On decentralized oracles for data availability

Jason Teutsch
TrueBit Estsblishment
jt@truebit.io

December 25, 2017

Abstract

Nakamoto consensus, the protocol underlying Bitcoin, has the potential to secure a new class of systems which agree on non-mathematical truths. As an example of this capability, we propose a design for a trustless, data availability oracle. This exposition reduces the problem of determining whether or not a registered datum is publicly available to the problem of constructing a network in which either almost all nodes can download a given datum, or almost none of them can.

1 Silo effect

Traditional blockchains, like Bitcoin [1] and Ethereum [4], live a lonely life. By design their miners, who constitute the "eyes and ears" of the network, don't view the outside world at all! While Oraclize [9] and Reality Keys [11] successfully pipe Internet information into Ethereum, their bridges rely on trusted authorities. These efficient, oracle methods suffice for many applications, however they also introduce central failure points in Ethereum's otherwise autonomous, reputationless system.

Unstoppable, decentralized applications, such as Livepeer [7] demand not only trustless computation oracles, like TrueBit [15], but also un-erasable data storage. Ethereum provides scarce but immutable "on-chain" storage, however prohibitive expense limits on-chain storage to only the most laconic of messages. Ethereum's security depends on every miner storing every byte of blockchain data until the end of time, hence substantive data uploads on this system remain unwieldy. Moreover, propagating more than a little data in each block could delay time-sensitive miners, resulting in a so-called Verifier's Dilemma [16] which could break underlying consensus. Indeed,

rational miners might not wait for downloads. We therefore consider off-chain alternatives.

We set ourselves the task of constructing a trustless oracle which can confirm off-chain data availability. This use case fundamentally differs from that of traditional, cloud storage platforms, such as Dropbox [2], Storj [13], Sia [12], and Maidsafe [8], which place users in control of the data that they upload. Dropbox, in particular, allows uploaders to decide whether and when to share their data with others. In a data availability scenario, on the other hand, disappearance of data can destroy high-stakes financial transactions as well as critical computations. The network must therefore counteract users' incentives to *pretend* that they have "published" data. For illustration, a Bitcoin transaction should not magically disappear after the system confirms its funds as spent, and neither should the input for a TrueBit computation vanish before the system has had time to process it.

To distinguish the present approach from existing projects like Filecoin [5] and Swarm [14], we explicitly isolate the data availability problem from other commonly associated concepts such as data sharding, privacy, computation, scalability, and Ethereum. We aim for a dead simple, modular design which is amenable to a clean and rigorous security analysis.

Overview of technical contributions. Under the peer-to-peer network assumptions of Section 3, the proposed construction in Section 4 achieves a trustless, Nakamoto-based system which correctly reports on whether or not registered data are available during a given epoch. Our construction maintains integrity without resorting to either distinguished nodes or a reputation scheme. Any anonymous node can participate in the system with a deposit, and like Bitcoin, security improves as more honest and rational miners nodes join the network. The main idea is straightforward: use Nakamoto consensus [18] to create a report of what's available and what isn't.

In the present system, every functioning Miner node stores and propagates each and every registered datum. We do not attempt to "shard" data among various parties. Rather than attempting an infinitely scalable solution, we instead content ourselves with designing a system where each node securely stores some orders of magnitude more data than "on-chain" storage permits. Unlike a blockchain, which permanently stores all data forever, the present system releases data after its registration period ends, hence its storage space is reusable. We shall further explore scaling methods in Section 6.

Building on an independent blockchain offers some flexibility over Ether-

eum. As noted earlier, Ethereum has limited storage space and its miners do not see external inputs. Thus our data availability system requires some consensus beyond Ethereum mining. Moreover, a system which allows a set of ERC20 tokens [3] to determine consensus forfeits the character of open participation. Finally, potential abstinence among token holders complicates the notion of "majority vote."

2 Warm-up: fiat-crypto exchange rates

Verifying an Ethereum transaction amounts to checking signatures, logical conditions, and self-evident mathematical truths. Thus Ethereum miners, who are tasked with verifying these transactions, need only pay attention to local announcements on the blockchain. One can, however, imagine that miners could agree on other kinds of objective, global facts, whose validity depends on time. In this section, we outline a simple "consensus computer" [16] application which extends beyond agreement on mathematical facts.

Consider the following concrete example. Some decentralized applications require a bound on the exchange rate between a stable, fiat currency and a native, cryptocurrency token. Since miners in traditional blockchains do not observe fiat exchanges, some other mechanism must necessarily feed this information to the blockchain. Rather than relying on an authority, Teutsch and Reitwießner proposed to use a blockchain consensus protocol to agree on external exchange costs [19, Section 5.5]. We shall now extrapolate on this idea and then, in Section 4, transmogrify it into a data availability protocol.

Network nodes agree on exchange rates using a variant of Nakamoto consensus [18], the protocol underlying Bitcoin. Rather than collating financial transactions into blocks, each Miner simply includes, in addition to a proof-of-work, a real interval allegedly containing the value of the current current exchange rate. A block, then, is valid if both the proof-of-work is correct and its interval contains the true exchange rate. By convention, Miners keep an eye on real exchange rates and mine on top of blocks they perceive as valid.

How large should the exchange rate interval be? While in general, Miners might be able to agree, say, that the price of an ether lies between 650 and 700 USD, during moments of wild volatility or low liquidity, the market may not afford such precision. For this reason, each Miner independently chooses the size of of the interval and receives a block reward inversely related to the

size of the interval. Smaller intervals have higher reward potential, however they also have a greater chance of being ignored by Miners who may perceive the associated block as invalid.

3 The consistency problem

Let us now return to our original problem. We seek a decentralized system which correctly reports on data availability. Our consensus protocol construction hinges on the following crucial assumption about the underlying peer-to-peer network.

Consistency Axiom. Either almost all nodes in the network can download a given datum, or almost none of them can.

The World Wide Web, for example, largely exhibits this property. Either the website `truebit.io` is "up" and everyone in the world can see it, or else most people agree that the site is "down" regardless of their Internet access point. For the purpose of this exposition, we shall assume that a peer-to-peer network exists and exhibits this desired consistency property. We shall not concern ourselves here with the construction of the peer-to-peer layer but rather explore a cryptoeconomic structure on top of it.

The Consistency Axiom above permits us to make the following well-defined notion.

Definition 1. Let x be a datum. If most nodes can see x, then x is *publicly available*. If few nodes can see x, then x is *not publicly available*.

Due to the gap afforded by the Consistency Axiom, the above definition covers all possible cases for all data. We now introduce a secondary network assumption.

Upload Axiom. Any computer can join the network and, with high probability, propagate data into a publicly available state.

Without the Upload Axiom, our Consistency Axiom might vacuously describe a network without any nodes. A new upload need not become instantly available, however we shall assume a bounded lag on its propagation. We conclude this discussion with one final requirement.

Directory Axiom. Nodes can efficiently determine whether or a datum x is publicly available from $\text{hash}(x)$. If x is available, then $\text{hash}(x)$ tells where to download x.

Thus any node which wishes to upload or download a publicly available datum can use the datum's hash as a directory. The IPFS [6] filesystem, for example, satisfies the Directory Axiom. Aside from exceptions described in Section 5, the system relies on Nakamoto consensus for security.

4 A consensus protocol for data availability

Under the axioms of Section 3, we now devise a decentralized system which indicates whether or not a registered datum is publicly available. We assume familiarity with Nakamoto consensus [18]. The network consists of two types of parties: Storers who wish to provably publish data and Miners who both confirm and guarantee availability of data. Let c be the number of blocks to confirm a storage request for a datum and propagate it through the network. For simplicity of presentation, we assume that this time bound suffices for all datums, regardless of size. The Upload Axiom from Section 3 now allows us to introduce the following upload interface.

Storer interface. A Storer who wishes to publish a datum x broadcasts a *registration* of x to the network consisting of the following components:

1. hash(x) which, by the Directory Axiom, doubles as an address for downloads,
2. the number of block epochs for which x should be publicly available (excluding network lag time c), and
3. a *reporting fee* payable to Miners based on the size of x and the registration duration specified in item 2.

We say that datum x is *registered at time* t if the blockchain contains a registration for x that persists at block epoch t. Note that registered data may or may not be publicly available. The Storer must propagate his registered datum, lest the network report it as unavailable.

Definition 2. A *report*, denoted $R(x)$, is a formal, plaintext assertion that datum x is publicly available. A Miner who includes $R(x)$ in the t^{th} block signals that, "datum x is publicly available in both block epochs t and $t+1$." A set of reports S in block t is called *complete* if for every registered datum x, $R(x) \in S$ if and only if x is publicly available in block epochs t and $t+1$.

Using Definition 2, we shall obligate each Miner who propagates a block claiming that x is publicly available at time t to further ensure that x remains

available at time $t+1$ According to the first item of Definition 3, a Miner's block remains valid in the successor block only if no publicly available data disappears. See "Space economy" in Section 5 for further details.

Miner interface. Any Miner who wishes to join the network first identifies the "longest" blockchain, namely the one containing the greatest proof-of-work [18]. The miner obtains the longest blockchain from peer nodes and retraces it from its genesis block, observing each Storer transaction along the way in sequence, and noting which Storer requests are still registered at the present moment. The Miner attempts to download, store, and propagate all currently registered data. The Miner locally considers any data successfully downloaded to be publicly available. She can determine validity of the current header block valid, according to the criteria below, after silently observing the chain for c epochs. We assume that the initial, altrustic Miners in the system converge to a consistent world view during the first c blocks following the consensus genesis.

Definition 3. A *valid* block at time t consists of the following elements:

1. the complete set of reports at time t (taking into consideration network lag time c),
2. a collection of new, cryptographically signed Storer requests, and
3. a value *nonce* witnessing a proof-of-work. More specifically, the concatenation of the following components must hash to a small value:
   (a) the mining nonce [18],
   (b) items 1. and 2. above.
   (c) a private key at which to receive the block reward and network fees, and
   (d) the hash of the previous block header.

Main Loop. The protocol steps for the Miner now roughly follow Nakamoto consensus [18].

1. In each block epoch, the first miner to find a valid block broadcasts it to the network and receives a block reward plus applicable reporting fees.

2. Upon verifying a new valid block, each Miner downloads, stores, and propagates all data registrations contained in the new block. The Miner locally considers any data successfully downloaded to be publicly available and propagates the data as expediently as possible.
3. The mining race begins anew on top of the new block.

Miners always mine on the "longest" chain whose most recent c blocks are all valid. "Longest" here formally means the chain with the greatest proof-of-work, since block difficulty may change over time. Miners need not store data which ceases to maintain registered status.

Unlike Bitcoin, the present blockchain construction has no notion of validity for individual "transactions." Indeed reports are only valid as sets. We remark that the validity of a block also depends on time because data presence can vary. We thus realize a powerful application of the consensus computer [16] which both requires and permits agreement on facts external to mathematics.

5 Security analysis

In this section, we argue that the blockchain construction in Section 4 accurately reports data availability. More specifically, we argue that a report for a registered datum x appears in the t^{th} block if and only if x was publicly available during the t^{th} block epoch. In other words, the system records both when a registered datum is publicly available and when it isn't. The "Main Loop" in Section 4 dictates that Miners always broadcast valid blocks consisting of complete reports. The security doubts we must resolve in order to establish our desired property above are twofold:

1. Can rational Miners and Storers gain by deviating from the consensus protocol?
2. Does the consensus withstand peer-to-peer layer failures?

We shall assume that an adversary of the first type wishes to either:

1a) convene a published report that some datum was publicly available when it actually wasn't, or

1b) receive a block reward without actually committing resources to the network.

Nakamoto consensus largely inhibits attacks of types 1a) and 1b). The Consistency Axiom permits us to circumvent potential attack vectors of the second type, however precise protocol adaptations for handling these attacks depends on the specific implementation of the underlying peer-to-peer network. It remains a crucial open problem to design and construct a peer-to-peer layer satisfying the three axioms in Section 3.

Data withholding. A Storer who registers a datum x could potentially take any of the following actions:

1. neglect to publish x itself,
2. delay his public reveal of x until the final moments of c-block propagation period, or
3. publish x at first but then hide this datum from the network.

In case 1, Miners never see x, unless it was otherwise publicly available, and hence they never report x in a block. Since the blockchain witnesses no report of x as publicly available, this type 1a) attack fails. Cases 2 and 3 do not impact functionality of the network, care of the Consistency Axiom. Indeed, either x becomes publicly available, in which case most Miners would report it as such, or else it would not be publicly available, in which case they would not. Miners have incentive to propagate data (Section 4), which reduces the chances that x could transition from publicly available to not publicly available during its registration period. In short, the Storer cannot incite a fallacious or controversial report on the blockchain.

Space economy. Although the consensus protocol in Section 4 obligates Miners to store and propagate data, a Miner might neglect this obligation in attempt to conserve resources in accordance with attack type 1b). A Miner who fails to upload and propagate publicly available data risks orphaning his block as other Miners who cannot access x may perceive the block as invalid. Similarly, the Miner who mines the subsequent block at time $t+1$ bears responsibility for the availability of x until the next block epoch $t+2$. This overlapping storage requirement reinforces the Consistency Axiom by incentivizing Miners to download and propagate all registered, publicly available data. Indeed, a Miner who witnesses but does not propagate data must rely on the faithfulness of other anonymous Miners and Storers to maintain validity of her reports in the successive mining race.

We remark that with some small probability, a Miner may not be able to see a publicly available datum and therefore might unknowingly propagate an invalid block. This invalid block would then not be accepted by other Miners, who have the "true" world view. As in Bitcoin, timely, well-intentioned blocks occasionally "uncle," or never reach the blockchain. Confirmed reports, however, remain consistent with publicly available data because the majority of Miners share correct perceptions.

Mining pools. Miners often share computing resources and block rewards via *pools* in order to collectively reduce income variance. The pool *operator*, who coordinates a pool's cooperation efforts, typically chooses the reports for all members in the pool. Under such circumstances, pool members would rely on their pool operator to determine whether a datum is publicly available, thereby reducing the total number of eyes on the peer-to-peer network and weakening consistency. A simple solution is to require all mining pool members to choose their own reports rather than relying on operators for selection. To this end, the underlying consensus mechanism could, for example, mandate universal participation in SmartPool [17].

6 Scalable implementation

How much data could a system like the one proposed here actually monitor in practice? While the actual capacity depends on the structure of the underlying peer-to-peer network, the present storage mechanism clearly has some finite limit for the same reasons that Bitcoin and Ethereum have bounded transaction volumes [16]. The finite limit of this system should, however, greatly exceed what the system could securely store directly on the blockchain itself. What about more storage — does this construction scale? Fortunately, two independent storage systems can store twice as much as a single one!

In theory, one can store an unlimited amount of data through system replications, however, if one wishes to use the availability of data in some root system, such as Ethereum, the root system's underlying consensus protocol must keep an eye on each individual, data availability system. Suppose that a Task Giver in TrueBit [19] were to provide a computational task whose off-chain input Solvers and Verifiers could not see (i.e. they witness only a hash on-chain). Then the Task Giver could potentially provide a bogus solution to his own task, and neither Solvers nor Verifiers would have any recourse to object against the data unavailability unless the authoritative

Judges, or miners, collectively expand their myopic, on-chain, world view.

Finally, we remark that maintaining the same level of proof-of-work security on two independent blockchains requires twice as much mining resources as a single blockchain. Thus a hierarchical system such as Plasma [10], in which the integrity of each child blockchain relies on proper function of its parent, could provide a useful scaling model for data availability. Each leaf in Plasma's blockchain hierarchy would manage a modest amount of data, while the root system would only monitor data availability disputes for escalated situations.

References

[1] Bitcoin. https://bitcoin.org/.

[2] Dropbox. https://www.dropbox.com/.

[3] ERC20 token standard. https://theethereum.wiki/w/index.php/ERC20_Token_Standard.

[4] Ethereum. http://ethereum.org/.

[5] Filecoin. https://filecoin.io/.

[6] IPFS. http://ipfs.io.

[7] Livepeer. https://livepeer.org/.

[8] MaidSafe. https://maidsafe.net/.

[9] Oraclize. http://www.oraclize.it/.

[10] Plasma. https://plasma.io/.

[11] Reality Keys. https://www.realitykeys.com/.

[12] Sia. https://sia.tech/.

[13] Storj. https://storj.io/.

[14] Swarm. https://ethersphere.github.io/swarm-home/.

[15] TrueBit. https://truebit.io.

[16] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)*, pages 706–719, New York, NY, USA, 2015. ACM.

[17] Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. SmartPool: Practical decentralized pooled mining. In *26th USENIX Security Symposium (USENIX 17)*, pages 1409–1426, Vancouver, BC, 2017. USENIX Association.

[18] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

[19] Jason Teutsch and Christian Reitwießner. A scalable verification solution for blockchains. Manuscript.