The greening of blockchain mining
Revised version 2020

Bill Roscoe and Bangdao Chen

University College Oxford Blockchain Institute and The Blockhouse Technology Ltd

Abstract
We analyse the proof of work mining model and show how to simulate many of its properties in a form of proof of stake called **Work your Stake**. In doing so we create innovations on timing, branch prevention and achieving an unbiasable random oracle that will be useful in other Blockchain models and decentralised systems.

1. Introduction

At the time of writing (mid 2019) the best known public blockchains are maintained by proof of work, a protocol which is simultaneously magical and disastrous. We will see why it is magical later. The best-known objection to it is that proof of work wastes energy. Mining a block of the bitcoin blockchain currently means that the network has to compute a truly astronomical number of hashes: about $2.2*10^{22}$. For comparison, the Andromeda galaxy is only about $2.5*10^{22}$ metres away and there (via Avogadro's number) are about the same number of atoms in a gram of silicon or six grams of gold. This work does no good whatsoever except for controlling the rate and the distribution of who gets to create blocks and making various attacks difficult or impossible. The second objection is that it concentrates mining power in the hands of those with highly specialised equipment and cheap electricity. More recently a second model has become apparent: mining by people who do not have to pay for the electricity they are using. Thus, we read of scientists misusing supercomputers, of botnets of miners, and of hijacked web servers mining. These things do not contribute positively to the world.

One often sees suggestions that the energy consumption would not be so serious if we would find a more positive use for it. This might simply be a use for the waste heat or doing useful calculations instead of hashing. We observe, however, that if this created economic benefit for the miner, the effect would be to reduce the net cost of mining. Thus, more miners would join the more profitable game: it seems to us that the net, rather than gross cost of mining, will naturally settle at a level determined by the reward model. Creating public good is more defensible, but in this world any major unnecessary sink of energy must be questionable at best.

Yet we will see that PoW has positive qualities that are hard to replicate, such as reliably controlling the rate of mining and allowing a very distributed implementation.

The objective of this paper is to introduce and justify a mining model with as much as possible in common with PoW, but does not require anything like so much energy usage.

The core idea is to retain the hash competition and its economics but make it vastly easier by dint of controlling how many entries can be made. Each has to be paid for. Broadly speaking we want to force miners to spend money on mining tokens of some sort rather than electricity. Of course, we then have the pleasant problem of deciding what to do with the money.

The rest of this paper is structured as follows. In the next section we remind ourselves of the basics of blockchains and their evolution. We consider what the positive qualities or PoW are and show how to restructure the chain so that one of these is no longer important. We then consider how to have a hash competition limited by tokens and how the structure of these tokens can amplify incentives to behave correctly. Next, we discuss a means of limiting the rate at which blocks are produced. We then examine how to make the puzzles that drive the hash competitions fair, in the sense that they cannot be influenced to favour anyone. We next consider the rules that motivate the creation of a linear and inclusive blockchain.

This paper differs from the original draft by (I) improving on the security of the creation of the random seeds used for things like mining lotteries and (II) building backlinks into the Blockchain that make it secure against the late addition of branches.

In the final analysis this creates a flexible mining environment with a number of parameters that can tune how the overall system behaves. We believe that from the point of view of security it has considerable advantages over PoW, as well as eliminating its primary drawback.

2. Blockchain background

A blockchain is a data structure in which a sequence of files, or *blocks*, are created, and each contains the cryptographic hash of its predecessor. The essential property that the hash function must satisfy here is 2nd preimage resistance, in other words given that we have hash(x)=y, then for all but a negligible fraction of xs it is practically impossible (i.e., computationally infeasible) to find a second value z such that hash(z)=y. In other words, based on the nth block of a blockchain there can be no debate about what the previous n-1 blocks are once we have plausible candidates.

As a measure of extra security, given that hashes are relatively cheap to compute, one will normally use a hash that is believed to have the property of strong collision resistance, namely that it is infeasible to find any x and y such that hash(x)=hash(y). Furthermore, given that we expect blockchains to live for a long time, the hash should have a long expected life, tolerating for example the advent of quantum computers and the anticipated effect of these on hashing.

For us, a blockchain will be for use in distributed systems, an agreed record of what has happened between a number of widely spread users. There will always be many copies of the chain to maintain accessibility in such a network. No party in this network is regarded as totally trustworthy by everyone, in the sense that it will always follow the rules and always be available. If there were such a party, then everything can be delegated to it and there is no need for the complexity. So, a blockchain is a way of engineering trust and availability from a network of not-necessarily trustworthy and not-always available parties, who may indeed come and go.

In principle blockchains can contain any information. However, in practice this information is likely to consist of:

(I) Transactions of assets held on the blockchain.
(II) Hashes of potentially large amounts of data held off it, including pointers to the data. Note that this, given a sufficiently strong hash, can guarantee that the said information does not change, but does not guarantee that it remains available unless the data is replicated with the blockchain itself.
(III) Registrations of assets such as real estate or gemstones.
(IV) Smart contracts: programs that are guaranteed to be run, when some condition triggers them, by the blockchain itself. These will normally create transactions.

Moreover, blockchains contain information needed to maintain their own structure and integrity, such as the previous block's hash. We will make some detailed suggestions about this later.

One of the most important issues in blockchain is how the next block is created. This question essentially divides blockchains into two categories. The first are ones where block formation is carefully organised and the job of a restricted group who must approve each new block meaning, that the chain grows completely linearly. In the second, the creation of blocks is something of a free-for-all and blocks are created by individual parties when they are able to do so. A number of schemes have been devised for this, the two most prominent being Proof of Work (PoW) and Proof of Stake (PoS).

In PoW each block B implies a puzzle that must be solved to create the next block. In most cases the puzzle is as follows: find a nonce N (i.e. string of bits) such that hash(hash'(B)C(N)) $\leq \varepsilon$ for some preset value ε (where the value hashed may be some variation on this) and C is a given context. Here, hash and hash' may or may not be the same cryptographic hash function. (Careful thought reveals that they are playing quite different roles, on the assumption that hash'(B) is the linking hash that will be placed in the new block.) hash'(B) and C(N) together comprise a header which typically includes a description of the body of the proposed block (as a hash) and the context in which the block sits.

Finding such an N gives a miner the right to place the new block, including this N as a certificate, on the blockchain.

It is possible for a fork to appear in the chain, either by accident, because a second node gets its own certificate at about the same time, or by villainy, via nodes finding further certificates to follow B even when they know one already exists. The assumption is that such action will be to influence the structure of the chain, though exceptional gains for some piece of mining (e.g. transaction fees) could have this effect also.

Rules are created, such as always backing the longest valid chain, that are designed to have correct (i.e. rule following) miners converge quickly to a linear chain, and furthermore overwhelm a minority of dishonest ones simply through being able to create more certificates thanks to the greater amount of work they can do.

We will discover more about proof of work as we try to emulate it.

There are a variety of proposals for proof of stake that typically use cryptographic mathematics to form a fair competition based on the various nodes' holdings (stakes) in the blockchains. One of the main problems of PoS is that it is potentially too easy to create "fake" chains and branches, which can appear to be the genuine extension of the chain from some real block.

3. The magical qualities of PoW

Before designing our own mining protocol, it is as well to reflect on why PoW is so successful. The following sets out our interpretation of this.

1. **Attractive and understandable.** The concept of a lottery is instantly understandable without specialised cryptographic knowledge. Similarly, the simplicity of only being to win by extreme effort is appealing. The concept of making mining attractive by offering rewards is similarly attractive.
2. **Controlling block rate.** The mechanism by which PoW achieves this is simply making sure that all the computing resources thrown at the problem will not generate solutions faster than we want. It can, as in Bitcoin, be adjusted to maintain this rate.
3. **Fairness of puzzles.** By this we mean that the agent or agents who actually set each puzzle do not gain a meaningful advantage from this. In PoW where the puzzle is the hash of the previous block, the miner of this possibly does have a small advantage from knowing this in advance, as of course do any allies. But because hashes are all separate trials, here the only advantage can be a short start in a long process and delaying the use of a solution carries the risk of losing out to someone else.
4. **Countering forks.** The greatest danger to the integrity of a Blockchain is that it might branch with two or more separate series of blocks emerging as successors to some node and being believable. The rule in PoW is that the longest correct chain must be backed. This strongly discourages both branches appearing around the head of the chain, and ones being added in further back. To understand the first of these, consider the situation of node A who has been mining for a solution to the puzzle to create block n. If someone else has found a solution, then A has two choices: to continue to look for solutions to the same puzzle, or to look to the one that now exists for block n+1. Alice will almost always be exactly as likely to find a solution to either puzzle in a given time, and she knows that she already has a probably preferred rival as the winner at level n, but none yet for level n+1. Therefore, it will almost always be better to switch her attention to the longer chain that includes the rival's black n. Rather than competing at the head of the Blockchain, an attacker might decide to add a fork at an earlier point by finding an alternative solution to the puzzle embedded there. However, to be successful the attacker would then need to be able to grow his branch faster than the real chain or it cannot pass the test of

being the longest branch. But to do this it would need to command more computing power than the proper branch[1].
The bottom line is that good nodes following the rules can simply out-work a minority of bad ones and will do so because of the rewards involved.

5. **Ensuring payment for mining.** The economics of PoW are that each competitive hash performed costs the same whether it is supporting a universally agreed block or one of a number of options. You pay for it whether you succeed or not because you have to pay for the resources and energy. In fact, PoW's origin (pre Bitcoin) was as a way of ensuring everyone made micropayments in a context where the discouragement of having to pay was more important than collecting the money (namely sending spam emails).

6. **Allowing distribution**. PoW avoids the need for all potential miners to synchronise before the creation of each block.

Of course, the real test of an incentive structure is whether it effectively deters all misbehaviour. In such a structure we need to maintain vigilance even though it might be boring and a little costly, even though nothing is ever found.

If we contemplate the anti-branching discussion above, it is clear that the winning properties of PoW thanks to backing the longest chain depends on two things. The first is that anyone considering the believability of a candidate series of blocks must have the sight of what other candidates are out there to compare. The second is that whatever mechanism is defining the block production rate, such as the difficulty of puzzles, must be uniform over all branches. If an attacker could reduce the difficulty on its own branch it would not work. Both of these speak to a Blockchain being more than just the sequence of block values, plus an elective community of active nodes. In particular, it suggests that there must be common knowledge of a central location for storing control information and that parties in general should have a "heads up" perspective to judge security.

As long as most users of a blockchain use standard or checked software to run their activity, we have nothing to worry about. However, if people start to use a cheap and cheerful version without the vigilance there is a potential problem. They might think "I can avoid the costs of vigilance because I never find anything wrong, and other people will be doing it anyway." This strikes us as an interesting problem. Can one mandate software standards or otherwise ensure vigilance? *Quis custodiet custodies?*

In general terms all of this raises the issue of whether a given blockchain specification is stable and secure, or whether it is just the precise implementation(s) in use that are making it so.

[1] A possible flaw in this argument is that the difficulty of mining a block in the original blockchain and the attempted branch might differ. One can compensate for this by carefully defining "longest" to mean the aggregate expected hash cost.

4. Keeping your head down

In our view the strongest argument, with current technology, for PoW is its ability to suppress branching via the sheer force argument. In this it has a clear advantage over proof of stake models we are aware of.

In this section we propose a model that allows us to move the responsibility for eliminating branching, at least of a certain sort, away from the mining model and into the basic structure of the chain itself. Specifically, we seek to remove need for security to be heads up as described in the last section.

In its pure form a public Blockchain is the values of the blocks that make it up. It is not determined by where it is stored, because this changes (along with replication) as the Blockchain evolves. It is created by a population of participating nodes that varies over time, and over whom there needs to be some trust assumption. We probably need to assume that this assumption has always been true, not just that it is true now.

We would expect that at any time the chain will consist of a sequence of blocks, possibly numbering in the millions or more, starting at a block B0 that contains initial conditions and rules, linked by the fact that the n+1th block contains a strong cryptographic hash of the nth. Around the head, where new blocks are being created, there may be several contenders and other data structures. We would expect old blocks to become settled and uncontroversial, as indeed they must be thanks to the hash links for someone who knows a newer one.

The main purpose of the present exercise is to show how to ensure that someone who only knows the value of B0 can join the chain securely, depending solely on the state of the Blockchain rather than depending on properties of the network. The following sets out the assumptions we make:

- Everyone understands that the Blockchain is identified by its root block: the value of it, not any specific memory location.
- At any time, there will be a collection A of active nodes who are interfacing with the Blockchain. All have a currently reliable signature method. We assume that always a majority of these are reliable, in the sense that they are supporting rather than undermining the protocols.
- The nodes in A all have good knowledge of what is happening at and around the head of the chain.
- As blocks become immutable, the nodes of A become aware of this. Eventually the fact that a block is immutable will be common knowledge: all of A know it is immutable and all understand they have common knowledge of this.
- Nodes can leave and join A. The crucial feature we pay attention to is how a node joins: it needs to know the values of blocks at the head of the chain: we pay attention to what it must do to identify these.

- We anticipate that once a node has successfully identified the head of the Blockchain, it will become familiar with the majority of the present community A and the way the Blockchain is developing. It will know where to look based on its interactions.
- As previously implied, we can imagine two modes for a node intent of joining: in heads up node a node B is expected to be conscious all contenders for being the head of the chain. We can call this **heads up** mode because B is expected to diligently look around. In this mode B has to look at all contenders to see which one wins. This is the principle with PoW, where the longest chain wins. The second mode is **heads down**: in this B can simply be presented with a record purporting to be at or near the current head and can, by examining it and the implied chain from the acknowledged root, determine whether this block is near the real head, in the sense thar it is a block being monitored by the present A, and thus the key to discovering who these are.
- It is obvious that architecture that only requires a heads down check is more secure than one requiring a heads up one, even if some element of heads up checking is still done.

The basic structure of a Blockchain, namely blocks in which each contains the hash of its predecessor, with the ultimate successor being the start block B0 that represents the chain, implies a tree rather than a chain. It relies on ever-developing consensus to pick a single successor, but in the standard data structure there is nothing to enforce this. We regard the initial establishment of consensus amongst the currently active nodes A as a different subject from keeping that consensus visible to all for nodes outside A.

Imagine the following scenario: block Bn was established some time ago, and Bn+1 as its successor. Potentially the cryptographic signatures associated with the nodes involved have weakened and/or the nodes involved have gone offline. Whether using these things or not, an attacker constructs a plausible successor for Bn, namely B'n+1, and as many further successors B'm to this as it likes. We would like the blocks to contain structure that would allow the joining node B to recognise such a branch even though he has never seen any of the Bm for m>n.

Presented with one of the B'm as being at the end of the chain, with presumably all the A' used by it under the control of the attacker, how can he tell the difference? One way might be to look around (heads up) and see what other independent people think, but it would be better not to have to.

Our proposal, based on the above assumptions, manages to place upward pointing links (which we term *backlinks*) into the Blockchain in addition to the traditional downward pointing ones. These are individually not so secure since they depend to a greater extent on the trustworthiness of nodes, but we use them so that the presence of relatively few trustworthy nodes amongst successful nodes is sufficient.

The proposal is as follows: a number s is chosen so that it is believed certain that of any s consecutive blocks, a reasonable number will have been mined by a trustworthy and honest node. For example, on the assumption that at least half the mining power is honest, the chance that less than 10 of 100 consecutive blocks were mined by honest parties is infinitesimal (8σ discrepancy in binomial distribution or less than 10^{-15}.)

The creator of each block is obliged to include a fresh public key PKn in it, of which only she knows how to create the corresponding signatures. This is the as-yet unplaced ***hook*** that will eventually stiffen the blockchain. She is obliged to use that key only once, and using it twice attracts a great penalty. We imagine that the most likely form of the public key will be a Lamport one or a more space efficient hash-based one-time key because of their intrinsic quantum resistance.

She is also obliged to stay in A and watch the development of the chain until she sees that block Bn+s is immutable. At that point she makes the signature under PKn available for inclusion in the Blockchain, for example as a transaction, and to watch to make sure it is included. (Replication of the signature is allowed, but not needed.) Before such a signature is included in block r for r>n+s it is checked, so only accurate signatures make it in.
Once that signature has been placed, the hook has been placed: block Bn attests one successor, and that is Bn+s, so in effect there is a link from Bn to Bn+s in addition to the standard chain of links from Bn+s to Bn.

It is easy to incentivise both the creation of signatures and their inclusion in later blocks: all or part of the mining free for the block in which the key was registered can depend upon the signature, and attractive mining frees can be associated with including the signature in a later block. Where, as proposed elsewhere in this document, prospective miners have to offer a good behaviour deposit to be allowed to mine, we can amplify the incentive to provide signatures by forfeit of that.

As the chain evolves, it is vital that this signature protocol is complied with (in the sense that these *hook* signatures are present and correct) for a higher percentage of each window of M blocks than the largest conceivable number of bad-miner blocks in the M. We can reasonably assume that signatures by good nodes are present, as they have every motivation to provide them, and when good miners create a block, they have every motivation to include them. Non-performance on these two fronts can be taken as evidence of badness, which bad nodes will not want to provide.

In practice it will be necessary to choose parameters such that this is so *and* have a remediation protocol involving extra signers. For example, noting that it would only be bad miners that have motivation not to supply one, there might be back up signers for blocks.

In any context such as public mining or knowledge of a KYC protocol where disgrace will follow non-compliance, we can be more confident that even bad nodes will deliver their signatures.

We have the advantage that if, despite all this, insufficient signatures appear, some other action can be taken, since the lack of placed hooks will be apparent to all.

Now, if C is presented with a node B’m, he follows the Blockchain back to B0 using the conventional hash pointers. Furthermore, every time he encounters one of these signatures, he checks it. If B’m is a block of the original chain this will work. If, however, the nth block is in the correct chain and the n+sth is not, the signature of B’n+s-i can only check out if the creator of Bn-I (for I in {0..s-1}) has breached her promise to use PKn only once (for she certainly signed Bn+s.)

We have to ensure that an attacker, in creating a branch at block Bn cannot create sufficient signatures of members of the branch by bad miners before Bn to make it believable. In other words when C encounters a signature pattern which falls short of a criterion that the bad miners cannot achieve.

For the initial part of the chain we assume that the creator of B0 is honest and trustworthy and does sign the block or blocks it is expected to sign.

If this check succeeds, there can be no branch away from the true Bm sequence other than towards the end of the Bm where there is still at least one trustworthy miner in A, and C will perform checks with such people.

So overall C can check the true status of the block he is given following only pointers within it, on the assumption that C has shown that it does lead back to B0.

What we have done here is to provide the core of the solution. More research is needed on the details. See my paper Taking the Work out of Blockchain Mining *available at* www.tbtl.com.

Hooks provide a highly organised and long-life form of block attestation that resides entirely in the blockchain itself.

5. Work your stake

The idea behind this paper is simple: create a version of PoS that mimics proof of work except that now rights to mine are bought rather than gained by spending the same money on electricity. Thus, the economics of mining will be very similar, but energy will not be needlessly consumed. Anyone who wants to mine needs a token. Clearly such tokens need to be immutably bought. It follows that before they are used there must be indisputable transactions creating them.

We call this model **Work your Stake** or **WyS.** As this will be a proof of stake model, we assume than the anti-branching technology described in the previous section, which is independent of the details of how block creators are chosen, is employed.

It follows that our Blockchain will contain a form of transaction by which a node purchases, probably using the tokens/coins of the chain, the right to mine at a later stage. We will call these *mining tokens* and refer to general currency as coins to avoid ambiguity.

The value of a mining token must exceed some defined minimum, which will be the sum of the maximum sum one can be charged for mining a single block and any deposit that is paid. For efficiency in creating and committing tokens a single transaction might create multiple tokens which can be used together.

Before a token is actually used it must be committed to mining on a specific block, identified by its index. This can be done as part of the same transaction that creates it or later. This commitment is required to ensure that miners spend money even when they do not win the game. (Of course, in a variant game we may allow a token to be used until it does win.)

The commitment must be done far enough in advance that the commitment is immutable when the chosen block is mined. We note that in PoW the actual cost to anyone attempting to mine a block is proportional to the time taken for someone to solve the puzzle, whether they solve it or not.

It is illegal for a single identity (and thus a single token) to back two different blocks at the same level. This would be a strange action, and without this rule there would be the formal possibility of doing this with the same token committed to this level. To police it we might require public backing of a block by each identity that is mining to extend it. We do not believe this is necessary, though.

Suppose token t has been committed to mining block n, and block n-1 has now appeared. A miner must first decide if it wants to back this block by extending it. In general terms it should do so if the previous one (as well as its predecessors) follows the rules. After all, the miner has already invested in this process and wants his mining to bear lasting benefit. We will discuss what happens in the negative case later. The model we discuss in detail below has nodes winning the right to mine and only then deciding which block to mine if there is a choice. However, we ensure that the choice is always clear, and our node knows where his duty lies.

Once block n has become immutable, then the deposit (subject to good behaviour) and any balance of the fee element can be

(I) Carried on as a mining token subject to there being a sufficient balance.
(II) Redeemed back into coins.
(III) Carried on in whole or part as security against the performance of some other duty.

The crucial thing is that each token only allows a single entry into the hash competition for block n. It follows that the agent doing the entry can have no choice over what is hashed. We assume that the blockchain creates some value X which forms the basis of the next competition, and that the said competition is based on the hashes of the combinations as t varies over the tokens committed for this round, or something very close to this. It is essential that X is completely unpredictable before the current block is published, even to the agent who is publishing it. For otherwise that party may be able to bias the competition in its own favour by the way it builds the block. We will discuss this later.

6. The puzzle game

Suppose agent A is creating a block B that must contain a puzzle X which defines the game for the right to build the next. All the tokens that are going to be used are committed and apparently visible to A. A has perfect information about her own committed tokens and importantly perhaps also those of A's allies. Therefore, she might be able to manipulate B so that it suits one of her and her allies' tokens.

To avoid this, we need to have X unbiassable by A, certainly at the time it constructs B and preferably at all. It follows that if X is indeed in B, neither A nor any small coalition can choose it or bias its value constructively.

There is no great problem of this sort with PoW because A is likely to disadvantage herself in the current competition by spending resources working out what block to submit with her winning entry for it. In other words, it is such a difficult process to anticipate that it makes cheating essentially pointless. This is the core of the security of PoW: building fake chains or doing computations to bias future competitions is simply too expensive. The first is simply not possible relative to the rule that the longest chain is best, and any attempt to choose a more advantageous solution to a competition is unlikely to be worth the effort. It is, however, hard to argue in any generality that it cannot be done at all.

However, in our world there is a fundamental problem in puzzle setting if the value of the puzzle (typically a random number) is not to be bias able by any of the competitors or their allies. If the value of the puzzle is known in advance of committing mining tokens, that would undermine the model as nodes would mine only with tokens likely to win. If the value is computed from the state of the Blockchain at some later point, then whoever last influences that gets an unfair advantage.

We need an unbiasable random oracle, either (contrary to the above argument) from within the Blockchain itself, or from a trustworthy external source that is available to all.

The question of creating, trusting and securely using an external source is beyond the scope of this paper. We will therefore concentrate on the pronken on generating unbiasable random numbers in a blockchain consisting of biassed nodes. We will discuss two approaches to this in Section 8.

7. Telling the time: the hash clock.

We take it as given that all nodes actually know the time to within a reasonably close tolerance, certainly much less than the expected inter-block time. We can also assume that whenever agent A timestamps something, she signs the combination, thus making her to some degree liable if the timestamp is demonstrably (sufficiently) wrong. What we cannot do is prevent nodes lying about timestamps unless they are suitably protected cryptographically.

A has two options for lying: she can claim it is either earlier or later than it really is.

For example, if told not to submit an entry into a hash puzzle and build a block before time T, she might, at an earlier time, put later time stamps on them. This carries the danger that a small group of miners might very quickly add a number of blocks, taking control.

On the other hand, some facility might only be available to those who bid with timestamps less than T. Now A might add an earlier timestamp to her late entry. We cannot stop A putting in an entry late that it could have entered earlier, and we are aiming to avoid our design being computationally bound. The best approach in this direction is to be able to prevent early timestamps being accepted for messages that could not have been created earlier because they depend on later information. So, we will concentrate on the problem of preventing nodes doing things early.

We do not have a single TTP, and those complaining about A's timestamp might themselves be lying, so the blockchain is not necessarily an ideal context for enforcing accuracy here. We do, however, offer an interesting option for preventing falsely late timestamps.

At any time, we have a chain of blocks and it is to be expected that most of those who created the recent ones are both present and trustworthy. What we can do is embed, in each block, a sequence of committed but unrevealed nonces. For example, we can store $\text{hash}^{100}(N)$ in the block, which effectively conceals a sequence of 100 starting at $\text{hash}^{99}(N)$ and ending with N. The crucial thing is that whenever these values are seen they can be checked against the commitment in the blockchain. This is a great deal more efficient than using conventional cryptographic signature.

We can then tell the creators of the blocks to reveal successive values of this sequence into some universally readable space (e.g. with a chosen instance of their block) linked to their observing a series of later events and some formulaic delay based on history there; or simply to the occurrence of certain times. To assert that the system has moved to a given time a node must know a defined number of these values.

So, for example we could use this mechanism to release a group of previously secret values every minute, or a minute after each block is finished, or whatever. The point is that now, to be believed, a timestamp will need to include enough of the values that should have been released shortly before that time. How many will depend on the trust model.

It is of course in a miner's interest to perform this new timing duty diligently. A penalty for not doing so, or worse releasing early, can potentially be exacted against the coins that it has involved in the block he has mined including his deposit, or simply in terms of reputation.

This provides an excellent mechanism for controlling the rate at which our blockchain grows. A block will not be trusted unless there is evidence that its creator knew sufficient values recently released at the time when it was supposed to be issued.

There is of course no need for the parties involved in this mechanism to be simply the last few block creators, though for the application above this seems perfectly sufficient. Suppose, however, that we need to be able to rely on all the hash inverses we are expecting being released. The above mechanism does penalise parties that do not do this, but we might want to make additional precautions such as selecting parties who are believed to be more reliable and less prone to communications failures, or allowing selected parties to resign from their duties in some controlled and secure way.

We can even add some redundancy by having multiple allies releasing the same series or relaying values between centres.

This mechanism of the regular release of values should be seen as a service — maybe the stiffening of the blockchain — that mining nodes have to perform as part of the service they provide for the fee they receive. We will term this mechanism the ***hash clock.***

8. Setting fair puzzles

There are many reasons why a Blockchain would benefit from a reliable source of unbiasable random numbers. This is true every time we want *the blockchain* to pick anything, as opposed to letting a participant do so. Clearly setting our hash-based lottery is one of these.

If we imagine that the random numbers are to be calculated from the present state of the blockchain, then this will have been put together over time by the nodes involved. Imagine the last node that can affect what the choice is. Our node can potentially see the contributions of the others at the time he makes his. He therefore can potentially choose a contribution which will result to his own advantage.

So, we have to be careful, particularly in the asynchronous communication environment surrounding a blockchain, where a malicious player might delay his contributions. One idea for countering this behaviour is to force the participants to commit their contributions in advance, as with the hash clock discussed above. Indeed, it makes sense to potentially use the node's entries in the hash clock as its contributions to successive puzzles.

There is, however, a further complication. The decentralised communication of blockchains means that a node's communications can get at least temporarily blocked through no fault of its own. A bad node might therefore decide not to contribute its value to the hash clock for time t is that would result in a better puzzle for it. Furthermore, this advantage could grow exponentially for a coalition of bad nodes.

We offer two solutions to these problems. The first is directed specifically at creating puzzles for WyS, the second creates a generally usable source of random numbers: a *random oracle*. Both use the idea of cryptographic delay: making the final value uncomputable by all of those contributing at the point they have to supply (but may fail) a contribution to the result. The latter only becomes visible later when it is needed.

Solution 1
To set the puzzle at block n, take the set H(n-r) of hash clock values in the earlier block n-r that has now become immutable. Let p0 be the XOR of this set. The puzzle p is some value seeded by p0 that takes a significant amount of sequential calculation so that no node can

know within the period when it can meaningfully release its hash clock value for n-r what the resulting p would be. Then a bad node must release (or not) the value blind, so p cannot be biased.

The reader should ensure that he or she clearly understands the difference between a node contributing to a random value at a time when it has no idea what effect it will have on the result, and doing the same thing at a time when its knowledge of the ultimate distribution is changed by its action. The former does not allow the node to bias the result, the latter does.

The delay itself is most likely an iterated hash. Anyone wanting to know the result of the mining competition will either have to perform this or have someone else do it for them. Sequential hashing like this looks a bit like PoW, in the sense that otherwise pointless hashing is being done to ensure fairness. Fortunately, the restriction of it to a sequential calculation means the actual amount of energy wastage will be far smaller.

There is no guarantee that all nodes will know the puzzle at the same time. What we require is

1. The competition is completely defined (I.e. the mining tokens committed) before anybody knows it. We will use a hash clock from after this commitment is closed.
2. Everyone relevant can calculate it before it is needed.

This idea for creating random numbers will only work where the choice that the Blockchain is resolving has been well established for some time.

Solution 2

In the second solution we create a series of random numbers which are available to all at essentially the same time. The source of the numbers remains the H(n), but the new protocol dispenses with the sequential computation by removing a bad node's ability to contribute or not in such a way that might lead to bias.

To do this we take advantage of the Blockchain implementation of *delay encryption* set out in our earlier paper[2] This allows any party to create *delay(x,t),* a value that reveals x to anyone at and beyond t, but which gives no information about x before that. That implementation uses asymmetric encryption and threshold cryptography plus the underlying trust model.

We remove each node's ability to bias the result by the choice of contributing its previously committed value or not. It is now obliged to place a delay encryption of what it will release at time t on the Blockchain by some deadline t'<t. By t the set of such timed commitments will be immutable, and *only those nodes who have a timed commitment in place* will count for the final calculation. If such a node does not succeed in releasing at t, the delay encryption can be opened, and all nodes will agree.

Biassing this is impossible without publicly attributable misbehaviour. If any node produces a value (either via open release or the delay) which does not agree with the original hash clock value, then it should be punished. This protocol therefore guarantees that either an

[2] *Delay and escrow in the blockchain,* available from www.tbtl.com. The latest version includes description of this unbiasable oracle.

unbiased random number is produced, or some node has introduced a value which is demonstrably wrong: a sin of commission rather than omission.

9. Countering forks: consensus

Due to distribution, malevolence, or coincidence, forks can arise in public blockchains. Multiple blocks are introduced which continue from a single one, with the branching potentially persisting for a few blocks or for ever.

The measures described in this section are additional to the hooks protocol discussed earlier, which is intended to prevent, primarily, the insertion of branches a long way away from the current head. In this section we mainly consider ensuring unique successors close to that head.

We need to prevent branches persisting so that every block which has been present for sufficient time has become immutable and part of the blockchain for ever, being an ancestor of every new block.

This is traditionally achieved by stating unambiguous rules which miners must follow when they are being honest. We will do this in such a way that a benevolent majority will defeat a malevolent minority: the details will depend on the rules for resolving the hash competitions. For example, where the right to mine a block is split into a sequence of lotteries whose jackpot rolls over if there is no winner, backing the longest chain is correct.

We have another potential vector for this arising from our invention of the hash clock. For we can ask the authors of historical blocks to choose between successors: where the release of a nonce is dependent on a later block, it will presumably only be triggered by one at each level and the way it is released may back one choice. Thus, we can get the historical authors to check emergent blocks and back one using the same criteria as the miners. However, to be secured using only the nonces, this would be subject to the same timing constraints as temporal signature.

Again, we can potentially penalise these historical authors for not performing their roles here, should we wish to.

The most powerful weapon for consensus we have comes from the nature of the game for deciding who gets to create a block. Note that

1. The amount of work to compute who has won is minimal and, because mining tokens are on the blockchain, anyone can work this out once the puzzle is known. Thus, we can replicate this computation and essentially create common knowledge about who has won, and who has not.
2. We can mandate that only a block created by the true winner at each stage is valid, and it is only accepted if it passes validity tests. Backing a bad block – a non-winner or an invalid one — is a crime.

If we can enforce these rules, then there is no possibility of plausible branching: a branch in the chain where a well-behaved node can reasonably support either branch.

A remaining issue here is what happens if the winner of a mining competition fails to provide a timely block. We can divide this into three.

1. *What do we do if the winner provides no block?* This will surely be a rare event. We have got the choice of simply skipping that block, so that the block n+1 will follow block n-1, or get the second place party to provide one instead.
2. *How do we decide if the winner has provided a valid block?* In other words, how does the blockchain decide if a block provided by the winner gets included, as opposed to exercising whatever option is chosen for the preceding question. Either this will be because the block is not valid (which may recursively be because that block made the wrong decision about its predecessors) or it was not there in time. We address the second of these below. The former must be based on objective facts that can be seen in this block and its predecessors, that all honest parties will agree on. We can choose to depend on the new block creator to decide this or allow any party (or any party with a mining deposit to lose) to post reasons for disallowing the block or any predecessor. The reason for the latter is to make it obvious if a bad node has backed a bad block.
3. *How do we decide if a block offered by the winner is timely?* We cannot be completely objective about time in the decentralised environment, but we do need an objective way of deciding the answer to this question. This must be down to some sort of voting. One natural way of doing this is to delegate it to the same population of nodes implementing the hash clock. In the case where the inter-block time D and communication is such that the next block will normally be present in, say, D/3, we can get the voters to say whether it was present as part of their vote (e.g. by releasing one of two values). If we place a numeric requirement on how many votes are required, this becomes completely objective. We do not see it as essential that blocks are created before the winner of the next competition is decided, and in some cases a more relaxed regime will be required. However, there will still need to be some objective measure of timeliness, and in no case can we allow a logically later block to become fixed while an earlier one has not been.

10. Payment and reward

We are making miners pay for the right to mine by converting coins into mining tokens. The value of these is eroded every time they are used. There are many things that need to be decided about this.

a. How is the price of mining determined? The default value of this is to approximate (in terms of mining power, meaning likelihood of winning) the cost in a PoW system being simulated. Note that here mining is being paid for in coins rather than off chain in terms of power and equipment. Thus, the cost of mining automatically rises as the gains do.
b. What should deposits be?

c. What should mining rewards be? Again, the default is to be driven by a simulation. Evidently this can be a mixture of new coins and mining fees.
d. How long should coins be tied up in the mining process: how long should tokens exist before mining, and how long after are deposits and change released?
e. How much of the total asset pool is committed to mining at any time? Clearly this will be driven by the decisions above coupled with the popularity of mining. Ideally we would like most of the assets tied up (and note that having large deposits reduces the risk to nodes from mining: they are only risking a fraction of the assets they have committed to the process.)
f. Is the amount of mining power directed at a particular block visible at times which will allow miners to adjust their own stake in it? We believe it should be and that the way mining is managed might make subversion harder (e.g. by randomly spreading each payment across a number of blocks). We may choose to give miners who are prepared to do so publicly (and so risking reputation and penalty in a more severe way if they do not mine accurately) the right to bid more precisely and later.
g. What happens to the money generated by the purchase of mining rights? In the absence of transaction fees this will generally be less than the mining rewards, but with transaction fees we can be less sure. There are two obvious possibilities here: the coins can be cancelled to restrict the money supply, or the money can be used for good causes such as the support of the environment. We can also vary between these based on the state of the currency, presumably in a formulaic way. This potentially gives us a very useful inflationary or deflationary tool to help manage the stability of the currency.

We might note that by including someone else's mining commitment in block, A, who might want to mine that block herself, may be acting against her own self-interest. This will be balanced by appropriate transaction charges.

11. Crime and punishment

The fact that nodes have to pay to mine, and potentially pay a deposit too, gives us many opportunities to fine them if they demonstrably break the rules. Punishment can take the form of failure to back blocks, thereby denying the offender the benefits of mining, or depriving the miner (whether she has created a block or not) the right to claim back all of their deposit.

The former is automatic, the latter will require some extra form of transaction backed up by evidence of the crime. The said transaction can be created by anyone who is himself willing to back up his claim with resources but must only be adopted into a block if the evidence checks out. If it does the accuser may get a reward, if not he will lose the resource he put in. It follows that the evidence for any crime must be clear and unambiguous, and easy to verify, as with the following.

1. Inserting an inconsistent block, including one with bad transactions. Backing an inconsistent block, directly or indirectly.
2. Lying about some value.

3. Failing to perform some duty when otherwise present and active
4. Double mining

Note that such a transaction can refer to illegal activity on any fork of the blockchain, not just the winning one. Thus, this mechanism can exact penalties for doing bad things on branches now dead. This is highly desirable, as we do not want criminals to escape justice by hiding the evidence in a dead branch.

12.Conclusions

This paper has been a thought experiment. We have provided the details to show that our objective is attainable. There are many implementation details left to be worked out, including communication requirements, the structures of transaction pools and other common workspaces. However, we have provided some very concrete ideas, on things like avoiding branches being accepted and creating a random oracle, for making it work.

There are also many parameters to be worked out, and rules developed for the evolution of these parameters. There is plainly a need for experimentation and modelling on a large scale before this model “goes wild”. We see a role for computational game theory in the latter, and in other potentially competitive and adversarial situations in blockchains.

We hope that this model takes away the motivation for miners to concentrate and to steal other people’s compute resources. Our model also removes the near necessity to concentrate legitimate mining into massive specialist hardware. We are excited by the prospect that we can transform the economic model of PoW into an almost equivalent one that does positive environmental good.

By removing one of the main societal objections to blockchain we hope this work will contribute to the higher goal of *moving the blockchain to the mainstream.*

Acknowledgements

We are very grateful to our colleagues Cas Cremers, Jim Davies, Ivan Martinovic, Wang Lei and Peter Ryan for useful discussions and comments. Members of the OxHainan team have prototyped some of the concepts in this paper. We look forward to reporting on these experiments.