Taking the work out of blockchain security

A W Roscoe and Wang Lei
Chieftin Lab, Shenzhen, and
University College Oxford Blockchain Research Centre

February 13, 2020

Abstract

The most convincing argument for proof of work in the mining of public blockchains is, at least in our view, that it makes the creation of a fake branch too difficult or impossible thanks to the *follow the longest chain* principle. In this paper we suggest a modification to the structure and protocol for blockchains that provides an alternate possibility for this. Specifically we show how blockchains can develop links in the opposite direction to the standard hashes of the hash chain thanks to building in *hooks* that can later be used to attach further up the chain. These, together with limited use of timing, can eliminate any realistic possibility of a branch being plausible.

1 Introduction

A blockchain is a linked list of data blocks, each being within some size limits (typically up to a few Mb) built using a specified format though with no general limitation on what it can contain, with each containing a strong cryptographic hash of its predecessor. The identity of the blockchain contains the contents of "Genesis" Block 0, B_0. With that as its root, the blockchain is thus a tree rooted at B_0: it is limited to the set of compliant blocks where the hash pointers eventually lead to this root. Each blockchain is also associated with rules about how new blocks are created in a decentralised and distributed environment, which the said rules always encouraging agreement on a linear sequence of blocks starting with B_0, where all branching away from that sequence is eventually disregarded. The hash function is assumed to be preimage resistant and collision free, meaning that for the life of the blockchain it is believed that it is essentially impossible to calculate two values that hash to the same image, or a single value that

hashes to any image that has not itself been calculated with the function. Similarly, someone who knows the value $hash(x)$ but not x cannot calculate x unless they know x is drawn from a small enough range of values to try them all. By this means, if we know a series of blocks where each contains the hash of the previous one, and someone else contains a second such series, and one contains B_0 at the end, the two series are the same between any member the two have in common and their joint end at B_0. When B_n is in this sequence in the nth position, and no further branching is possible at its own or a lower level according to the protocol for building blocks, then we say that B_n is immutable. Looking at the structure from the outside or from the perspective of a single node, the immutable part can only grow. The immutable sequence seen by a node may either equal or be a prefix of the immutable sequence we see from the outside.

One of the core assumptions about blockchains is that they can contain a limited number of bad nodes who may or may not follow the protocol. Depending on the nature of the protocol it may be possible for such a node to post a block at any time whose hash field is any chosen block B and therefore a contender for B's successor, even though the latter may already have a successor and the new one has no hope of being accepted. When we say that a block is immutable in such cases, we more precisely mean that no block added by anyone at this or an earlier place would be believed by any trustworthy node from the community running the blockchain according to its rules.

The natural structure supported by hash links is not a chain but a tree rooted at B_0. Every block in such a structure has an unambiguous number of steps from B_0, and unless itself B_0 has a unique predecessor that is one step closer to B_0. The protocols creating blockchains are designed to ensure that, except perhaps for a little temporary confusion about young blocks, the substantive blockchain is a chain with no branching.

Though established mining protocols may create consensus and a growing immutable chain amongst an active community of users, some may be fallible when it comes to unambiguously identifying the present head of the chain to a potential new user. How does such a user know she is not being presented with a fallacious branch rather than the official chain? That is the problem this paper addresses.

In this paper we first set out our basic assumptions about blockchains (Section 2) and discuss the nature of forks that attackers might attempt in Section 3. In Section 4 we observe an important dichotomy in defence strategies, while Section 5 shows how to introduce *hooks* into a blockchain, our core idea that we assert can remove the need for PoW to eliminate

branching as well as improving basic security. These hooks are placed when a block is laid down, and later attached further down the chain. In order to give a strategy that covers all angles, we examine in Section 6 how to cover the possibility that a node is presented with a candidate that only forked recently from the official chain. In Section 7 we analyse an attacker's options for defeating this extra security and thereby derive parameters that the blockchain and mining community must satisfy to prevent such attacks.

We believe that since the focus of a blockchain's active participants is around the head where new blocks are appearing, we need to develop ways of making the older parts immune from attack. Hooks make a substantial contribution to that security. The structures set out herein can improve the security of any blockchain, but will be particularly applicable in public blockchains where miner selection is not based on Proof of Work (PoW) [**?**], but rather by some mechanism that makes a random selection of the next miner from a large pool of able and willing parties, as for example is provided by Proof of Stake (PoS) [**?**] and Work Your Stake (WyS) [**?**].

2 Blockchain assumptions

For us, a blockchain is a series of blocks each of which contains the strong cryptographic hash of its predecessor, and where each of the blocks contains a time-stamp which is strictly greater than that of the previous block and which is closely related to the real time at which the block was posted.

For blocks posted at and around the true head of the blockchain, we expect this to be managed by the general consensus and approval process. For public blockchains, which is the class we mainly consider, at any time there will be a community of interested parties, potentially miners and non-miners, who monitor it and provide consensus. It is assumed that incentive structures, which may be based on reward or reputation or both, motivate most of the mining activity to be according to the blockchain's rules.

A blockchain may be associated with some state that is stored relevant to the construction of new blocks or other information relevant to the regions around its head, such as the transaction pool of information that has been proposed for insertion into new blocks, assertions made by nodes about recent blocks, and information about the identities of participating nodes. We want the long-term security of the chain (as opposed to the identities using it) to depend only on structures within the blockchain itself.

In this paper we assume that miners are chosen by a stochastic process from the willing in such a way that in any reasonably long run of blocks, it

is practically certain that the majority were created by good miners, namely ones that follow the rules. Unfortunately we do not think it is realistic to assume that all the miners – particularly the *bad miners* – act independently of each other.

We expect there to be upper and lower bounds on the mining rate, namely the number of blocks produced over intervals of time. In particular it should not be plausible to have an extended time interval in which many fewer or many more than than the intended number of blocks appear[1]. We will make use of this assumption, in preventing accepance of one variety of forking. Many readers will be familiar with the fact that the parameters of the Bitcoin chain are adjusted regularly so that the rate of block production is controlled statistically. In this regard we expect good nodes to have a reasonably accurate notion of time, such as secure internal clocks, that they can use to judge time related issues. These can of course be supplemented with secure external sources of time information such as GPS when available.

The best-known way of managing the mining process is to allocate mining rights using PoW, which has many positive qualities as well as the well-known disadvantage of wasting energy. These include managing the block creation rate, generating reasonably trustworthy random numbers, and avoiding forking or branching.

In our view the strongest argument for PoW as opposed to other public mining protocols is its resistance to branching. The assumption supporting this is that good miners outweigh bad ones in sheer mining power and actually choose to use their power for this purpose, motivated by rewards. Working together they can therefore create a longer chain satisfying the conditions of PoW than the bad ones, meaning that if a potential user is aware of both the genuine chain and a fraudulent one created as a branch, he or she will always almost pick the genuine one, with the advantage growing as time passes since the fork.

We are not aware of any watertight means of supporting this type of security with other mining protocols such as PoS, where potentially a bad miner could create a fork many blocks before the present head, possibly helped by some weakening in the cryptography used at the time that part of the real chain was created, of using some other strategy such as the one we develop in the next Section. He could then extend the branch to the present day with many false blocks (incorporating false time-stamps) when

[1] Alternatively we could imagine that if there were, exceptionally, a long time gap, additional security measures will be placed in the block after the gap, which could include approvals by many participants according to some rules.

he did not have the actual speed limit that PoW imposes, as opposed to a rule-imposed one.

In creating a blockchain, we are free to lay down rules for how blocks are formatted and what the fields of a block are intended to be used for. We can set out rules for what conditions must hold for something to be accepted into the chain, whether it is a piece or data, a transaction or a block. Essentially what we are doing here is proposing extra data to be included in blocks and rules for what is expected of miners and under what conditions they receive mining rewards.

The main contribution of this paper is showing how the judgement of the population of block creators in the run up to a given one can contribute sufficient backing to the true chain to allow those to joining nodes to distinguish between it and competitors indefinitely.

3 Forks and branching

In a public blockchain anyone can propose an extra node that points at some block other than the last one. Where this is near the proper head in steps through the tree, this can be part of the normal mining protocol and it is possibly proposed as an alternative, hoping to win a legitimate place in the chain in place of others. We do not regard this as a *fork*, which (when not officially sanctioned) means an attempt to create an alternative and presumable bogus history as a plausible alternative to the official one that already exists. We expect the process of selecting the correct branch to follow and getting consensus on it to be the business of the mining protocol. The details of the mining protocol are not the topic of this paper, but we term the growing chain of blocks it produces the *official* chain. Our business is to find ways of preventing anyone from being deceived into thinking that a block that is not part of the official chain is one.

What we need to do is avoid forks of the sorts illustrated in Figure 1, where the branch may occur well behind the current mining face. An attacker might create a new block that points directly at a block in the official chain, as with the forks leading to F_{long} and F_{short} in the figure. Or it might use a pre-existing block that was discarded in the mining process, as in the fork leading to F_{imb}. In either case the attacker can make use of identities and computing under his control to build a chain beyond this, containing whatever he likes within his power to create it.

In the figure, real time, as approximated by time stamps, increases from left to right, and hash links point from right to left where shown.

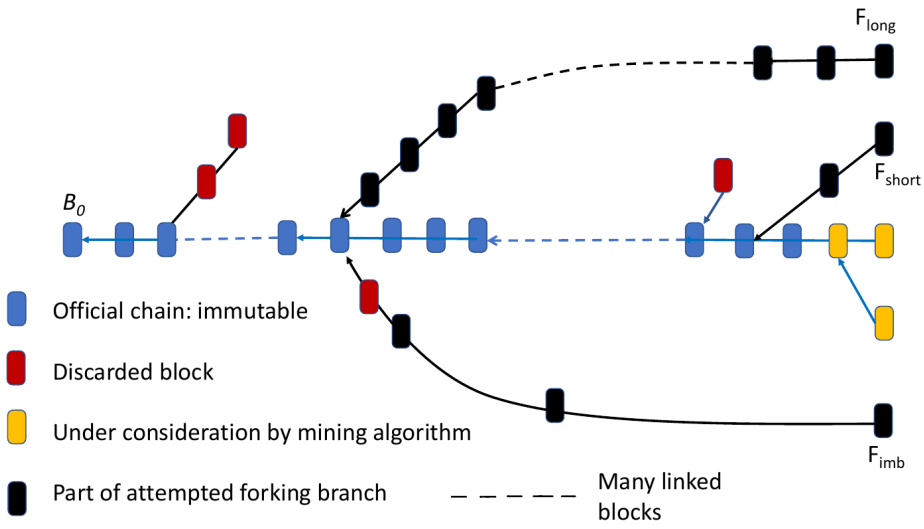

Figure 1: Types of fork

The mining protocol may or may not give rational and locally checkable reasons why it was correct to follow the official branch rather than the one leading to F_{long}, F_{imb} or F_{short}, or it may not be so explainable. Even if it is, we might fear that as time progresses and cryptographic primitives used at the time that the root of the respective fork was created. In that case it might be possible for an attacker to create block(s) with forged time-stamps much earlier than they were actually built, which are plausible successors to blocks already present.

An interesting alternate attack strategy is that our attacker C might legitimately gain the right to propose a block B_n: the prior blocks may thus confirm his right to do so. But he does not use it, and the chain selects a reserve party to create the official block B_n. Much later, C might create an alternate history starting from his own version of B_n: under many assumptions this will be possible and not require any broken ciphers or hashes. PoW beats this attack (which there would be an attacker finding a solution to the hash puzzle posed by B_{n-1} and not using it till much later), as will our own approach.

In the methods developed in this paper, we do not discriminate between the cases of a fork being built on the main chain or on a discarded node, noting that depending on circumstances either might give the attacker an

advantage. What we do distinguish are long forks, as shown here to F_{long}, and short ones as shown to F_{short} and F_{imb}. Note that in F_{imb} there is a huge imbalance between the lengths of the two arms of the fork, with *imb* standing for *imbalanced*.[2]

We concentrate mainly on making long forks unbelievable, but will also address short ones.

4 Heads up, heads down

Though the longest chain model under PoW gives a powerful security argument, a little thought reveals that it only works when whoever is presented with another candidate is aware of the existence of the head of the real chain as a second candidate and can analyse it. In other words if Alice, who is trying to decide whether a proposed head is genuine or not, has not kept her "head up" to see what is going on, she may not be aware of longer chains that beat the one she is testing.

Of course this may not be because of laziness on her part: because of the vagaries of decentralised systems she may be temporarily disconnected from the part of store where the head of the official chain sits. There is no such uncertainty in checking that a putative block is a descendant of the root block B_0: simply follow pointers back. If in the latter process some data is temporarily unreadable, there is no ambiguity that it is still the correct data and has to be read.

This last process is "heads-down", in the sense that the algorithm simply follows a path it is led down without any searching for "better" data. In a world of poor communication, either sort may be blocked from proceeding, but a heads-down process will know it is blocked whereas a heads-up one may give the wrong answer. We have no doubt that heads-down security checks are preferable to heads-up ones, since they should both be more efficient and more certain.

Of course in avoiding all sorts of forks, it may or may not be possible to avoid heads-up checks completely. We will see later that heads-up is still a valuable back-up to our proposal for a structure that makes it less necessary.

[2] In this context, to be imbalanced, there should be a significant difference in the two legs: in fact one of them should break the rules on how fast blocks may be generated on the assumption that the block at the head of each is reasonably new.

5 Hooks

In this section we propose a model that allows us to move the responsibility for eliminating forking, 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 the need for the main argument 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 beyond, starting at a block B_0 that contains initial conditions and rules, linked by the fact that the $n+1$th 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. It must never be forgotten that thanks to the properties of hashing, any block determines a unique series of predecessors, and that any assurance that block B_n is genuine member of the chain automatically ensures that each of these predecessors is too.

The main purpose of the present exercise is to show how to ensure that someone who only knows the value of B_0 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 B_0: that means the value of it, not any specific memory location.
- The creator of B_0 is assumed to be completely trustworthy at the time B_0 was created and for a suitable period afterwards.
- At any time there will be a collection *Active* 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 both responsive and supporting rather than undermining the protocols. So they are *good* rather than *bad*.

- The nodes in *Active* all have good knowledge of what is happening at and around the head of the chain. At least the good nodes in *Active* will, unless they have very recently become active, be aware of meaningful activity at and around the head.

- As blocks become immutable, the nodes of *Active* become aware of this. Eventually the fact that a block is immutable will be common knowledge: all of *Active* and any users who are simply observing know it is immutable and that all understand they have common knowledge of this.

- Nodes can leave and join *Active*. 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 to identify these.

- We anticipate that once a node has successfully identified the head of the blockchain, it will become familiar with the activities around it and be able to follow the development of the blockchain accurately. It will follow the development of the official chain as long as it remains *Active*.

The basic structure of a blockchain, namely blocks in which each contains the hash of its predecessor, with the ultimate predecessor being the start block B_0 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 *Active* as a different subject from keeping that consensus visible to all for nodes outside *Active*.

Imagine the following scenario: block B_n was established some time ago, and B_{n+1} shortly after 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 B_n, namely B'_{n+1}, and as many further successors B'_m $(m > n+1)$ to this as it likes. Such a series of nodes is almost certainly associated with a community of identities controlled by the attacker. We would like the blocks to contain structure that would allow the joining node C to recognise such a branch even though he has never seen any of the B_m for $m > n$.

Presented with one of the B'_m as being at the end of the chain, with presumably all the agents apparently creating these fake blocks under the control of the attacker, how can C 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 manufacture upward pointing links (which we term *hooks* because of the way they work, into the blockchain in addition to the traditional downward pointing ones. They have essentially the qualities of links pointing in the other directions from the usual ones in a blockchain. Since we cannot modify existing blocks these have to be placed subtly. This is done by showing they have been created by the same parties as the preceding block and not requiring any other information about the parties or from outside the blockchain. These are individually not so secure since they depend to a greater extent on the trustworthiness of the said nodes, but we use them with considerable redundancy so that the presence of a proportion of bad miners amongst successful miners does not destroy security. How large a proportion will be the topic of later sections.

The proposal is as follow: a number s is chosen so that it is believed certain that of any s consecutive blocks of the official chain, a reasonable number will have been mined by trustworthy and honest nodes. We will examine what "reasonable" means here later. On the assumption that the mining protocol picks the miners of blocks independently and at random in such a way that each is good with probability at least p and bad with probability at most $q = 1 - p$ (q thus being an assumed upper bound on the proportion of mining power as time progresses[3]), the number of good miners in any s is binomially distributed, so it is never completely *certain* that $k > 0$ of any s will be good, but we can get close. For example if s is 100, $p = q = 0.5$ and "reasonable" means *at least 10* then the standard deviation of the distribution is 5 and the probability that there are not a reasonable number is less than 10^{-15}. If $p = 0.75$ so $q = 0.25$ and $s = 160$ and "reasonable" means at least 81 (i.e. more than half) then the standard deviation σ is $\sqrt{(}480/16) = \sqrt{3}0$ which is close to 5.5 and the probability

[3]In general this calculation and subsequent ones in this paper based on assumed proportions of good and bad miners may need to be adjusted for two factors. The first is that some good nodes may choose not to mine, if this is permitted. Of course we could make mining compulsory or highly incentivised. In either case we would need to ensure that, unlike PoW where specialised hardware is in reality required, it is accessible. The second is that the selection will in all likelihood be biased by ownership of some resource such as stake or equipment.

only slightly greater. (Both require exceptions of about 8σ.)

In our hooking protocol:

- The creator A_n of each block B_n is obliged to include a fresh public key PK_n in it, of which only she knows how to create the corresponding signatures. (Note that though the B_n are necessarily different from each other, the A_n need not be.) She is obliged to use that key only once for anything that could be construed a hooks, and only over a highly constrained time. In fact we will normally assume she uses it exactly once in total. Using it outside these restrictions, or not at all when it should, attracts a 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 (e.g., using the Winternitz scheme [**?**] or a variant) because of their intrinsic quantum resistance. The choice will be based on the trade off between the sum of the sizes of the public key and signature (each of which appears once in the blockchain, a replicated structure) and the creation (once per signature) and checking (probably many times) per signature. For this particular purpose, structures such as Merkle trees that allow multiple use of hash based keys are probably redundant. At the time the key is placed in the block we think of it as a hook dangling from the chain which still has to be placed higher up.

- A_n is obliged to stay in *Active* and watch the development of the chain until she sees that block B_{n+s} is immutable.

- At that point she signs B_{n+s} using PK_n and makes the signature under available for inclusion in the blockchain, for example as a transaction, and watches to make sure it is included. (Replication of the signature over multiple blocks is allowed, but not needed.)

- Before such a signature is included in block m for $m > n + s$ it is checked, so only accurate signatures make it in: a block with an inaccurate signature will be rejected as invalid.

- The creators of block B_n and B_m thus cooperate in taking the hook that is dangling from B_n and attaching it to B_{n+s}.

- Once that signature has been registered and the hook completed, the role of the agent A_n creating it disappears: A_n can forget the secret key, and it would be better if she did since then if A_n is later corrupted the key cannot be compromised. It is best not to think that A_n has

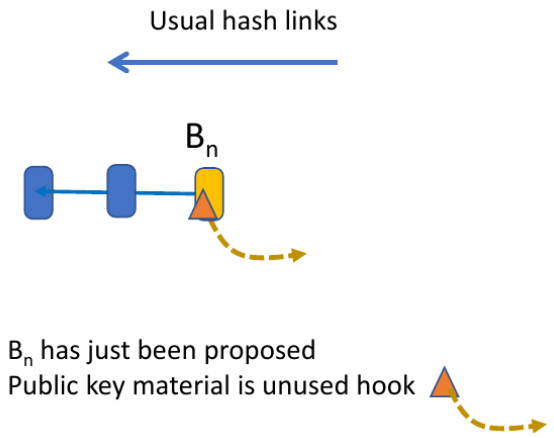

Figure 2: The root of the hook anchored in B_n.

signed B_{n+s} but rather that B_n has: the hook exists solely within the blockchain, just like regular links do.

The development of a single hook is illustrated in the following figures. In practice the gaps between the blocks involved would probably be rather larger than is illustrated here. Figure 2, shows the initial state of the hook, namely the public key material that has been placed in B_n. Naturally this block itself does not change, but the key material forms the basis of the hook that A_n will later make available about B_{n+s} for A_r to place in B_r. Figure 3 shows the state where B_{n+s} has been placed in the chain, but is not yet settled enough to be signed.

Figure 4 shows the signature created but waiting to be placed. Note that the two blocks following B_{n+s} were probably placed before the signature was available to include in them. The signature that represents the hook is proposed by one node (A_n) and checked several times: the first necessary one is the agent A_r who includes it on B_r. Finally, in Figure 5 the hook has been adopted into block B_r, which can be expected to become immutable before too long. The acceptance criteria for B_r will include ones about the hook: it must be well-formed and be a correct signature using B_n's key material of B_{n+s}, which is a predecessor of B_r: thus this is a further round of checking of the hook.

Figure 5 shows that the hook from B_n to B_{n+s} has components which sit in B_n and B_r, but none in B_{n+s}.

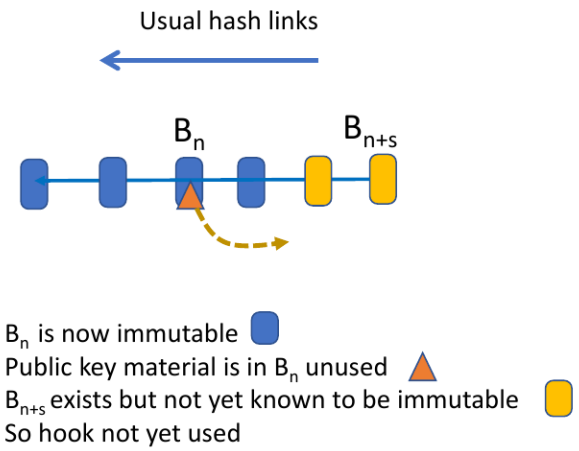

Figure 3: A_n can now see B_{n+s}, but not yet sign it.

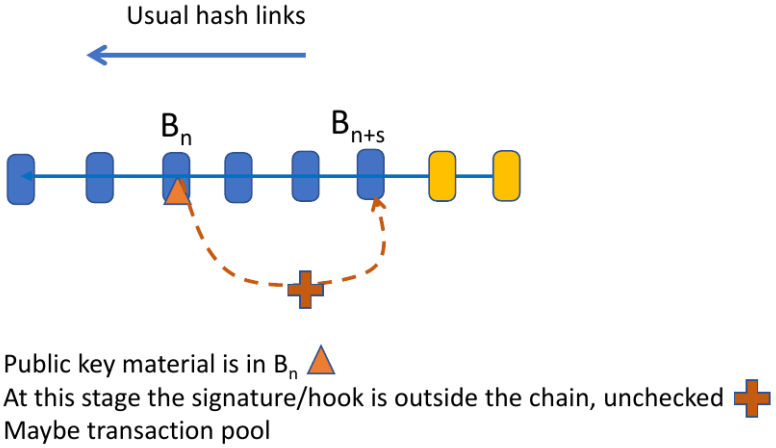

Figure 4: B_{n+s} has been signed, but the hook has not yet been adopted into the chain.

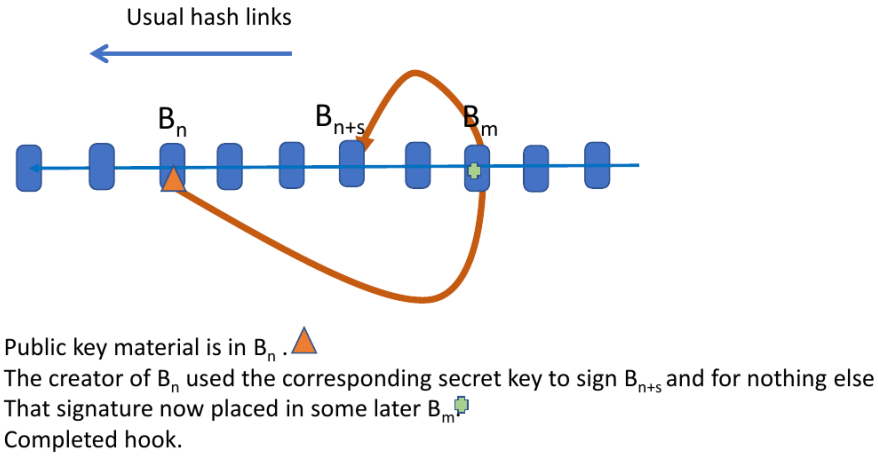

Figure 5: The hook is now in block B_r. the chain.

This process can be seen evolving along the chain in Figure 6, where the further right one goes the less evolved the hooks are, since the blocks are younger. Note that the first hook is into B_s. Note also that the assumption that the creator of B_0 is trustworthy implies that any block history with a B'_s that is not the one certified in B_0's hook must be fraudulent.

For simplicity this picture assumes that r is a constant greater than $n + s$ for all hooks, and so each block contains only one. This makes the figure more understandable. Note also that we have coded the status of each hook in colour and possible dashes and dots. In future figures we will omit the explicit role of B_r and have the hook involving only B_n and B_{n+s}, but maintain the colour and line-type coding. Figure 7 shows exactly the same state as Figure 6.

It is easy to incentivise both the creation of hooks and their inclusion in later blocks: all or part of the mining fee for the block in which the key was registered can depend upon the signature, and attractive mining fees can be associated with including the signature in a later block. Where, as proposed in [**?**], 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.

We can thus rely on good miners (a) generating hooks and (b) including

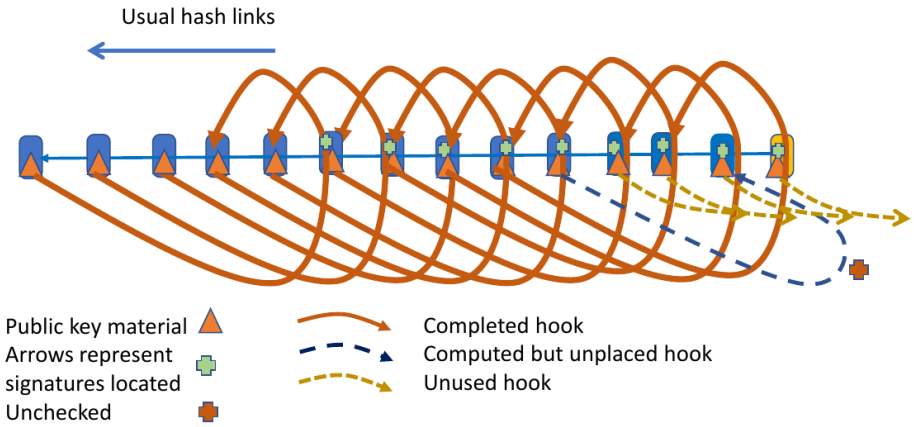

Figure 6: Chain showing multiple hooks at different stages.

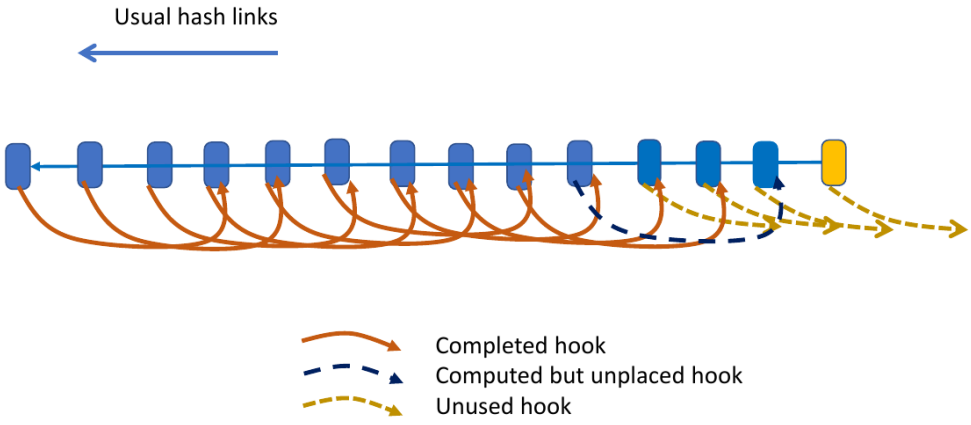

Figure 7: Chain of Figure 6 in simplified notation.

available ones in blocks unless they are out of action or have their communication blocked. We can rely on them not re-using their PK_ns. The failure of bad miners to generate hooks or failure to include others' hook signatures in the blocks they build will advertise their bad status, which we would not generally expect unless an attack with a good chance of success is under way. Note also that if a hook is available for inclusion in a block, and a (bad) miner fails to include it at block m, then in the next few there will probably be a good miner who will include it.

The fact that bad miners are highly incentivised, in the normal run of events, to generate good hooks, can play a positive role in the protocol for detecting forks.

Initialisation

We can say that the hook from B_n to B_{n+s} covers each node B_j between these two, and implies that B_n backs the fact that B_m is followed in the chain by B_{m+1} whenever $n \leq m$ and $m + 1 \leq B_{n+s}$, as B_{n+s} is certainly preceded by the sequence from B_n to B_{n+s-1}.

The observant reader will have noted that while there ought to be s distinct hooks covering every consecutive pair of blocks in the central portion of the blockchain, this is not true at the beginning and end of the chain because this would have to stretch s blocks to both left and right for this to be true. Protecting links near the end of the chain is covered in Section 6 as they give rise to short forks.

We should also pay some attention to the initial portion of the chain. Until s blocks have been established there are no completed hooks at all, but there can be no long forks.

Working in our favour is the fact that the creator of B_0 is completely trusted. Thus the fact that the first hook is created by the creator or B_0 seems to give it complete authority and, once this hook is present, no possibility of a fork succeeding in the first s blocks: we remarked above that to be believed every possible version of the chain must include the unique B_s certified by B_0. Until the hook from B_0 is present, all block creators remain in *Active*.

Testing a block

The effect of hooks on a fork is shown in Figure 8. If the signature protocol has been complied with throughout a blockchain's evolution in the sense that every good node creating a block has introduced a hook to block B_{n+s},

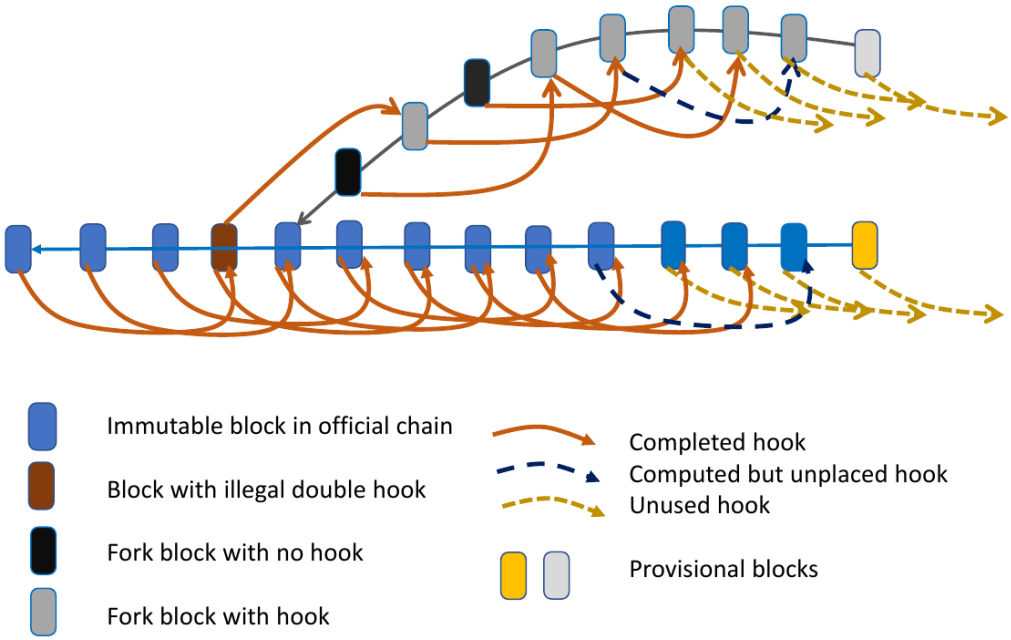

Figure 8: A fork, assuming complete hooks in official chain

then a tester presented with a supposed block B of it can follow the following algorithm:

- The hash pointers are followed back to their root. If this is not B_0, then B is not part of its blockchain.

- If B is at level n we will call it B_n, and its level-m predecessor ($m < n$) B_m. If B_n is not part of the official chain then this is true for some first B_m amongst its predecessors, or B_n is itself the first such not to be in the official chain. We will assume that the tester carries out a search for such an m starting from s, which it tests using the known-to-be-trustworthy attestation of B_s by B_0: if this checks out then all blocks up to B_s are official. If not then B_s and hence all B_m for $m > s$ are not. So below we assume $m > s$.

- Then for some known constant r, if $m \leq n - r$, there should be a hook from B_{m-s} to B_m to be found in a block between B_{m+1} and B_n: namely a signature of B_m checkable using the public key found in B_{m-s}. If $m \leq n - r - s$ there should be hooks from blocks $B_{m-s}, \ldots, B_{m-1}$ to $B_m, \ldots, B_{m+s-1}$, each of which represents a separate attestation to B_m by an earlier block. The checker should count how many of these are actually present. If they all are, then B_m can be assumed to be official, if none are then it can be assumed to be bogus.

  In effect Figure 8 shows that if the official chain has a complete set of hooks, and we assume that any link without all hooks is suspect, then this gives perfect discrimination unless there are s consecutive bad miners: ones who might double sign. In the illustrated example $s = 3$ (much smaller than in practice) and one of the s nodes required to certify the fork has done so, but not the other two, leaving two uncertified blocks in positions where we would expect them to have been certified.

  If some are present and some are not and we are not sure the complete-hook assumption is valid, then we will need a protocol for deciding the answer based on the circumstances of the blockchain and possibly additional tests. We will discuss this topic in later sections.

- If $m > n - r - s$ then there may not be enough hooks in the chain to determine the above. It may be possible to find some in the place that hooks are put before being transferred to the chain, e.g. the transaction pool, but still there may not be enough. If so then m is

within about s of the claimed current block and so if it is the first branching block then the fork is definitely short, and we have to guard against the cases F_{short} and F_{imb} from Figure 1. We will deal with this case in Section 6.

6 Rejecting shorter branches

Note firstly that if a node A is presented with a block B that is claimed to belong to the blockchain, she has no idea whether it is or not, and if it is not but does ultimately lead to B_0, she has no idea where the inevitable fork is. It follows that as well as using main hooking protocol to eliminate long forks, she also needs to eliminate the possibility that there are shorter ones.

Suppose we have a fork that is not many blocks behind the one that an attacker is falsely proposing as the head of the chain. As the length of this gap grows, the protection from hooks improves. It follows that one strategy that someone judging such a block might adopt is simply to wait until the hook-based protection will be strong enough because B itself will have developed multiple successors in a process that the checker can follow. The checker can also search for hooks not only in these blocks but also in the transaction pool.

We can include in the protocol to expect nodes that have not yet provided their hook to provide a signed copy of what they each believe is the end of the chain, which are accessible in some place like the transaction pool, but do no have to be stored in the chain or be long-lasting signatures.

Depending on the nature of the public key stored for the hooks, as it would have to be capable of supporting multiple signatures, such intermediate evidence might be signed with that. In that case these would have to be explicitly distinguishable from hooks. Or some other key could be used.

With such additional places to look where recent block creators can place such information, if this is included in the protocol we can simply aggregate the available hooks with the opinions of a given B_m's predecessors that do not yet have one and apply the same criteria we would if all relevant hooks were present.

We note that the case F_{imb} can be eliminated by refusing to accept candidate blocks whose path to B_0 fails block-creation requirements. Thus the attacker will not be successful if they present a short fork from an old block.

A node that is presented with a claimed head of the chain might thus seek to confirm it is not the result of a recent fork by some combination

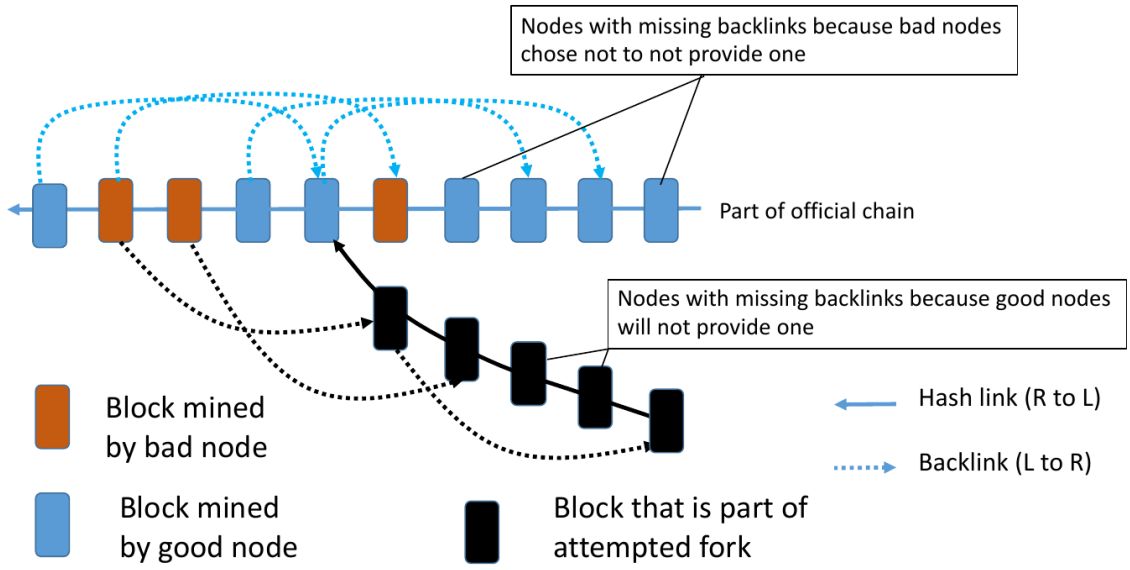

Figure 9: Possible behaviour of bad nodes *versus* hooks

of the above: looking at the existing links, hooks and approvals, examining the time-stamps on apparently recent nodes and perhaps examining the consensus process. And perhaps waiting for the chain to develop further.

For the rest of this paper we will analyse the game between good and bad miners that hooks inspire.

7 Attacking hooks

If the hooking protocol is complied with, at least as far as what is visible and checkable in the chain, then it provides watertight security against long forks. But what if it is not: what if bad miners, having successfully had their blocks adopted and become immutable, choose not to provide the signatures for inclusion further down the chain? What if they sign alternative blocks, as well as or instead of the ones they were meant to? What can we do about this? How much of this activity can we tolerate? In this section we answer the second of these questions.

Some of the concepts discussed here are illustrated in Figure 9, which shows that (made more plausible by the small range s used in it) control of

sufficient mined blocks by bad nodes can make the distinction between good blocks and forks difficult.

We assume that the fork detection protocol is modified so that it only accepts a chain as genuine if, in each s consecutive blocks, at least L of them have the expected signatures. Note that in the figure, bad node A has provided a double hook, namely one each for the official and fork chain. On the other hand bad node B has only provided one for the fork.

Two things can go wrong with the use of hooks thanks to such an attack:

- A false positive: the protocol certifies a fork as the genuine chain when it is not.
- A false negative: the protocol refuses to accept the genuine chain as genuine because the attacker manages to limit the number of signatures for some run to be less than L.

How big must L be to avoid both of these? In analysing this we assume that a good node always provides a hook which is adopted into the chain. This implies that r is sufficiently large together with the incentives for those doing the adoption.

Suppose L was strictly greater than G the minimum number of good miners we can conceive in any s consecutive blocks. Then if no bad miner ever provides a signature but otherwise behaves in exemplary fashion (in particular never introducing a fork), it is clearly possible that there might be false negatives.

Next, suppose that L was less than or equal to $s-G$ the maximum number of bad miners in in any s. Again suppose that no bad miner ever provides a signature, and that in the evolution of the blockchain, the s blocks up to and including B_n have at least L bad miners. The attackers can then conspire to create a successful fork by introducing their chosen $B'_{n+1}, \ldots, B'_{n+s}$ and have the bad miners duly sign the blocks they are "meant" to amongst this sequence; including these signatures in later B'_k.

From the two paragraphs above we can conclude that to be free of these two attacks we need $L > s - G$ and $L \leq G$. We can only find L satisfying both these inequalities if $G > s/2$, which is hardly surprising.

There are important points that need to made about the potential game that bad miners might play against the blockchain. Some are helpful to the attackers and some to those trying to keep it secure.

A On the assumptions that bad miners know each other and collaborate, they have luxury of knowing how many of them have seen selected for

a given run of s blocks before any of them have to commit themselves by not providing a signature to form a hook. Thus they know they have the numbers to attack the protocol before they arouse suspicions, lose fees or deposits for failing to supply signatures.

B It is important to remember that in creating a fork, we can assume that all blocks beyond the fork are built by bad miners, and on the assumption that these can lie about time, at any time. So there is no inspection or monitoring of these blocks by good nodes until the branch is presented them to try to gain acceptance.

This creates a challenge and quandary for attackers, whom we must assume are well aware of the progress of the official chain as well as trying to gain opportunities to attack. It is this: how do they behave when contributing to the official chain, other than presumably trying to get as many mining rights as they can. Do they behave like good nodes: this has the advantage of getting them their expected income from mining in the official chain and not drawing attention to their existence? Or do they withhold some of the hooks they are expected to provide? The argument for doing the latter is that if they do not, it will be easy for good nodes to suspect subversion and potential forking whenever they see missing hooks.

C In a blockchain with a technology such as KYC that can reveal the identities of participants who fail to follow protocol, which can exact financial penalties on non-performers, or both, it will be unattractive for anyone to carry out an attack which may be discovered, is blameable on them, and which might fail. This would seem to push bad miners in the direction of not showing their hands in the official chain. In other words it suggests they should supply hooks when required to do so in the official chain.

Clearly no improvement to the protocol is required if we can choose s large enough that the consequent L (which is assessed via probability calculations based on the assumed proportion of bad miners) is greater than $s/2$. Of course choosing s large increases the attention span required of miners and means that the members have to wait longer for the security that hooks bring to the blockchain. Interestingly it does not require more signatures: still one per mined block.

8 Prevention or cure?

Observations A and B above make it hard to prevent an attack on any region of the blockchain where sufficient bad miners are chosen. Even if these were picked in some stochastic way rather than the very predictable schedule we have chosen, there is nothing to stop an intruder noticing post hoc that he and his allies have got lucky.

Thus it seems hard to prevent an attack other than by a combination of the following

- Choosing the parameter L so that the likelihood of the attacker ever being able to conduct an attack (corresponding to a false positive) is infinitesimal.
- Ensuring that good nodes essentially always contribute their hooks.
- If this leaves a residual chance of a false negative, ensure that this is tolerable, most likely have additional and perhaps more expensive, possibly heads-up tests that can be used in such circumstances to distinguish between false negatives and attacks.
- An obvious heads-up defence in this case is a direct analogue of the PoW protocol: pick the block that has the most complete record of hooks, which possibly means the one that maximises the minimum number of hooked nodes in any s.

Provided these guidelines are followed, the only result of an attack is disruption and/or extra cost to the maintenance of the blockchain, rather than a direct fraud. Such an attack will also cost the attacker in (a) mining fees and (b) reputation once the chain has recovered. In pure game-theoretic terms, one should aim to set parameters such that it costs the attacker more to mount the attack than it costs the good nodes to recover from it.

We must therefore ask what a node can do to reassure themselves or otherwise about a chain when it is not sure of its correctness. It will have a fairly small range of blocks where it fears a fork may have occurred. This means that it can compare the constitution of the chain before and after this point. If this is the official chain then there is unlikely to be a great difference before and after. If it is a fake then all the blocks after the fork will be constructed by bad miners, who obviously will want to make it look as genuine as possible.

The possibilities for making such a discrimination include

- Where mining is public, a discontinuity in the population of miners.
- If we make each transition back an immutable recent block via signature of its hash, a discontinuity in the population of transactors as the attacker will be unable to copy transactions successfully.
- Having a bulletin board where suspicious blocks are posted inviting participants to post evidence including double signings.
- Performing a heads-up search for alternative histories with better records as discussed above.

In general we might well take missing hooks above a very conservative threshold as an invitation to do a heads up check to search for alternative histories of the blockchain. If one is found, a search for blocks that sign more than one hook may be extremely informative about who is not trustworthy. The knowledge that this happens will also make it more difficult for bad nodes to choose a sensible strategy: by not providing hooks during the development of the official chain, bad nodes will draw attention to themselves and lose incentives. But by providing these hooks, any attempt to create a believable fork will involve them in double signing, which also draws attention (in response to heads-up search) and presumably major penalties.

8.1 Using transactions for assurance

If C is an attacker trying to create a fork, he has to build a block for the fork, and subsequent blocks to follow it. We already noted that if the nature of the new blocks was significantly different from the ones immediately before the fork, this will give anyone testing the chain reasons for suspicion. A natural tactic for C to use is to copy transactions from official chain blocks at the same level as the rival one he is creating, so that the fake blocks are not grossly different from what the tester is expecting to see. Such a transaction may or may not refer to things that have happened earlier in the chain, such as previous transactions. If so C will need to make sure this still provides a consistent picture.

We can make this essentially impossible by requiring that each transaction, which as a whole must be signed by the node that proposes it, incorporates the hash of the most recent block at the point it is proposed. It is then made a consistency condition of the blockchain that the block-hash in each transaction in block B is either of the block that B links to or a predecessor of that.

Anyone checking the series of blocks leading to a given proposed B will then test the compliance of the series to the above rule, either in general or when there is any doubt arising from missing hooks.

The concept of including block hashes in transactions is also useful for ensuring the intention or regulation of a given transaction is not circumvented. The current state – represented by the most recent block – may well be essential in the decision process to execute a given transaction or ensure it is legal. After all the fact that the transaction was carried out may well have been conditional on the state of the blockchain, as in all likelihood is the fact that it satisfies regulations. A transaction that satisfies a regulation in one state established by one block may well not satisfy it in the state established by another.

If A and B operate a joint bank account, they may agree to keep the balance at least £500, and the bank may well not allow them to have a negative balance. If they both see that the balance is £1600, each of them might want to buy (different) things for £1000, seeing this is within their own limits and the rules. If they both launch their own transactions, they are both legal in the current state. However if a block with one of them is added, the other is not, even though it may have previously have been. This simply illustrates that for both personal and global rules, a check for validity is unlikely to be accurate unless the label that asserts it includes a commitment to the state where it was checked. This just emphasises the similarity between checking general rules and the no-double-spending property that is so associated with blockchains.

From a security point of view, this incorporation of prior blocks in transactions prevents a strategy for disguising attacks whose effectiveness will depend on circumstances. Since this idea depends on the regular signature mechanisms of the users, contingent on the availability of their own identity records and potentially weak in the long term, neither does it have the structural advantages of hooks. So from that standpoint we see it as more of a back up to hooks rather than assurance against forking in its own right.

9 Conclusions

In this paper we have showed how an evolving blockchain can develop hooks, what are in effect signatures by earlier blocks of later ones. This is self contained because all the signature material is contained in the blockchain itself. These signatures should be as strong and permanent as the core hash links. We believe that these hooks are a hugely powerful tool in ensuring the

integrity of a blockchain, and especially a public one where such questions are normally more more troubling

We have relied on the successful miners providing this assurance because we assume they have been fairly chosen and will on the whole behave well. However thanks to the inbuilt hash links we can have each miner back a run of s consecutive blocks back to itself, giving considerable redundancy in our protocol to cope with the possible presence of some bad nodes amongst the participants.

The protection of a link in the blockchain that these hooks provide grows until that link is roughly speaking the standard hooks length back from both the head of the chain and the head of the proposed forking branch. We have seen how to cover cases that do not satisfy this.

Our principal contribution is the construction of hooks and the analysis of the situation (including assumptions) on which they provide an exact decision method for whether a given block B is on a fork or not.

In an implementation we will have to decide how conservative to be in allowing for potential attacker action. By and large we expect to ensure that it is not worth it for attackers do do anything other than tow the line, and in practice impossible for them to successfully subvert or disrupt it, given if needed the countermeasures in place to attacks.

Heads up defence will not be necessary unless a node is presented with a block with a marginal record of hooks. In that case it should be possible to construct effective tests and penalise mis-behaving agents since they will have breached the hooking protocol.

In general terms we invite others to create other ways to use hooks to defend blockchains and to develop strategies to encourage good behaviours and penalise bad in chains that use them.