The greening of blockchain mining (Draft)

Bill Roscoe, Wang Lei and Bangdao Chen

Chieftin Lab Shenzhen and University College Oxford Blockchain Institute

1. Introduction

At the time of writing (mid 2018) 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 computes 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 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. The second objection is that it concentrates mining power in the hands of those with highly specialised equipment and cheap electricity. At the time of writing a second economic model is becoming apparent: mining by people who do not have to pay for the electricity they are using. Thus we read of scientists misusing supercomputers [1], of botnets of miners [2], and of hijacked web servers mining [3]. 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.

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 which does not require anything like so much energy usage. The core idea is to retain the hash competition 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 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.

In the final analysis this creates a flexible mining environment with a number of parameters that can tune how the overall system behaves.

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 second-preimage resistance, in other words given that we have hash(x)=y, then for all but a negligible fraction of x's 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).

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.

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. a string of bits) such that hash(hash'(B)CN) $\leq \varepsilon$ for some preset value ε (where the value hashed may be some variation on this). 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 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.

3. 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**

A possible target of a transaction will thus be a mining token which is created to an identity and which contains additional committed secrets that are used to introduce entropy later in steps associated with mining. The token can therefore contain hashes of these or a single iterated hash. These can also be introduced at the time of commitment. The token must be associated with the (perhaps anonymous) public identity of the token purchaser, who can prove ownership by signing things.

The coin value of a 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 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 know 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 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.

There are a number of choices about how to run the competition. We can invite entries by a time that all should be available for and the lowest one wins. We can set a threshold for winning which increases with time (with anyone entitled to claim a win in a round when the competition reaches it) or vary the contents of the hashes with time until one yields a winner, leaving the threshold constant.

All of these approaches, and some more than others, depend on a degree of agreement on the passage of time between the miners if there are not to be disputes about the results of competitions. We will address this later.

4. 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.

There is no such problem 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 pointless.

We can to some degree reduce A's advantage by having a secret part of each token that has previously been committed and can be revealed by the hash competition. To remove it we need to do one of two things: either add some randomisation to it post hoc that A has no control over or have A construct it to be a value she cannot predict.

The former is complicated by the assumed possibilities that blockchain nodes are unreliable and may be covertly collaborating. The latter can be achieved by carefully controlled delay: making A publish a recipe for the puzzle before she can actually calculate the literal value.

The actual calculations to determine the winner of the hash competition can be expected to be very fast, so we might schedule a block along the following lines.

A discovers she has the right to construct block B at n and posts a claim at t1, together with the transactions in her new block and a hash h of them.

This prompts the authors of K previous blocks to decommit values in them (see Section 5). If they all do then the hash of these and h becomes the puzzle and is placed in B to complete it. If not all of them do then a hash of those that are placed in B at t2 and the puzzle becomes

an iterated hash of this, designed so that A cannot know the value of it by t2. The actual value is placed in the block at time t3.

Thus, unless A and the K previous block authors are colluding, she has no control over the puzzle. This provides one solution to the problem of setting a fair puzzle. As second option is given below.

5. 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 to prevent nodes from 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 (trusted third party), 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 $hash^{100}(N)$ in the block, which effectively conceals a sequence of 100 starting at $hash^{99}(N)$ 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.

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.

Provided that all the block creators do perform, or that in some well defined way all that are expected to in some secure sense do so, and that there are sufficient such parties to give a practical guarantee that not all are corrupt, this provides the ideal way of setting the hash puzzles: these can be, for example, the hash of all the specified clock values of the previous step. Using a hash rather than XOR is important because it removes a number of potential algebra based attacks.

There is a potential advantage in not including the previous block in this hash, as it means that a miner has no advantage in backing the wrong block simply because he wins the puzzle it creates.

There is an interesting contrast between what we need for the three different roles of the released nonces. For timing properties, what we need is that a majority are released at the right time. For creating a puzzle we needs all the expected values to be released, but only one of these needs to be a properly chosen random input that is kept properly secret.

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

A second mechanism we can use for creating puzzles and delays is complex sequential computation. The assumption here is that we can place a lower bound on how long some essentially sequential computations will take. The most obvious example to use here is many-times iterated hashing. For a delay of one second today we will need to demand something

like 40 million hashes. This is not so environmentally friendly, but at least it only works a few cores hard, rather than thousands. It is a few rather than one because miners will need to check each other.

Thus we can have a node specify a puzzle string that it does not know by taking some information it can only just have learned and specifying more hashes than it can have done yet.

This in itself will impose a minimum time between block creations, and of course we can elaborate on this. We believe, however, that the hash clock is the best mechanism for regulating block formation.

Whatever protocols we design around sequential computation delays need to allow for variation in processing speed. The amount of variation might be reduced by imagining service providers who do fast hashing for subscribers.

6. 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.

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.

7. 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 a 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.

8. 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.

9. The magical qualities of PoW

Now that we have elaborately constructed our alternative model, it is as well to reflect on how well the relatively simple concept of proof of work achieves similar goals

1. **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. On the other hand we need to rely on some sort of consensus mechanism about whether it is time to allow a new block yet. Our favourite solution is the hash clock.
2. **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. In WyS the number of hashes performed is small. So if the creator of the puzzle can vary the puzzle in a way he can calculate this might have a great effect. Our most elegant solution here is to use the hash clock again.
3. **Countering forks.** PoW does this with the combination of hash power and clear rules. To dominate in WyS one must invest more in mining than all the good nodes. In this sense it is clearly a form of PoS. The fact that the amount and distribution of mining is visible in advance in WyS offers additional guards against attacks. We have an additional weapon to use in this battle, namely payments of deposits which require good behaviour. The fact that it takes several layers of covering for a block

to become immutable is also helpful here in that any sudden injection of effort by a bad group of participants does not have to be countered in the same hash competition (for block r, say) but can be countered by the good majority a block or two later. Thus any sudden commitment to mine in block r can be countered by good nodes in blocks r, r+1, r+2 say.

4. **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. The primary mechanism we have used for this is committing to mining in advance. We have also required nodes to declare their allegiance before mining a block to provide protection against the illegal activity of double mining.

5. **Allowing distribution**. PoW avoids the need for all potential miners to synchronise before the creation of each block. While WyS requires broadcasts by those who happen to be involved in the hash clock, it otherwise has the same property because it is so closely based on PoW.

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.

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. This strikes us as an interesting problem. Can one mandate software standards or otherwise ensure vigilance? *Quis custodiet custodies?*

One idea is a honey trap in which illegal material is placed to see if it gets caught. But that seems to require a privileged party to run the trap, which is not desirable.

In general terms it 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.

10. 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.

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 and Peter Ryan for useful discussions and comments.

References

[1] https://www.independent.co.uk/news/world/europe/russia-bitcoin-cryptocurrencies-sarov-supercomputer-federal-nuclear-centre-latest-a8204161.html

[2] https://www.infosecurity-magazine.com/news/cisco-cryptomining-botnets-100m/

[3] https://qz.com/1288291/new-crypto-mining-malware-is-using-amazons-cloud-to-hijack-computers/