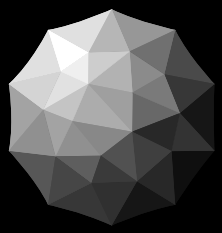
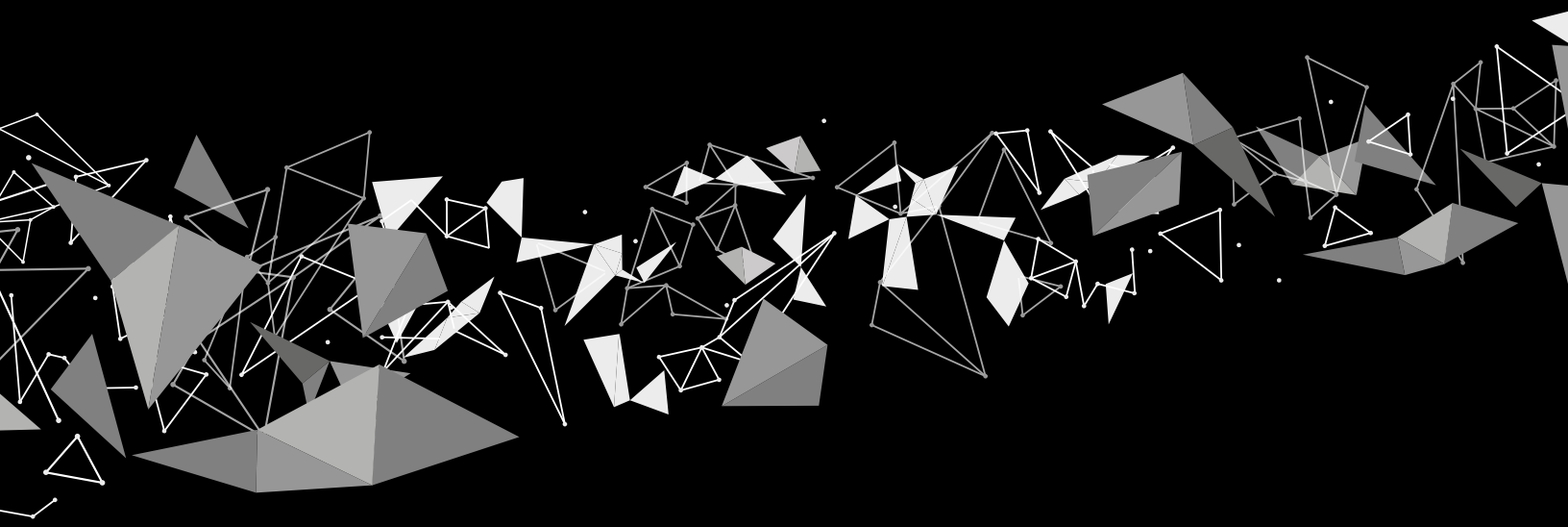


XAIN.

Proof of Kernel Work

A Resilient & Scalable Blockchain Consensus Algorithm for Dynamic Low-Energy Networks

Leif-Nissen Lundbæk, XAIN
Daniel Janes Beutel, XAIN
Michael Huth, Imperial College London
Laurence Kirk, University of Oxford
Stephen Jackson, XAIN



Yellow Paper: Version 1.3

Contents

1 Introduction

1.1 Motivation & Aims

This yellow paper proposes the notion of *Practical Proof of Kernel Work*. Proof of Work (PoW) [2] is a by now well known concept that precedes the development of Bitcoin [15] and was originally proposed as a potential solution to contain spam emails [5]: a mail server would have to solve a certain cryptographic puzzle before gaining the permission to send a certain number of emails, and one could then dynamically increase or decrease the level of difficulty for solving such puzzles to adaptively manage email traffic.

Proof of Work is, of course, a central ingredient of cryptocurrencies such as Bitcoin and Ethereum [20]. However, the utility of Proof of Work has been questioned, given that it requires a lot of energy and so appears to be wasteful. Yet, Proof of Work seems to be the only currently available mechanism that has been tested in real-world applications and by which we can guarantee stronger security for the practical immutability of a Blockchain – particularly in an enterprise setting in which we face an increasing threat of insider attacks, but in which we also may restrict which agents or devices may participate in the solution of such puzzles. Proof of Work is also a very simple mechanism that is, therefore, easy to implement, whereas the complexity of alternative approaches may make their correct implementation and change management more challenging.

There is therefore an incentive to seek alternative usage modes for Proof of Work that retain its desirable security whilst also reduce the amount of energy that Proof of Work consumes within a closed or open Blockchain system. We here propose and develop such an approach which we base on two pillars:

- **Reduction to a Kernel:** a means of reducing the set of nodes that can participate in the solving of Proof of Work puzzles such that an adversary cannot increase his attack surface because of such a reduction, based on Cryptographic Sortition [4]
- **Practical Adaptation of Existing Technology:** a realization of this Proof of Work reduction through an adaptation of existing Blockchain and enterprise technology stacks

These pillars are coordinated based on models for optimization, and they are consistent with the use of machine learning for adaptive resiliency of system parameters and networks in more closed blockchain systems.

We call this reduced version of Proof of Work the *Proof of Kernel Work* (PoKW). Furthermore, the realization of the combination of these pillars lets us create a Blockchain-based communication platform for energy-critical, low-latency platforms – supporting the dynamics of a majority of nodes that consistently enter or leave the network. We face such high dynamics in many IoT systems, specifically in potential vehicle networks and their intelligent infrastructures. Thus, this work is particularly focused towards building a secure, scalable, and stable information auditing channel between such objects.

1.2 Intended Audience

This yellow paper means to be accessible to a broad range of readers: scientists, engineers, Blockchain developers, practitioners in Information Security and Enterprise Systems, as well as decision makers in Business, Policy, and Governance. As a consequence, we will at times oversimplify technical presentations but also strive to provide enough technical content so that experts may judge the merits of the suggested overall approach.

1.3 Outline of Yellow Paper

Section 2 introduces our particular approach to a smaller yet still resilient mining race based on Proof of Work and a smaller but randomly determined kernel of nodes. Our models and tools for optimization of performance, security, and cost trade-offs are described in Section 3. How PoKW may be combined with PoET for more advanced resiliency is the subject of Section 4. Some attack vectors for this system and overall approach, and a discussion of how we may mitigate or prevent such attacks, are provided in Section 5. The practical reasoning and implementation of our system is featured in Section 6 and we conclude in Section 7.

2 Proof of Kernel Work: PoKW

Proof of Work, as currently used in Bitcoin and Ethereum, has been very successful as a consensus mechanism for cryptocurrencies and Blockchain systems. However, it consumes a lot of energy and leads to centralization of mining in standard incentivization structures. Therefore, it seems desirable to retain the advantages of PoW while also containing its energy consumption and mitigating, if not eliminating, centralization of mining. The work in [13] already developed means of minimizing energy consumption of PoW in the "governed Blockchain" setting [14], at a guaranteed level of security. We now want to scale up these abilities.

A Blockchain $B^0, B^1, \ldots, B^{r-1}$ consists of a linearly ordered list of blocks, where B^i has block number i and block height $i-1$, the number of blocks that precede B^i in that chain. Block B^0 is the *Genesis Block*. Each block B^r with $r > 0$ is determined by a *mining race* that also makes B^r depend on B^{r-1}.

2.1 Algorithmic Introduction to PoKW

The process of electing a leader – who can propose the next block on the chain – relies on Proof of Work (PoW). We restrict participation in the PoW mining race for the next block by three mechanisms – which we can interpret as three filters run in the listed order:

F1 A dynamic White List L, authenticated on the Blockchain, maintains the public keys from which it chooses participants in the next PoW mining race.

F2 Rule-based policies that additionally constrain which nodes from the White List L this second mechanism is, in principle, allowed to select from.

F3 An adaptive node selection mechanism, illustrated in Figure 1, based on *Cryptographic Sortition* as introduced in Algorand [8, 4]. This randomly selects a subset of expected size n_w from the set of eligible public keys computed by the above two filter mechanisms.

We may also apply these staged filters to tasks other than mining. For example, a machine learning task may be desired that is periodically subsumed by the task of mining. In particular, the above filtering technique lends itself to other realizations of task divisions and dependencies. Implementations may blur the first two filters or not use F2 at all.

The second mechanism offers a means of controlling the change of the overall system and its configuration state, as we discuss in a later section. Similarly, it can control the set of eligible public keys. For example, a rule may say that the public key that won the last mining race is not permitted to join the next 2 mining races. Another rule (conjoined to all other rules) could say that a public key that makes its first appearance on the blockchain in the last k blocks is not permitted to perform any task on the next block.

We can assure the trustworthiness of the interaction of these rules and the other two mechanisms by formal modeling and analysis. For example, if PK is the set of public keys computed

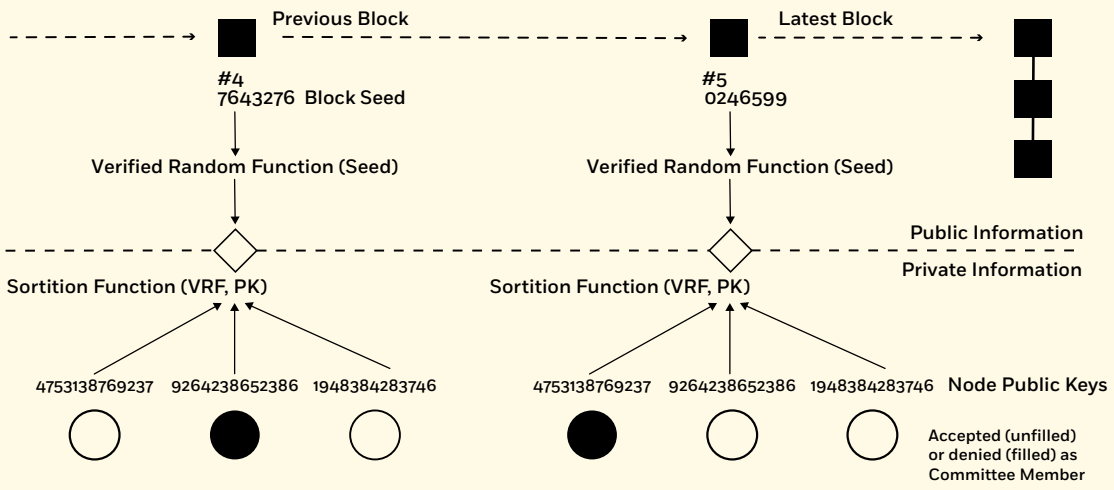


Figure 1: An illustration of the Cryptographic Sortition process used in PoKW

by the above two filters F1 and F2, then we want that the quotient $n_w/|\,PK\,|$ is in $[0, 1]$ to denote a meaningful probability threshold. Formal models can verify such invariants, and such invariants also find their way into `assert` statements in the specification of PoKW in Figure 3.

The third mechanism F3, Cryptographic Sortition, randomly selects at each blockheight $r \geq 1$, the set of nodes eligible to complete a task – such as mining a block B^r – from the set of all public keys computed by the first two mechanisms. The selection process has the following properties:

- it is sufficiently random, and only the owners of the private keys corresponding to the selected public key know the process has selected them
- an adversary who can compromise nodes cannot exploit this selection mechanism to inform which nodes/public keys it aims to compromise, and
- the expected number, n_w, of the number of selected public keys is a control parameter of the Blockchain system.

This combination of three mechanisms for controlling access to public keys to tasks offers advantages. For one, a system can control the expected number of public keys that participate in specific tasks such as mining or administering the whitelist L. This process may save energy costs as fewer miners participate, and so fewer lose a mining race.

It can also change the game theory of incentive structures for application domains that incorporate such incentives (e.g., cryptocurrencies). The pooling of mining resources as seen in Bitcoin may no longer be effective or would require novel strategic behavior. These advantages apply equally to Blockchain systems that are open (any node may join the system) or closed (only specified nodes can participate in the system).

To summarize, Cryptographic Sortition determines the actual set of nodes eligible to participate in a specific task at blockheight r as a deterministic function of the White List L and rule-based policies at blockheight $r - 1$, and of the local Blockchain.

2.2 Algorithmic Specification of PoKW

This section details how to use PoKW as a consensus mechanism in a given blockchain framework.

Notation We use the familiar tuple notation $a = (a_1, \ldots, a_n)$ and its projections $\pi_i(a) = a_i$ in the pseudo-code below. We use standard set-theoretic notation. If A denotes a set, then $|A|$ denotes its size, and $a \in A$ denotes that a is an element of A. Set comprehension $\{a \in A \mid \phi(a)\}$ defines the subset of A of all elements that satisfy ϕ. We also strive for clarity over code elegance in this yellow paper, implementations optimize algorithmic function. Whenever we mention 'destroying keys', this means secure memory erasure.

Block structure Using the notation in [8, 4], we let a block have *conceptual* structure

$$B^r = (r, TSet^r, Q^r, H(B^{r-1}), nonce, k, p, n_w, \ldots) \tag{1}$$

where

- r is the blockheight,
- $TSet^r$ is the payload of transactions of the application domain,
- Q^r is the seed of that block – a concept introduced in [8, 4],
- $H(B^{r-1})$ is the hash of the previous block,
- $nonce$ is the value that demonstrates Proof of Work,
- k is a security parameter similar to that used in Algorand [8, 4],
- n_w is the expected size of the set of nodes that are eligible to participate in the mining race for the next block B^r, and
- other components "..." may be populated with additional configuration information.

The notation $sig_{pk}(m)$ refers to the digital signature of the message m with the private key sk that corresponds to the public key pk.

The definition of $SIG_{pk_i}(m)$ given in Figure 2 embeds the public key pk_i into the signed message. An alternative is to embed identity information. Such implementation details of identity management may vary with the application domain.

Some instantiations of PoKW may choose to replicate parameters within the block structure in (1), e.g., one may choose bespoke seeds Q^r_{task} for each task $task$ and corresponding expected sizes n_w^{task}. For the sake of illustration, we present the above simple setting.

The notation $0.H(m)$ refers to the real number in the open interval $(0, 1)$ obtained by interpreting $H(m)$ as the mantissa of that real number over the binary representation of reals (base $b = 2$). As already stated, the filter F3 uses *Cryptographic Sortition*, an important technical ingredient of Algorand [8, 4] to select a dynamically adjustable group of nodes for a specific task $task$, e.g. the next mining race. A node pk_i may perform task $task$ at blockheight r if the formula below holds:

$$0.H(SIG_{pk_i}(r, task, Q^{r-1})) < \frac{n_w}{|PK^{r-k}|} \tag{2}$$

That is to say, if the hash of its signature of $(r, task, Q^{r-1})$ is less than the quotient of parameter n_w, and the size of the set PK^{r-k} of public keys, where PK^{r-k} is the result of performing the two filtering operations F1 and F2 described above.

The current Blockchain $B^0 \dots B^{r-1}$ creates consensus for the values of Q^{r-1} and n_w, and indirectly for the set of keys PK^{r-k}.

Functions $getEligibleKeys_{pk_i}$ and $getTransactionsForNextBlock_{pk_i}$ refer to the local state of node pk_i. Its local Blockchain $B^0, \dots, B^{r-1}$ for the former function, and a set of transactions (of the chain's application domain) that node pk_i has already seen and validated (the validation logic is also application-specific) for the latter function.

The pseudo-code below indicates which functions may want to extract values of local variables such as k and n_w from the Blockchain. We cannot do this at each blockheight or step in an implementation or shared across functions. However, we need to embed such parameter values in the chain and periodically verify them, not least because machine learning may change them over time.

Figure 2 shows some functions that we use as primitives:

(a) $SIG_{pk_i}(m)$: specification of digital signatures for node pk_i – where $sig_{pk_i}(m)$ refers to the digital signature of message m under the private key for pk_i.

(b) $getEligibleKeys_{pk_i}(r, k)$: function that extracts the set of eligible public keys from local Blockchain state by running filters F1 and F2 in that order.

(c) $mayPerform_{pk_i}(r, task)$: checks whether node pk_i is eligible, in principle, to participate in task $task$ (where task $mine$ refers to a mining race) for the block with blockheight r.

(d) $getTransactionsForNextBlock_{pk_i}(r)$: function that computes local transactions that should be in the next block (should node pk_i happen to win the mining race).

The determination of set $getEligibleKeys_{pk_i}(r, k)$ is also a function of the security parameter k:. Filter F2 ensures that only entities that interacted with the Blockchain in its history $B^0, \dots, B^{r-k}$ may be elements of that set. In particular, this prevents an adversary from introducing such entities in more recent blocks to rapidly gain significant influence – which then may also increase the number of nodes it would control in the randomly selected mining nodes for blockheight r. Computation of outputs for (c) and (d) are application-specific.

We are now in a position to specify the code that nodes run to determine whether they are eligible nodes for the next mining race, and what tasks performed if indeed eligible. We can see this in Figure 3. Node pk_i waits until it learns the latest block B^{r-1}. Then it extracts from the local Blockchain the security parameter k, the period p at which machine learning and system parameter adaptation take place, the level of difficulty d for Proof of Work, and the expected size n_w of the set of nodes permitted to mine the next block B^r.

Next, it checks the integrity of these values, with appropriate exception handling (not shown). It also extracts the seed Q^{r-1} from block B^{r-1} and assigns to PK the set of all eligible public keys as returned by function call $getEligibleKeys_{pk_i}(r, k)$.

The function $mayPerform_{pk_i}$ is then used with these computed inputs to determine whether node pk_i is indeed eligible to participate in specific task $mine$, i.e., the mining race for B^r. Only node pk_i knows if it is eligible, assuming that it does not share the signature $SIG_{pk_i}(r, Q)$ with any other node. This signature is the input for the hash function H, whose output decides whether node pk_i is indeed eligible. In particular, no other nodes can produce this signature as they are not in possession of the corresponding secret key sk_i.

An attacker who compromises node pk_i and so also knows the secret key sk_i then knows whether this node can participate in the specified task. However, this selection process is dependent on the seed Q^{r-1} in a non-predictable way. Thus, the adversary doesn't know which nodes to compromise before the next mining race is about to start. This technique provides crucial resiliency to this Blockchain system architecture.

$SIG_{pk_i}(m)$ { % (a)
 $return\ (pk_i, m, sig_{pk_i}(m))$;
}

$getEligibleKeys_{pk_i}(r,k)$ { % (b)
 $assert\ (k \leq r-1)$;
 $return\ set\ of\ public\ keys\ obtained\ from\ filters$ (F1; F2) $on\ local\ blocks\ B^0, \ldots, B^{r-k}$;
}

$mayPerform_{pk_i}(r, task)$ { % (c)
 $assert\ 0 < r$;
 $extract\ k, n_w, Q^{r-1}\ from\ B^{r-1}$;
 $assert\ (0 < k) \&\& (0 < n_w)$;
 $PK = getEligibleKeys_{pk_i}(r,k)$;
 $assert\ n_w < |PK|$;
 $return\ (pk_i \in PK)\ \&\&\ (0.H(SIG_{pk_i}(r, task, Q^{r-1})) \leq \frac{n_w}{|PK|})$;
}

$getTransactionsForNextBlock_{pk_i}(r)$ { % (d)
 $return\ set\ of\ local\ transactions\ to\ be\ included\ in\ block\ B^r$;
}

Figure 2: (a) Specification of digital signatures for node pk_i, $sig_{pk_i}(m)$ refers to the digital signature of message m with the private key for pk_i. (b) Function that applies filters F1 and F2 in that order to compute superset of public keys that are eligible for the next task. (c) Specification of checking whether node pk_i is a potential participant in the task $task$. (d) Function name for the computation of the set of local transactions that should get into the next block (should node pk_i become leader). Outputs for (c) and (d) will be application-specific

Returning to the code for mining, if node pk_i is not selected for mining the block with block-height r, function $computePoW_{pk_i}$ stops execution. Otherwise, node pk_i computes the set of transactions $TSet^r$ to include in block B^r – dependent on the application logic for transactions.

It computes candidate blocks as follows. The new seed Q^r is computed as the signature of the previous seed Q^{r-1}. This process ensures that the new seed is sufficiently random and that an adversary cannot influence the value of the next seed – in a model of an adversary similar to that used for Algorand in [8, 4].

If it finds PoW (tested with function $isPoW_{pk_i}$), it sends the block (which contains the nonce for which it found PoW) together with a signature of its hash and a credential (a signature of the blockheight, nonce, and previous seed) across the network. Otherwise, node pk_i either exhausted its possible nonce values without finding PoW (and so stops the entire function); or node pk_i has learned from another node a verified PoW block B^r that it adds to its local Blockchain, and its mining efforts stop.

Function $raceEnded_{pk_i}$ captures the ability to detect either of these events (nonce space exhausted or verified block learned from another node). Of course, this program logic might be implemented in a completely different way. The message broadcasted contains a signature of the hash of this next block but signed with an *ephemeral private key*, which also gets destroyed once it sends the message. This ensures that an attacker who compromises nodes can no longer manipulate signatures of such hashes, for example, to recreate a segment of a chain assuming that it compromised the long-term public key pk_i.

The use of ephemeral keys requires a sufficient number of such key pairs such that other

Algorithm 1: $computePoW_{pk_i}()$

begin
  $assert\ (0 < r)$;
  $waitTill(block\ B^{r-1}\ is\ locally\ known)$;
  $extract\ k, p, d, n_w\ from\ local\ Blockchain$;
  $assert\ (k \leq r)\ \&\&\ (0 < n_w)$;
  $extract\ Q^{r-1}\ from\ B^{r-1}$;
  $PK = getEligibleKeys_{pk_i}(r, k)$;
  if $!mayPerform_{pk_i}(r, mine)$ **then**
    $stop\ this\ function$;
  end if
  // pk_i can participate in the PoW mining race for this blockheight
  $TSet^r = getTransactionsForNextBlock_{pk_i}(r)$;
  $nonce = initialValue_{pk_i}()$;
  repeat
    $nonce = nextValue_{pk_i}(nonce)$;
    $B^r_{pk_i} = (r, TSet^r, SIG_{pk_i}(Q^{r-1}), H(B^{r-1}), nonce, newSysValues)$;
  until $(isPoW_{pk_i}(H(B^r_{pk_i}), d)\ ||\ raceEnded_{pk_i}())$;
  if $(raceEnded_{pk_i}())$ **then**
    if $(received\ verified\ PoW\ block\ B^r\ of\ blockheight\ r\ from\ other\ node)$ **then**
      $add\ B^r\ to\ local\ Blockchain$;
    end if
    $stop\ this\ function$;
  end if
  // PoW found, and no other node reported PoW solution
  $generate\ ephemeral\ key\ pair\ (pk^r_i, sk^r_i)$;
  $m^r_{pk_i} = (B^r_{pk_i}, SIG_{pk^r_i}(H(B^r_{pk_i})), SIG_{pk_i}(r, nonce, Q^{r-1}))$
  $destroy\ sk^r_i$;
  $propagate_{pk_i}(m^r_{pk_i})$;
end

Figure 3: Specification of PoW for potential leaders for blockheight r. Nodes pk_i start mining only if they are eligible.

nodes can learn the corresponding set of public keys. There are standard solutions to this, for example, the use of identity-based public-key cryptography.

3 Robust Consensus Optimization

3.1 Motivation

For PoKW, we have the means to optimize the Blockchain network and the underlying consensus mechanisms for minimum security and stability requirements. Using mathematically assured optimization allows us to better understand trade-offs between data security, availability, and cost. Such trade-offs may be a function of the application; e.g., vehicle data needs to be resilient to attacks for an unforeseen time. However, our approach, also requires a certain network stability and resiliency, which we may not get by using standard Blockchain architectures, such as the open Ethereum network.

Therefore, the optimization of the consensus mechanism plays the largest role in this stabilization and is defined according to general assumptions realized at the protocol level. With such assumptions integrated into our system, we can build faithful models that measure and influence the security, cost, and stability of the entire system, to find an optimal equilibrium of the system state, based on the required parameters and within a monetary budget. The latter is particularly important since open Proof of Work consensus systems do not fit within any budgetary boundaries in the long term. Other approaches, such as Proof of Stake and Practical Byzantine Fault Tolerance (PBFT) [3], are not suitable to function in systems that require high stability, resiliency, and long-term data security.

3.2 Mathematics for Governed PoKW

This modeling is based on the work done in the publications [12, 13]. Our governed Blockchain approach means that our Blockchain framework also manages which network nodes can participate in system aspects, such as consensus creation. Our mathematical model assumes a cryptographic hash function: $h\colon \{0,1\}^p \to \{0,1\}^n$ Where $p \geq n > 0$ such that h has *puzzle friendliness* [16]. The *level of difficulty* d is an integer satisfying $0 < d < n$. PoW has to produce some x where $h(x)$ has at least d many leftmost 0 bits. We write $T > 0$ to compute a sole hash $h(x)$ and to decide whether it has at least d leftmost zeros. As values of d will be relatively small, T is a device-dependent constant.

Our probabilistic modeling treats h in the *Random Oracle Model* (ROM). Function h is chosen uniformly at random from all functions of type $\{0,1\}^p \to \{0,1\}^n$, that is to say, h is a deterministic function such that any x for which h has not yet been queried will have the property that $h(x)$ is governed by a truly random probability distribution over $\{0,1\}^n$.

We assume that x consists of a block header which contains some random data field – a nonce $nonce$ of bit length r, that this nonce is initialized, and then increased by 1 each time the hash of x does not obtain PoW. In particular, this yields that $\{0,1\}^p \cong \{0,1\}^{p-r} \times \{0,1\}^r$ where $0 < r < p$. The input to h is of form $x = data \mid\mid nonce$ where $data$ and $nonce$ have $p - r$ and r bits, respectively. Our use of ROM will rely on the following assumption:

Assumption 1 (Invariant)**.** *The mining of a block with one or more miners uses an input to h at most once, be it within or across miners' input spaces.*

This assumption and ROM give us hash values that are always uniformly distributed in the output space during a mining race.

Consider having $s > 1$ many miners that run in parallel to find Proof of Work, engaging in a *mining race*.

Our use of Cryptographic Sortition for PoKW suggest that s represents the expected value n_w of how many nodes are eligible to participate in the mining race for the next block.

We assume these miners run with the same configurations and hardware, for the sake of simplicity of presentation; we have refined mathematical models that allow nodes to have finitely varied hash rates and so nodes may have different hardware and mining capabilities.

As already discussed, in our approach miners do not get rewarded.

Assumption 2 (Miners)**.** *Miners are a resource controlled by the governing organization or consortium and have identical hardware. In particular, miners are not rewarded nor need incentive structures.*

However, miners may be corrupted and misbehave, e.g., they may refuse to mine. To simplify our analysis, we assume miners begin the computation of hashes in approximate synchrony:

Assumption 3 (Approximate Synchrony)**.** *Miners start a mining race at approximately the same time.*

For many application domains, this is a realistic assumption as communication delays to miners would have a known upper bound that our models could additionally reflect if needed.

Next, we model the *race* of getting a PoW where each miner j has some data $data_j$. To realize Assumption 1, it suffices that each miner j has a nonce $nonce_j$ in a value space of size

$$\lambda = \lfloor 2^r/s \rfloor$$

such that these nonce spaces are mutually disjoint across miners. Our probability space has $(data_j)_{1\leq j\leq s}$ and d as implicit parameters. For each miner j, the set of basic events E^j is

$$E^j = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda\} \cup \{\mathsf{failure}\} \tag{3}$$

Basic event failure denotes the event that all λ nonce values $nonce_j$ from $\{(j-1)\cdot\lambda, \ldots, j\cdot\lambda-1\}$ for miner j failed to obtain Proof of Work for $data_j$ at level of difficulty d. Basic event $\otimes^k \cdot \checkmark$ models the event in which the first k such nonce values failed to obtain Proof of Work for $data_j$ at level d but the $k+1$th value of $nonce_j$ did render such Proof of Work for $data_j$.

To model this mining race between s miners for $(data_1, data_2, ..., data_s)$ and d as implicit parameters, we take the product $\prod_{j=1}^s E^j$ of s copies E^j and quotient it via an equivalence relation $\equiv$ on that product $\prod_{j=1}^s E^j$, which we now define formally.

Definition 1. *1. The s-tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$ models failure of this mining race, and it is $\equiv$ equivalent only to itself.*

2. All s-tuples $a = (a_j)_{1\leq j\leq s}$ other than tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$ model that the mining race succeeded for at least one miner. For such an s-tuple a, the set of natural numbers k such that $\otimes^k \cdot \checkmark$ is a coordinate in a is non-empty and thus has a minimum $\min(a)$. Given two s-tuples $a = (a_j)_{1\leq j\leq s}$ and $b = (b_j)_{1\leq j\leq s}$ different from $(\mathsf{failure}, \ldots, \mathsf{failure})$, we define

$$a \equiv b \text{ if and only if } \min(a) = \min(b)$$

Two non-failing tuples are equivalent if they determine a first (and so final) Proof of Work at the same attempt of the race. This defines an equivalence relation $\equiv$ and adequately models a synchronized mining race between s miners.

The interpretation of events $\otimes^k \cdot \checkmark$ in the mining race is then the equivalence class of all those tuples a for which $\min(a)$ is well defined and equals k, all mining races that succeed first at attempt k. The meaning of failure is the overall failure of the mining race, the equivalence class containing the only tuple $(\mathsf{failure}, \ldots, \mathsf{failure})$.

The set of events for the PoW race of s miners is thus

$$E^s = \{\otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda\} \cup \{\mathsf{failure}\} \tag{4}$$

In (4), expression $\otimes^k \cdot \checkmark$ denotes an element of the quotient

$$(\prod_{j=1}^{s} E^j)/\equiv$$

namely the equivalence class of tuple $(\otimes^k \cdot \checkmark, \mathsf{failure}, \mathsf{failure}, \ldots, \mathsf{failure})$. Next, we define a probability distribution $prob^s$ over E^s.

To derive the probability $prob^s(\otimes^k \cdot \checkmark)$, recall that:

$$\tilde{p}(\otimes^k) = (1 - 2^{-d})^k$$

As the probability that a given miner won't obtain PoW at level d in the first k attempts. By Assumption 1, these miners work independently over disjoint input spaces. By ROM, expression:

$$\left[(1 - 2^{-d})^k\right]^s = (1 - 2^{-d})^{k\cdot s}$$

Therefore models the probability that none of the s miners obtains PoW in the first k attempts. Appealing again to ROM and Assumption 1, the behavior at attempt $k+1$ is independent of that of the first k attempts. Therefore, we need to multiply the above probability with the one for which at least one of the s miners will obtain a PoW in a sole attempt. The latter probability is the complementary one of the probability that none of the s miners will get a Proof of Work in a sole attempt, which is $(1-2^{-d})^s$ due to the ROM independence. Therefore, we get

$$prob^s(\otimes^k \cdot \checkmark) = (1-2^{-d})^{k \cdot s} \cdot [1-(1-2^{-d})^s] \tag{5}$$

This defines a probability distribution with a non-zero probability of failure. Firstly,

$$\sum_{k=0}^{\lambda} (1-2^{-d})^{k \cdot s} \cdot [1-(1-2^{-d})^s]$$

is in $(0,1)$. To see this, note that this sum equals

$$[1-(1-2^{-d})^s] \cdot \frac{1-[(1-2^{-d})^s]^{\lambda+1}}{1-(1-2^{-d})^s} = 1-(1-2^{-d})^{s \cdot (\lambda+1)}$$

Since $0 < d, s$, the real $1-2^{-d}$ is in $(0,1)$, and the same is true of any integral power thereof. Secondly, $prob^s$ becomes a probability distribution with the non-zero probability $prob^s(\mathsf{failure})$ being $1 - prob^e(E^s \setminus \{\mathsf{failure}\})$, that is:

$$prob^s(\mathsf{failure}) = (1-2^{-d})^{s \cdot (\lambda+1)} \tag{6}$$

That this failure probability is almost identical to that for $s = 1$ is an artifact of our modeling. If each miner has 64 bits of nonce space, e.g., then our model would have $r = 64 \cdot s$, so failure probabilities decrease as s increases.

3.3 Mathematical Optimization in Mining Design Space

Generality of Approach We want to optimize the use of $s > 1$ miners using a level of difficulty d, and a bit size r of the global nonce space concerning an objective function. The latter may be a cost function, if containing cost is the paramount objective or if we seek a first cost estimate that we can then transform into a constraint to optimize for a security objective. For example to maximize the level of difficulty d, as seen below. Higher values of d add more security. It takes more effort to mine a block and so more effort to manipulate the mining process and used consensus mechanism. However, we may need lower values of d, for example, in high-frequency trading where performance is an important issue. We want to understand these trade-offs, and we want to explore how the corruption of miners or inherent uncertainty in the number of deployed miners or the level of difficulty across the lifetime of a system may influence the above trade-offs.

Optimizing Cost and Security The flexibility of our approach includes the choice of an objective function for optimization. Let us consider the following function that models cost as a function of the number of miners s, the bit size of the nonce r – implicit in random variable $E^s(noR)$, and the level of difficulty d; where we want to *minimize* cost:

$$\mathsf{Cost}(s,r,d) = \mathsf{TVC} \cdot E^s(noR) \cdot s + \mathsf{TFC} \cdot s \tag{7}$$

The real variable TVC models the *variable* cost of computing *one* hash for *one* miner, reflecting the device-dependent speed of hashes and the price of energy. The real variable TFC

models the *fixed* costs of *having one miner*. We can see this as modeling procurement and depreciation. Variables s, r, and d are integral, making this a *mixed integer* optimization problem [11]. The expression $E^s(noR)$ denotes the *expected number of attempts* (of approximately synchronous hash attempts) needed to mine a block in a mining race that uses s miners, level of difficulty d, and nonce bit size r. The derivation of this expression below shows that it is non-linear, making this a mixed-integer nonlinear programming (MINLP) optimization problem [19, 11].

We can use other objective functions. One of these is the expression d, which we would seek to *maximize*, the intuition being that higher values of d give us more trust into the veracity of a mined block and the Blockchains generated in the system.

Figure 4 shows an example of a set of constraints and optimizations of security and cost for this.

$$
\begin{array}{rcl}
0 & < & s_l \leq s \leq s_u \qquad 0 < d_l \leq d \leq d_u \qquad 0 < r_l \leq r \leq r_u \qquad \kappa \geq prob^s(\mathsf{failure}) \\
\tau_u & \geq & T \cdot E^s(noR) \geq \tau_l \qquad\qquad \delta_2 \geq prob^s(disputes\ within\ \mu) \\
\delta & \geq & prob^s(PoWTime > th) \qquad \delta_1 \geq prob^s(PoWTime < th')
\end{array}
$$

Figure 4: Constraint set $\mathcal{C}$ for two optimization problems: (a) *minimize* Cost(s, r, d) as in (7) subject to constraints in $\mathcal{C}$; and (b) *maximize* d subject to $\mathcal{C} \cup \{\mathsf{Cost}(s, r, d) \leq budget\}$ for cost bound $budget$. This is parameterized by constants $0 \leq \delta, \delta_1, \delta_2, \kappa, th, th', \tau_l, \mathsf{TVC}, \mathsf{TFC}$ and $0 < T, s_l, r_l, d_l$. Variables or constants $s_l, s_u, s, d_l, d_u, d, r_l, r_u, r$ are integral

We use integer constants s_l, s_u bound variable s, and similar integer bounds to constrain integer variables r and d. The constraint for κ uses it as upper bound for the probability of a mining race failing to mine a block. The next two inequalities stipulate that the expected time for mining a block is within a given time interval, specified by real constants τ_l and τ_u.
The real constant δ_2 is an upper bound for:

$$prob^s(disputes\ within\ \mu)$$

The probability that more than one miner finds PoW within μ seconds in the same, approximately synchronous, mining race. The constraint for real constant δ says that the probability:

$$prob^s(PoWTime > th)$$

δ above bounds the *actual* time for mining a block (for the expected number of miners) being above a real constant th. This constraint is of potential interest.
Some systems may also need assurance that blocks are almost always mined in time *exceeding* a specified time limit th'. We write:

$$prob^s(PoWTime < th')$$

To denote that probability and add a dual constraint, which the actual time for mining a block (for the expected number of miners) has a sufficiently small probability $\leq \delta_1$ of being faster than th'.

Constraints as Analytical Expressions We derive analytical expressions for random variables occurring in Figure 4. Beginning with $E^s(noR)$, we have:

$$E^s(noR) = \sum_{0 \leq k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) \cdot (k+1) \tag{8}$$

Which we know to be equal to:

$$\sum_{0 \leq k \leq \lambda} (1-2^{-d})^{k \cdot s} \cdot [1-(1-2^{-d})^{s}] \cdot (k+1)$$

We can rewrite the latter expression so that it eliminates reduces summations to exponentiations. We rewrite $\sum_{0 \leq k \leq \lambda} prob(\otimes^k \cdot \checkmark) \cdot (k+1)$, the right-hand side of (8), to $\lambda+1$ summations, each one starting at a value between 0 and λ, where we use the formula:

$$\sum_{k=a}^{b} x^k = \frac{x^a - x^{b+1}}{1-x}$$

This renders as:

$$E^s(noR) = \frac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1-y) \cdot y^{\lambda+1}}{1-y} \tag{9}$$

Where we use the abbreviation:

$$y = (1-2^{-d})^s \tag{10}$$

The expected time needed to get a proof of work for input $data$ is then given by:

$$E^s(poW) = T \cdot E^s(noR) \tag{11}$$

We derive an analytical expression for the probability $prob^s(PoWTime > th)$ next. $(th/T)-1 < k$ models that the actual time taken for $k+1$ hash attempts is larger than th. Therefore, we capture $prob^s(PoWTime > th)$ as:

$$\sum_{\lceil (th/T)-1 \rceil < k \leq \lambda} prob^s(\otimes^k \cdot \checkmark) = y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1} \tag{12}$$

Assuming that $\lceil (th/T) - 1 \rceil < \lambda$, the latter becomes a constraint that we need to add to our optimization problem. We may choose the value of δ based on the Markov inequality, which gives us:

$$prob^s(PoWTime \geq th) \leq T \cdot E^s(noR)/th$$

But the upper bound $T \cdot E^s(noR)/th$ depends on the parameters s, r, and d. For example, the analytical expression for $E^s(noR)$ in (9) is dependent on λ and so dependent on r as well. The representation in (12) also maintains that expression:
$y^{\lceil (th/T)-1 \rceil + 1} - y^{\lambda+1}$
Is in $[0,1]$, i.e. a proper probability. Since $y = (1-2^{-d})^s$ is in $(0,1)$, this is already guaranteed if $\lceil (th/T) - 1 \rceil + 1 \leq \lambda + 1$, i.e. if $\lceil (th/T) - 1 \rceil \leq \lambda$. But we already added that constraint to our model.
Similarly to our analysis of $prob^s(PoWTime > th)$, we get:

$$prob^s(PowTime < th') = 1 - (1-2^{-d})^{s \cdot (\lfloor (th'/T)-1 \rfloor + 1)} = 1 - y^{\lfloor (th'/T)-1 \rfloor + 1} \tag{13}$$

Which needs $0 < \lfloor (th'/T) - 1 \rfloor$ as additional constraint.

To derive an analytical expression for $prob^s(disputes\ within\ \mu)$, each miner can perform $\lfloor \mu/T \rfloor$ hashes within μ seconds. If we set:

$$w = (1-2^{-d})^{\lfloor \mu/T \rfloor + 1} \tag{14}$$

The probability that a given miner finds PoW within μ seconds is:

$$\sum_{k=0}^{\lfloor \mu/T \rfloor} (1-2^{-d})^k \cdot 2^{-d} = 2^{-d} \cdot \frac{1-(1-2^{-d})^{\lfloor \mu/T \rfloor + 1}}{1-(1-2^{-d})} = 1 - w \tag{15}$$

Therefore, the probability that no miner finds PoW within μ seconds is:

$$prob^s(0\ PoW\ within\ \mu) = (1-(1-w))^s = w^s \tag{16}$$

The probability that exactly-one miner finds PoW within μ seconds is:

$$prob^s(1\ PoW\ within\ \mu) = s \cdot w^{s-1} \cdot (1-w) \tag{17}$$

Thus, the probability that more than one miner finds PoW within μ seconds is:

$$\begin{aligned} prob^s(disputes\ within\ \mu) &= 1 - prob^s(0\ PoW\ within\ \mu) - prob^s(1\ PoW\ within\ \mu) \\ &= 1 - w^s - s \cdot w^{s-1} \cdot (1-w) \\ &= 1 - w^s - s \cdot w^{s-1} + s \cdot w^{s-1} \cdot w \\ &= 1 + (s-1) \cdot w^s - s \cdot w^{s-1} \end{aligned} \tag{18}$$

Figure 5 shows the set of constraints $\mathcal{C}$ from Figure 4 with analytical expressions and their additional constraints, e.g., $0 \leq \lfloor \mu/T \rfloor$ for the analytical representation of $prob^s(disputes\ within\ \mu)$.

$$\begin{aligned} s_l &\leq s \leq s_u \qquad d_l \leq d \leq d_u \qquad r_l \leq r \leq r_u \qquad \lambda = \lfloor 2^r/s \rfloor \\ y &= (1-2^{-d})^s \qquad w = (1-2^{-d})^{\lfloor \mu/T \rfloor + 1} \qquad 0 \leq \lfloor \mu/T \rfloor \\ \kappa &\geq y^{\lambda+1} \qquad \lceil (th/T) - 1 \rceil < \lambda \qquad 0 < \lfloor (th'/T) - 1 \rfloor \\ E^s(noR) &= \frac{1 - y^{\lambda+1} - (\lambda+1) \cdot (1-y) \cdot y^{\lambda+1}}{1-y} \\ \tau_u &\geq T \cdot E^s(noR) \geq \tau_l \qquad \delta_1 \geq 1 - y^{\lfloor (th'/T) - 1 \rfloor + 1} \\ \delta &\geq y^{\lceil (th/T) - 1 \rceil + 1} - y^{\lambda+1} \\ \delta_2 &\geq 1 + (s-1) \cdot w^s - s \cdot w^{s-1} \end{aligned} \tag{19}$$

Figure 5: Arithmetic version of set of constraints $\mathcal{C}$ from Figure 4, with additional soundness constraints for this representation. The feasibility of (s, r, d) and $r_u \geq r' > r$ won't generally imply feasibility of (s, r', d) due to the constraint in (19).

3.4 Robust Design Security

Our model described above captures design requirements or design decisions as a set of constraints, to optimize or trade-off measures of interest, subject to such constraints. We can extend this model to also *manage uncertainty* via robust optimization [1]. Such uncertainty may arise during the lifetime of a system through the possibility of having corrupted miners, needing flexibility in adjusting the level of difficulty, and so forth. For example, corrupted miners may refuse to mine, deny their service by returning invalid block headers, pool their mining power to get more mining influence or break down. Robust optimization treats such uncertainty as a non-deterministic choice and refers to it as *strict* or *Knightian* uncertainty.

Consider $1 \leq l < s$ corrupted miners. We can model their *pool power* by appeal to ROM and the fact that the mining race is approximately synchronized. The probability that these l miners win $c > 0$ subsequent mining races is then seen to be $(l/s)^c$. We can, therefore, bound this with a constant δ_3 as in Figure 5.

We model uncertainty in the number of miners available by an integer constant u_s as follows. If we deploy s miners, then we assume that at least $s - u_s$ and at most s many miners participate reliably in the mining of legitimate blocks. They will not mine blocks that won't verify and only submit mined blocks that do verify to the network. Constant u_s can model aspects such as denial of service attacks or a combination of attacks with faults. $u_s = 3$, e.g., subsumes the scenario in which one miner fails and two miners mine invalid blocks.

Integer constant u_d models the uncertainty in the deployed level of difficulty d. Our analysis should give results that are robust in that they hedge against the fact that values d' satisfying the below may be the running level of difficulty:

$$|\, d - d' \,| \leq u_d$$

This enables us to understand a design if we are unsure about which level of difficulty we will deploy or if we want flexibility in dynamically adjusting the value of d in the running system. We can see the corresponding robust optimization problem for cost minimization in Figure 6. It adds to the constraints we already consider further requirements on constants l, c, and δ_3 as well as the constraint:

$$l^c \leq \delta_3 \cdot s^c$$

We achieve robustness of analysis by changing the objective function from $\mathsf{Cost}(s, r, d)$ to

$$\mathsf{Cost}^{u_s}_{u_d}(s, r, d) = \max_{s - u_s \leq s' \leq s,\ |d - d'| \leq u_d} \mathsf{Cost}(s', r, d') \tag{20}$$

The latter computes a worst-case cost for triple (s, r, d) where s and d may vary independently, subject to the strict uncertainties u_s and u_d, respectively. We call a triple (s, r, d) *feasible* if it satisfies all constraints of its optimization problem. Costs such as the one in (20) for a triple (s, r, d) are only considered for optimization if all triples (s', r, d') used in (20) are feasible – realized with predicate $feasible^{u_s}_{u_d}$. Robust optimization guarantees [1] that the feasibility of solutions is invariant under the specified strict uncertainty (here u_s and u_d).

$$\begin{array}{l}
\min\{\mathsf{Cost}^{u_s}_{u_d}(s, r, d) \mid feasible^{u_s}_{u_d}(s, r, d)\} \\
\text{subject to the set of constraints } \mathcal{C} \text{ from Figure 5 together with} \\
\begin{array}{lll}
4 = l < s & c = 6 & 0.001 = \delta_3 \\
l^c \leq s^c \cdot \delta_3 & u_s = 5 & u_d = 3
\end{array}
\end{array}$$

Figure 6: Robust cost optimization for the constraints from Figure 5, where up to $u_s = 5$ miners may be non-functioning or miss-behaving; where the level of difficulty may vary by up to +/–3; and where the probability of any mining *pool* of size $l = 4$ winning $c = 6$ consecutive mining races is no larger than $\delta_3 = 0.001$. Predicate $feasible^{u_s}_{u_d}(s, r, d)$ characterizes *robustly feasible* triples. It is true if all triples (s', r, d') with $s - u_s \leq s' \leq s$ and $|\, d - d' \,| \leq u_d$ are feasible.

We refer to the publication [13] for a more in-depth discussion of this approach and some of its experimental results for this optimization framework. We next discuss the node selection mechanism based on Cryptographic Sortition, and how it relates to the above model.

3.5 Discussion of Cryptographic Node Selection

The above model seeks to compute an optimal number of miners s for the realization of important trade-offs in security, cost, performance, and resiliency. Initial use of this optimization framework would, therefore, inform the choice of appropriate values for system parameters such as n_w, k, and perhaps others. For example, an optimal value reported for s is a sensible initial choice for n_w to be embedded into the Genesis Block B^0, as this means that trade-offs are made optimally for the expected number of nodes that will participate in the mining race.

It is important to realize that use of Cryptographic Sortition does not guarantee a fixed such number of nodes, and so we work with an expected value instead. However, our approach is well integrated with the White Listing mechanism. Each node on the network can extract the set PK of nodes that are, in principle, eligible to participate in the next mining race. And the determination of this set through function $mayPerform_{pk_i}(r, mine)$ already incorporates the White List L as a filter. The threshold $n_w/|\,PK\,|$ for Cryptographic Sortition adapts automatically to changes in the White List L since the threshold always expresses an expected size of n_w no matter what size PK is at present state of the Blockchain and White List L.

For example, if the set PK becomes smaller through a shorter White List L, then the ratio $n_w/|\,PK\,|$ becomes larger, meaning that it becomes more likely that Cryptographic Sortition selects a node from the smaller set PK. Conversely, a longer White List L increases the size of PK, and then $n_w/|\,PK\,|$ becomes smaller – meaning that Cryptographic Sortition is less likely to select nodes in PK.

In the above model, we also expressed that there might be l of the s miners corrupted by an adversary. We may think of l/s as the percentage of miners that the adversary manages to seize control of for a given mining race. This percentage is meaningful in the context of our Cryptographic Sortition as a node selection mechanism. If we let N be the number of nodes of the total network, $n_w \ll N$ holds.

Let us further assume that an adversary can control at most N_1 out of N nodes. Define $q = N_1/N$. We claim that q is a realistic percentage of the adversary's ability to corrupt miners that are eligible to participate in the next mining race for block B^r. To the adversary, the selection of such nodes looks random. If function $mayPerform_{pk_i}(r, mine)$ selects s nodes for mining, then we know that the expected value of s is n_w. If we set $l = \lceil q \cdot s \rceil$, then the percentage l/s is as close to q as we can get, given that l and s are integral.

We can determine meaningful values of l based on assumptions of an adversary's global abilities, and this determination is valid no matter how large the actual or expected number of nodes selected for mining turns out to be. To make this more concrete through an example, let us say that an attacker can compromise at most 25% of the nodes of the entire network. Consider the use of $s = 4$ miners, minimal value for reasonable security.

An adversary can compromise one in four nodes. Since a random process selects the four selected miners, we may assume that the adversary's choice of nodes and the choice of nodes determined by the Cryptographic Sortition are independent. Therefore, the attacker is expected to compromise $0.25 \cdot 4 = 1$ node of the four that do the mining. In other words, the percentage of nodes that an attacker can compromise in a mining race is more or less equal to the percentage that the attacker can compromise on the entire network.

One consequence of this is that if an adversary wants to control 50% of a mining race of, say, 4 miners. Then they need to compromise $5,000$ nodes in a network of $10,000$ nodes, but they need to compromise $50,000$ nodes in a network of one $100,000$ nodes.

Therefore, large networks make it harder for an adversary to gain control even though the size of the sub-network which does the mining does not need to grow with the size of the overall network. The reasoning for the probability of wining k consecutive mining races through nodes controlled by an adversary remains valid and local to the selected nodes for each of these c races, as expressed in the above optimization framework.

4 Advancing PoKW Resiliency through PoET

In 2017, Intel introduced a novel Nakamoto type consensus protocol called *Proof of Elapsed Time* (PoET). This protocol elects a leader, who is allowed to produce the next block, as follows:

- nodes sample from an exponentially distributed variable a duration that they then commit to waiting before claiming leadership,
- a timer is created that checks whether the node waits for this long,
- a node cannot cheat this timer since it performs its execution and cryptographic attestation within a Trusted Execution Environment (TEE), and
- the node with the shortest and verifiably attested waiting time is elected as leader.

Intel released a reference implementation of PoET, for an abstract TEE, for Hyperledger, and an implementation for Intel's Software Guard Extensions (SGX) as TEE was then also provided as part of the Sawtooth distributed ledger software. The PoET consensus mechanism appears to be an attractive alternative to the Byzantine protocols that are part of the Hyperledger Fabric. For one, the protocol has lower communication complexity than those other protocols and does not consume as much energy as Proof of Work does – Proof of Work being the original, first Nakamoto type consensus protocol.

There is broad interest in integrating TEEs in a wide range of compute devices, e.g., microcontrollers for IoT, so that such environments could support PoET. We share this interest, but also share the concern that PoET places considerable trust into the security of the used TEE. By way of example, if we were to use PoET for SGX, then any vulnerabilities discovered for SGX might have a negative impact on the security of PoET. Some also expressed concerns that PoET requires too much trust in the manufacturers of the TEE used – whomever they might be – and in the nature of their relationship with government agencies and other third parties.

We believe in the potential of PoET and mean to explore now how we may combine it with PoKW so that we can harden the reliability of each of these protocols through their judicious interaction. One aim in this combination is to harden PoKW so that attackers are dis-incentivized from cloning Proof of Work efforts onto other devices whenever the next mining committee elects a public key controlled by attackers.

The intuition behind this hardening is that we impose a minimal waiting time mD so that all nodes need to wait $\mathrm{mD} + noise$ seconds where $noise$ is node-specific and not controllable by an attacker. The value mD is chosen to provide a balance between the mining power of all benign, low-end devices on the network and the mining power of all devices (often high-end specifications) compromised by, or owned by, attackers.

The validation logic for a block would then be the "conjunction" of the logic for PoET and PoKW. A leader can only propose the next block if they were elected into the mining committee by PoKW and produced a Proof of Work solution, and if they can produce an attestation that they waited for $\mathrm{mD} + noise$ seconds before announcing their solution. This conjunctive semantics of block validation means that there is no point in cloning Proof of Work effort across fast devices for a public key that happens to be on the committee for the next block. The waiting time is chosen so that the totality of low-end, non-compromised nodes have as good of a chance of having a leader elected among them than the set of attackers would have.

4.1 Composition of PoET and PoKW

We now develop a simple model and its analysis that allows us to understand the issue above, and the potential of a combination of PoET and PoKW better.

Let d be the level of difficulty for Proof of Work and let ρ be the hash rate of a miner, meaning that the miner can compute ρ Proof of Work attempts per second. The probability that a miner can find Proof of Work is then governed by an exponential distribution. Specifically, let $p(Succ[\rho, m] \leq t)$ denote the probability that $m \geq 1$ mining devices, each with hash rate ρ, will find Proof of Work for level of difficulty d within t seconds. Thus we have:

$$p(Succ[\rho, m] \leq t) = 1 - e^{-(\rho \cdot m/d) \cdot t} \tag{21}$$

We now want to explore how we can fruitfully combine PoET with PoKW to strengthen the resiliency of PoKW and ensure that the composition guarantees low energy needs and more democratic cryptographic puzzle solving. Since both PoET and PoKW are Nakamoto type consensus protocols, their combination is of that type and so does provide eventual consistency – but with less energy demands and more democratic puzzle solving than PoKW or PoET alone.

Let n be the number of nodes in the blockchain network, a value that evolves. We assume that an attacker can control at the most pc percent of these nodes, and so it can control, conservatively, at most:

$$m_A = \lceil pc \cdot n \rceil \tag{22}$$

nodes in that network. It is also conservative to assume that an attacker controls all these m_A nodes and are operating at the same high hash rate ρ_A – with $\rho_A = 10^9$ being a typical value. Therefore, the remaining:

$$m_B = n - m_A \tag{23}$$

nodes are benign, and it is conservative to assume that all these m_B nodes are operating at a lower hash rate ρ_B, where the value of ρ_B reflects common end-user compute devices such as a standard smart-phone and micro-controllers found in embedded systems. For example, we may have $\rho_B = 4 \cdot 10^3$ for a typical Android phone and similar rates for a micro-controller.

This abstraction is conservative for assessing the abilities of an attacker, and has the benefit of simplifying the model of this network. Malicious nodes all have the same high hash rate ρ_A, and benign nodes all have the same low hash rate ρ_B.

In our subsequent model and analysis, we make the following assumptions:

- all miners have a TEE that can attest that it waited for a duration of $\mathsf{mD} + noise$, with $noise$ determined similarly as in Sawtooth's PoET,
- all miners can perform Proof of Work computations while their TEE waits for a specific duration $\mathsf{mD} + noise$, and
- the value of $noise$ is node-specific, cannot be adversely manipulated by a node and provides sufficient variance for actual waiting times across the network.

The first requirement gives us the ability to force nodes to wait for a specified amount of time before performing a particular action, here announcing a solution for Proof of Work. The second ensures that nodes can mine for such a solution while their TEE performs the waiting required and its attestation. The third is there to prevent all nodes from sending Proof of Work solutions at the same time and that they cannot adversely influence the uncertainty in such variance. This situation would be problematic as it may create frequent, unnecessary or malicious forks.

For modeling and analysis, it is a conservative abstraction to assume that the wait times for all m_A nodes of the attacker are only mD. We may similarly stipulate that the attacker has an additional advantage by saying that the wait times (rather the time available to find Proof of Work) for all m_B benign nodes is $\mathsf{mD} - \epsilon$ for some value $0 \leq \epsilon$. Setting $\epsilon = 0$ eliminates such an advantage from the model and its analysis.

To model the effects of PoKW, we require the parameter s_k^r which denotes the size of the set PK^{r-k}, the superset of eligible keys computed by the first two filters F1 and F2 before applying cryptographic sortition. Like n, the value of s_k^r evolves over time. In our model, we need the constraint:

$$0 < n_W < s_k^r \tag{24}$$

As this guarantees that the threshold n_W/s_k^r for cryptographic sortition is an element in the open interval $(0, 1)$ and thus represents a non-trivial probability for selection to the committee.

We can now compute the expected number n_A of nodes that the attacker controls in the committee of expected size n_W. The probability that any node is in that committee is independent of whether an attacker controls that node. Therefore, we have:

$$n_A = \lceil pc \cdot n_W \rceil \tag{25}$$

The application of $\lceil \cdot \rceil$ is conservative as this may add 1 to that expected size of committee nodes controlled by the attacker. We can model the expected number n_B of nodes in the committee that are benign as:

$$n_B = n_W - n_A \tag{26}$$

What is the probability that the expected n_A nodes in the committee and controlled by the attacker find a Proof of Work within t seconds? Since they all have the same hash rate ρ_A, we compute that as:

$$p(Succ[\rho_A, n_A] \leq t) = 1 - e^{-(\rho_A \cdot n_A/d) \cdot t} \tag{27}$$

Similarly, we have:

$$p(Succ[\rho_B, n_B] \leq t) = 1 - e^{-(\rho_B \cdot n_B/d) \cdot t} \tag{28}$$

As the probability that the expected n_B benign nodes on the committee find Proof of Work within t seconds. We also add a constraint:

$$\rho_B < \rho_A \tag{29}$$

Saying that the attacker has a higher hash rate, and this may be further qualified, e.g., as in $10 \cdot \rho_B < \rho_A$ to say that the attacker's hash rate is at least one order of magnitude greater than that of benign nodes.

Given the wait time $\mathsf{mD} + noise$ and its above abstraction, we are interested in the following two probabilities:

$$p_A = p(Succ[\rho_A, n_A] \leq \mathsf{mD}) = 1 - e^{-(\rho_A \cdot n_A/d) \cdot \mathsf{mD}} \tag{30}$$

$$p_B = p(Succ[\rho_B, n_B] \leq \mathsf{mD} - \epsilon) = 1 - e^{-(\rho_B \cdot n_B/d) \cdot (\mathsf{mD} - \epsilon)} \tag{31}$$

We want to use these probabilities to derive a conservative probability for the attacker to win a mining race. We can derive this probability based on 4 possible events:

E_{AB} at least one of the expected n_A nodes finds Proof of Work within mD seconds, and at least one of the expected n_B nodes finds Proof of Work within $\mathsf{mD} - \epsilon$ seconds,

E_A at least one of the expected n_A nodes finds Proof of Work within mD seconds, and none of the expected n_B nodes finds Proof of Work within $\mathsf{mD} - \epsilon$ seconds,

$\epsilon = 0$	$n = 10^6$	$n_w = 64$
$\rho_A = 10^9$	$\rho_B = 4 \cdot 10^3$	$pc = 0.3$

Table 1: Values that illustrate how we may determine a suitable value for the minimum wait time mD for specific values of d

E_B at least one of the expected n_B nodes finds Proof of Work within $\mathsf{mD} - \epsilon$ seconds, and none of the expected n_A nodes finds Proof of Work within mD seconds, and

$E_\emptyset$ none of the expected n_A nodes finds Proof of Work within mD seconds, and none of the expected n_B nodes finds Proof of Work within $\mathsf{mD} - \epsilon$ seconds.

Intuitively, event $E_\emptyset$ should have a probability close to 0 if we want Proof of Work solutions found within mD seconds. Event E_B should also have small probability unless the attacker controls so few nodes that the benign nodes compensate for that even with lower hash rates. The events E_A and E_{AB} are intuitively the most likely ones for desired configuration values such as mD and d.

If event E_{AB} occurs, we may say that both malicious and benign nodes can propagate a solution at or soon after time mD. It also seems reasonable to say that the probability that a solution from a malicious node accepted globally is 0.5 – conditional on event E_{AB} and given that there are competing "malicious" and "benign" solutions propagated in the network.

If event E_B occurs, we can say that a Proof of Work offered by a benign node is accepted with probability close to 1. We can express the probability that a benign node wins the next mining race as:

$$0.5 \cdot p(E_{AB}) + 1 \cdot p(E_B) \tag{32}$$

Since the search for Proof of Work is independent across miners, we can compute:

$$\begin{aligned} p(E_{AB}) &= p_A \cdot p_B \\ p(E_B) &= p_B \cdot (1 - p_A) \end{aligned} \tag{33}$$

Adding this to (32) yields:

$$\mu = 0.5 \cdot p_A \cdot p_B + 1 \cdot p_B \cdot (1 - p_A) = p_B \cdot (1 - 0.5 \cdot p_A) \tag{34}$$

as the "probability" for a benign node to determine the next block in this mining race.

Let us illustrate this with some concrete numbers by setting values as shown in Table 1. To illustrate the metric μ, Table 2 shows values for it when d is 16 and when d is 64, we computed these results by evaluating the expression for μ in (34) in Haskell.

These results suggest that we need to choose mD carefully to get a fair balance between the faster mining power of an attacker and the lower mining power of benign nodes. The values in these tables also suggest that a combination of PoET and PoKW holds promise. Moreover, for larger values of d we seem to be able to decrease the minimal wait time without compromising much on the fairness of this combined leader election process: for $\mathsf{mD} = 0.001$ the value of μ increases from about 0.22 for $d = 16$ to about 0.936 for $d = 64$.

4.2 Sybil attacks

In the above model, we implicitly assume that an attacker cannot replicate compute devices to increase the probability of solving Proof of Work within t seconds. This assumption may not be

d	mD	μ
16	1.0	1.0
16	0.1	0.999999999986112
16	0.01	0.917915001376101
16	0.001	0.221199216928595
16	0.0001	0.0246900879716674
64	1.0	1.0
64	0.1	1.0
64	0.01	0.99999999999886
64	0.001	0.936072138793292
64	0.0001	0.240427876775032

Table 2: Illustration of the value μ in (34) for the values shown in Table 1

valid at all. An attacker may clone public keys, elected into the committee for mining the next block on other devices, and so may amplify their computational power in this leadership race. Our model still makes sense in such a setting if we choose n_A to be the estimated number of such devices that an attacker would bring to this task.

Capital expenditure costs inform estimates for devices, monetary or other incentives in winning a leadership race, and other factors such as the applications supported by the blockchain system. The public keys used in Cryptographic Sortition for PoKW may be those of the TEE, which would make cloning and thus Sybil attacks much harder to operationalize.

4.3 Optimization for PoKW + PoET

We can now use similar methods as developed in Section 3 to optimize the composition of PoKW and PoET. Intuitively, we would like to:

- maximize the value of d for better security,
- minimize the value mD so that the announcement of new blocks is not delayed too long,
- minimize a (weighted combination of) capital expenditures (e.g., cost of mining units and TEEs) and operational expenditures (e.g., energy costs for mining), and
- minimize the probability that attackers can mine a certain number of successive blocks.

This as a multi-objective optimization problem. We follow the approach of Section 3 when exploring trade-offs between these objectives for a public version of our blockchain system in which each node has a TEE such that blocks are signed with a private key from that TEE.

The latter seems to make it hard to launch effective Sybil attacks. Also, the combination of PoKW and PoET within such a public, open network and the apparent hardness of any Sybil attacks mean that we do not have to model the energy costs of mining. Instead, we can rely on the low-energy aspect of PoKW and favor lower values of n_w to higher ones when looking for optimal solutions.

We could therefore consider a MINLP optimization problem over the set of constraints depicted in Figure 7. These constraints stipulate that we assume the attacker is able to control just over a third of all network nodes, all which having a high hash rate ρ_A. Nodes not controlled by the attacker have hash rate ρ_B consistent with use of a smartphone. The expected committee size n_w is bounded by above and below, as is the level of difficulty d, and the minimal waiting time mD for PoET. The probability that some node not controlled by the attacker finds Proof of

Work within mD seconds is specified to be at least 0.9999, and the probability that the attacker wins 6 consecutive mining races is made sufficiently small.

$$
\begin{array}{rcl}
pc & = & 0.34 \\
16 & \leq & n_w \leq 64 \\
4 & \leq & d \leq 64 \\
0.001 & \leq & \mathsf{mD} \leq 15.0 \\
\epsilon & = & 0 \\
n_A & = & \lceil pc \cdot n_w \rceil \\
n_B & = & n_w - n_A \\
\rho_A & = & 10^9 \\
\rho_B & = & 4 \cdot 10^3 \\
p_A & = & 1 - e^{-(\rho_A \cdot n_A / d) \cdot \mathsf{mD}} \\
p_B & = & 1 - e^{-(\rho_B \cdot n_B / d) \cdot (\mathsf{mD} - \epsilon)} \geq 0.9999 \\
\mu & = & p_B \cdot (1 - 0.5 \cdot p_A) \\
(1-\mu)^6 & \leq & 10^{-6}
\end{array}
\tag{35}
$$

Figure 7: Example of a MINLP system of constraints for finding suitable parameters for a combination of PoKW and PoET consensus mechanism. We assume the attacker is able to control just over a third of all network nodes, all which having a high hash rate ρ_A. Nodes not controlled by the attacker have hash rate ρ_B consistent with use of a smartphone. The expected committee size n_w is bounded by above and below, as is the level of difficulty d, and the minimal waiting time mD for PoET. The probability that some node not controlled by the attacker finds Proof of Work within mD seconds is at least 0.9999, and the probability that the attacker wins 6 consecutive mining races is sufficiently small

We can then solve optimization problems over this set of constraints, using the robust optimization techniques of Section 3, and by using appropriate objective functions. For example, constants u_w and u_d may model the strict uncertainty in the actual value n'_w of n_w and d' of d, respectively:

$$
\mid d' - d \mid \leq u_d \qquad\qquad \mid n'_w - n_w \mid \leq u_w \tag{36}
$$

As before in the governed blockchain setting, we can consider tuples – here of form (d, n_w) – and define a tuple (d, n_w) to be feasible if all d' and n'_w that satisfy (36) make the set of constraints in Figure 7 feasible for d' and n'_w instead of d and n_w. We can then list these tuples in a preferred lexicographical order, for example those that maximize d and then minimize n_w.

It is less clear whether or how to optimize the value of mD, other than constraining it within defined bounds, and to demand specific guarantees, such as those for μ in Figure 7. Making mD too small may increase the attack surface on the network and its latency aspects. Making μ too big may slow down the combined PoET + PoKW consensus mechanism.

An attractive feature of PoKW is that its protection against an attacker's ability to control nodes is scale invariant in the sense discussed in Section 3.5. This MINLP model reflects this since it reasons about networks of any size.

Our model of the attacker of PoET + PoKW is conservative in its classification of hash rate capabilities for malicious and benign network nodes. The need for *robust* optimization may,

therefore, be less pronounced. However, lack of robustness may flag issues that we would have to address with further modeling prior to implementation.

5 Mitigating Potential Attack Vectors

We now discuss potential vectors through which an adversary or a group of adversaries could attack this PoKW-based system, and how we may mitigate such threats. First, let us state the underlying security model.

> **Security Model:** The overall security model is that all nodes trust the current Blockchain that they know, and may not trust anything else. Also, we assume that an adversary or group of adversaries cannot compromise more than a certain percentage pc of *all* network nodes, at any given point in time.

5.1 Cryptographic Sortition

Let us analyze the use of Cryptographic Sortition. We assume that the initial seed Q^0 in the system's Genesis Block B^0 will be generated by a cryptographically strong pseudo-random generator. If the Genesis Block is created by a central system authority, the latter could create the seed for the generation of Q^0 from a high-entropy source. One could alternatively generate this seed using decentralized protocols for verifiably generating secrets, e.g., the JF-DKG protocol [7], if there is a well defined initial set of nodes who would participate in this process. The latter approach may produce more trust into the initial block data B^0, including the value of Q^0.

Similar considerations apply for the creation of the next k blocks on the chain. These initial k blocks may, for example, have empty transaction payloads apart from seeding the blockchain with a suitable set of public keys that would form the superset of eligible keys for mining and other system actions. Access to mining and other actions would then be governed by the PoKW specification for blocks of blockheight larger than k.

The generation of subsequent seeds Q^r is such that it is a deterministic function of the previous seed Q^{r-1} – whose value we trust based on the consensus of the current Blockchain – and of the digital signature of some node pk_i. An adversary cannot change the format of this new seed since block validation would otherwise fail. An attacker may only try to create this signature with a key pair (pk_i, sk_i) of his choice, assuming he is able to compromise a certain percentage of all nodes.

But all such signatures would use a specific algorithm, e.g. ECDSA for a particular Elliptic curve. And we assume that an adversary cannot exploit any changes in digital signatures based on changes of private keys. This is a reasonable assumption as the creation of signatures acts like a (key-dependent) pseudo-random generator.

Let us now consider how an adversary could exploit knowledge of the new seed value Q^r, for example in a setting in which the adversary would first learn the new block B^r and could ensure that other nodes receive this block only after some delay. For those nodes which the adversary has already compromised, we may assume that the adversary knows the private keys of these nodes. Assuming that the set of public keys PK that are, in principle, eligible to participate in a mining race for B^{r+1} is computable from the current Blockchain, the adversary therefore can evaluate the entire body of $mayPerform_{pk_i}(r, mine)$ for compromised nodes pk_i to determine which of these nodes are eligible to mine.

However, if they have not yet compromised a node pk_j, then they won't know its secret key sk_j and so cannot evaluate the body of $mayPerform_{pk_j}$. Therefore, the attacker cannot know which of the nodes they have not yet compromised will be able to participate in the next mining race. This means that they have no additional information that may help them determine which

additional nodes to compromise next – assuming that such actions take effort, time, and can only be done for a limited number of nodes. Therefore, an adversary may as well resort to compromising nodes randomly for the purpose of influencing mining races.

This means that they have to compromise a percentage pc of *all* network nodes to guarantee that they compromise an expected percentage of pc of eligible nodes for each mining race. To make this concrete, assume that n_w equals 50 and that there are $100,000$ nodes in the network. Then an adversary needs to compromise $10,000$ nodes if they expects to compromise 5 of the 50 nodes that will be eligible to mine the next block.

5.2 Sybil attacks

Another threat is that an adversary who knows a private key sk_i can duplicate this key across fast PoW devices to engage in a private, parallel thread of the current mining race. Depending on the number and types of devices used for this, the adversary may gain a considerable advantage in winning the mining race. Specifically, the probability that the adversary will compromise at least one node that is eligible to mine the next block can be seen as equal to:

$$1 - (1 - pc)^{n_w} \tag{37}$$

For example, for $n_w = 50$ and $pc = 0.1$ this gives us a probability of 0.99484622479268 that the adversary controls at least one of these eligible nodes. However, the discussion in Section 4 suggests that this may not be a grave concern whenever the public network has sufficiently many benign nodes of similar hash rates available – especially in combinations of the PoKW and PoET consensus mechanisms. This is certainly less of a concern in an enterprise setting.

Moreover, for highly sensitive applications, we may also mitigate against or even prevent such usage of more powerful mining machine by embedding such private keys securely into devices, using Physical Unclonable Functions (PUFs).

5.3 Degrading quality of transaction sets

Another threat stems from the ability to propose empty, sparse or low quality transaction sets $TSet^r$ within blocks B^r. One could mitigate against this by insisting in the validation logic for blocks that only blocks that meet deterministic metrics (e.g. sufficiently many transactions in payload) be validated by network nodes.

The filters F1 and F2 described above could also be used to identify problematic behavior and thereby remove nodes from the white list. However, such adaptive mechanisms such as machine learning are themselves attack surfaces (see e.g. [18]) and so would have to be designed and verified with extreme care.

5.4 Attacking system control mechanisms

Naturally, these control mechanisms themselves may be subject to attack and manipulation. Control mechanisms, including those based on policy, would need to be internally consistent so that they would not offer denial-of-service type attacks on their own access-control logic. Formal verification has already been applied extensively in the area of access-control policies, and such validation could therefore also be done for smart contracts that embody such policies.

5.5 Reconciling needs for change management and resiliency

The strength of smart contracts, as compute engines that are immutable and for which there is system consensus, also reflects their weakness, the limited ability to manage change. This needs to be reconciled with support of the entire life cycle of a system. We may achieve this,

e.g., through voting mechanisms within smart contracts, similarly to how Bitcoin handles major system configuration changes through a vote of miners. We think that such voting mechanisms will offer an ability to recover from major system incidents, should an attack of an adversary ever produce systemic damage that requires stability-preserving system repair.

6 Practical Reasoning & Implementation

The Ethereum platform offers several beneficial features that make it a great fit into the XAIN software stack. Currently, the public Ethereum network reaches consensus based on PoW. However, the Ethereum platform does not mandate the use of PoW for reaching consensus on the next block. For example, the Ethereum Foundation plan to replace PoW, on the public Ethereum Blockchain, with a PoS consensus mechanism in the future [9]. Also, both the Rinkeby and Kovan public test networks have replaced PoW with a PoA consensus mechanism. Furthermore, JP Morgan's project to create a permissioned Ethereum network for Enterprise use, Quorum, offers the choice of multiple consensus mechanisms.

Implementing a consensus mechanism – such as PoKW – for a distributed network is not trivial. Leveraging the core components of Ethereum, such as smart contracts, make this task easier as they add an element of certainty about the state of the network as it evolves – helping to form part of an elegant solution.

Throughout this section, we will discuss the changes made to the Geth Ethereum client in implementing PoKW, and how we have leveraged smart contracts in our solution.

6.1 Modifying Geth

We chose to modify the Ethereum Foundation's go-ethereum (Geth) implementation of the Ethereum protocol in our initial implementation of PoKW for the following reasons:

- It is the most well maintained of the Ethereum Foundation's clients.
- Roughly 70 percent of the Ethereum nodes on the public network use the Geth client.
- Geth provides a programming interface for alternative consensus mechanisms to code against, in order to seamlessly plug in to the rest of the Ethereum protocol.

The consensus process within a Blockchain protocol can roughly be split into two loosely related parts:

- Verifying incoming, new blocks that have been communicated by peers.
- Rules for establishing when a new block can be proposed.

High-level overviews of how the aforementioned aspects of a consensus mechanism currently work in Geth can be seen in Figures 8 and 9, respectively.

Block verification process (a) If a newly received block has already been seen or is not considered valid, then the process terminates; the block is discarded and no updates are made to the local copy of the Blockchain. (b) The "validate block" activity as seen in Figure 8 represents the steps of ensuring that the contents of the newly received block conform to the specifications as laid out in the Ethereum Yellow Paper [20]. In reality, a block is not always communicated as a single entity; therefore, the validation of a block's header and the validation of a block's body are treated as separate functions in Geth for code re-usability, but are treated as a whole in Figure 8 for sake of brevity.

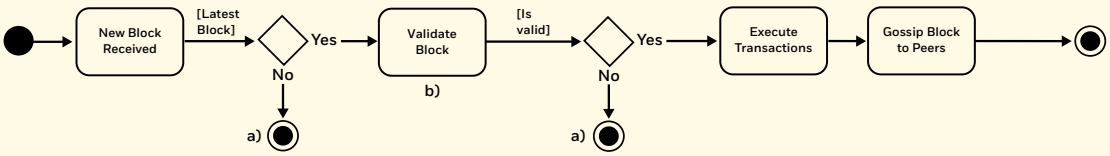


Figure 8: Block Verification Process

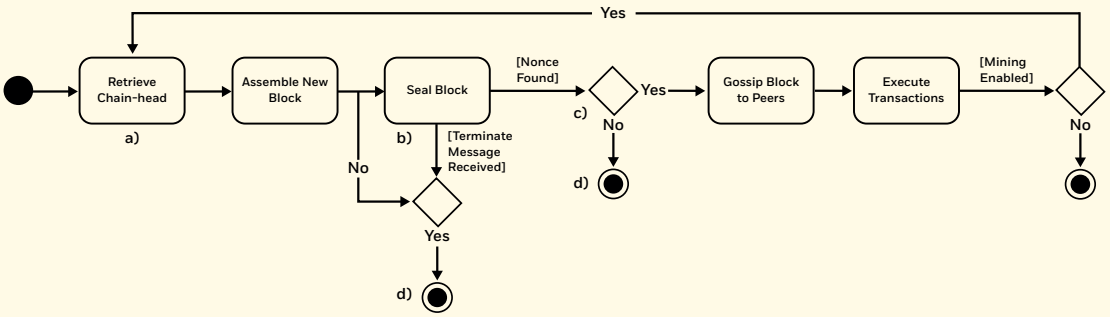


Figure 9: Geth's Block Proposal Process

Geth's block proposal process (a) A new block will always be built upon the latest block of the longest chain with the greatest total difficulty. The process of proposing a new block starts by retrieving from the node's local Blockchain this latest block; which could be the block just mined by this node. (b) The act of "sealing" a block, as seen in Figure 8, is specific to the chosen consensus mechanism, for example it could be achieved by finding an appropriate nonce using PoW or by adding a digital signature in PoA. Sealing a block can take several seconds (depending on the computational capability of the node) and during this time a new, valid block could have been mined by a peer or a node's administrator could have issued a command to terminate mining. It is, therefore, necessary to be able to prematurely terminate the sealing process - and discard the block that was being sealed by the node. (c) It is possible that a sealing process on a node could exhaust the assigned range of nonce values without finding a solutions that creates a valid block - if this happens, the process will wait for a new, latest block to build upon; before starting the process of proposing a new block again. (d) If the block proposal process terminates before a new, valid block has been successfully sealed, then the block that was being worked upon will be discarded and no updates to the local Blockchain will result from that block proposal process.

6.1.1 High-level Overview of Modifications

Block verification process The modifications to Geth required to implement PoKW, at a high-level, did not require any changes to the block verification process as outlined in Figure 8. However, the implementation of PoKW did require additional information to be added to a block's header and as such the block header validation phase of the block verification process had to be extended. The additions to the block header are detailed in Figure 11 and details about the extra validation can be found in Figure 8.

Geth's modified block proposal process Figure 10 provides a high-level overview of the changes that were made to Geth in order to modify the block proposal process to implement

the PoKW algorithm.

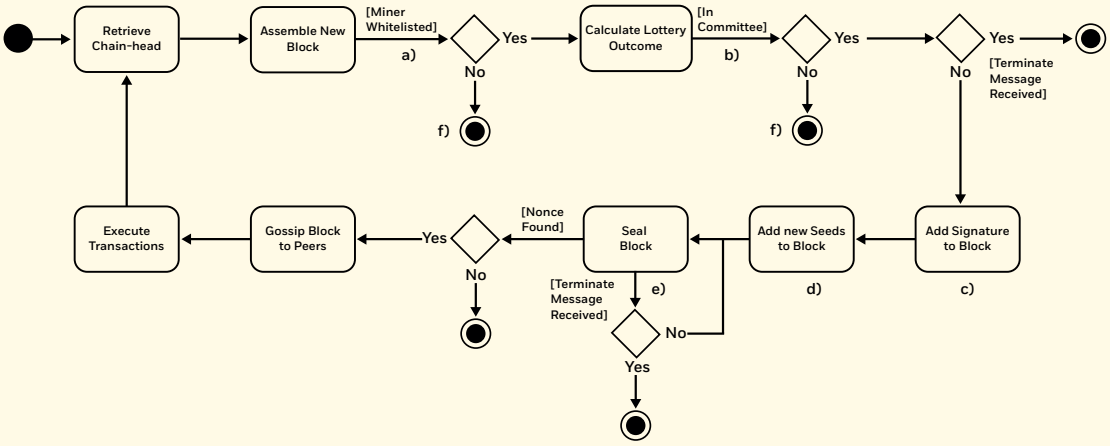


Figure 10: PoKW Block Proposal Process

(a) The super-set of all block proposers is maintained in white list L. A node must first ensure that it is in L by querying the smart contract that publicly governs this list. (b) If a node is in L, it must run a secret lottery to see if it has been selected to be part of the committee for sealing block B^r. (c) A signature, generated in the secret lottery process, using elements of the block's header, is created and added to the block to authorize the block. (d) The new seed Q^{r+1} is computed as a hash of the signature of Q^r by the leader who mined the new block. The new block structure is described in more detail in Section 6.2. (e) The existing PoW sealing algorithm has been largely kept the same with the addition of new attributes to the block header to cryptographically secure them. (f) If a node is not required as a block producer for this blockheight, then they will move on to the next step of checking to see if they are in either of the committees performing the machine learning algorithms for B^r.

6.2 Comparison of block structure

In Ethereum, the header of a block can be passed around without its corresponding block – as defined in the official ĐΞVp2p Wire Protocol [6]. This is particularly useful for clients that do not want to store the entire Blockchain, but still want to participate in the network operations. This approach maintains integrity, even without access to the full Blockchain, since every block contains the root of a Patricia Merkle tree for the corresponding aspects of the block body (see Figure 11 for more details). The result is that, when a client who does not store the entire Blockchain wants to verify that a transaction was indeed mined within a given block, that client only need to request the respective block body – thus saving on the data transfer and storage overhead. The corresponding, verified, Merkle tree root in the block header (and block hash) can then be checked against the received block body to confirm, with overwhelming certainty, that the data received is valid and has not been modified.

The implementation of PoKW, therefore, requires additions to the structure of the Block Header. Figure 11 shows a comparison of the standard Ethereum block structure and the modifications necessary to support the PoKW consensus mechanism:

Block structure additions: The extended block header contains the a number of seeds, one for each task, required to carry out PoKW. These seeds are all deterministically derived from

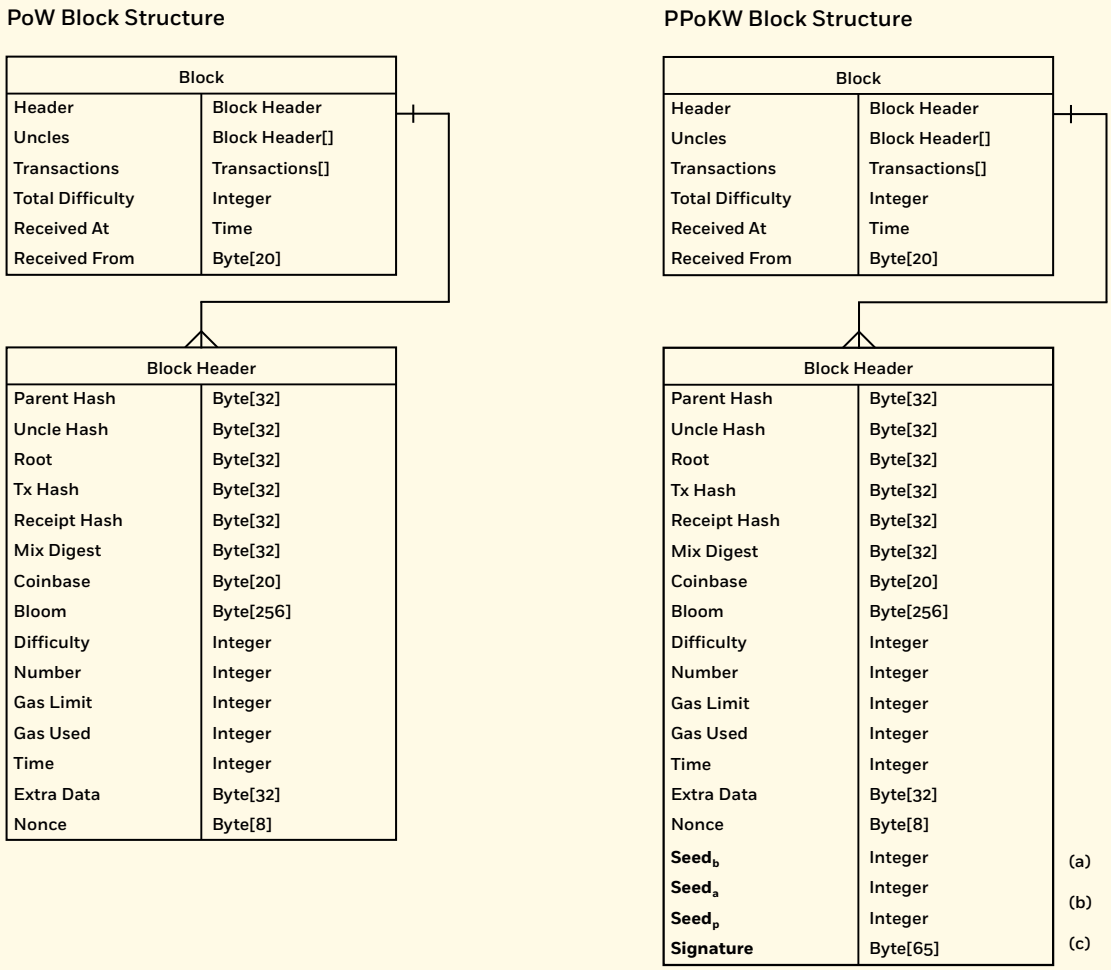


Figure 11: Block Structure Comparison

the seed Q^r as a function of the task to be performed: (a) The seed Q_b^r is used to determine the committee members for mining the next block. (b) A seed Q_a^r may be used for determining the anomaly-detection committee members. (c) A seed Q_p^r may be used for determining those committe members that mean to alter Blockchain parameters. The signature that is used to verify the block producer's public key and that the block producer was selected to be part of the committee has also been added to the block header.

6.3 Verifying Blocks

Upon receiving a newly proposed block B^r from a peer, the contents of B^r are first scrutinized to see whether its aspects are considered valid. If so, necessary changes are applied locally and B^r will be gossiped to any known, connected peers. Otherwise (i.e. the block header or any transaction contained within the block is deemed to be invalid), the validation process fails fast: the block is dropped and the node's local copy of the Blockchain remains unchanged.

PoKW extends the validation rules defined in Ethereum's Yellow Paper by also requiring the block producer to be present in white list L and having been selected to be part of the

respective committee for blockheight r to consider B^r as valid.

6.3.1 Identifying the Block Producer

The header of B^r is expected to contain a digital signature of form $SIG_{pk_i}(m)$. Given that we know the elements, contained within the block header, that were used to create m, and since we know that the private key used to sign m lies upon the Elliptic Curve (secp256k1), we are able to establish pk_i from $SIG_{pk_i}(m)$ - without knowing sk_i.

We can be certain that it was improbable that anyone could have created $SIG_{pk_i}(m)$ without knowing sk_i, giving a strong guarantee that B^r originated from the node identified by pk_i.

An Ethereum address related to pk_i (denoted A_{pk_i}), that also lies on the same Elliptic Curve (secp256k1), can also be deterministically established. Address A_{pk_i} can, therefore, also be used to identify a node, rather than its public key directly. Using A_{pk_i} to identify a node has the benefit of being natively supported by smart contracts, similarly to how JP Morgan's Quorum white-lists nodes [10]. We expect that the coinbase address in the block headers will match A_{pk_i}, but this is not necessary in order for a block to be considered valid.

6.3.2 Approaches to White-listing Nodes

The white list of Ethereum addresses, eligible in principle to be part of a committee, is expected to change over time – new entities will join the network; and will eventually have their addresses added to the white list. Moreover, existing, known entities may be compromised or may no longer desire themselves to be part of the service and will have their addresses removed from the white list. Properties that we desire from the white list L are that:

- L should be known by all nodes on the network, and agreed upon.
- L should be dynamic.
- It should be possible to restrict access to which nodes can change L or provide logic for under what conditions L can change.
- All nodes should agree upon any changes made to L.
- L should exist already by the time the Genesis Block B^0 has been created and distributed.

Using a smart contract to represent L, therefore, seems to be an elegant solution. The white list, smart contract and the initial white-listed nodes are embedded in the Genesis block – at a fixed and preselected address. Given that we are able to identify, with a large degree of certainty, the identity of a nodes pk_i (from the set PK), it is then relatively easy to look up the related address in the smart contract, white-list and quickly establish if this node pk_i is in the superset PK of all possible block producers. If this is the case, then the protocol proceeds onto the next step of establishing the eligibility of this node to create the block at this height, that is where they are secretly selected to form the committee C_r of those who are eligible to participate in the mining race for block B^r, as specified in function $mayPerform_{pk_i}(r, mine)$.

A node pk_j can verify whether some other node pk_i from PK was a member of that committee C^r by running $mayPerform_{pk_i}$ with the parameters that pk_j has access to on the current Blockchain and set PK. If this function returns `false`, the block B^r is rejected and no updates to the local version of the Blockchain are made.

```
pragma solidity ^0.4.18;

contract AuthorisedMinersWhitelist {

    mapping(address => bool) whitelist;
    uint32 public size;

    event AddedToWhitelist(address miner);
    event RemovedFromWhitelist(address miner);

    function isAuthorisedMiner(address miner) public view returns (bool) {
        return whitelist[miner];
    }

    function authoriseMiner(address miner) public {
        require(! whitelist[miner]);
        whitelist[miner] = true;
        size = size + 1;
        AddedToWhitelist(miner);
    }

    function removeMinersAuthorisation(address miner) public {
        require(whitelist[miner]);
        whitelist[miner] = false;
        size = size - 1;
        RemovedFromWhitelist(miner);
    }
}
```

Figure 12: A simple implementation of a White list that meets most of the requirements listed in Section 6.3.2

6.3.3 Smart Contract, White-List Code

The Solidity code fragment in Figure 12 shows a simple implementation of L that fulfills most of the criteria outlined in Section 6.3.2, with the exception of restricting which addresses are able to modify L. This can be addressed in extended versions of this contract. Another enhancement would be to switch from a binary `true` or `false` membership of the White list to a more sophisticated, e.g. quantitative, solution that would be more difficult for an adversary to attack.

6.4 Proposing a New Block

Upon receiving a new, valid Block B^r that becomes the head of the chain, each of the nodes looks to extend the Blockchain by building upon B^r.

It is possible that the latest block could have included a transaction which modified L, so nodes cannot presume their eligibility (or lack of) to be part of the committee of block producers.

Each node pk_i queries the read-only function of the whitelist L, smart contract created in the Genesis block, to establish whether pk_i is in the super-set of block producers and should proceed to determine whether pk_i is in the selection committee C_r for mining the next block B^r.

6.4.1 Establishing the Committee

The process of determining whether a node has been selected to be part of the committee C_r is ran by every node that is in L. Each such node independently executes the function $mayPerform_{pk_i}(r, mine)$ which will output whether they have been elected as part of C_r.

The signature in (38), calculated in (2) as part of the Cryptographic Sortition function in $mayPerform_{pk_i}(r, mine)$, is used as the authorization signature in the block header – that validating node pk_j can use to retrieve pk_i, but it can then also be used by a pk_j to verify that pk_i was elected to be part of C^r.

$$SIG_{pk_i}(r, Q^{r-1}) \tag{38}$$

6.4.2 Seed Generation

Since the seed Q_b^{r-1} and optional ones for other tasks, e.g. Q_a^{r-1} and Q_p^{r-1}), are integral to the PoKW consensus mechanism, it is important to make sure that an adversary cannot manipulate them to gain an advantage or undermine the system.

The integrity and trust in the seed Q^{r-1} is ensured by first embedding the initial seed Q^0 in the Genesis block. All subsequent seeds Q^r are derived from Q^{r-1} and this process can be traced all the way back to Q^1 relating to Q^0. Seed Q^0 is generated using a trusted and verifiable function across all of the addresses that will be present in the initial white list L^0. The seed value added to the block header is an integer that is arrived at using the expression:

$$0.H(SIG_{pk_i}(Q^{r-1})) \tag{39}$$

6.4.3 Finding the Nonce

PoKW uses the existing PoW logic in Geth to find a nonce that results in a block hash with the appropriate difficulty. Keeping this code the reduces the technical-footprint; including the amount of changes required to the validation logic.

7 Summary & Conclusion

In this yellow paper, we described a new consensus algorithm *Proof of Kernel Work* (PoKW) that is well suited for dynamic low-energy networks such as those found in IoT and in intelligent transportation systems. Proof of Kernel Work is based on Bitcoin's Proof of Work but uses Cryptographic Sortition (as proposed in the work on Algorand [17]) to create a publicly verifiable random function written onto the blockchain. This function is then used to randomly select an expected number of network nodes to perform critical system tasks, notably the mining of the next block.

This approach greatly reduces the energy consumption of the mining process whilst also forcing an adversary to scale the effectiveness of node compromises to actual network size, independent of the expected number of nodes selected for mining. We presented a mathematical model for robust optimization that explores the trade-offs between security, cost, and availability of this PoKW-based approach to compute optimal, initial system parameters. This analysis was also extended to a combination of PoKW with *Proof of Elapsed Time* (PoET), since such a combination offers attractive means of making the consensus mechanism more democratic and even less amenable to Sybil attacks.

We also discussed attack vectors for this approach and measures for their mitigation. Finally, we reported on a practical implementation of a PoKW blockchain system based on Ethereum and the Geth client.

To summarize, the XAIN PoKW-based framework rests on two pillars:

- a new consensus mechanism, Proof of Kernel Work (PoKW), a variant of Proof of Work based on Cryptographic Sortition, the latter was introduced for Algorand in [17], and
- an adaptation of existing Blockchain and enterprise technology stacks for the XAIN framework, based on the governed Blockchain approach in [14].

We expect to publish separate research papers about the XAIN PoKW-based framework, as well as its open-source software, in due course.

References

[1] BEN-TAL, A., GHAOUI, L. E., AND NEMIROVSKI, A. *Robust Optimization*. Princeton University Press, 2009.

[2] BONNEAU, J., MILLER, A., CLARK, J., NARAYANAN, A., KROLL, J. A., AND FELTEN, E. W. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015* (2015), pp. 104–121.

[3] CASTRO, M., AND LISKOV, B. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999* (1999), pp. 173–186.

[4] CHEN, J., AND MICALI, S. ALGORAND. *CoRR abs/1607.01341* (2016).

[5] DWORK, C., AND NAOR, M. Pricing via processing or combatting junk mail. In *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings* (1992), pp. 139–147.

[6] ETHEREUM COMMUNITY. ÐΞVp2p Wire Protocol. Online at `https://github.com/ethereum/wiki/wiki/ÐΞVp2p-Wire-Protocol`, 2017.

[7] GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology 20*, 1 (2007), 51–83.

[8] GILAD, Y., HEMO, R., MICALI, S., VLACHOS, G., AND ZELDOVICH, N. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017* (2017), pp. 51–68.

[9] HERTIG, C. A. Ethereum's Big Switch: The New Roadmap to Proof-of-Stake. Online at `https://coindesk.com/ethereums-big-switch-the-new-roadmap-to-proof-of-stake/`, 2017.

[10] J.P. MORGAN CHASE. Quorum. Online at `https://jpmorgan.com/global/Quorum`, 2017.

[11] JÜNGER, M., LIEBLING, T. M., NADDEF, D., NEMHAUSER, G. L., PULLEYBLANK, W. R., REINELT, G., RINALDI, G., AND WOLSEY, L. A., Eds. *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*. Springer, 2010.

[12] LUNDBAEK, L., D'IDDIO, A. C., AND HUTH, M. Optimizing governed blockchains for financial process authentications. *CoRR abs/1612.00407* (2016).

[13] LUNDBÆK, L., D'IDDIO, A. C., AND HUTH, M. Centrally governed blockchains: Optimizing security, cost, and availability. In *Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday* (2017), pp. 578–599.

[14] LUNDBAEK, L., AND HUTH, M. Oligarchic control of business-to-business blockchains. In *2017 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2017, Paris, France, April 26-28, 2017* (2017), pp. 68–71.

[15] NAKAMOTO, S. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2008. Published under pseudonym.

[16] NARAYANAN, A., BONNEAU, J., FELTEN, E., MILLER, A., AND GOLDFEDER, S. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[17] NUÑEZ, D., AGUDO, I., AND LOPEZ, J. Proxy Re-Encryption: Analysis of constructions and its application to secure access delegation. *J. Network and Computer Applications 87* (2017), 193–209.

[18] RUBINSTEIN, B. I. P., NELSON, B., HUANG, L., JOSEPH, A. D., LAU, S., RAO, S., TAFT, N., AND TYGAR, J. D. ANTIDOTE: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference, IMC 2009, Chicago, Illinois, USA, November 4-6, 2009* (2009), pp. 1–14.

[19] VIGERSKE, S. MINLP Library 2. Online benchmark repository available at `http://www.gamsworld.org/minlp/minlplib2/html/`.

[20] WOOD, G. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. On Github. EIP-150 REVISION.