acknowledged, and then reflected in the set of data of all other nodes. A connection of nodes based upon these requirements forms a distributed network.

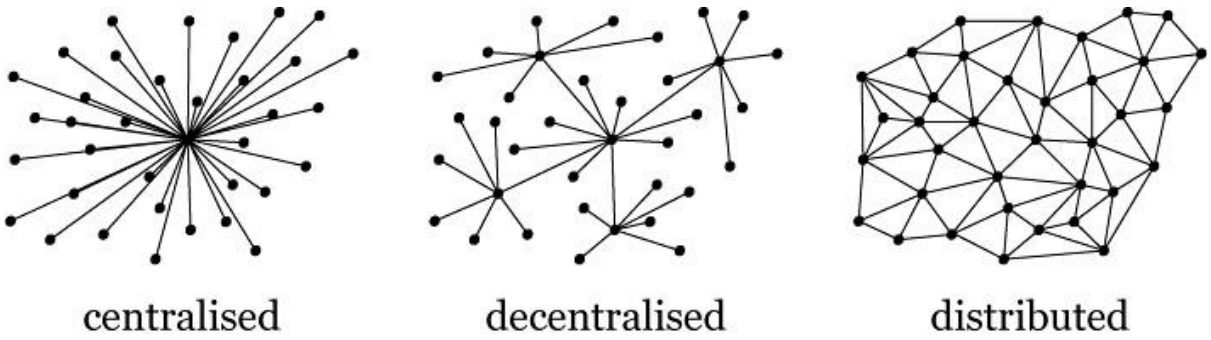

Figure 1: Network Structures

Due to the need for a blockchain to be transparent, the choice of a distributed network is simple. If based on a centralised or decentralised network, it'd be impossible for the ledger that a blockchain represents to be updated in full among

all parties. A centralised network would require such an update to be done via the central node, before being 'passed' to all other parties who themselves would not be able to update the ledger themselves, thus breaking the decentralisation that the blockchain relies upon. Further, a strictly defined decentralised network would not reflect updates to the ledgers from all parties, but rather just the parties a part of a given cluster where the ledger is being update. With a distributed network, all parties will have access to a complete ledger regardless of where in the network they connect to the blockchain from. Also, any update to the ledger can be undertaken by any party, with such an update being recognised through all parties connected to the network due to the completeness of such network. Because of this, its seen that a distributed network base for a blockchain provides full transparency through providing a complete ledger to all parties in which updates can occur from any party, and be examined by all other parties.

2.2. Underlying Architecture of a Blockchain

For a blockchain, it is defined so that it meets key criteria in terms of security, anonymity, transparency, and be non-dependent on 'trust' through verification. For the underlying architecture of blockchains, it is structured so that its components meet such criteria.

The fundamental architecture of blockchains are built upon linking together blocks, as result storing a complete history of transactions as described prior. As transactions occur, new blocks are created to record the existence of such transactions, and once created, are appended to the previously made block.

Continuation of such process as transactions occur creates what is defined as a blockchain.

For a given block, it is vital for the formation of a blockchain with criteria listed prior. Such a block can be viewed as a data structure (Fig. 2) in which it contains a unique subset of all transactions recorded on a blockchain. However, blocks do not only store just a subset of all recorded transactions. To ensure the criteria of a blockchain is met, blocks also contain a hash of itself, timestamp, the hash of the previous block as a reference, and a nonce value[2]. For these components, each has its place in helping ensure validity of the block through one form or another. In each case, cryptographic hashing is applied either directly or indirectly to ensure security and defer from tampering

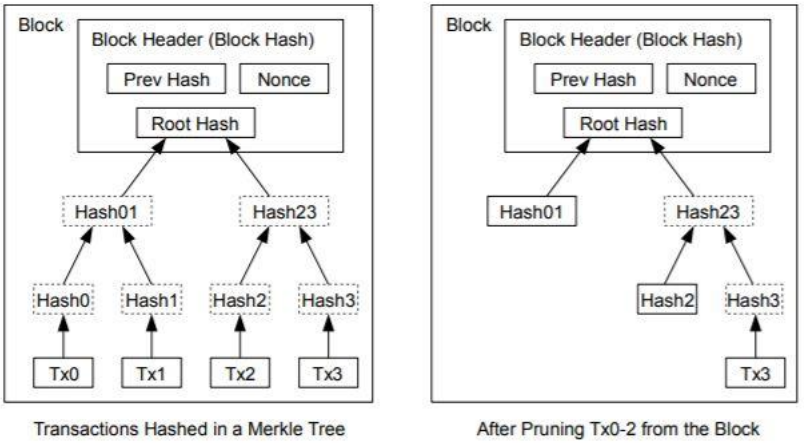

Figure 2: Blockchain Block Structure – Nakamoto [1]

Taking the hash of the block itself, otherwise known as the *block header*, the header is formed through a hash on all other components of the block. Such header is then given to the next block formed as a reference, and so for a given block, it is

[2] Nonce values refer to a single use arbitrary number used for cryptographic communication. [2].pdf]

used in the construction of the next block, and the block before it is used in the construction of itself. From this, it ensures security within the block by dissuading tampering, as tampering for a single block would require one to then alter every following block due to a subsequent change in the hash of the previous block stored, and thus a change in that following blocks header[3].

Looking at the nonce value, its application in blockchains work similarly to a block header to ensure security, and also has a use for verifiability. Nonce values are introduced to blocks to allow for the use of proof of work concepts. For blockchains, this is used in block creation, where *miners* calculate[4] the nonce value required to form the correct block header. In doing so computational efforts are shifted from the blockchain network to the miners[5], and so any tampering done to the blockchain would ultimately cause this work to be lost, and require recalculation of all nonce values in following blocks. Due to the large computational work required to calculate these nonce values, it helps to verify the blockchain as a whole, as if the majority of *miners* are honest, they will generally find the correct nonce value prior to those dishonest and create the next block, thus dissuading tampering.

In terms of transactional data, for a block it stores only a small subset of all transactions. Due to the many reasons for transferring 'currency', when a transfer

[3] For last block manipulation, it would have to be manipulated and added prior to the addition of new blocks to the blockchain, which in reality isn't realistic with blocks being added in at most minutes for the most profitable (and thus largescale) blockchains [1].
[4] Miners in this case don't necessarily 'calculate', rather they repeatedly attempt to guess the nonce value due to its randomness. Such randomness requires an unsophisticated way to get the solution – trial and error.
[5] Computational work is still work, and let's face it no one works for free. Those who do the work like to get paid it – the caveat being that since there's only once solution and thus one solver, they get all the recognition, and the prize – so maybe they don't.

between two parties is wanting to take place, it exists in an unvalidated form. When a block is created, it collects some of these unvalidated transactions to store within its partial ledger, thus validating them. The method in which blocks stores transactional data is through the use of a Merkle tree. The subset of transactions for a given block is hashed to form the leaf nodes of the Merkle tree, with each transaction representing a leaf . From this, the parent is formed by hashing the leaf node of this transaction with 1) the leaf node based upon the transactions of the previous block if an even block or 2) itself if an odd block. This process of forming the parent from the hash of the child nodes is repeated until the root node is formed. Thus, we store the transactions (Fig. 2)[6] through the use of a Merkle tree in such a way that it is secure.

For different implementations, certain components of blocks that do not affect its overall functionality may also be introduced. Take for example a timestamp, which simply stores roughly when such block was made. Simply, its purpose is not to ensure the necessary features of a blockchain, but to provide some additional information other than transactions to a party for soundness reasons of the underlying currency. However, due to its introduction, it acts as a buffer in which through the used cryptographic hashing algorithm, it alters the block header, thus indirectly improving security of the block. This idea can be generalised, and applied to any component that exists within a block to provide some other purpose, and thus has an indirect effect on helping a blockchain meet its criteria overall.

[6] Here there are 2 representations of how transactions can be stored. Either in its entirety, or with just 'route' to get from transaction Tx to the root node. Full nodes store the entirety of the block, whereas lite nodes the block header. For validation on Tx, lite nodes request the necessary nodes of the transaction Merkle tree – ones in the latter representation – to ensure it matches with the root node, and is thus valid.

Underlying Concepts

With many revolutionary ideas in Computer Science, the underlying concepts of such idea may not entirely be by the author, but are killer applications of concepts that are already in circulation. For blockchain technology, this notion applies, with use of Cryptographic and Computer Science based concepts[7] through the form of cryptographic hashing and Merkle trees. Understanding these underlying concepts is key to understanding the academia behind blockchains as a whole. As such, a low-level description of both concepts is given below.

3.1. Cryptographic Hashing.

3.1.1. A low-level introduction to Cryptographic Hashing.

To introduce, cryptographic hashing is based upon the millennia old field of cryptography. Concepts of cryptography have existed since the earliest manuscripts themselves have been written [3]. This is of no surprise; multitudes of reasons exist as to why one would want information to only be understandable by certain parties. Because of this, the field of cryptography has existed for long periods of time and inherently has led to the development of cryptographic hashing.

[7] One could argue the extent of which blockchains are the killer applications of these concepts; in reality it's probably not, with many uses for these concepts. On the same note one could also argue if Cryptocurrency is the killer application of Blockchain Technology. Maybe there exists an even better (profitable) killer application of blockchains that trumps over its usage in cryptocurrency.

Cryptographic hashing has the aim of creating a trace[8] of a message in such a way that it easily verifiable but hard to manipulate. This is achieved by showing that given the underlying message and the trace produced, if a different underlying message were to be used, it would result in a different trace to be produced. To illustrate mathematically[9], let h define an arbitrary hash function and x some data. Applying h to x, the outputted trace would be $y = h(x)$. As above, if a different underlying message x' were used, then it is expected that some output y' would be produced by applying h to x'. By comparing trace y and y', it can be seen that the altering a message would result in a different trace being produced. Thus, just by using any arbitrary hashing function, it is easy to produce verifiability.

Unmentioned above is the manipulation of such cryptographic hashing functions. For cryptographic hashing, as much as it is important to be verifiable, it is equally important for it to be hard to manipulate. In this context, the extent of manipulation of a cryptographic hash functional can be defined through the computational feasibility of finding an invalid pair, such as (x', y) from the example above. Such manipulation can be illustrated through three problems[10].

The first problem, more formally known as 1st-preimage, is defined as taking pre-specified outputs, the 'solution' is to find any input that hashes to such output. Building on this, it takes the idea that if the outputted trace of a hash function is known, the solution to such problem would involve computing an input so that it produces such output. To reduce manipulability in regard to this aspect, it should

[8] Here a trace simply represents the outputted hash value from a hash function. It represents a fingerprint of what the underlying message is in a non-contextual way.
[9] Basic mathematical Illustration referenced from Rosen [4].
[10] Definitions of such problems are referenced from the Handbook of Applied Cryptography [5].

be computationally difficult to produce such an input based upon knowing what the output, and its respective hash function that produced it is. If it is computationally difficult, then the hash function can be considered secure in this aspect.

The second problem, namely 2nd-preimage like that of 1st-preimage, is defined so that given an input, find a second input which produces the same output as the former. Unlike 1st-preimage, the focus of this problem is that if the message provided to the hash function is known, the solution to this problem involves calculating a trace equal to that of the first message, from an entirely new message. To ensure little manipulation, the hashing algorithm must make computationally impracticable an individual to calculate such a trace.

The third problem, the collision problem[11], is defined as find a matching output for two distinct inputs. Namely, the aim is to find a pair of messages, where such messages are distinct, that both result in the same trace being produced from a given hash algorithm. To ensure secureness against manipulation, such hashing algorithm must, like the other problems, make it so that the solution to this problem is impractical to calculate.

Thus, for a cryptographic hashing algorithm to be considered secure against manipulation, it must be secure individually against these problems. By having

[11] The resistance a hashing algorithm has to the collision problem implies its resistance to the 2nd-preimage problem, however it does not guarantee it. This is due to the expectation in bounds for the 2nd-preimage problem, in which the collision problem can only guarantee a bound lower than expected. For example, say 2nd-preimage resistance expects a bound of 2^n whereas the collision resistance only guarantees $2^{\frac{n}{2}}$, thus implying 2nd-preimage only to $2^{\frac{n}{2}}$, lower that what is expected. For further explanation, please refer to the paper *Cryptographic Hash-Function Basics* by Rogaway and Shrimpton [6].

secureness against these three problems in conjunction, chance of manipulation by a party in any sense will be computationally impracticable. Therefore, in a generalised sense there should be no weakness to manipulation for a given cryptographic hashing algorithm as long as it resists against such problems.

Hence for a hashing algorithm to be sound, it has to meet such constraints based on its verifiability and secureness towards manipulation. By allowing a trace, the hashed output, via a hashing algorithm is reproducible its can easily be seen that doing so would allow verifiability of an underlying message to ensure it occurs. By using a hashing algorithm such that it has resistance to the three problems listed, it ensures its secureness in a general context. Thus, following these constraints would produce a sound cryptographic hashing algorithm of a generalised form.

3.1.2 Specifics of Cryptographic Hashing Construction.

3.1.2.1. A General Model of Cryptographic Functions.

Continuing on with the generality of cryptographic hashing functions, the use of a model is a sound basis in which to build constructions of the hashing for more specified purposes on. To begin, take such a model, in this case of an iterated hashing algorithm, which can be used to build constructions on to obtain resistance against problems listed above. Taking a message that needs to be hashed, the first step is to pre-process the message so that it can be hashed by the compressing hash function. To pre-process, the message needs to have padding bits and a length block applied (when required) as to ensure that it is of correct size for the iterated process of the hashing algorithm. Using this newly formatted version of the message, it is then passed through an iterative process, which

outputs a hash of the formatted message compressed in length. This compressed hash is then used as the input of the next iteration, and is repeated until the hash produced is of the correct length required for the hashed output of the general hash function. Optionally, this output is passed through a final hash function which transforms it into the optimal output. Thus, a model (Fig. 3) is formed which gives a general representation of iterated hashing for which to base specific cryptographic constructions on.

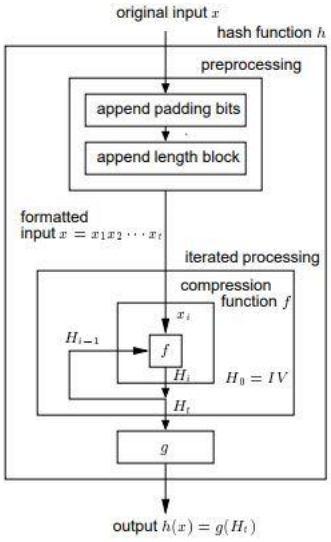

Figure 4: Iterated Hashing Model – Handbook of Applied Cryptograph [Menezes].

Using such a general model, specific constructions of cryptographic hashing algorithms have been formed so that they are applicable for particular uses in which they are designed for. Such constructions can be defined in two ways; through their use of a key or not. For a keyless hashing algorithm, it has specific uses in manipulation detection, thus giving the name *manipulation detection code*. Keyed hashing algorithms, by having a use of a key allow for authentication of a message itself, and so in turn is given the name *message authentication code*. For

these constructions, they take the assumption that Kerckhoff's Principle[12] holds true.

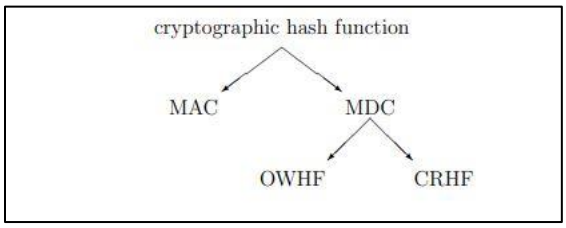

Figure 5: Taxonomy for Cryptographic Hash Functions - Preneel

3.1.2.2. Manipulation Detection Code Construction.

As stated briefly, manipulation detection code is built off the lack of need for a key. As such the only input for hashing is the message itself, and so the focus is on manipulation of the message, and not the author of the message. By Kerckhoff's principle, the outputted hash of this message is public and thus known, and so, as to ensure that the underlying message has not been tampered with, by applying the same hashing algorithm to the same message, it should produce the hash that is expected. For any different message, it should produce a hash that is different to that of the original message. From this, *manipulation detection code* can be broken down further into two constructions to achieve this. The first is through a *one-way hash function*, and the second through a *collision-resistant hash function*.

A one-way hash function involves easy computation in one direction, and is computationally difficult in the other. Applying this, it should be simple for the hashing algorithm to produce an outputted hash, but by having the hashed output,

[12] States that the "security of a cryptosystem must lie in the choice of its keys only, everything else including the algorithm itself should be considered public knowledge" as defined by Petitcolas – [7]

it should be 'impossible' to revert it back to the original message using such hashing algorithm. To define formally, a one-way hash function must satisfy the following conditions [8]:

1. *The given input x can be of any length, but its resulting hash $H(x)$ has to be of a fixed length equal to n bits where $n \geq 64$*
2. *When given a message x and hashed output $H(x)$, it must be computationally difficult to find such another distinct input x' such that it produces $H(x)$. Thus, resistant to 2nd-Preimage.*

Such rules assure that the output produced of the hash is of fixed length, and that it is computationally difficult to find another input that would give the same hash output, leading it to be manipulatable without detection. Defining 'computational difficulty', it requires that such resistance to 2nd-Preimage requires an upper bound of $O(2^n)$ as based on the *Random Oracle Model* [4]. This asserts that for an adequately large value of n, in the case that a 2nd-Preimage is found is less than 2^nattacks, it is still computationally difficult.

Next is the collision-resistant hash function, which entails a subset of rules similar to that of one-way hash function, and then builds on it to include collision resistance. So, collision resistance hash functions have properties similar to that of one way hash functions, but require collision factoring constraints. Defining CRHF as given by Damgard [8], the formal definition is as follows:

1. *The given input x can be of any length, but its resulting hash $H(x)$ has to be of a fixed length equal to n bits where $n \geq 128$*

2. *When given an message x and hashed output $H(x)$, it must be computationally difficult to find such another distinct input x' such that produces $H(x)$. Thus, resistant to 2nd-Preimage.*
3. *For the given hash function, it must be computationally difficult to find a pair of distinct message x and x' such that the outputted hash function of both sees $H(x) = H(x')$; It is collision resistant.*

Here, the rules of CRHF also assert that a fixed length output is produced. Likewise, it also defines resistance to 2nd-Preimage. However, a separate definition exists due to the definition of 'computational difficulty' occurring here. In this case, computational difficulty is defined so, again by the *Random Oracle Model,* 2nd-Preimage resistance is requires an upper bound of $O(2^n)$, however, collision resistance by this same model requires only an upper bound of $O(2^{\frac{n}{2}})$ [4]. Due to this, a collision and thus a 2nd-Preimage can be found in less than $2^{\frac{n}{2}}$ attacks, and thus, for an adequately large n, computationally difficulty in the case this is found. Hence, this definition of 'computationally difficulty' separates this construction from that of the one-way hashing construction.

So, for modification-detecting code based algorithms, there exists two constructions applications can be built on. For this, the construction chosen is based on the definition of computationally difficult required.

3.1.2.3. *Message Authentication Code Construction.*

As mentioned, message authentication code construction relies on the use of a keyed hashing algorithm. Here a key is defined as an arbitrary value subject to the constraint of being mathematically difficult to compute. With this

understanding, keyed hashing algorithms here use such key to authenticate the validate the source of a message so that it matches source of the party it is intended to have come from. As again, by Kerckhoff's Principle, the key mentioned here is private [7], as to ensure that for a given private key, and public message, if repeating the hashing process, it will produce the same hash as the original hash. In the use of a different key, the outputted hash value will not be equal to that of the original. Hence, informally defining the construction being message authentication code[13].Taking this informal explanation, a formal definition can be given as follows [8]:

1. *The given input* x *can be of any length, but its resulting hash* $H(x,k)$ *has to be of a fixed length equal to* n *bits where* $n \geq 32 \dots 64$
2. *When given a message* x *and hash function* H*, it should be computationally difficult to calculate the hash function* $H(x,k)$ *such that the chance of success is much greater than* $\frac{1}{2^n}$*.*

   Expanding, even if a large number of selected messages and their respective hash value are known, $\{x_i, h(x_i,k)\}$*, it should be computationally difficult to calculate the key from these pairs, or to calculate any hash value without the key. This here is named 'adaptive chosen text attack'.*

Essentially two rules give a cryptographically sound definition of message authentication code. Of which, the first rule here implies that MAC produces a fixed hash output, much like MDC. The second rule here, simply states that for a key, using MAC it should be collision resistant, and also impossible to calculate

[13] As an aside, MAC has been in use in the banking industry for a long time. But only after open research began, have MACs been introduced with sound cryptographic properties – Preneel [8].

for a person only holding the messages and outputs hashed the key, and not the key itself.

Hence, for message authentication code construction, its requirement that a key ensures the hashing of such messages can only be done by parties with access to such key. For any other party without such key, they will not be able to hash a message in a way that it would match the output of the same message hashed with the key without computational difficulty, thus ensuring authenticity of a messages source.

3.1.3 Relevant Applications of Cryptographic Hashing Functions.

In terms of relevancy, as the focus is on Blockchain Structures, the applications of the above constructs mention will be necessary for a robust understanding of Blockchain Technology. As such, included will be applications for Information authenticity, and application in proof protocols.

3.1.3.1. Applications for Information Authenticity.

Information authentication refers to both the protecting the authenticity of the source and also modification of the message such source is sent. Ensuring information authenticity can occur in two ways. The first via ensuring information authenticity based on the key, whereas the second involves ensuring such authenticity through the outputted hash.

The first method entails using a hash algorithm to compress a given message, and then append the outputted hash to the original method. By doing this, a party can determine whether or not the information is authentic by verifying the appended hash much like a signature at the end of a physical contract. For this, a key is used

in the hashing process, to ensure that only holders of that key can produce the required output hash. If there is no existence of such hash, or the hash does not match the one expected to be outputted with use of a key, a party can determine that the information is unverifiable, and thus is unsafe to use. In terms of MDC and MAC, approaches to this can be done by incorporating the key in the compression stage for an MDC based hashing algorithm, or using the key as the key required for MAC input. Thus, through appending, this provides a verifiably sound way of applying a cryptographic hashing algorithm with focus upon the authenticity of an appended 'proof of ownership'.

In terms of the second method, focus is not so much on an appended trace to a message, but rather outputted hash of the method. Here, the reliance is on keeping the sources' outputted hashes private for information authentication. By doing this, when the underlying message is sent to another party, the receiving party can hash such message. Of which, they obtain the hashes from the author of the message who originally hashed it, and compare the outputted hashes to determine if the message has been modified by some other party along its 'travels'. In the case that the hash produced by the receiving party does not match that of the one produced by the author, then they can determine some modification has occurred by some unknown party, and thus the information cannot be authenticated. Opposite to this, if equal hashes are produced, it is then authentic.

For these applications, they rely on sending such private information, either through a key or the correct output hash, to that of other parties for them to ensure security. There exist cases where additional third parties will want to verify information authenticity, but the author of such message is unwilling to give up a

private information to prevent modification by such third parties. To deal with these cases, the use of what is known as digital signature schemes come into play.

A digital signature scheme invokes the use of two keys; a private one for that of the author, and a public one which third parties can use to ensure authenticity. More so, a third party can prove to a subsequent third party that the copy message they received is authentic, even if the owner original copy from public domain, as they have all elements required to prove such authenticity.

For construction of digital signature schemes, it can be related back to one-way and collision resistant functions. As mentioned above, collision resistant functions are robust in terms of their level of computational difficulty. However, one-way functions are easier to implement and generally cheaper. In terms of which to use for digital signatures, it is done case by case, depending on the requirements. A case where one-way hash function is where all parties completely trust the other [8]. In the case of blockchains it focuses on a trustless system, and so the use of a CRHF implementation is used.

In terms of the details of a digital signature algorithm, a general form can be produced through the following method. 1) Create a set containing a public and private key through a key generation algorithm. 2) Optionally hash a given message for the purpose of uniformity and compression. 3) Apply said (hashed) message through a signing algorithm given the private key and hash key as input to create a signature of said message. 4) When verifying, apply the relevant verifying algorithm to ensure said signature is valid through the use of the public key. By following said steps, a digital signature can be formed in a general sense.

Using such model, a more formal approach can be given for a digital signature. In this case, the approach used is based off RSA encryption, with the mathematical construct[14] as follows:

1. *Randomly select two large primes* p *and* q *such that* $n = p \times q$*. Here* n *will be in the public domain.*
2. *With this calculate*[15] $\varphi(n) = (p-1) \times (q-1)$. Using $\varphi(n)$, determine to integers e and d such that $a\ b \equiv 1\ (mod\ (\varphi(n))$.
3. *For values* e, d *and* n*, form a public and private key, such that the public key is the set containing* $\{a, n\}$ *and private key contains* $\{b\}$*. Here the public key is in public domain, and the private key is only available to the signee.*
4. *Apply, if required, a compression hashing algorithm to the given inputted message* m*.*
5. *With said message* m*, produce a signature* σ *where* $\sigma \equiv m^b (mod\ n)$*. When verifying, verify* σ *with* $\sigma^a \equiv m\ (mod\ n)$*. In the case this is not congruent, the authenticity can be assumed to be invalid.*

Such use of an RSA based signature algorithm gives light into the difficulty in attempting to forge a signature. Due to the complexity in trying to reverse factors from the products of two primes, it becomes computationally difficult to do so, and thus makes it computationally difficult to find the private key based on step 2. Here, when even knowing n and e, to calculate $\varphi(n)$ without knowing its factors, it becomes a computationally difficult task to complete. As such, finding d based

[14] Following formal definition is based off *Simple Secret Sharing and Threshold Signature Schemes – Tang* [8].
[15] Here φ denotes Euler's Phi, which for a integer n, it will calculate the number of integers i up to n, such that $\gcd(n, i) = 1$

off its modulo subsequently becomes difficult as well. Hence, the RSA based signature algorithm illustrates the protectiveness of such digital signature schemes in terms of information authenticity.

3.1.3.2. Applications for Proof Protocols.

Proof protocols form one of the underlying fundamentals of many cryptographic algorithms designed over the years. Defining, a proof protocol can be seen to illustrate that a proving party has to convince a verifying party that it knows 'a secret of some form, or that a certain mathematical statement holds true' [Proof of Work]. From this cryptography based idea, the *proof of work* concept was developed.

Proof of work entails that the prover proves that they have completed enough work to a verifier through the form of an output. Expanding, this ensures that without seeing the prover complete such work, they have achieved some result that can be used by the verifier to verify such work knowing that to achieve that result requires a large effort on the prover's part. This here is essentially the notion of Jakobsson's proof of work concept Hashcash, the cryptographic hashing based application of proof of work.

Hashcash incorporates proof of work through the calculation of a token. From this, here a token that is computationally heavy to calculate by a prover is required by that prover to prove to the verifier that they have completed such calculation. By finding the correct token, they have proven completed the required work. Doing this, moves the onus of 'proof' from the verifier to the prover. As the computational expenditure is on the provers end, they create a simple value in which verification

can occur with little expense through a simple check. In terms of the token, this can be set in two ways [10]. The first is interactive, in which the verifying creates a challenge that the prover has to solve, whereas the second is non-interactive, in which such challenge is created by the prover themselves. Here, we can define both interactive and non-interactive proof of work system from the Hashcash model as follows [10]:

$$Interactive = \begin{cases} c \leftarrow Chal(s, w) \\ t \leftarrow Mint(c) \\ v \leftarrow Val(t) \end{cases}$$

$$Non - Interactive = \begin{cases} t \leftarrow Mint(s, w) \\ v \leftarrow Val(t) \end{cases}$$

In this definition, there exists a challenge function, which produces a said challenge by the verifier; A cost function $Mint$, that given a challenge c calculates a token t by the prover; and a token evaluation function, which given a token calculated by a prover, will determine if that token is correct. For both implementations, they require the verifier s to set a pre-determined amount of work w to ensure enough computational work is done.

Relating to blockchains, proof of work is done through an interactive setting. As such, taking the interactive definition, cryptographic hashing can be applied in the following setting. When determining a challenge for a prover, the challenge requires that the hash produced by the prover meets some specific constraint. To do this, the token when applied directly, or indirectly through appending it to a message, to the hashing function, the subsequent hash produced meets the constraints of the challenge. Here, finding such a token that produces the correct hash of the challenge is what requires computational work. In verifying, the

verifier simply takes the token given by a prover, and applies it to the hashing algorithm in its respective way. If the outputted hash matches that required by the challenge, then the prover has proved to the verifier he has completed the required work. If not, it cannot be verified that he has done so.

Merkle Trees

Merkle trees form an essential part of a block's, and subsequently a blockchain's structure. Here, Merkle trees are act as data structure for compressed storage for a subset of transactions that are stored within a block. Due to their design, they allow for easy verification of a transaction to see if it is valid within a block. As such, a low-level description of Merkle trees shall be given below.

3.2.1. Low-level introduction to Merkle Trees.

To start, Merkle trees can be seen by another name; that of a hash tree. Generally speaking, based off the latter name, a Merkle tree can be seen as a binary tree structure in which each of its node is labelled via a hash. For such a tree, it allows for compressed representation of the data set through the root node's hash. Further, it allows a quick verification of each element in the set of data through an efficient traversal time.

To build such a Merkle tree structure that allows for the given benefits, it requires a hash to branch between parent and child node. To start, the leaf nodes of such tree are formed from the set of data through a cryptographic hashing algorithm with OWHF features, where each leaf stores the hash of its respective element in the set. From this, the tree can is built in such a way that the hash formed through hashing the hash of the two child nodes. Repeating such process until a root node is found will form such tree. More formally, as defined by Ederov, this process can be represented as [11]:

$$P[i,j] = h(< P\left[i, \frac{i+j-1}{2}\right], P\left[\frac{i+j+1}{2}, j\right] >)$$

Here P defines an assignment, mapping it's respective two children, of which h is the hashing function used. This can be further simplified for illustration to define the left, right and parent to n_{node}, forming:

$$P(n_{parent}) = h\left(P(n_{left}), P(n_{right})\right)$$

Thus, it can be simple to see from the formal definitions the required assignments and hashing functions to build such a tree. However, not addressed by this is the case for leaf nodes and as such is currently undetermined. For a set of leaf nodes, they can exist in two 'states'. One where the underlying data set and so leaf set has an even number of elements, and the case where both are odd. To account for this, the following is introduced:

$$P(b_{i,i+1}) = \begin{cases} h(l_i, l_{i+1}) : & l_{i+1}\ exists \\ h(l_i, l_i) : & l_{i+1}\ does\ not\ exist \end{cases} \quad i \in even\ values\ of\ |\{l\}|$$

Where $b_{i,j}$ represents the branch of two leaf nodes l_i and l_j, for a leaf set of length $|\{l\}|$.

From this it can be deduced that if given a data set of odd length, for the final leaf in such set, it's parent will be formed with the hash of its hash twice. Thus, ensuring that a complete binary tree can be formed. Hence, following the notation given, a sound Merkle Tree can be formed which can be used to meet the benefits given.

In terms of such benefits, the first forms in the case of compressed representation. For the a given Merkle tree built in the way specified above, the resulting root node will be a representative hash of a given set of data. As each parent is formed off the hash of the two children, by building up, each higher level will incorporate

all leaf nodes that fall in the sub-tree with such node as root. Take the following for example; Given a set of data {A,B,C,D} which are paired as defined above. The resulting parent nodes will be formed as P(AB) = h(h(A), h(B)), and the same for P(CD). From this, the root node will be formed as P(ABCD) = h(P(AB), P(CD)). This causes the resulting hash of the root node to incorporate A,B,C and D, as required.

From this, the second benefit can be illustrated. Here, the second benefit entails an efficient method for verification based off a given Merkle tree. Given an element in the data set that, at some other point in time different to the tree's creation, needs to be verified, it can simply be done through rehashing it into a leaf, and checking it against all other hashes that are required for that element to get to a root node. If the root node's hash is equal to that of the original Merkle tree, then that element is verified. Basing an example off the one given above; Take the element C, in order to verify the underlying data has not been modified in say a copy, start by rehashing that supposed element of data and define it C'. For this, then calculate $P(C'D)$ with the leaf of D. Next, the root node $P(A\,B\,C'D)$ can be calculated through the use of $P(AB)$. If the given node $P(ABCD)$ is equal to the new root node produced above, then it can be certain that element C' in this copy of the data set is the same as the original. This example can be further generalised to any verify any element in such a data set.

Furthermore, it can be generalised for a Merkle Tree of any height. As such, the computational effort is based on the height of the tree, in which case, for verifying a single node, the number of calculations required is equal to the height of such

tree + 1 (for the comparison check). Therefore, the time complexity of such problem is equal to $O(\log n)$, and so is an efficient method for verification.

Another thing to note is that for a given check, it only requires the nodes that are used in hashing a node 'containing' the element checked. For instance, the above example only required P(AB), and l(D) to calculate the root hash. This allows for memory saving to be conducted by trimming the Merkle Tree, so it only contains the nodes necessary to get to the root node (Fig. 1). When there exist multiple copies of the data structure containing a Merkle tree, only certain copies need to store the full tree. Others can contain only the final hash, and, ensuring the final hashes are equal, they can simply request the nodes necessary from a full tree to take out such verification. As a result, in a dormant state they only need to store the root node, and when verifying, they only 'grow' to include the complementing branch from each level. Such a feature therefore allows for the majority of the copies of a Merkle tree to be stored in a further condensed format, increasing memory saving to a bound of $O(\log n)$, compared to a full tree with a memory bound of $O(n)$.

Blockchain Block Structure

As mentioned, blocks of a blockchain form the entirety of its underlying structure in such a way that ensures security of the overall blockchain. Taking this, blocks incorporate in their construction the above cryptographic concepts that give it such features so that it is secured. As such, a low-level analysis will focus on the components of a block, explaining its features and how the incorporation of such concepts gives it the constraints required of a blockchain.

4.1. Architecture of a Block.

In terms of the architecture of a block, it can be broken down into its components which incorporate the above cryptographic concepts either directly or indirectly. As such, these components are chosen, and implemented in such a way that they ensure the soundness of the block they form, and in turn the entire blockchain they fall into. For analysis, blocks will be discussed through their components and the soundness to blockchain's they provide.

The components of required of a block can be given as followed. For each block, it is composed[16] of a block header, a record of its transactions, a nonce value, a record of the previous block, and a timestamp. By having such components, the block can be formed through their use in such a way that the block is sound in terms of security, anonymity, and transparency.

[16] Other optional components may also exist depending on the application of such blockchain to fit its requirements.

Beginning with the block header, it simply represents the entirety of the block. For this, it is composed of a hash of a conjunction of all other components of the given block that it represents. In doing this, it allows for a holistic approach to verification of the block, as if any change occurs to each subcomponent, a resulting change will occur to this block header. Block headers also have a second use; they're used as the connection between the current block and its successor. By storing the block header of the current block in the following one, it forms a link between the two. In terms of security, if one were to alter the current block, it's block header would change, thus having a domino effect on all following blocks, as for each block, its block header would change from its predecessor's block header changing. Such a process deters against tampering due to the expensiveness in terms of rehashing each subsequent block to account for the change of the block tampered.

The next component of blocks to discuss is the record of a set of transactions that each one stores. To first explain, a transaction stored is a part of a record for an amount of currency. In each transaction there exists the public key of the party receiving currency, a signature from the sending party, and a hash of the receiving party's public key to store as record of ownership, as well as multiple inputs and outputs referring to the amount transferred [1]. By having it formed this way, when the receiving party later transfers the currency to another party, the former's public key can be used to verify such transaction, and their private key used when they send such currency to ensure it is them doing so. This method of storing transactions allows for a verifiable way of checking where currency lies, and if the sending party has enough of the currency to do so.

From this, a block, when formed, gathers unconfirmed transactions and adds them to a set of transactions in which it stores. However, for a currency with high trading volume, storing such currencies can be costly memory wise in terms of a full blockchain. To account for this, a block does not store the transaction in its entirety, but rather stores a record of such transaction which can be used for verification at a later time. Such storage occurs through a Merkle tree[17]. In such a Merkle tree, a leaf node is formed using the hash of a receiving party's public key stored in a transaction as discussed prior. Having the leaf nodes structured this way, the set of leaf nodes represents a set of transactions for a block, where leaf represents one transaction stored in such sets. Building a Merkle tree with these leaves forms a root node with a hash representing all transactions confirmed by such block. As such 'lite' blocks can be formed in which only the root node of this tree is stored within the block [1]. For verification, the block can request from a 'full' version block in the network the required path to verify a transaction, and so storage is significantly minimised (Fig 1).

The required component of a block is its nonce value. Nonce values come into play for the proof of work concept given above. By having one, it allows proof of such an effort to be moved away from that of the network, to those who wish to add a new block to it. As such, it helps to ensure security by requiring both honest, and dishonest parties wishing to add to the blockchain a complete a certain amount of work before they can. In terms of this nonce value, it simply represents the token that solves the challenge set by the blockchain network. Such a challenge exists in the form of the required output of a block header, in which the challenge is to

[17] Refer back to Merkle Tree subheading for further explanation of Merkle Trees.

find a nonce value that produces a block header with a required number of leading zeros. By doing this, the number of zeros directly correlates to the probability that a guessed nonce value is the correct one, where the greater the number of zeros, the harder the challenge is exponentially [1]. From having this nonce value based challenge, it eliminates the number of users trying to create nodes, and incentivises honesty. In the case of tampering, the nonce value for each preceding block would have to be recalculated, thus increasing the overall effort to successful tamper with a blockchain.

A timestamp is the last required component of the blockchain to be stored within the block. Timestamps, although not directly involved in the security of the block itself, help to keep record of blocks within a blockchain, and also provide security to the currency itself in terms of double spending. For a given amount of currency, by having timestamps, if multiple transactions occur with that currency, the timestamp can be used to solve where it is transacted, and thus eliminate the double spending. Using the timestamp, when there are two transactions for such currency, the timestamp would simply ensure that only the first transaction is accounted for and the second ignored, so that a party holding it cannot spend it twice.

Thus, by forming a block with such components, it ensures the soundness of the blockchain containing it itself. Due to the nature of cryptographic hashing, the block's header, and subsequently its use in the next block, provides a way in which it protects against tampering through the requirement of recalculating the hashes of all following blocks. Further, the addition of the nonce value as a proof of work, makes it so recalculating the such block headers require all previous calculation

for proof of work to be done again, and so any previous computation spent is sunk, and thus lost as result of tampering. Also, due to adding a timestamp, although a simple feature, it ensures a fix against double spending, making the underlying currency of the blockchain useful in a contextual environment. As such, blockchains are secure based on their block components and implementation.

Block Time

Block time can simply be referred to as the time taken to create a given block. For blockchains based on a proof of work scheme, this time taken is directly correlated to the difficulty of the 'work' problem, and collaboration between solving it. For other schemes such as the Proof of Stake, there is no reliance on 'work' and so no delay due to it, and so block time is controlled by the design of the blockchain. As such, block time can be determined through either the 'work' required for a block, or the time for Proof of Stake.

5.1. Difficulty of the 'work' problem

As previously mentioned, the difficulty of the work problem has a direct affect on the time it takes to solve a network set challenge, and thus create a block. By creating such a problem, it requires any party trying to add a block to the network to first solve the 'work' problem, and thus sacrifice some of their computing time to add to their authenticity.

In determining the difficulty of such problem, it is required to consider the probability of success that a party tampering with the blockchain will succeed. For a blockchain to be successfully tampered, it requires the last block[18] to have its nonce value changed by the party tampering it. A version of such a problem can be given as derived by Nakamoto [1]:

[18] The last block is considered as any previous block requires the change of all following blocks. As more blocks are added, these chains are longer than the one being tampered, and so when the tampering is complete, that chain will be shorter than the current one, and thus rejected by the general consensus.

$$p = probability\ an\ honest\ node\ finds\ the\ next\ block$$

$$q = probability\ a\ dishonest\ node\ finds\ the\ next\ block$$

$$q_z = probability\ the\ dishonest\ node\ can\ catch\ up\ from\ z\ blocks\ behind$$

$$q_z = \begin{cases} 1 : & p \leq q \\ \left(\frac{q}{p}\right)^z : & p > q \end{cases}$$

It can be seen then that as long as the probability an honest node succeeds in finding the next block before a dishonest one as more blocks are added to the network it becomes exponentially more unlikely for a dishonest node to succeed in tampering the blockchain.

Thus, the difficulty of the work problem must be tailored so that it is long enough to favour probability of an honest node finding the block before a dishonest one. In determining this, the following is defined:

$$p = s * \frac{\sum d}{n}$$

Here, the probability of a dishonest node finds a block on a given attempt is determined by the chance of success of finding a node with the number of dishonest nodes in the system relative to the number of nodes in total. By choosing s such that it minimises p entails the difficulty of the work problem. However, as change to s has a linear effect on honest nodes, it is necessary to ensure that the problem is not of a difficulty that it becomes too impractical for a new block to be added.

5.2. Collaborative efforts for solving Proof of Work.

As discussed, up until now, current means of the proof of work concept have been done in a one on one manner, with representation being between the network server and one party. When generalised, it sees parties trying to form a new block compete against each other in order to be the first to create such new block. However, proof of work can be approached in a new manner which sees collaboration occur between parties. As a result, the workload of a challenge, although still the same, is dispersed across all members involved, thus reducing the total time taken to create a block.

Such method of collaboration can be done through that of a virtual machine. In which each member of the collaborative effort can 'donate' their resources to the virtual machine to produce the nonce value. In this case, the nonce value is not guessed, but rather is the amount of work required to produce a block. As such, the 'challenge' set by the network is not a random value, but the computational cost a transaction has associated with it to be 'confirmed' and placed within the block. As so, it is done state to state, and a finite amount of values represent the nonce, unlike the non-collaborative form of proof of work, in which it is stateless, and thus a single guess does not reduce the probability of success in finding a 'nonce'.

5.3. Proof of Work vs. Proof of State.

There also exists another concept which essentially eliminates cost in terms of block time; proof of stake. Here proof of state defines the 'finder' of a block based

on a random lottery of which all parties holding the currency of the block chain can take part in. Here, in a rudimentary implementation, the more currency 'bided' the higher the chance that party has of being the one to implement the block. To make it more complex so that block calculation is not solely reliant on the amount of currency a party has, a randomised hash can be added and given weight to reduce such reliance.

The underlying problem with proof of stake, however, is that it provides no computational deterrent to tampering, as no computational work is required. So, in a network where the blockchain is based on proof of stake block creation, although it reduces block time, it still has to conform to the rules to prevent a '51% attack [12]. That is, if the probability of a dishonest block is greater than that of an honest one, as defined above, then the blockchain is likely to become 'infected' with fraudulent data. Due to the lack of a 'work' requirement, one can expect that the number of dishonest nodes in the system will also rise, as there is no cost for them to act dishonest, and so by luck, and the law of large numbers, if this dishonest nodes rise above 50%, they will infect such network.

Hard Forks

In terms of blockchains, a fork of any type occurs when there is simply a split along the chain. The reasonings behind such a split varies due to the intentionality in such fork occurring. Due to this, the forks can either become permanent, or are temporary, with any temporary fork dying off. Take for instance a block being found simultaneously; here the general consensus will build off one fork, and the other will die out, thus creating a *temporary* fork in the system. In terms of a hard fork, it is an intended permanent fork of the blockchain.

For a hard fork to occur, it requires a change in the underlying protocols set by those running the blockchain network. What such change to the protocols does is cause all previous blocks to be invalidated, unless they update to the network change. Here, the hard fork is developed if not all nodes accept the new protocols, and update to it. As a result, some nodes will continue the blockchain using the old protocols from the block where such protocols were introduced. The other nodes that accepted the changes however, will, from the same block, continue their own fork of blockchain with blocks based on the new network changes. Further, this fork with the new set of rules will only account for transactions in blocks up to the block where the protocol change was introduced.

The reasoning behind a hard fork can vary due to many reasons; the two of focus however will be for straightforward software updates, and the other due to successful interference into the blockchain (potentially indirectly) by a third party. In the case of a straightforward update, blockchains tend to be actively developed over time [13], and so they often need updating to reflect such development. In

this case, due to the planned nature of the change, the block selected for the hard fork tends to be a future block to be created, as to give nodes on the network time to prepare for such update [13]. In most cases, as long as the changes are relatively straightforward, and not of dire effect, they are accepted by nearly all nodes on the blockchain. In the case there is some delay in the nodes updating, a small hard fork is formed, of which relative to the updated blockchain, there are only a few blocks added while the nodes update to the new protocol.

For interference in the network, say if a hacker steals a whole bunch of private keys from a big crowdfunding platform, then the protocols may be introduced to a block earlier in the blockchain. Here, not all nodes may agree with the changes, such as the party who steals the blockchain's currency and the other parties affected by the change happening at an earlier block, making their transaction invalid. In this case the hard fork is more visible, as both branches of the fork are continually added to; one with the new protocol by those who made the switch, and one with the old one that didn't.

Modern Applications

Modern applications of blockchain technology focus on those where the underlying data wants to be decentralised, verifiable and secure by a party that is potentially anonymous in the network itself. As such, the most prominent application of blockchain is in cryptocurrencies. However, due to the relative newness of blockchain technology, further applications are currently being developed with many new proposals for the use of blockchain forming every day.

7.1. Application of Blockchain Technology via. Cryptocurrencies

Blockchain Technology see's its most prominent application in the form of cryptocurrencies. A cryptocurrency is simply a medium of exchange, much like any other currency. However, cryptocurrencies, by being based on blockchains through a decentralised network, are in turn decentralised, immutable and transparent much like that of the features of a blockchain. The cryptocurrency at the forefront of financial markets is Bitcoin, with approximate 70% market share[19]. This is of no surprise, as the introduction of Bitcoin saw the birth of blockchain technology.

In application, cryptocurrencies rely on blockchain technology, as well as other cryptography concepts to ensure such requirements of decentralisation, immutability and transparency are met. Starting, transactions of cryptocurrencies rely on signatures to record and later verify the exchange. Here, a party 'giving'

[19] Based off Market cap of $130,000,000 as of December 2019. Figures taken from coinmarketcap.com [14]

their cryptocurrency to another user signs a signature with their private key. As such, their public key, as well as the receiving party's public key, as well as the amount given, are stored within the transaction. Finally, a hash is taken of the public key of the elements stored in the transaction to represent such transaction. For transactions that occur like this, they are then stored in a block as describe, which when created confirms the transaction of the cryptocurrency to the new owner. Parties requiring verification of the transaction can do so using the methods for verifying a transaction as listed above.

Due to the profitability of cryptocurrencies, there has been an exponential growth in the number of different ones available to the public. As a result, there has also been a proportionate growth in the number of people trying to exploit such systems. In such a case, it has led to the case where hard forks have been applied, as seen with the second-most popular cryptocurrency Ethereum. One of the collectors of Ether, Ethereum's stored currency, DAO, faced the issue where it's private keys were compromised [15]. As a result, approximately $50,000,000 worth of Ether was hijacked [15]. As the transactions that took place were not of compromise of the Ethereum system, it came to the debate on whether the transactions should be allowed. Such debate led to the hard fork, where there now exist two forks of Ethereum, one still known as Ethereum, where the transactions were rejected, and the other Ethereum classic, which kept the transactions of the attacker. As so, it can be seen that properties of the Blockchain have come into play in the real world.

7.2. Further Applications of Blockchain Technology.

Although blockchain's main use as of now still lies in Cryptocurrencies. Due to its relative newness there has been plenty of research into applying blockchain technology to other applications. The main focus of these have been on using blockchains in some sort of ledger format. More specifically, it has seen the development of blockchain based smart contracts, and development in supply chain management . As such alternative uses of blockchains are still in their early stage, it is yet to be known if the killer application of blockchains has been found.

First looking at smart contracts, these are contracts that are directly facilitated through a computer protocol. As such, the facilitation, and subsequent verification and enforcement of the contract can calculated automatically through the protocol, separating it from the need of any third party. By basing it on blockchain, smart contracts are able to track transactions that occur within the network. When a transaction that 'triggers' the smart contract occurs, it can then carry out the appropriate steps as required by the contract its based on. Such an application of a smart contract (although conceptual at this time) can be seen as an automated escrow. Here, instead of a third party delivering funds when terms of a contract are met, the smart contract replaces such third party to deliver such funds when the terms of the contract are triggered.

In terms of supply chain logistics, blockchain technology can be improve traceability within such supply chain. By using RFID technology to physically track elements within the supply chain, the data produced can be stored within a blockchain based ledger system to ensure verifiability of all products within such

supply chain [16] To expand, take the food industry; here, each element in the supply chain represents facilities in which produce has to pass through, and be RFID scanned. At the point of sale, each batch of produce has a full block in which it stores the supply chain which it has passed through. In the case of a food scare, it can be easily verified through the block that batch is in where it came from, and using the same blockchain, find all currently moving batches and halt their progress in reaching the shelf.

Such practice has use in the real world by 'retailing giant' Walmart, which incorporated a similar system after a food scare involving romaine lettuce [17]. By having this system, and adequately storing produce by its batch, it minimises cost by removing having to throw out all of that produce, but rather, only the produce from that batch. By also preventing more produce from reaching the shelf, it can further prevent any more health risks from happening from that product, and possibly stop people from eating needle-filled strawberries [18]. As such, this practice be beneficial in both protecting the health of customers, as well as minimising the damage to the pockets of businesses.

Conclusion

Blockchains form a new technology to the world of computer science in which they have plenty of potential for vast improvements in the world in the future, and have already seen a substantial amount of success. Such success relies on the constructions and applications of Blockchains, and their innovative approach in applying cryptography concepts to create a distributed system. As such, this paper has delved into the functioning of Blockchains and their inherent features. Thus, this paper has focussed on the concepts of blockchains as a whole, and their applications of discussed cryptographic concepts in terms of their blocks. Further, it has discussed such successful applications through cryptocurrency, as well as potential uses through smart contracts and supply-chain management. From this, one can understand the functionality of Blockchain, and see its potential for new uses in the future based on its current success.

References

[1] Nakamoto, S 'Bitcoin: A Peer-to-Peer Electronic Cash System.', 2008.

[2] Rogaway, P., 2004, February. Nonce-based symmetric encryption. In International Workshop on Fast Software Encryption (pp. 348-358). Springer, Berlin, Heidelberg.

[3] D'Agapeyeff, A 2008 *Codes and Ciphers – A History of Cryptography*, Hesperides Press, Hong Kong.

[4] Rosen, K 2005 *Cryptography: Theory and Practice*, CRC Press, United States.

[5] Menezes A, Oorschot P, Vanstone S. 2001 *Handbook of Applied Cryptography*, CRC Press, United States.

[6] Rogaway, P. and Shrimpton, T., 2004, February. 'Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance'. In *International workshop on fast software encryption* (pp. 371-388). Springer, Berlin, Heidelberg.

[7] Petitcolas, F.A., 2011. 'Kerckhoffs' principle'. *Encyclopedia of cryptography and security*, pp.675-675. Springer, United States

[8] Preneel, B., Govaerts, R. and Vandewalle, J., 1993. 'Cryptographic hash functions: an overview'. In Proceedings of the 6th International Computer Security and Virus Conference Vol. 19.

[9] Tang, S., 2004. 'Simple secret sharing and threshold RSA signature schemes'. *Journal of Information and Computational Science*, 1(2), pp.259-262.

[10] Jakobsson, M. and Juels, A., 1999. Proofs of work and bread pudding protocols. In Secure Information Networks (pp. 258-272). Springer, Boston, MA.

[11] Ederov, B., 2007. 'Merkle tree traversal techniques'. Bachelor Thesis, Darmstadt University of Technology Department of Computer Science Cryptography and Computer Algebra

[12] Bastiaan, M., 2015, January. 'Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin'. *Available at http://referaat.cs.utwente. nl/conference/22/paper/7473/preventingthe-51-attack-a-stochasticanalysis-oftwo-phase-proof-of-work-in-bitcoin. pdf*.

[13] Kiffer, L., Levin, D. and Mislove, A., 2017, November. 'Stick a fork in it: Analyzing the Ethereum network partition'. In Proceedings of the 16th ACM Workshop on Hot Topics in Networks (pp. 94-100). ACM.

[14] Top 100 Cryptocurrencies by Market Capitalisation, accessed 2 December 2019, <coinmarketcap.com>

[15] Siegel, D 2016 'Understanding the DAO Attack". *Coindesk*, 25 Jun 2016. Accessed 2 December 2019.

[16] Tian, F., 2018. *An information system for food safety monitoring in supply chains based on HACCP, blockchain and Internet of things* (Doctoral dissertation, WU Vienna University of Economics and Business).

[17] Corkery M, Popper N 2018 'From Farm to Blockchain: Walmart Tracks Its Lettuce'. *The New York Times,* 24 Sept 2018. Accessed 6 December 2019.

[18] Silva K, Sibson E 2018 'Strawberry needle contamination: Accuse woman motivated by spite, court hears'. *ABC News*, 12 Nov 2018. Accessed 6 December 2019.