Practical Proof of Kernel Work & Distributed Adaptiveness

A Resilient & Scalable Blockchain Platform for Dynamic Low-Energy Networks

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

1 **Introduction** 4  
1.1 Motivation & Aims ........................................ 4  
1.2 Intended Audience ........................................ 4  
1.3 Outline of Yellow Paper .................................. 5  

2 **Computational Backgrounds** 5  
2.1 Cryptographic Foundations ................................. 5  
2.1.1 Finite Fields ........................................... 7  
2.1.2 Cryptographic Functions ............................... 8  
2.1.3 Elliptic Curve Cryptography ........................... 11  
2.1.4 Post-Quantum Cryptography ............................ 15  
2.1.5 Random Oracle Methodology ........................... 17  
2.2 Linear Programming ....................................... 17  
2.2.1 Solution Algorithms .................................... 18  
2.2.2 Integer Linear-Programming & Relaxation .......... 18  
2.2.3 Mixed Integer Non-Linear-Programming ............. 20  
2.3 Reinforcement Learning for Adaptive Engines .......... 20  
2.4 Foundations of Anomaly Detection ....................... 22  
2.4.1 Motivating Example .................................... 23  
2.4.2 Basic Notations for Anomaly Detection ............. 24  
2.5 Distributed Ledger Technology .......................... 24  
2.5.1 History of Distributed Ledgers ....................... 24  
2.5.2 Introduction to Blockchain Technology ............ 25  
2.5.3 Comparative Analysis: Consensus Algorithms .... 26  

3 **Approach to Access Control & Privacy By Default** 28  
3.1 Key Management Procedures ............................... 29  
3.1.1 Privacy .............................................. 29  
3.1.2 Key Management ....................................... 29  
3.2 Node White List ......................................... 29  
3.2.1 Regeneration of Keys .................................. 30  
3.2.2 Key Revocation Schemes ............................... 30  
3.2.3 GDPR Compliance ...................................... 31  

4 **Practical Proof of Kernel Work: PPoKW** 32  
4.1 Algorithmic Introduction to PPoKW ....................... 32  
4.2 Algorithmic Specification of PPoKW ..................... 33  

5 **Robust Consensus Optimization** 37  
5.1 Motivation .............................................. 37  
5.2 Mathematics for Governed PPoKW ......................... 38  
5.3 Mathematical Optimization in Mining Design Space .... 40  
5.4 Robust Design Security .................................. 43  
5.5 Discussion of Cryptographic Node Selection ........... 44
6 Automated Run-Time Stabilization of Network
   6.1 Machine Learning Engines in Permissioned Blockchains 46
   6.2 Environment Agent Model for PPoKW 46
   6.3 Q-learning 47
   6.4 Adaptive Neural Network Engines 48
   6.5 Training Adaptive Neural Network Engines 48
   6.6 Prior Work 50
   6.7 Action Validation 50
   6.8 Feature Adaption for Governed Blockchain Engines 51

7 Automatic Anomaly Detection for Node White-listing 51
   7.1 Detecting Anomalies in the Context of Blockchain 51
      7.1.1 Feature Representation for Training and Inference 51
      7.1.2 Feature Aggregation for Inference 53
      7.1.3 Feature Aggregation for Training 53
      7.1.4 Inference 54
   7.2 Decisions Space of Detector Nodes 54
   7.3 Distributed Inference in the Context of Blockchain 54

8 Mitigating Potential Attack Vectors 55

9 Practical Reasoning & Implementation 57
   9.1 Modifying Geth 57
      9.1.1 High-level Overview of Modifications 58
   9.2 Comparison of block structure 60
   9.3 Verifying Blocks 60
      9.3.1 Identifying the Block Producer 60
      9.3.2 White-listing Nodes 61
      9.3.3 Smart Contract, White-List Code 62
   9.4 Proposing a New Block 63
      9.4.1 Establishing the Committee 63
      9.4.2 Seed Generation 63
      9.4.3 Finding the Nonce 64
   9.5 Storage Implementation with IPFS and NuCypherKMS 64
      9.5.1 Controlling Access to Data 64
      9.5.2 NuCypher KMS an Implementation of Proxy Re-Encryption 65
   9.6 Distributed Storage Implementation 65

10 Summary & Conclusion 68
1 Introduction

1.1 Motivation & Aims

This yellow paper proposes the notion of Practical Proof of Kernel Work. Proof of Work (PoW) [4] is a by now well known concept that precedes the development of Bitcoin [38] and was originally proposed as a potential solution to contain spam emails [14]: 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 [50]. 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 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 three 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 [10]

- **Practical Adaptation of Existing Technology**: a realization of this Proof of Work reduction through an adaptation of existing Blockchain and enterprise technology stacks

- **Machine Learning for Adaptive System Resiliency**: the use of techniques from Artificial Intelligence to make our approach adaptive to system, network and attack dynamics.

These pillars are coordinated based on models for optimization. We call this reduced version of Proof of Work the *Proof of Kernel Work* (PoKW). The practical outlook reflected in the last two items above is referred to as *Practical Proof of Kernel Work* (PPoKW).

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

We review computational background in Section 2: cryptographic foundations, linear programming and optimization, reinforcement learning, and anomaly detection. Therefore, this section can well be used for a better understanding of the methods used in the later sections. In Section 3, we introduce a general setting of our system, while also discussing privacy aspects. Section 4 introduces our particular approach to a smaller yet still resilient mining race based on Proof of Work and a smaller kernel of nodes. Our models and tools for optimization of performance, security, and cost trade-offs are described in Section 5. How such optimization and machine learning supports automated run-time stabilization of the network is discussed in Section 6. The use of anomaly detection to manage white listing of network nodes is featured in Section 7. 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 8. The practical reasoning and implementation of our system is featured in Section 9 and the yellow paper concludes in Section 10.

2 Computational Backgrounds

2.1 Cryptographic Foundations

We briefly review those aspects of Cryptography most relevant for Blockchains.

Confidentiality How can an agent communicate with some other agent confidentially, so that no third agent is aware of the content of the transmitted message? This is a problem which also occurs in everyday life. Principally, one can solve it in three different ways:

1. Organizational measures: This includes the transmission of a message by a trustworthy messenger or the classification and access restriction of confidential stored documents.

2. Physical measures: This hides a message in a vault or sends it in a sealed letter. Or it may conceal the existence of the message by using secret ink.

3. Cryptographic measures: This distorts a message in such a way that it appears completely nonsensical to any third agent, but the agent who is the legitimate recipient can easily “decrypt” it. For this to work, the receiving agent requires a so called key.

These cryptographic methods can be grouped with respect to their underlying encryption process. They can be set up so that the sending agent also needs the secret key of the receiving agent; then both agents protect the confidentiality of their communication by their common secret. In this approach, the receiving agent can also send messages to the other agent since they share this secret. Therefore, these are so called symmetric encryption methods, since the roles of the sending and receiving agent can be reversed and both possess the same key.

However, there are also cryptographic methods in which the secret key remains exclusively with the receiving agent and where the sending agent encrypts the message only with the aid of publicly accessible data, for example with a so called public key. This approach is referred to as asymmetric encryption or public-key cryptography.

Authentication How can an agent be sure that a message he received really came from the specified sender? In everyday life, the answer to this question can be very obvious. For example, you may recognize your friend’s letter by her handwriting. However, such everyday mechanisms usually no longer work when we communicate with electronic media or when
some participating agents are machines or devices. The methods for solving these problems are referred to as achieving authentication. Hereby, we differentiate between

- peer-entity authentication, which intends to prove the identity of a person or entity, and
- message authentication, in which both the origin of a message, the changes in the message and perhaps its “freshness” are to be detected.

A typical example of peer-entity authentication is the verification of your identity against a computing device such as a smart phone. In general, it is obvious that one can prove one’s identity by showing that one possesses something that no one else has. We differentiate between three kinds of things that we may possess:

- firstly, a clearly distinguishable and unique biological property, e.g. a fingerprint
- secondly, the ability of proving to possess something unique, such as a particular device
- thirdly, the possession of unique knowledge, referred to as a secret, such as a password.

In cryptography, the latter is still the most widely used means of establishing one’s identity in practice, for example to prove one’s identity through a personal identity number (PIN).

These approaches are not perfect solutions. One may extract, e.g., someone’s fingerprint with some tape, and use that “stored information” as the (no longer unique) credential for proving identity. Putting such issues aside, a basic problem is how one can convince another party that one possesses that unique secret. Sending that secret, even in encrypted form, may be problematic as the receiving agent then also knows that secret and could use it to impersonate the sending agent. Cryptography has developed solutions to this, for example zero-knowledge protocols in which a sending agent can prove knowledge of a secret to a receiving agent without revealing any information about the secret in question. There are also methods by which the receiving agent can verify authenticity based on publicly accessible information.

**Anonymity** In many situations, not only the content of a message should remain secret, but also the identity of its sender and recipient should be secret – even the fact that these agents communicated at all. For example, in monetary transactions, one of the fundamental principles of cryptocurrencies, we expect such stringent secrecy and privacy guarantees.

Generally, in electronic communications it is difficult to reconcile the demands for anonymity and reliability. As such, the usual procedures implemented for today’s systems for electronic cash do not guarantee any anonymity. However, we will see that modern cryptographic protocols provide very efficient methods to provide a very high degree of anonymity, which we will discuss in more depth in latter parts of this yellow paper.

**Availability** We need to ensure that systems and their core functions remain available and that such availability cannot be compromised by an adversary who interacts with the system. Distributed Denial of Service attacks are a prominent example of attempts to compromise availability, and countermeasures include de-centralization of services.

**Cryptographic Protocols** If several people or entities share a common goal, they must cooperate and communicate meaningfully to realize such intent. In order to achieve a goal through communication, these agents must, however, adhere to certain mutually agreed upon rules. As such, the aggregation of such rules is commonly referred to as a protocol, for example:

*After the bank card is inserted, secret PIN number has to be entered (if the card is still valid and functions correctly). If the entered PIN is correct, the customer can*
select the amount, perhaps the denominations of bills or access additional services such as checking account balances; and then the requested money is dispensed by the ATM; both the bills and the bank card must be removed from the ATM within a certain period of time – otherwise the device recollects them for security reasons.

This example conveys that such protocols can be subtle, complex, and hard to reason about. Public key cryptography was originally designed to solve a main problems of classical cryptography: how to securely exchange a key for symmetric encryption between two agents and how to securely sign a digital object (see [13]). It soon became clear, however, that this new method had much greater potential beyond its original vision. The basic mechanisms of public-key cryptography serve as building blocks for today’s complex cryptographic protocols.

In recent years, however, the need and opportunity arose to not only implement basic services, such as encryption and authentication, but also complex cryptographic applications, such as secure mobile radio systems, digital wallets and even digital contracts. In order to implement such services, complex cryptographic protocols with high assurance guarantees are needed. Furthermore, today’s cryptographic services are not restricted to small, closed user groups, but are also applied to large and open systems, such as the Internet itself. Therefore, even traditional applications become service-oriented and so require protocol support.

A minimum and core requirement for any such systems is a sophisticated, reliable, and resilient approach to key management. The latter refers to systems and services that support the entire life cycle of cryptographic keys, including their creation, secure storage, and controlled retrieval. This management is implemented via appropriate cryptographic protocols.

There is an apparent circularity in this approach: cryptographic protocols that manage keys require keys themselves. This circularity is resolved by noting that the most important secrets are actually stored in unencrypted form: these are the keys for which no other keys protect their storage and retrieval. Risk Management of systems better be aware of this inescapable fact.

2.1.1 Finite Fields

Before specifying cryptographic functions that we require in the work reported below, we introduce the mathematical background of finite fields, which are used, e.g. in various protocols. A group \( G = (G, \oplus, e) \) is a set \( G \), a function \( \oplus : G \times G \rightarrow G \), and an element \( e \) in \( G \) such that

- \( e \oplus g = g \oplus e = g \) for all \( g \) in \( G \)
- \( \oplus \) is associative: for all \( g, h, k \) in \( G \) we have \( (g \oplus h) \oplus k = g \oplus (h \oplus k) \)
- for all \( g \) in \( G \) there is some \( h \) in \( G \) with \( g \oplus h = h \oplus g = e \).

The element \( h \) in the last item above is unique and called the inverse of \( g \), denoted by \( g^{-1} \). Group \( G \) is commutative if \( \oplus \) is a commutative operation: \( g \oplus g' = g' \oplus g \) for all \( g, g' \) in \( G \).

A field \( \mathbb{K} = (K, +, *, 0, 1) \) is an algebraic structure such that both \((K, +, 0)\) and \((K \setminus \{0\}, *, 1)\) are commutative groups and that the operations \( + \) and \( * \) satisfy the familiar distributivity relations. Examples of fields are the rational numbers \( \mathbb{Q} \), the real numbers \( \mathbb{R} \), and the complex numbers \( \mathbb{C} \) with their usual addition \(+\), multiplication \(*\), zero \(0\), and one \(1\).

An important class of fields for Cryptography are those fields \( \mathbb{K} \) for which \( K \) is a finite set. There are three basic but non-elementary facts about finite fields:

1. For every prime number \( p \) and \( n > 0 \), there is a field \( \mathbb{F}_p^n \) of size \( p^n \).
2. The size of every finite field \( \mathbb{F} \) is a power of a prime, i.e. of form \( p^n \) for some prime \( p \) and \( n > 0 \).
3. Two finite fields are isomorphic (i.e. they have the same size and algebraic structure) if and only if they have the same size.
Thus, a finite field is determined by its size, and only and all prime powers are sizes of finite fields. The former point also leads to performance optimizations, as one can move between different representations of a given finite field, e.g., to minimize the number of times one needs to compute a multiplicative inverse modulo a prime $p$ – a relatively expensive operation to call.

Subsequently, we write $\mathbb{F}_q$ to denote the uniquely determined finite field of prime power size $q = p^n$. A special case is $\mathbb{F}_p$ when $n$ equals 1: one representation of that finite field is the set of numbers $\{0, 1, \ldots, p-1\}$ where $+$ and $\ast$ are interpreted modulo the prime $p$.

### 2.1.2 Cryptographic Functions

The above described general aims of cryptography need sophisticated mechanisms for their realization. Therefore, before discussing cryptographic protocols, we first give a brief introduction to symmetric-key and asymmetric-key cryptography, as well as to hash functions.

#### Symmetric-Key Cryptography

In contrast to an asymmetric cryptosystem, in a symmetric cryptosystem both participants use the same key – where “same” means that the key of each party is deterministically determined by the key of the other party, and with a method known to both parties. Formally, a symmetric encryption method is a tuple $(M, C, K, E, D)$, where $M$ is the set of possible plain-texts (unencrypted messages), $C$ is the set of possible ciphers (encrypted messages), and $K$ is the set of allowed keys. Then $E$ and $D$ are functions of type $E: K \times M \to C$ and $D: K \times C \to M$ where $E$ is the encryption function and $D$ the decryption function. These functions must interact correctly in that possession of the secret key $k$ and the ciphertext $E(k, m)$ of message $m$ allows one to recover message $m$:

$$\forall k \in K, m \in M: D(k, E(k, m)) = m$$

(1)

The methods for symmetric encryption can be divided into block ciphers and stream ciphers,

![Diagram of block cipher](imaginary-block-cipher-diagram)

![Diagram of stream cipher](imaginary-stream-cipher-diagram)

Figure 1: Comparing block and stream ciphers

see their comparison in Figure 1. With stream ciphers, the plaintext $m$ is encrypted character by character with a key stream $k$ that is as long as $m$ itself, to obtain the ciphertext or decrypt a ciphertext into the plaintext.

A prominent symmetric-key method is the Advanced Encryption Standard (AES), which is also used in our amended Geth clients introduced further below. In October 2000, the US National Institute of Standards and Technology (NIST) announced AES as the new standardized
cipher to succeed the Data Encryption Standard (DES). According to the developers of AES, Joan Daemen and Vincent Rijmen, AES is also called the Rijndael algorithm. Rijndael offers a very high level of security: even after more than ten years of its standardization, only one theoretically interesting, yet practically irrelevant, attack on AES has been found.

Asymmetric-Key Cryptography  How can one solve the so called key-exchange problem? A manual key exchange in a face-to-face meeting with a handful of communication partners would certainly not be a problem as such. Such an exchange over the phone seems already more problematic. But today’s IT Systems require many different keys, involve many communication partners, and many of these partners are machines or devices.

This key pair \((P, S)\) of a public-key encryption method is closely related via a mathematical algorithm. Data encrypted with the public key \(P\) can only be decrypted with the private key \(S\). Therefore, the private key must be kept secret by the owner of the key pair. The solution of the key-distribution problem through use of a public key, as described above, raises the question of whether the key \(P\) is actually the real public key of a specific communication partner. Therefore, this solution to key distribution creates another problem that of entity authentication.

Let us now give a prominent example of how to generate such a key pair \((P, S)\) – using the so called RSA method. This method uses a function that is very easy to calculate, but that is very hard to invert. Such functions are referred to as “one-way functions with a back-door” since the possession of some secret allows one to also invert the function easily.
This RSA function is rooted in number theory and modular arithmetic. While multiplication of integers is not a problem for modern computers, the inverse operation of decomposing a natural number into its prime factors is believed to be a hard computational problem. This is the so-called FACTORING problem. So while it is easy to compute the result of $1873 \times 3089 = 5785697$, finding its factors is much harder. There are methods that can achieve this, for example if all the prime factors are sufficiently small. But such methods don’t work when all prime factors are very large. We will revisit this topic when discussing Quantum Computing further below.

Overall, few one-way functions with back-doors are known that seem to offer sufficient security. Therefore, relatively few asymmetric encryption methods are used in practice, mostly RSA and Diffie-Hellman key-exchange protocols and their variants such as the Station-To-Station protocol. But Elliptic curves (discussed below) are becoming more popular as alternatives now.

Let us illustrate the workings of a one-way function with a back door through the RSA function. Hereby, the main features of RSA are as simple as this:

1. choose prime numbers $p$ and $q$ of roughly equal size, say 2048 bits,
2. compute the modulus $N$ as the product $p \times q$,
3. select $d$ and $e$, such that $d$ is the multiplicative inverse of $e$ modulo $(p-1) \times (q-1)$, i.e.
   - $d$ is relatively prime to $(p-1) \times (q-1)$, and
   - $e \times d \pmod{(p-1) \times (q-1)} = 1$
4. discard $p$ and $q$ – which are the back-door to this RSA function,
5. output the public key as the pair $(e,n)$ and the private key as the pair $(d,n)$, where $n$ is also public information.

Using this set-up, we may perform the following two encryption and decryption operations:

- encrypt message $M$ as $C = M^e \pmod{n}$
- decrypt ciphertext $C$ as $M = C^d \pmod{n}$

The set of plain texts is $\{0, 1, \ldots, N-1\}$. Although this encryption method is correct, this scheme is not secure because the ciphertext is deterministic in the plaintext. Certain parameter choices such as low values of $e$ (which are attractive as they involve less multiplications for encryption) are also insecure in certain use contexts. Standards such as the suite of Public Key Cryptography Standards issued by RSA Security Inc. ensure that RSA is used securely.

Nonetheless, asymmetrical methods are slow and complex compared to symmetric methods (and therefore very prone to implementation errors). They are not resilient to scalable quantum computers (should such machines ever be developed), and they require much more processing power than symmetric methods. For example, when comparing RSA and AES, RSA is about 1,000 times slower than AES in software-based implementations.

**Hash Functions**

Hash functions are work horses of Cryptography. We can think of a hash function $H$ as a function of type $\{0, 1\}^k \rightarrow \{0, 1\}^n$ for natural numbers $0 < n \leq k$ so that $H(x)$ is an $n$-bit output for any $k$-bit input $x$. Cryptographic applications of hash functions often require one or several security properties of such functions $H$, and we list some prominent ones here:

1. It should be computationally infeasible to compute some $x'$ in $\{0, 1\}^k$ with $H(x) = y$ for a given $y$ in $\{0, 1\}^n$.
2. It should be computationally infeasible to compute, given some $x$ in $\{0, 1\}^k$, some other $x' \neq x$ in $\{0, 1\}^k$ with $H(x) = H(x')$. 

10
3. It should be computationally infeasible to find $x \neq x'$ in $\{0, 1\}^k$ such that $H(x) = H(x')$.

4. The hash function $H$ should be puzzle friendly [39].

Computational infeasibility is a concept that can be formally defined, for example the inability to compute something within probabilistic polynomial compute time. But it may also refer to more practical metrics, such as the inability to compute something within 6 weeks given a budget of 1 million US dollars and requiring 1 week to spend that money to start the computation.

The first property above says that someone who does not already know a matching input for some output of $H$ cannot produce such a matching input. The second property says that someone who knows a matching input for some output of $H$ cannot find another such matching input. The third property says that one cannot find two different inputs of $H$ that have the same output. And the fourth property is a more modern one, saying that the hash function is suitable for usage in the solving of cryptographic puzzles, such as those based on Proof of Work.

In typical applications, we have $n << k$, for example $k$ may be $2^{64}$ and $n$ may be $256$. In such cases, there are of course many collisions $H(x) = H(x')$ for $x \neq x'$. So the above security properties don’t rule out the existence of such collisions but state that it must be impractical to find any of them. The challenge in the design of hash functions is to make this so.

Because of the first security property, we may think of a hash function as a one-way function. But it is unclear whether such functions do exist if we take a formal definition of infeasible computation – their formal existence proof would also have $P \neq NP$ as a consequence.

Furthermore, it is also not clear whether currently used hash functions have these security properties for practical interpretations of infeasibility. The discovery of a collision for the SHA-1 function [46](which turned a theoretical attack into a practical one with one of the largest compute efforts in history) is a good example thereof.

The recent hash function standardization competition for SHA-3 ensured that the winner of said competition prevents known weaknesses of existing hash functions, notably the ability to conduct length-extension attacks on hash functions using the Merkle-Damgård construction.

### 2.1.3 Elliptic Curve Cryptography

Elliptic Curves have been studied in Mathematics for centuries, and the analytical techniques developed for them were famously used in the proof of Fermat’s Last Theorem. But Elliptic Curves have important and exciting applications in Cryptography (see e.g. [3]), including digital signature schemes and identity-based public-key cryptography.

Given an Elliptic Curve $E$, defined below, and a finite field $\mathbb{F}_q$, the set of rational points $E(\mathbb{F}_q)$ on the Elliptic curve $E$ over $\mathbb{F}_q$ is a finite, commutative algebraic group (discussed below). Such groups are equivalent to the finite product of some additive and multiplicative groups of finite fields together with some Jacobians of curves. We will not go into the details of Jacobians, suffice to say that they play a crucial role in making group operations cryptographically secure.

A key attraction of the use of the groups $E(\mathbb{F}_q)$ in Cryptography is that these representations seem to provide stronger security for a given bit size of group parameters when compared to other representations of such groups. For example, it is believed that a public key with 313 bits – for an appropriately chosen Elliptic curve and finite field – provides as much security as an RSA key with 4096 bits. The clear benefit here is that the signature generation and verification can then be implemented much more efficiently, but still at a sufficient level of security.

Such beliefs are rooted in assumptions about the hardness of certain computational problems or beliefs that certain results in Complexity Theory hold. We will explore some potential issues of such assumptions when discussing Post-Quantum Cryptography further below. However, even the believed security of schemes that do not use Elliptic curves, e.g. RSA, rest on similar assumptions or beliefs. Therefore, the use of Elliptic curves has clear practical benefits.
We now discuss the Elliptic curve used in both Bitcoin and Ethereum, before we describe the algorithm for digital signatures ECDSA that uses that curve for a particular finite field. This is the Elliptic Curve $\text{Secp256k1}$, as defined in “Standards for Efficient Cryptography SEC 2: Recommended Elliptic Curve Domain Parameters” by Certicom Research [9]. This curve was chosen over the corresponding US NIST standards of Elliptic curves because $\text{Secp256k1}$ allows for more efficient computations, and its constants are determined in a predictable way – making the existence of “back doors” that compromise security much less likely.

The shape of this curve is indicated in Figure 3. Its algebraic equation is

$$Y^2 = X^3 + 7$$ (2)

when specified in what is called the Short Weierstrass Form. The finite field $\mathbb{F}_p$ used for points on this curve is given by the prime

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$ (3)

Note that this prime is of the form $2^n + a$ where $a$ is small compared to $2^n$. In fact, there are few 1 bits in the binary representation of $p$. This makes sure that algebraic operations, even the basic operation “modulo $p$”, have very efficient optimizations that may also be implemented in dedicated hardware – as much of it reduces to shifts and other hardware-friendly operations.

We can now define the set of points $E(\mathbb{F}_p)$ for the above curve and choice of $p$ as the set of all pairs $(x, y)$ of integer coordinates for which

$$y^2 = x^3 + 7 \pmod{p}$$ (4)

holds; plus an additional element $\mathcal{O}$ that is not a pair of coordinates. We write $P$, $Q$, and so forth for pairs satisfying (4); these and $\mathcal{O}$ are the points of this curve over $\mathbb{F}_p$. Also, let

$$-P = (x, -y) \quad \text{where } P = (x, y)$$ (5)

and set $-\mathcal{O} = \mathcal{O}$. The commutative group structure on $E(\mathbb{F}_p)$ is defined as an additive operation $P + Q$ on points. The geometric intuition of this operation is illustrated in Figure 4.

We use the convention that vertical lines that intersect points $P$ and $Q$ (counting multiplicities) also intersect the curve at $\mathcal{O}$. With that convention, we may define $P \circ Q$ for two points $P$
and \( Q \) on the curve as the third point (counting multiplicities) on the curve that lies on the line between \( P \) and \( Q \). We then may define the group law \( P + Q \) as

\[
P + Q = O \circ (P \circ Q)
\]

(6)

We can define this formally in algebra. Clearly, \( O + Q = Q + O \) for all \( Q \) in \( E(\mathbb{F}_p) \), including \( O \) itself. Let \( P = (x_P, y_P) \) and \( Q = (x_Q, y_Q) \) be different from \( O \). If the line through \( P \) and \( Q \) goes through \( O \), then \( P + Q \) equals \( O \). Otherwise, we set

\[
P + Q = (\lambda^2 - x_P - x_Q, -\lambda \cdot (\lambda^2 - x_P - x_Q) - \mu)
\]

(7)

where \( \lambda \) and \( \mu \) are determined as follows:

- For \( x_P = x_Q \) we have that \( Q \) is not equal to \(-P\). Then we set

\[
\lambda = \frac{3x_P^2}{2y_P} \quad \mu = \frac{-x_P^3 + 14}{2y_P}
\]

(8)

- For \( x_P \neq x_Q \), we set

\[
\lambda = \frac{y_Q - y_P}{x_Q - x_P} \quad \mu = \frac{y_P x_Q - y_Q x_P}{x_Q - x_P}
\]

(9)

Chart 1 of Figure 4 shows the case in which \( P \) and \( Q \) are points on the curve (other \( O \) at infinity on that curve) and the line between \( P \) and \( Q \) intersects a third point \( R \neq O \). By (6) \( P + Q \) equals \( O \circ R \) which is \(-R\). That figure also illustrates other cases for computing \( P + Q \).

We note that the fractions above are really operations in the finite field \( \mathbb{F}_p \); \( a \cdot b^{-1} \) denotes \( a \cdot b^{-1} \) in that field. Such computations can be done more efficiently by a change of representation that uses the so called Long Weierstrass Form of the curve, where the points are triples \((x, y, z)\) such that at least one of these three coordinates is non-zero.

The choice of parameters, the coefficients of the curves and the size of the finite field, also require care so as not to generate cryptographically weak, commutative groups \((E(\mathbb{F}_q), +, O)\). In particular, the size of \( E(\mathbb{F}_q) \) needs to contain a large prime factor – computing these sizes is far from trivial. Also, the Trace of Frobenius at \( q \), the value of \( q + 1 - |E(\mathbb{F}_q)| \), must not equal 1. The curve Secp256k1 and its choice of \( p \) in (3) meet these requirements. We refer to [34] for a more in-depth discussion of the security of Elliptic curve choices and their implementations.

We can now explain the ECDSA, the Elliptic Curve Digital Signature algorithm, instantiated for the above group \((E(\mathbb{F}_q), +, O)\). For \( k > 1 \) and a point \( P \), we define

\[
k \star P = P + \cdots + P \quad (k \text{ summands})
\]

(10)
Let $G$ be a base point of the group $E(\mathbb{F}_p)$. The sub-group it generates has size $n$ for a 256-bit prime $n$: so $k \cdot G \neq \mathcal{O}$ for all $0 < k < n$ but $n \cdot G = \mathcal{O}$. The security of ECDSA relies on the hardness of the corresponding Discrete Logarithm problem, saying that it is computationally infeasible to compute $k$ given a point $P$ in $E(\mathbb{F}_p)$ of form $k \cdot G$ for random $k$.

A private key is a random 256-bit number $k$. The corresponding public key is an element of $E(\mathbb{F}_p)$, namely $K = k \cdot G$. Let $m$ be a message we wish to sign. This requires a function $f$ that takes points on the Elliptic curve to the integer interval $[0, n-1]$. Let $f(P)$ be $x$ where $P = (x, y)$. The signature of $m$ is then computed as follows, using the hash functions SHA-256 and RIPEMD-160 with 256-bit, respectively, 160-bit outputs:

1. we hash $m$ as $h = \text{RIPEMD-160}(\text{SHA-256}(m))$ — using two different hash functions in this way is managing the risk that one of them may become insecure

2. choose a random $u$ with $0 < u < n$ as ephemeral key

3. set $r = f(u \cdot G)$ and repeat step 2 above if $r$ equals 0

4. set $s = (h + k \cdot r) \cdot u^{-1} \mod n$ and repeat step 2 above if $s$ equals 0.

The signature of message $m$ is then the pair $(r, s)$. To verify that such a pair $(r, s)$ is indeed a signature of some message $m$, the following protocol is executed:

1. compute $h$ as $h = \text{RIPEMD-160}(\text{SHA-256}(m))$

2. set $a = h \cdot s^{-1} \mod n$

3. set $b = r \cdot s^{-1} \mod n$

4. set $v = f(a \cdot G + b \cdot K)$ where $K$ is the public key.

Accept signature $(r, s)$ if $v$ equals $r$; otherwise, reject the signature. These protocols are well-defined since signatures ensure that both $r$ and $s$ are non-zero and that $n$ contains at least as many bits as the 160-bit output of the hash function. The use of an ephemeral $u$ (used only once) for signing is crucial: using the same $u$ on two different signatures allows an adversary to compute the private key $k$ (or realize a collision of the hash function used in the signature scheme). This flawed usage led to a security compromise in the PlayStation 3 gaming console.

It is also possible to reduce storage of the public key $K = (x, y)$: it suffices to store $x$ and the sign (a sole bit) of $y$ as this determines the point $K$ in $E(\mathbb{F}_p)$.

Let us now discuss the security of ECDSA based on the curve $\text{Secp256k1}$. This is a Koblitz curve [27], a special kind of complex multiplication Elliptic curve. Koblitz curves over fields of form $2^m$ (with characteristic 2) are known mostly for giving rise to anomalous curves for certain values $a$ and $b$ in $\{0, 1\}$ when the curve equation is

$$y^2 = x^3 + ax + b \quad (11)$$

and these curves would be cryptographically weak. Note that $\text{Secp256k1}$ is in a different setting though: $b$ there equals 7 and the field $\mathbb{F}_p$ has characteristic $p$. But all these Koblitz curves have very efficient implementations of the algebraic operations needed for group multiplication $(P + Q)$ and exponentiation $(k \cdot P)$. In particular, the doubling operation $2 \cdot P$ is very efficient, and this operation is the bases for efficient implementations of $k \cdot P$ using iterative squaring.

The efficiency of Koblitz curves is allowed by their non-random construction. This sets them apart from Elliptic curves that are generated in a pseudo-random way, where curve parameters are chosen randomly until a group of points with desired properties (e.g. being a prime field or having a sub-group whose order is a large prime) is found. One concern with the latter
approach, though, is that some parties may have privileged knowledge of cryptographic weaknesses of such curves, where this knowledge is not in the public domain. And they may then influence the pseudo-random source that generates such curves so that the process stops on a curve with weaknesses that only these parties are aware of and they then could exploit.

Allegedly, the US NSA did supply pseudo-random number generators to US NIST for use in such processes, were the generators might contain some “back doors” that the NSA could exploit as described above. This may sound like conspiracy theory, but a paranoid attitude can be very healthy in Cryptography. Perhaps this is why the designers of Bitcoin did not adopt the curve $\text{Secp256r1}$ that was generated by such a random process and chose $\text{Secp256k1}$ instead.

A general security concern, important for standardization efforts, is whether an Elliptic curve was generated with sufficient $\text{Rigidity}$. Intuitively, this means that the process – be it random or otherwise – that determined the chosen curve only generated a limited number of such curves, limiting the risk of ending up with a curve that an attacker would recognize as being weak.

### 2.1.4 Post-Quantum Cryptography

Information Security is a vital defense line against the increasing share of crime that is conducted online, referred to as cyber crime. The means of attack are varied, ranging from Ransomware and Distributed Denial of Service Attacks to the Undue Influence on Elections through a combination of Machine Learning and Perception Management.

An Intelligent Blockchain Framework stands to offer benefits here, as it has the potential of creating much better resiliency against at least some of these types of attacks. However, theCryptographic Primitives that Blockchain uses must equally provide sufficient security so used primitives can realize the aforementioned system resiliency. For Blockchain systems that are expected to operate for several years or decades, for example those that host medical data, this poses a risk since theoretical and technical progress may make a Cryptographic Primitive, that was deemed to be secure at the time of system creation, completely insecure.

We will now discuss one of these potentially disruptive advancements, Quantum Computing. We will assess the risk that this approach poses in the long term, and point out efforts that aim to mitigate against – if not eliminate – that risk by inventing new approaches to Cryptography, bundled together under the research area called Post-Quantum Cryptography.

Without going into the details of this approach, Quantum Computing exploits peculiar properties of quantum physics to entangle so called quantum bits, so that it becomes possible to superposition $2^n$ computational states in $n$ such quantum bits and perform computation over these superpositions much more efficiently than any classical computer based on Turing Machines and von Neumann architectures could.

For example, there are quantum algorithms with which one can factor integers efficiently. In particular, this would allow one to factor an RSA modulus $N = p \cdot q$ which would completely destroy the security of RSA. But RSA is still secure today since, so far, we have not been able to develop quantum computers that can entangle $n$ quantum bits for sufficiently large values of $n$ so that these entanglements are stable long enough in order to perform the quantum computations and to produce the classical output (here, the factors $p$ and $q$).

While this state of affairs seems to be re-assuring, there is a global effort under way to realize such scalable quantum computers, let us mention work by IBM and by D-Wave. The latter appear to claim to have achieved scalability, but this seems to be for approximative computations, not for exact ones. A D-Wave Quantum Computer may therefore be able to considerably advance the reach and tool box of mathematical optimization and machine learning, domains in which computation is of an approximative nature due to inherent reasons. And such advances may also pose threats to existing approaches in Information Security, e.g. by allowing novel forms of cryptanalysis. By the same token, these advances may also allow the design of more resilient Cryptographic Primitives such as Hash Functions.
In any event, one has to understand the risks of scalable quantum computers for Information Security of current and future Blockchain systems. The bad news is that, in such a possible future world, Cryptography based on Finite Fields and Elliptic Curves (notably the digital signature algorithms DSA and ECDSA) will no longer be secure. Moreover, RSA encryption and signature schemes but also Diffie-Hellman key-exchange protocols will become insecure.

The news is better for symmetric-key encryption and hash functions, and so better for message authentication codes that rely on hashes or symmetric-key encryption. The latter will only have to use longer keys (about twice as long), and hash functions will require longer outputs.

From the perspective of Risk Management, Michele Mosca’s Theorem states that your organization should worry if \( x + y > z \) where

- \( x \) is the number of years that cryptographic keys your systems uses should be secure
- \( y \) is the number of years it will take to re-tool your system infrastructure with quantum-safe approaches
- \( z \) is the number of years it will take to build a scalable, exact quantum computer.

Some think that \( z \) equals about 20 at the time of writing. Many application domains have little control over the value of \( x \), e.g., it may be determined by compliance. Organizations may have an opportunity and the means of decreasing the value of \( y \), though. This obviously depends on the availability of quantum-safe approaches developed by Post-Quantum Cryptography.

It is therefore no surprise that standardization agencies such as US NIST call for proposals for new public-key standards that are quantum-safe. The specific NIST call is not a competition, such as the competition seen for the newest Hash Function standard SHA-3. This is partly because the design space of quantum-safe cryptography needs to be better explored and understood before requirements that would be a good basis for a competition can be formulated with high confidence. The submissions for the aforementioned NIST call are due in November 2017, and their results are hoped to make significant contributions to furthering such understanding. But it will take several years of analysis and community-based discussions before standards drafts may be ready, perhaps by 2024.

Let us illustrate the use of quantum-safe solutions based on a cryptographic hash function \( H \) as a primitive. If we think of \( H \) as a black box, Grover has shown that there is a quantum algorithm that, with high probability, will compute an input \( x \) such that \( H(x) \) equals a particular, given output \( y \). Moreover, Gover proved that the running time \( \mathcal{O}(\sqrt{n}) \) is asymptotically optimal: there is no quantum algorithm that can achieve this in fewer operations where \( n \) is the size of the domain of \( H \). This is why we stated above that a hash function can be made quantum-safe by doubling the (bit) size of its output: this will compensate for the abilities of Grover’s algorithm.

Public-Key Cryptography based on the use of such hash functions can thus be designed to also be quantum-safe. However, these safe schemes have an inherent time-memory trade-off that is further constrained by the desired security level of the primitive. For example, we would have to double the key size for AES to make it quantum-safe but this doubling may severely impair system performance. There is therefore an effort under way to find schemes that trade off time and memory in optimal ways. Winternitz-type, one-time signature schemes are such an example, for which shorter signatures have been developed in recent years [21].

It will thus take time to gain optimal time-memory trade-offs whilst also ensuring quantum-safety of the developed primitive. These post-quantum solutions also will have to monitor risks of technological advancements. For example, the asymptotic optimality of Grover’s algorithm is shown in a certain quantum computational model, and there are quantum algorithms of time complexity \( \mathcal{O}(\sqrt[3]{n}) \) that solve the above black-box problem for a hash function \( H \) using the model of a quantum computer with non-local hidden variables.

Post-Quantum Cryptography should not be confounded with Quantum Cryptography, including the research area that uses quantum effects for generating and sharing secret keys such
that an attacker cannot learn anything about the exchanged key without having an observable effect on the quantum protocol that would then be detected by the honest parties. Such an approach has been used, e.g., to demonstrate the feasibility of quantum-safe authentication mechanisms used in a Blockchain based on an urban fiber-optic network [26]. In that context, we think it is important to get a better understanding of realistic attack models for Blockchains (quantum-safe or otherwise) when attackers have scalable quantum computers at their disposal.

2.1.5 Random Oracle Methodology

A standard practice in Cryptography is to first design an ideal system in which all parties, honest and malicious ones, have access to an oracle that realizes a truly random function. Then one replaces that oracle with a so called Cryptographic Hash Function, such as SHA-256, when implementing the system. This methodology is known as the Random Oracle Model (ROM).

The security proof is done by appeal to the properties of the oracle, and so this does not in and of itself guarantee that the implementation has the proven security property. Indeed, in [7] Canetti et al. give an example of a signature scheme that can be proved to be secure in the ROM but for which all implementations yield insecure signature schemes. This therefore means that one needs a better understanding of good practice in using and implementing such oracles in real systems – as discussed in [7].

But ROM is a core tool that provides a first step in understanding a protocol or cryptographic system. A random oracle \( H \) may have type \( \{0,1\}^m \rightarrow \{0,1\}^n \), so that an \( m \)-bit input produces an \( n \)-bit output. The true randomness of this oracle can be understood in two equivalent ways:

- we may think of \( O \) as being drawn according to a uniform distribution from the set of all functions of type \( \{0,1\}^m \rightarrow \{0,1\}^n \)
- \( O \) replies with the same \( O(x) \) for repeats of some input \( x \); otherwise, the initial output \( O(x) \) is drawn according to a uniform distribution from set \( \{0,1\}^n \).

We will make use of ROM in our optimization work described below.

2.2 Linear Programming

In addition to graph theory, we also make extensive use of (non-)linear programming in our approach, which we introduce in the following. Let us first discuss linear programming, whose objective is to optimize a linear function that consists of \( n \) degrees of freedom and is restricted by linear equations or inequalities. Thus, linear programming represents a very general method for optimizations. Therefore, we often also speak of linear optimization. A linear optimization problem is also referred to as a linear program (LP) and is defined in the following way:

**Definition 1 (Linear Program).** Let \( A \in \mathbb{R}^{m \times n} \) be a matrix as well as \( b \in \mathbb{R}^m \) and \( c \in \mathbb{R}^n \) two vectors. A feasible solution for \((A,b)\) is a vector \( x \in \mathbb{R}^n \), for which it holds that \( Ax \leq b \) and \( x \geq 0 \). A solution for \((A,b)\) is a feasible solution that maximizes the product \( c^T x \):

\[
\max\{c^T x \mid Ax \leq b, \ x \geq 0\}.
\]  
(12)

As such, \( x \geq 0 \) means that all coordinates of \( x \) are non-negative, \( c^T x \) stands for the standard scalar product of \( c \) and \( x \), whereby \( c^T \) represents the transposed vector of \( c \).

Furthermore, we define the set of all feasible solutions of an LP problem given by \((A,b)\) as the feasible range. Hereby, an optimal solution represents a feasible solution for which the objective function \( c^T x \) has a maximum. However, an LP problem may not always give rise to an optimal solution. Thereby, we differentiate between three different cases:
1. The LP problem has no solution, as the feasible range is empty, the inequalities cannot be fulfilled. This LP problem is then called **infeasible**.

2. The LP problem is **unrestricted**: the set in (12) is unbounded and so the maximum is not defined – i.e. for each value \( v > 0 \) there exist feasible solutions \( x \) with \( c^T x \geq v \).

3. The LP has at least one optimal solution \( x \): the maximum in (12) exists and is attained by at least one solution \( x \).

Hence, an LP is **feasible**, if it is unrestricted or comprises at least one optimal solution.

### 2.2.1 Solution Algorithms

The first practical algorithm to solve linear optimization problems has been presented by the US mathematician George Dantzig in 1947 [12]. As such, his so-called Simplex algorithm is still one of the most frequently applied methods to solve LPs. The main idea of the algorithm is to run from an arbitrary vertex (corner) of the polyhedron, which is defined by the optimization task, along the edges to an optimal vertex. Thereby, the algorithm always jumps to the neighboring vertex with an objective function value that is not smaller than the actual value. A maximum is found if all neighboring vertexes have smaller objective function values, so that the algorithm terminates. Hereby, we can also prove that this local maximum is also a global maximum, such that the algorithm is also correct.

However, there can be exponentially many vertexes in a polyhedron, in the number of variables and inequalities. Thus, the Simplex algorithm has theoretical worst-case exponential running time. Yet, practical applications rarely exhibit such long running times, the algorithm usually terminates after a moderate number of steps. As such, it is not known whether there exists a variant of this Simplex method with polynomial running time for *all* LP problem instances.

In 1979, the Soviet mathematician Leonid Khachiyan was able to show – with a different approach – that linear optimization problems are solvable in polynomial time [25]. However, his famous ellipsoid algorithm is not suitable for practical applications, due to a comparably very long running time. Hereby, the algorithm tests, if the center of an ellipsoid lies within a special polyhedron. As such, it can be shown that solving an LP is equivalent to finding a point in a suitably defined helper-polyhedron.

A further algorithm was developed by the Indian mathematician Narendra Karmarkar [24]. The Karmarkar’s algorithm was the first method that guarantees a polynomial running time while at the same time being nearly as efficient in practice as the Simplex algorithm. Its idea is to approach the optimum of the LP through the inside of the polyhedron. As such, further details of these algorithm can be found in the extended literature, such as [11, 12, 45].

### 2.2.2 Integer Linear-Programming & Relaxation

A special case of linear programming is the so-called integer linear programming (ILP). Hereby, for all feasible solutions it shall also hold that \( x \in \mathbb{Z}^n \), meaning that all variables \( x_i \), may only be integer values. An ILP problem is defined as

\[
\max \{ c^T x \mid Ax \leq b, \ x \geq 0, \ x \in \mathbb{Z}^n \}
\]  

(13)

If some of the the variables have to be integers and others may be reals, we refer to mixed-integer linear programming (MILP), which we introduce in Subsection 2.2.3. The limitation to integer values as such also restricts the range of feasible solutions, which may be favorable – for example when a value \( x_1 \) denotes how many planes need to be bought to complement a fleet portfolio. Further, in comparison to linear programming, which can be solved in polynomial time, finding an arbitrary feasible solution for an ILP is already NP-hard.
Theorem 2.1. Solving ILP problems is NP-hard.

Proof. It suffices to show that there is an efficient reduction of the NP-complete problem 3-SAT (a special instance of satisfiability of propositional logic) to ILP; formally, that $3\text{-SAT} \leq_p \text{ILP}$ holds. Let $F = C_1 \land \cdots \land C_m$ be a 3-SAT function for the variables $x_1, \ldots, x_n$. We create the inequalities $Ax \geq b$ for the solution vector $x = (x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n)^T$, represented as follows:

- For each $i = 1, \ldots, n$ we set up four inequalities
  
  $$
  x_i + \overline{x}_i \geq 1, \quad x_i \geq 0,
  
  -x_i - \overline{x}_i \geq 1, \quad \overline{x}_i \geq 0.
  $$

- For each clause $C_j = l_{j_1} \lor l_{j_2} \lor l_{j_3}$ with $l_{j_k} \in \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ for $j = 1, \ldots, m$ and $k = 1, 2, 3$ we set up the inequality:
  
  $$
  l_{j_1} + l_{j_2} + l_{j_3} \geq 1.
  $$

The inequalities for the integer variables $x_i, \overline{x}_i$ are fulfilled, if and only if one variable is 0 and the other 1. The clause inequalities ensure that at least one literal is true in each clause $C_j$. Now we can easily verify that each solution $x$ of $Ax \geq b$ matches a fulfilling occupancy of $F$ and vice versa. Moreover, this conversion of problem instances into other problem formats is efficient in the size of these problem instances.

Hereby, we cannot easily see that the ILP is in NP. Although we could of course apply a nondeterministic algorithm to guess the solutions, it is not clear whether this solution is polynomial in the length of the input $(A, b)$. However, using methods of linear algebra, we can show that each solvable ILP instance $(A, b)$ comprises a solution $x$, being polynomially solvable in the length of $(A, b)$. Thus, it follows that the solution of an ILP is NP-complete.

Furthermore, ILP problems can be solved exactly through cutting-plane methods or branch and bound (BnB). For the cutting-plane method, we solve an analogous linear problem without the integer condition. Hereby, the evolving non-integer edges of the polyhedron are cut by adding further suitable inequalities. Thus, the set of permissible solutions shrinks until all edges are integers. In the contrast, the BnB method strives to search the set of permissible solutions as efficiently as possible, whereby, however, in the worst case all possible values need to be tested. Thus, there exists a variety of heuristics, since ILP algorithms comprise an exponential running time. As such, often various methods are combined in solving ILP problems.

A further specialization of such kind of problems is discrete linear programming or 0-1 integer linear programming (0/1-ILP), as specified in (14), whereby it holds that $x \in \{0, 1\}^n$; i.e. the variables $x_i$ can also only be either 0 or 1. Due to the proof of Theorem 2.1, we conclude that also 0/1-ILP is NP-hard.

$$
\max \{c^T x \mid Ax \leq b, x \in \{0, 1\}^n\}
$$

(14)

A further possibility to efficiently solve ILP problems is to omit or loosen some constraints. Indeed, this can produce solutions that may not be solutions to the original problem. However, in some cases we can also show that through such relaxations no further solutions are added.

Definition 2 (Relaxation). Let $X, X' \subseteq \mathbb{R}^n$ be sets, $f, f' : \mathbb{R}^n \to \mathbb{R}$ functions and $P, P'$ two optimization problems with

$$
P : \text{compute} \ max \{f(x) \mid x \in X\}

P' : \text{compute} \ max \{f'(x) \mid x \in X'\}.
$$
Further, $P'$ is called a relaxation of $P$, if

$$X \subseteq X' \quad \text{and} \quad f(x) \leq f'(x) \quad \text{(for all } x \in X)$$

Evidently, the optimum $v(P')$ of the relaxation is at least as large as the optimal value $v(P)$ of the original problem. A relaxation is called exact, if it holds that $v(P) = v(P')$. Exact relaxations are particularly interesting in many problems.

For an ILP program, the LP-relaxation arises from omitting the integer condition, such that the optimization problem in (15) becomes the one in (16):

$$\max \{ c^T x \mid Ax \leq b, x \geq 0, x \in \mathbb{Z}^n \} \quad \text{(15)}$$

$$\max \{ c^T x \mid Ax \leq b, x \geq 0 \} \quad \text{(16)}$$

Correspondingly, for a binary program as given in (14) we assume $x \in [0, 1]^n$ instead of $x \in \{0, 1\}^n$, so coordinates of $x$ may have real numbers between 0 and 1 as values. Thus, this is now a linear program, which can be solved efficiently. As such, please note that each feasible solution of an integer linear program is also a feasible solution of a relaxed problem; however, this is not the case the other way around. Yet, there exist cases in which the optimum of a relaxation is also the optimum of the original problem. Moreover, it follows that infeasibility of the relaxation implies that the integer linear programming program it relaxes is also infeasible.

### 2.2.3 Mixed Integer Non-Linear-Programming

Now, having introduced linear programming, in comparison to ILP our approach deals with mixed-integer problems for which some variables denote integers but there the objective function is non-linear. Such problems, called mixed-integer, non-linear-programming or MINLP are not only of interest in this particular work but are also specifically important in practice, e.g., for facility location optimization problems. As such, we can specify MINLP problems as

$$\min f(x, y)$$

subject to

$$g(x, y) \leq 0$$

$$x \in X$$

$$y \in Y \text{ (integral)}.$$  

where $X \subseteq \mathbb{R}^k$ and $Y \subseteq \mathbb{Z}^l$. The function $f: \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}$ is a non-linear objective function, and function $g: \mathbb{R}^k \times \mathbb{R}^l \to \mathbb{R}$ is a non-linear constraint function. The variable vectors $x, y$ contain the decision variables, whereby $y$ is required to be an integer vector.

MINLP problems are thus very hard to solve in theory and in practice, as they combine all of the preceding difficulties, such as the combinatorial issues of mixed integer problems and the (non-)convexity of non-linear programs – which both lead to NP-completeness. As such, we need to combine specific algorithmic approaches, involving both cutting-plane methods and BnB techniques. We will revisit MINLP problems in our research applications in Section 5.

### 2.3 Reinforcement Learning for Adaptive Engines

A crucial element of our engine is a machine-learning model that informs decisions on how to adapt run-time parameters. The representation of this model is discussed in Section 4. Here, we first discuss possible training methods. One approach would be to use supervised learning: Using a large amount of labeled data, we would train a model that can reproduce parameter adjustment suggested by the training data. This approach however has several drawbacks:
Non-trivial data collection: Gathering the required amount of training data is non-trivial and expensive. A human operator would have to monitor a running Blockchain and take (best guess) decisions that get recorded.

Inherent bias: The resulting data is tightly coupled to the ambient system: a Blockchain running in a data center reacts very differently to parameter changes than one running on low-powered IoT clients (due to different characteristics, e.g. miner spin-up latency).

Static nature: The model will only be able to take decisions implied by the training data and it is unlikely to surpass the performance of those decisions.

To overcome those drawbacks, we suggest to use reinforcement learning (RL) to train machine-learning engines. RL is positioned in between supervised learning and unsupervised learning: Whereas supervised learning has labels for each training example and unsupervised learning has no labels at all, reinforcement learning has sparse and time-delayed labels.

The unique properties of RL enable an agent to build a model of an environment solely by observing its state transitions. As an agent interacts with an environment, it builds an internal representation of how it believes the environment to be working and adapts its own behavior to be successful in this environment. Two types of adaptiveness are worth highlighting:

Systemic adaptiveness: Any system that a Blockchain can run on will exhibit different characteristics (e.g. network latency). An ideal model adapts to such characteristics, expressing the full set of performance-influencing factors as model parameters is infeasible.

Temporal adaptiveness: Some intrinsic characteristics of a system evolve over time (e.g. decreased network latency due to hardware upgrades). Such changes cannot be fully representable as model parameters, so an ideal model adapts to evolving environments.

Reinforcement Learning Introduction We will use the example of self-driving cars to introduce the basic ideas used in reinforcement learning. As briefly mentioned, there are two entities interacting with each other: Agent and Environment (Figure 5).

In the case of autonomous driving, the agent would be an autonomous car, and the environment would be an actual environment the car is placed in: streets, traffic signs, other cars, pedestrians, and other factors. The agent has a set of actions it can carry out in the environment, e.g. accelerate, decelerate, turn left, turn right. It does so after observing the state of the environment. An autonomous cars’ observation of the environment (the state the environment is in) is the data it receives from e.g. cameras, GPS, and ultrasonic sensors (see Figure 6).
Terminal states  The agent carries out actions in the environment and observes state changes. States can be terminal: Once the agent caused the environment to be in a terminal state, the simulation episode is over. This can be either because the agent was successful in accomplishing a task (say, the self-driving car arrived at destination), or because the agent caused an undesired outcome (e.g. a self-driving car accident).

Rewards  State changes are sometimes (but not always) associated with feedback from the environment (be it positive or negative). Those sparse and time-delayed labels in RL are called rewards: A self-driving car will get rewarded when it arrives safely at the destination. The sparse and time-delayed nature of rewards leads to the credit assignment problem: Which action (and to which extend) was responsible for receiving a reward? Usually it was not the last action just before the reward, but a combination of previous actions: Safely overtaking the bicycle, stopping at the red light, taking the correct turns, for example.

Defining (and assigning) the right rewards is essential for reinforcement learning: To avoid undesired terminal states, one usually defines a robust target to optimize for. In the case of overtaking a cyclist, contact between car and cyclist can be considered a terminal state. One would set e.g. a 1.5m distance as optimal robust target, with decreasing (and even negative) rewards the further the agent diverges from this target: We don’t want the car to overtake a cyclist with just a few centimeters of distance as it might scare the cyclist and not account for surprising movements, so a negative reward is assigned. We also don’t want the distance to be too big: The maneuver might otherwise surprise other drivers.

2.4 Foundations of Anomaly Detection
The overall goal of anomaly detection is to find a model that represents the “normal” state of a particular subject. Anomaly detection works on datasets of the form \( \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \),
with \( x^{(i)} \) being a single training example. Each training example \( x^{(i)} \) consists of \( n \) features \( x_1, x_2, ..., x_n \). Individual features represent basic or aggregated properties of the domain.

### 2.4.1 Motivating Example

A motivating use case for this is that of monitoring machines in a factory. Anomaly detection can be used in this domain to predict whether a particular machine is likely to break, called **predictive maintenance**. Individual features \( (x_1, x_2, ..., x_n) \) represent properties of a particular machine, e.g. \( x_1 = \text{temperature during operation} \), \( x_2 = \text{noise volume} \), and so forth.

To train an anomaly detection model one would then record such examples during operation of such a machine. Most examples will represent “normal” operation: The machine works as expected. In the rare cases that a machine fails, the recorded information is quite valuable as it can be used to find clues in preceding data points that indicate the upcoming machine failure.

With the recorded data at hand, the aim is to train a model that identifies anomalies with high accuracy. A model that incorporates just two features \( x_1 = \text{temperature during operation} \) and \( x_2 = \text{noise volume} \) can be visualized as in Figure 7.

![Figure 7: Anomaly detection decision boundary, which is non-linear](image)

An important parameter used in anomaly detection is the real number \( \epsilon \): It defines the probability threshold used to classify examples as anomalous or non-anomalous. The decision boundary between anomalous and non-anomalous situations is defined by the resulting model and \( \epsilon \). Too large values of \( \epsilon \) result in false positives, too low values result in false negatives.
2.4.2 Basic Notations for Anomaly Detection

We fix some standard notation or anomaly detection needed in Section 7.

- \( X_{\text{train}} \in \mathbb{R}^{m_{\text{train}} \times n_x} \): Training set used to find a model.
- \( X_{\text{cv}} \in \mathbb{R}^{m_{\text{cv}} \times n_x} \): Cross-validation set used to find optimal hyper-parameters.
- \( X_{\text{test}} \in \mathbb{R}^{m_{\text{test}} \times n_x} \): Test set used to judge real-world performance of a trained model.
- \( x \): Individual training example consisting of \( n \) features \( x_1, x_2, \ldots, x_n \)
- \( m_{\text{train}}, m_{\text{cv}}, m_{\text{test}} \): Number of examples in the respective data set.
- \( n_x \): Number of features used for training and inference.
- \( \epsilon \): Threshold below which a prediction is said to be anomalous.

2.5 Distributed Ledger Technology

Distributed Ledger Technology (DLT) enables a special form of electronic data processing and storage. A Distributed Ledger is a decentralized database or data processing platform in which network users may share “read” and “write” permissions. Unlike a centrally managed database, this network does not require a central instance that makes new entries in the database. New records can be added at any time by the participants themselves. A subsequent update process ensures that all participants have the latest version of the database. A special form of the DLT is a Blockchain. Yet, other DLT expressions exist, e.g., a Tangle network [43], as in IOTA.

Configurations of DLT systems are dependent on the access possibilities of the participants in a network. Distributed Ledgers can be subdivided into “permissioned” and “unpermissioned” ledgers. While the latter are openly accessible to anyone (such as in the Bitcoin network), access to the ledger is regulated at the former. Participants in networks with permissioned ledgers are generally registered and meet certain requirements in order to access data or to get involved in the consensus computation. The choice of the circle of authorized users (open or limited circle of participants) also involves the choice of the consensus mechanism. For example, Proof of Work (PoW) algorithms are primarily used for unpermissioned ledgers (yet not exclusively, as we will see in the XAIN framework), since the validation of entries requires no trust among the participants. On the other hand, permissioned ledgers use Proof of Stake (PoS) or Probabilistic Byzantine Fault Tolerance (PBFT) consensus mechanisms that require less computational power. The establishment of a basis of trust in this case already takes place through the admission of the participants to the network.

2.5.1 History of Distributed Ledgers

The history of DLT has started long before the introduction of Bitcoin along with Blockchains. The first bases for the cryptographically secured linking of individual blocks were described in 1991 by Stuart Haber & W. Scott Stornetta, 1996 by Ross J. Anderson, and then in 1998 by Bruce Schneier & John Kelsey. Furthermore, in parallel, Nick Szabo also worked on a mechanism for a decentralized digital currency in 1998, which he called Bit Gold. As such, the history of DLT is vastly driven by the application area of digital payments and currencies, which can be seen as a very natural but non-exclusive use case of DLT. When dealing with digital currencies and payments in a completely distributed system without a central administrator, like a bank, there arises the requirement for automatic audibility checks that prevent double spending and verify transactions. This has been a very hard problem to solve, mostly since...
traditional cryptographic protocols have only focused on building secure channels, yet have not addressed honest behavior for reaching a system consensus.

In the further development of DLT, in the year 2000, Stefan Konst developed a general theory on cryptographically secured links and derived various solutions for implementation. In 2003, Buldas and Saarepera from Guardtime formally designed the properties for hash-based distributed data structures, building linked time-stamping, as one of the central concepts behind Blockchains. As such, most of the parts to construct functioning Distributed Ledgers have been established several years prior to the introduction of Bitcoin. However, these constructs only worked in low-scale and highly specific corporate infrastructures and not within public networks.

The last missing piece was then introduced in 2008 by anonymous Satoshi Nakamoto with the white paper for Bitcoin [38] and its programmatic publication in 2009. Hereby, the game-theoretic aspect of rewarding good behavior was a central aspect together with linked time-stamping of Blockchain data structures. In the following years, thousands of further cryptocurrencies and second-generation, as well as third-generation, Blockchain systems were introduced, such as Ethereum in 2014. Furthermore, the IOTA white paper was published in 2016 as a different approach using a directed acyclic graph (DAG) instead of chaining blocks.

2.5.2 Introduction to Blockchain Technology

Although the concept of Blockchain already appeared in 2008 [38], it took another nine years until we could recognize the start of a mass adoption in 2017, mostly regarding Bitcoin and Ethereum as well as a whole range of domain-specific, Blockchain-based business ideas using tokens. Hereby, it seems that tokens are a very central aspect of these systems. Yet, the core techniques of Blockchains are centered around a range of well-known cryptographic concepts (which we introduced in Section 2.1) paired with novel algorithmic approaches of reaching consensus in a fully distributed network of nodes, partly also involving game-theoretic aspects.

Cryptographic methods are already heavily applied in various productive systems and are generally considered to be secure. But consensus protocols (discussed in Section 2.5.3) and, particularly, advancements of second-generation Blockchain systems such as smart contracts (an idea going back to contract-oriented programming [35]) or state-channel approaches for so-called off-chain transactions and the interrelation of different Blockchains as third-generation protocols are perceived as not being production-ready and are thus under active research. Our introduction of Blockchain Technology (BCT) thus decouples the different concepts behind them. For this, we start with the core concepts of Bitcoin as the most prominent example and switch to Ethereum when it comes to advancements, specifically regarding smart contracts.

**Bitcoin** Bitcoin can be defined as a homogeneous randomly distributed peer-to-peer network without a central control of nodes. Each of the nodes generates one or more private keys in order to derive also a public key, which is used to generate an address. Using this information, nodes can exchange information, particularly “Bitcoins” as a means of digital tokens between each other. This is done by signing an output with a private key of the input address as a spending condition. Thus, an output consists of the number of Bitcoins to be spent and a spending condition while an input refers to a previous output and the linked condition (signature) as a permission to spend this very input. Now, a transaction is nothing more than the sum of one or more inputs and one or more outputs between two nodes, resulting in a new balance. Hereby, the concept of linked time-stamping appears, as every transaction integrates a time stamp, whereby the specific inputs of a transaction, say 5 Bitcoin, need to be used as a whole, e.g. to send 4 Bitcoin to another address and 1 Bitcoin back to the own address.

As the Bitcoin network does not involve an administrator, all transactions are broadcasted and processed by every node. This, however, touches upon the first crucial problem: The
network needs to achieve a consistent state within a short period of time to prevent double-
spending. In an obvious scenario, an dishonest participant tries to spend the very same Bitcoin
for different goods and services, exchanging it to varying nodes at the same time. This issue
can theoretically be resolved by algorithms like Paxos [29] or Zyzzyva [28] for authenticated
messages as a strong consistency. The problem thereby, however, lies in the immense com-
putational complexity and time to solve this in a distributed network, particularly when involving
the entire global Internet instead of a single corporate network. As such, Bitcoin uses a form
of weak consistency by assuring that the nodes eventually achieve a consistent state, while
temporarily nodes may disagree, which is achieved through the PoW algorithm.

Ethereum After the initial use of BTC to support virtual currencies, a second generation of
Blockchain applications became possible when smart-contract implementations became sim-
pler with the introduction of Ethereum in 2014 [5]. A developer can create a smart contract and
deploy it to the network, every node on the network will receive the byte code of the contract,
and make it available in its virtual machine. Like Bitcoin Ethereum currently uses Proof of Work
as a consensus mechanism, but will migrate to Proof of Stake in 2018.

2.5.3 Comparative Analysis: Consensus Algorithms

Proof of Work PoW is a process of inverse hashing: finding a nonce $n$ that, combined with
block data, produces a hash value within a certain range. This requires considerable computa-
tional effort, proving an economic disincentive to dishonest behavior of miners. A number
$d > 0$ – the so-called level of difficulty – is applied to adjusted this computational hardness.

Proof of Burn PoB relies on possession by the miner of an existing amount of
cryptocurrency, it is often used to bootstrap a new cryptocurrency from an existing one. The
proof shows that some coins have been transferred from the miner’s account to an unreachable
account (or involving a script that cannot return a success value). Figure 8 shows Proof of Burn
as implemented in Slimcoin [41], with current block height $r$, block height at burn transaction $r_t$
multiplier $m$, burn hash $bh$, number of coins burned $n$, target difficulty $d$.

Algorithm 1: computePoB, $pk_i()$

begin
  waitTill(block $B^{r-1}$ is locally known);
  Calculate multiplier $m$;
  $m = r - r_t/n$;
  internal hash $ih$ is calculated from $B^r$;
  $bh = m\times ih$;
  if $bh < d$ then
    add blockheader;
    $m_{pk_i} = (B_{pk_i}, SIG_{pk_i}(H(B_{pk_i})))$;
    propagate$(m_{pk_i})$;
    add $B^r$ to local Blockchain;
  end if
end

Figure 8: Specification of Proof of Burn
Proof of Stake  Proof of Stake (PoS) requires validators to provide a stake in the cryptocurrency, the probability of a validator being chosen to vote on an upcoming block is proportional to the amount staked. The stakers are rewarded when a block they voted for reaches finality. Various mechanisms that result in loss of the validator’s stake were proposed to discourage dishonest behavior. Given a choice of blocks to vote for, validators are incentivized to vote for the block most likely to win the vote, creating a positive feedback loop which leads to convergence. Figure 9 shows the algorithm of the Slasher PoS proposal: for account $a_i$ with balance $b_i$, system difficulty $d$ at current block height $r$, with validator set for block $r$ denoted as $v_r$.

Algorithm 2: $computePoS_{pk_i}()$

begin
  waitTill(block $B^{r-1}$ is locally known);
  if $SHA256(SHA256(r-1) + a_i + timestamp) \leq 2^{256} * b_i/d$ then
    $a_i$ can join the validator set for $r + 3000$;
    $a_i$ must send a transaction accepting this for block $r + 3000$;
  end if
  if $a_i \in v_r$ then
    collect transactions;
    add blockheader;
    $m^r_{pk_i} = (B^r_{pk_i}, SIG_{pk_i}(H(B^r_{pk_i})))$;
    propagate$_{pk_i}(m^r_{pk_i})$;
  end if
  else
    waitTill(block $B^r$ is locally known);
    assert $SIG_{pk_j}(r)$;
    assert $SIG_{pk_j}(r)$ is unique at this block height;
  end if
  add $B^r$ to local Blockchain;
end

Figure 9: Specification of Proof of Stake

Practical Byzantine Fault Tolerance  Practical Byzantine Fault Tolerance (PBFT) [8] is a means of achieving consensus of the replicated state machines in asynchronous environments that exhibit arbitrary, possibly malicious faults. Consensus is achieved by clients voting for changes to the state machine in a series of rounds until a majority decision is achieved.

Proof of Authority  Proof of Authority (PoA) predetermines trusted nodes to seal blocks. At each round, a leader node is chosen that is entitled to seal and broadcast a block. The mechanism used to choose the leader varies with different PoA implementations. It can be as simple as a round robin through the set of validators. Blocks are sealed on top of the latest known block in the canonical chain. The block header includes the step and the leader’s signature against the block hash. The Ethereum test network Kovan, set up by a consortium of Ethereum companies, uses Proof of Authority as its consensus mechanism. Figure 10 shows this based on the AuRa PoA mechanism for block $r$ and step $s$ from $nv$ validators at Unix time $t$.

A summary of these different consensus mechanisms is seen in Table 1.
Algorithm 3: computePoA\(_{p,k}\)()

\begin{algorithm}
\begin{algorithmic}
\itembegin
\item waitTill(block \(B^{r-1}\) is locally known);
\item \(s = t / \text{step\_duration}\);
\item The primary node is the node \(p_j\) where \(j = \%nv\);
\item Chain Score \(cs = U128max * r - s\);
\item if \(i = j\) then
\item collect transactions;
\item add blockheader;
\item \(m^r_{pk_i} = (B^r_{pk_i}, SIG_{pk_i}(H(B^r_{pk_i})))\);
\item propagate\(_{pk_i}(m^r_{pk_i})\);
\item \end if
\item else
\item waitTill(block \(B^r\) is locally known);
\item assert\(_{SIG_{pk_i}}(s)\);
\item \end if
\item add \(B^r\) to local Blockchain;
\item \end
\end{algorithmic}
\end{algorithm}

Figure 10: Specification of Proof of Authority

<table>
<thead>
<tr>
<th></th>
<th>PoW</th>
<th>PoB</th>
<th>PBFT</th>
<th>PoS</th>
<th>PoA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction finality</td>
<td>Probabilistic</td>
<td>Probabilistic</td>
<td>Immediate</td>
<td>Economic</td>
<td>Immediate</td>
</tr>
<tr>
<td>Transaction rate</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Token needed</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Adversity Tolerance</td>
<td>49%</td>
<td>49%</td>
<td>33%</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>Scalability Nodes</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Transaction Scalability</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
</tbody>
</table>

Table 1: Properties of different consensus mechanisms. The adversity tolerance of PoW may well be as low as 33\% [17]

3 Approach to Access Control & Privacy By Default

Access controls restrict who may use what resources in which ways. Blockchains control the access to information, the permission to conduct transactions, and so forth. Our approach supports such mechanisms but strengthens them with additional layers of access control: a white list \(L\) of nodes that may be eligible to participate in the network; PPoKW, which determines an additional necessary condition for nodes to be eligible to perform certain tasks; and machine learning that makes the system and its access control adapt based on past system behavior.

These access controls and their management are also meant to be resilient to adversarial manipulation. For example, PPoKW (discussed in detail in Section 4) selects a random group of nodes for a new task such that this selection process can neither be manipulated by an adversary nor can he exploit that process to amplify his existing system influence.

Another important aspect of the XAIN framework is its support for Privacy By Design and Default. We now sketch the privacy-preservation techniques of our XAIN framework and its
pertinent access-control layers, which interact with PPoKW in a manner discussed in Section 4.

3.1 Key Management Procedures

3.1.1 Privacy

A core feature of Blockchains is transparency: every node can inspect the content of each block on the chain. This makes it challenging to realize general or application-specific privacy requirements. Data on the Blockchain is visible to anyone, both as the current state of the EVM and as details of any transactions that produced that state. Typically, data that is required to be private needs to be encrypted before it is added to the Blockchain or IPFS.

For audibility requirements, only the hash of data needs to be on the Blockchain. Storing hashes only also has privacy benefits: mere possession of a hash gives no knowledge other than that some data has been added by (the Ethereum address of) the originator.

3.1.2 Key Management

System Keys At the protocol level, keys used by participants are Ethereum public/private key pairs. The private key \( sk_i \), denoted by \( pr_i \) in [50], is a randomly selected positive integer represented as a byte array of length 32 in big-endian form in the range \([1, \text{secp256k1n} - 1]\). We assume that an adversary could generate any amount of such key pairs at negligible cost.

Application Keys We consider 2 scenarios:

1. Users have direct control over their public/private key pairs. The identity of users is then defined by this key pair.

2. Users have an intermediary client system to restrict access to the XAIN network. This client (centralized and integrated with other legacy systems) is responsible for the creation of public/private key pairs. But users are identified and access is controlled by the key specific to the client system. In this scenario, it is possible that multiple Ethereum addresses could be mapped to one user, or possibly ephemeral key pairs would be used.

3.2 Node White List

We restrict access to the system in 2 ways

1. At a network level, we specify those nodes which are allowed to participate in the system, via a combination of a public key and an IP address, for example

   \( \text{enode} : //e847d...b2d707e77b71d6fbc8@52.59.97.160 : 30303 \)

   We expect changes to this list to be rather infrequent.

2. We maintain a white list of public keys that are permissioned in our system. This white list \( L \) specifies which public keys are, in principle, eligible to be members of blockheight-specific committees, such as the mining committee.

   The white list \( L \) is realized as a set of mappings \( \{ P_k : W \} \) between Ethereum public keys \( P \) and a weighting \( W \) in interval \([0, 1]\). Permission to partake, in principle, in tasks such as mining is given to a key \( P_i \) with \( W_i > T \) where \( T \) is a threshold determined at system startup.

   A pre-compiled smart contract contains the white list of nodes \( L \), and the mechanisms used to implement actions to secure and stabilize the system. The functions of this contract are called by special transactions generated by the anomaly-detecting nodes. The advantages of implementing this functionality in a smart contract, rather than in the client software, are
1. *Enforced applicability:* all clients, however implemented, use the same rule sets and data.

2. *Transparency:* the rules governing the system can be easily inspected and understood.

3. *Well-defined change procedures:* rules around changing system parameters, e.g. the white list, can be defined in one place, reducing the complexity of the attack surface.

This smart contract, being part of the Blockchain, can be queried by any of the nodes on the network. The white list of nodes $L$ is initially populated with approved addresses in the genesis block $B^0$. Thereafter, nodes can be added and removed from that list, based on special transactions created by anomaly-detection nodes.

### 3.2.1 Regeneration of Keys

**Generation of the public key from a known private key**  An Ethereum address is defined as a 160-bit string. For a given private key, $sk_i$, the Ethereum address $A(sk_i)$ corresponding to that private key is a 160-bit value, and is defined as the right-most 160 bits of the Keccak hash of the corresponding ECDSA public key [50]:

$$A(sk_i) = B_{96..255}(KEC(ECDSA(PUBKEY(sk_i)))) \quad (17)$$

where PUBKEY($sk_i$) computes the public key $pk_i$ corresponding to $sk_i$. Thus, this involves

1. Derivation of the public key from the private key (512 bits).
2. Derivation of the address from this public key (160 bits).

**Generation of private key from a seed phrase**  DApp clients that create key pairs for users may produce a 12-word seed phrase to allow users to regenerate private keys, in case they are lost. The mnemonic code converter from the BIP39 project is used to create the phrase [42].

### 3.2.2 Key Revocation Schemes

Keys can be removed from white list $L$ based on a series of decisions made by detector nodes, see Section 7 for details. The detector nodes produce a transaction $T$ which results in

1. an unchanged white list
2. For node $i$ in the list, a transition in its weight $W_i$ to $W_i - \epsilon$

Here, $\epsilon$ denotes a system parameter that is constrained such that

- $\epsilon$ is large enough so that malicious nodes can be removed in a timely manner
- $\epsilon$ is small enough so that the number of malicious transactions required for successful removal of an honest node is large enough to have a sufficiently low probability.

**Storage Keys**  The permissioning of data access is controlled through proxy re-encryption nodes. Through these, a data owner can revoke access for a particular key to a data item.
3.2.3 GDPR Compliance

The EU General Data Protection Regulation (GDPR) replaces the Data Protection Directive 95/46/EC and will be intacted after May 28, 2018. GDPR is meant to harmonize data protection across Europe and to set a global standard for regulating privacy. It brings significant changes:

- **Extra-Territorial Scope**: It applies to companies that process data of citizens residing in the EU, regardless of the company’s own location.

- **Penalties**: Each GDPR violation may incur punitive fees of up to 4% of a company’s global annual turnover.

- **Consent**: Consumers get stronger rights regarding use of their personal data.

- **Privacy By Design**: This technical aspect is now a legal requirement for data controllers when designing their data processing systems.

- **Breach Notification**: Stronger requirements are in place when the notification of a breach becomes mandatory.

**Privacy by Design**  GDPR compliant software needs to apply so called *Privacy by Design and Default* (Article 25). This addresses privacy risks, not only as a legal restriction for processing personal data, but to also meet privacy concerns in early-stage design of IT architecture. This includes principles such as minimizing data collection to its intended use or use of incorporated mechanisms for data deletion (e.g. for the Right To Be Forgotten), updating, and portability.

Regulated use of private and permissioned Blockchain solutions will facilitate the exploration of GDPR issues for fully decentralized and public Blockchains, such as Bitcoin or Ethereum. The European Data Protection Supervisor has called for the development of a privacy-friendly BCT [6] in order to understand how data protection principles can be applied to it.

**Proof of Compliance**  Hence, we propose two industry-ready solutions in order to enable a privacy-friendly Blockchain technology architecture:

1. Our data is addressed by the hash of its content. Thus, after data removal from decentralized storage (and unpinning the file from any nodes), we can prove that data removal by re-requesting it from the storage system, and by showing that that data does not exist. This re-request and its result can be stored in the Blockchain as a Proof of Revocation.

2. The approach described in the previous item lays the foundation for enabling other privacy mechanisms, such as

   (a) updating data, which would be achieved by appending (new) updated data after revocation;

   (b) sharing data between users by re-encrypting the data to allow access;

   (c) porting data to another service via proxy re-encryption, followed by the process outlined in item 1. above.

For all these privacy mechanisms we would store the corresponding facts of the actions taken as Proofs of Compliance on the Blockchain.

A GDPR-compliant process – specific to a given use case – can then be defined, individualized and built into the user interface by mapping out or combining processes for consent, access, update and erasure policy.
4 Practical Proof of Kernel Work: PPoKW

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 [31] already developed means of minimizing energy consumption of PoW in the “governed Blockchain” setting [32], 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 $B_{r-1}$.

4.1 Algorithmic Introduction to PPoKW

The process of electing a leader – who can propose the next block on the chain – relies on Proof of Work (PoW). We restrict the PoW mining race for the next block by two mechanisms:

- A dynamic White List $L$ which is authenticated on the Blockchain and maintains those public keys that are, in principle, eligible to participate in a PoW mining race.
- An adaptive node selection mechanism, illustrated in Figure 11, based on Cryptographic Sortition as introduced in Algorand [19, 10]: determines the superset of nodes that may be eligible to participate in specific tasks of Blockchain construction and management.

These tasks include mining, machine learning, and management of the white list $L$. Below, we will make machine learning a task that is (periodically) subsumed by the task of mining. Our approach lends itself to other realizations of task divisions and dependencies, though.

The above cryptographic eligibility is necessary but not sufficient for engaging in a task: white list $L$ or other security state may override such eligibility as discussed below.

The utility of the first mechanism has already been described in the previous section, including how it can support adaptive access control informed by anomaly detection. The second mechanism provides the following:

- at each blockheight $r \geq 1$, the set of nodes eligible to complete a task – such as mining a block $B_r$ – are randomly selected from the set of all nodes on the white list $L$
- this selection process is sufficiently random and only the selected nodes themselves will know that they are selected
- an adversary who can compromise nodes cannot exploit this selection mechanism to inform which nodes it aims to compromise
- the expected number of selected nodes is a control parameter of the Blockchain system.

This second mechanism therefore offers many advantages. For one, a system can control the expected number of nodes that participate in specific tasks such as mining or administering the white list $L$. This can save energy costs as fewer miners will lose a mining race.

It can also modify 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. We note that these advantages apply equally to Blockchain systems that are open (any node may join the system) or closed (only specified nodes are permitted to participate in the system).
Moreover, the Blockchain has a system parameter $p \geq 1$. Its ensures that all mining races of blockheights $r$ that are multiples of $p$ are also doing machine learning that will inform the modification of system parameters, whose changes would be embedded in the next block $B^r$. One important such system parameter is the expected number of nodes that participate in mining races for the next $p$ or so blocks. The actual set of nodes eligible to participate in a specific task at blockheight $r$ is a function of the White List $L$ at blockheight $r - 1$, the local Blockchain, and the superset generated by the second mechanism above.

### 4.2 Algorithmic Specification of PPoKW

We will now specify PPoKW in sufficient detail so that we can explain further below how the Ethereum platform can be instantiated with PPoKW as chosen consensus mechanism.

**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 $\phi$. We also strive for clarity over code elegance in this yellow paper. An implementation will optimize algorithmic function. Whenever we talk about destroying keys, this means secure memory erasure.

**Block structure** Using the notation in [19, 10], we let a block have *conceptual* structure

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

where

- $r$ is the blockheight,
- $TSet^r$ the payload of transactions of the application domain,
- $Q^r$ the seed of that block – a concept introduced in [19, 10],
• $H(B_{r-1})$ the hash of the previous block,

• \textit{nonce} is the value that demonstrates Proof of Work,

• $k$ a security parameter similar to that used in Algorand [19, 10],

• $p$ a period for system parameter adjustments, based on machine learning, which are meant to happen every $p$th block,

• $n_w$ 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 $\text{sig}_{pk}(m)$ refers to the digital signature of message $m$ with the private key $sk$ that corresponds to the public key $pk$. The definition of $\text{SIG}_{pk_i}(m)$ given in Figure 12 embeds the public key $pk_i$ into the signed message. One could alternatively embed identity information. Such implementation details of identity management may vary with the application domain.

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 $b = 2$). We make use of the concept of Cryptographic Sortition, an important technical ingredient of Algorand [19, 10], 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

$$0.H(\text{SIG}_{pk_i}(r, task, Q_{r-1})) < \frac{n_w}{|PK_{r-k}|}$$

(19)

that is, 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 that are – in principle – eligible to perform task $task$, based on the blocks from $B^0$ up to $B_{r-k}$. The current Blockchain $B^0 ... B_{r-1}$ creates consensus for the values of $Q_{r-1}$ and $n_w$, and indirectly for the set of keys $PK_{r-k}$.

Functions $\text{getEligibleKeys}_{pk_i}$ and $\text{getTransactionsForNextBlock}_{pk_i}$ refer to local state of node $pk_i$: its local Blockchain $B^0, ..., B_{r-1}$ for the former, 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). Note that an implementation that uses machine learning to update system policy may make functions $\text{getEligibleKeys}_{pk_i}$ and $\text{getTransactionsForNextBlock}_{pk_i}$ dependent on the current state of such policy; for example, this would allow one to ban or add public keys, block or prefer certain types of transactions, and so forth. Our White List, discussed below, is an example of such policy specification and enforcement. The pseudo-code below indicates which functions may want to extract values of local variables such as $k$ and $n_w$ from the Blockchain. This may not be done at each blockheight or step in an implementation or shared across functions. But such parameter values need to be embedded in the chain and periodically verified, not least because machine learning may change them over time.

Figure 12 shows some functions that we use as primitives:

(a) Specification of digital signatures for node $pk_i$ – where $\text{sig}_{pk_i}(m)$ refers to the digital signature of message $m$ under the private key for $pk_i$.

(b) Specification of checking whether node $pk_i$ is eligible, in principle, to participate in task $task$ (where task $\text{mine}$ refers to a mining race) for the block with blockheight $r$: $PK$ refers to a set of eligible public keys – a subset of the White List $L$.

(c) Function name for extraction of set of eligible public keys from local Blockchain state. The implementation of this function will also reflect the logic of the White List $L$. 

34
(d) Function name for computing local transactions that should be in the next block (should node \( pk_i \) happen to win the mining race).

The determination of set \( PK \) is also a function of the security parameter \( k \): the intuition is that only entities that interacted with the Blockchain in its history \( B^0, \ldots, B^{r-k} \) may be elements of \( PK \). In particular, this prevents an adversary from introducing such entities in more recent blocks in order 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) will be application-specific.

\[
SIG_{pk_i}(m) \quad \{ \quad \text{% (a)} \quad \text{return } (pk_i, m, sig_{pk_i}(m)) \; \}
\]

\[
\text{mayPerform}_{pk_i}(r, task) \quad \{ \quad \text{% (b)} \quad \text{assert } 0 < r; \quad \text{extract } k, n_w, Q^{r-1} \text{ from } B^{r-1}; \quad \text{assert } (0 < k) && (0 < n_w); \quad PK = \text{getEligibleKeys}_{pk_i}(r, k); \quad \text{assert } n_w < |PK|; \quad \text{return } (pk_i \in PK) && (0.H(SIG_{pk_i}(r, task, Q^{r-1})) \leq \frac{n_w}{n_{w}^{r-1}}); \}
\]

\[
\text{getEligibleKeys}_{pk_i}(r, k) \quad \{ \quad \text{% (c)} \quad \text{assert } (k \leq r - 1); \quad \text{return set of all eligible keys from local blocks } B^0, \ldots, B^{r-k}; \}
\]

\[
\text{getTransactionsForNextBlock}_{pk_i}(r) \quad \{ \quad \text{% (d)} \quad \text{return set of local transactions to be included in block } B^r; \}
\]

Figure 12: (a) Specification of digital signatures for node \( pk_i \), \( sig_{pk_i}(m) \) refers to the digital signature of message \( m \) with private key for \( pk_i \). (b) Specification of checking whether node \( pk_i \) is a potential participant in the task \( task \). (c) Function name for extraction of set of eligible public keys from local Blockchain state. (d) Function name for computing local transactions that should get into the next block (should node \( pk_i \) become leader). The manner in which outputs for (c) and (d) are computed will be 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 they will perform if indeed eligible. This is seen in Figure 13. Node \( pk_i \) waits till it knows 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 takes place, the level of difficulty \( d \) for Proof of Work, and the expected size \( n_w \) of the set of nodes that are permitted to mine the next block \( B^r \).

Then 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, a function of the local block state, security parameter \( k \), and the White List \( L \) (left implicit).

The function \( \text{mayPerform}_{pk_i} \) 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 \( B^r \). Note that only node \( pk_i \) will know 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
Algorithm 4: 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 ≤ 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;
// pk_i can participate in the PoW mining race for this blockheight
TSet^r = getTransactionsForNextBlock(pk_i, r);
if (r == 0 \% p) then
    newSysValues = newPar(pk_i, doMachineLearning(pk_i))
else
    newSysValues = (k, p, n_w, ...);
end if
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(B^{r}_{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, sk'_r);
\[ m^r_{pk_i} = (B^r_{pk_i}, SIG_{pk_i}(H(B^r_{pk_i})), SIG_{pk_i}(r, nonce, Q^{r-1})) \]
\[ destroy sk'_r; \]
\[ propagate_{pk_i}(m^r_{pk_i}); \]
end

Figure 13: Specification of PoW for potential leaders for blockheight r. Nodes \( pk_i \) start mining only if they are eligible. New systems values are the same as the old ones when \( r \) not a multiple of \( p; \) otherwise, the new system values are determined by local machine learning.

output decides whether node \( pk_i \) is indeed eligible. In particular, no other node can produce this signature as they are not in possession of the corresponding secret key \( sk_i \).

We note that an attacker who compromises node \( pk_i \) and so also knows the secret key \( sk_i \) will then know 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 won’t know which nodes to compromise before the next mining race, machine-learning process, and so forth is about to start. This provides crucial resiliency to this Blockchain system architecture.

Returning to the code for mining, if node \( pk_i \) is not selected for mining the block with block-
height $r$, function $\text{computePoW}_{pk_i}$ stops execution. Otherwise, node $pk_i$ computes the set of transactions $TSet'$ to be included in block $B^r$ – dependent on the application logic for transactions. Next, it checks whether the block $B^r$ is one in which system parameters may be updated according to machine learning insights.

If so, it computes $\text{newSysValues}$ – a tuple of form $(k', p', n'_w, ...)$ – as result of such machine learning (function $\text{doMachineLearning}_{pk_i}$) and how these machine learning outcomes inform decisions (function $\text{newPar}_{pk_i}$) that modify system parameters. Otherwise, $r$ is not such a periodic block, and then the value of $\text{newSysValues}$ defaults to those from the previous block $B^{r-1}$ so that system parameters won’t be adjusted in block $B^r$.

Candidate blocks are computed as follows. The new seed $Q^r$ is computed as the signature of the previous seed $Q^{r-1}$. This ensures that the new seed is sufficiently random, and that an adversary cannot really influence the value of the next seed – in a model of an adversary that is similar to that used for Algorand in [19, 10].

If PoW has been found (tested with function $\text{isPoW}_{pk_i}$), the block (which contains the nonce for which PoW was found) together with a signature of its hash and a credential (a signature of the blockheight, nonce, and previous seed) are sent 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 is then added to his local Blockchain, and its own mining efforts stop.

Function $\text{raceEnded}_{pk_i}$ 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.

Note that the message to be broadcasted contains a signature of the hash of this next block but signed with an ephemeral private key, which also gets destroyed once the message has been sent. This ensures that an attacker who compromises nodes, can no longer manipulate signatures of such hashes, for example in order 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 nodes can easily learn the corresponding set of public keys. There are standard solutions to this, for example the use of identify-based public-key cryptography.

5 Robust Consensus Optimization

5.1 Motivation

For PPoKw, we now also have the means to optimize the Blockchain network and particularly the underlying consensus mechanisms in terms of minimum security and stability requirements. Using such 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 long time.

But 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. As such, we develop a new network architecture – the eXpandable Artificial Intelligence Network (XAIN) – which uses an Ethereum structure, yet, integrates in it the PPoKw consensus algorithm and also supports an optimization framework with various tasks to fulfill: First of all, we allow for permissioning and network policies to integrate our defined KYC and access-control layers for further verifications. This permissioning restricts the possible nodes and, thus, wallets in the network as well as the mining activity. That all of these restrictions are in place and work effectively is absolutely crucial when dealing with highly sensitive corporate data. The optimization is then pursued as a distributed reinforcement learning process, see Section 6.
The optimization of the consensus mechanism plays hereby the largest role and will be defined according to general assumptions that are realized at the protocol level. Having such assumptions integrated into our system, we can build faithful models that measure and influence the security, cost and stability of the entire system, in order to find an optimal equilibrium of the system state, based on the required parameters and lying within a certain monetary budget. The latter is particularly important, since open Proof of Work consensus systems are not fitting within any budgetary boundaries in the long term. And other approaches, such as Proof of Stake and PBFT, are clearly not suitable to function in systems that require a high stability, resiliency, and long-term security of data.

5.2 Mathematics for Governed PPoKW

The modeling we now present is based on the work done in the publications [30, 31]. Recall that 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 \( h: \{0,1\}^p \rightarrow \{0,1\}^n \) such that \( h \) has puzzle friendliness [39]. 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 \( T > 0 \) for the time 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 will treat \( h \) in the Random Oracle Model (ROM): function \( h \) is chosen uniformly at random from all functions of type \( \{0,1\}^p \rightarrow \{0,1\}^n \); 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 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 \( h \) will be of form \( x = data || nonce \) 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 will use an input to \( h \) at most once, be it within or across miners’ input spaces.

This assumption and ROM give us that hash values are always uniformly distributed in the output space during a mining race. Now consider having \( s > 1 \) many miners that run in parallel to find Proof of Work, engaging thus in a mining race.

Our use of Cryptographic Sortition for PPoKW suggest that \( s \) represents the expected value \( n_w \) of how many nodes will be eligible to participate in the mining race for the next block.

We assume these miners run with the same configurations and hardware, for sake of simplicity of presentation; we have refined mathematical models that allow nodes to have one of finitely many 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 have the need for incentive structures.

But 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

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

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

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

Basic event failure denotes the event that all nonzero 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 $(data_1, data_2, \ldots, data_s)$ 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 3.**

1. The $s$-tuple $(\text{failure}, \ldots, \text{failure})$ 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 $(\text{failure}, \ldots, \text{failure})$ 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 $(\text{failure}, \ldots, \text{failure})$, we define

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

So 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 overall failure of the mining race, the equivalence class containing only tuple $(\text{failure}, \ldots, \text{failure})$. The set of events for the PoW race of $s$ miners is thus

$$E^* = \{ \otimes^k \cdot \checkmark \mid 0 \leq k \leq \lambda \} \cup \{ \text{failure} \}$$

In (21), expression $\otimes^k \cdot \checkmark$ denotes an element of the quotient

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

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

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

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

\[
\text{prob}^s(\otimes^k \cdot \check{	ext{✓}}) = (1 - 2^{-d})^k \cdot s \cdot [1 - (1 - 2^{-d})^s] \tag{22}
\]

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, \( \text{prob}^s \) becomes a probability distribution with the non-zero probability \( \text{prob}^s(\text{failure}) \) being \( 1 - \text{prob}^s(E^s \setminus \{\text{failure}\}) \), that is

\[
\text{prob}^s(\text{failure}) = (1 - 2^{-d})^{s \cdot (\lambda + 1)} \tag{23}
\]

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 do decrease as \( s \) increases.

### 5.3 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 with respect to an objective function. The latter may be a cost function, if containing cost is the paramount objective or if a first cost estimate is sought that can then be transformed into a constraint to optimize for a security objective – for example to maximize the level of difficulty \( d \), as seen further 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. But lower values of \( d \) may be needed, for example, in high-frequency trading where performance can become a real issue. We want to understand such trade-offs. Moreover, we want to explore how corruption of some miners or inherent uncertainty in the number of deployed miners or in 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 first consider an objective function

\[
\text{Cost}(s, r, d) = \text{TVC} \cdot E^s(\text{noR}) \cdot s + \text{TFC} \cdot s \tag{24}
\]

that models cost as a function of the number of miners \( s \), the bit size of the nonce \( r \) – implicit in random variable \( E^s(\text{noR}) \), and the level of difficulty \( d \); where we want to minimize cost.

The real variable \( \text{TVC} \) 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 \( \text{TFC} \)
$$0 < s_l \leq s \leq s_u \quad 0 < d_l \leq d \leq d_u \quad 0 < r_l \leq r \leq r_u \quad \kappa \geq \text{prob}^*(\text{failure})$$

$$\tau_u \geq T \cdot E^*(\text{noR}) \geq \tau_l$$

$$\delta \geq \text{prob}^*(\text{PoWTime} > \text{th})$$

$$\delta_1 \geq \text{prob}^*(\text{PoWTime} < \text{th}')$$

Figure 14: Constraint set $C$ for two optimization problems: (a) minimize $\text{Cost}(s, r, d)$ as in (24) subject to constraints in $C$; and (b) maximize $d$ subject to $C \cup \{ \text{Cost}(s, r, d) \leq \text{budget}\}$ for cost bound budget. This is parameterized by constants $0 \leq \delta, \delta_1, \delta_2, \kappa, \text{th}, \text{th}', \tau_l, \text{TVC}, \text{TFC}$ 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

models the fixed costs of having one miner; this can be seen as modeling procurement and depreciations. Variables $s, r,$ and $d$ are integral, making this a mixed integer optimization problem [23]. The expression $E^*(\text{noR})$ 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 MINLP optimization problem [47, 23].

We may of course use other objective functions. One of these is simply 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 14 shows an example of a set of constraints and optimizations of security and cost for this.

Integer constants $s_l$ and $s_u$ bound variable $s$, and similar integer bounds are used to constrain integer variables $r$ and $d$. The constraint for $\kappa$ 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 $\tau_l$ and $\tau_u$.

The real constant $\delta_2$ is an upper bound for

$$\text{prob}^*(\text{disputes within } \mu)$$

the probability that more than one miner finds PoW within $\mu$ seconds in the same, approximately synchronous, mining race. The constraint for real constant $\delta$ says that the probability

$$\text{prob}^*(\text{PoWTime} > \text{th})$$

of the actual time for mining a block (for the expected number of miners) being above a real constant $\text{th}$ is bounded above by $\delta$. This constraint is of independent interest: some systems need to assure that blocks are almost always (with probability at least $1 - \delta$) mined within a specified time limit.

Some systems may also need assurance that blocks are always mined in time exceeding a specified time limit $\text{th}'$. We write

$$\text{prob}^*(\text{PoWTime} < \text{th}')$$


to denote that probability, and add a dual constraint, that 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 $\text{th}'$.

**Constraints as Analytical Expressions** We derive analytical expressions for random variables occurring in Figure 14. Beginning with $E^*(\text{noR})$, we have

$$E^*(\text{noR}) = \sum_{0 \leq k \leq \lambda} \text{prob}^*(\otimes^k \cdot \checkmark) \cdot (k + 1) \quad (25)$$
which we know to be equal to
\[
\sum_{0 \leq k \leq \lambda} (1 - 2^{-d})^k \cdot [1 - (1 - 2^{-d})^s] \cdot (k + 1)
\]
We may rewrite the latter expression so that summations are eliminated and reduced to exponentiations: we rewrite \(\sum_{0 \leq k \leq \lambda} \text{prob}(\otimes^k \cdot \checkmark) \cdot (k + 1)\), the right-hand side of (25), to \(\lambda + 1\) summations, each one starting at a value between 0 and \(\lambda\), where we use the familiar formula
\[
\sum_{k=a}^{b} x^k = \frac{x^a - x^{b+1}}{1 - x}
\]
This renders
\[
E^s(\text{noR}) = \frac{1 - y^{\lambda+1} - (\lambda + 1) \cdot (1 - y) \cdot y^{\lambda+1}}{1 - y}
\]
where we use the abbreviation
\[(1 - 2^{-d})^s \quad (27)\]
The expected time needed to get a proof of work for input data is then given by
\[
E^s(\text{PoW}) = T \cdot E^s(\text{noR})
\]
We derive an analytical expression for the probability \(\text{prob}^s(\text{PoWTime} > th)\) next. Note that \((th/T) - 1 < k\) models that the actual time taken for \(k + 1\) hash attempts is larger than \(th\). Therefore, we capture \(\text{prob}^s(\text{PoWTime} > th)\) as
\[
\sum_{[(th/T) - 1] < k \leq \lambda} \text{prob}^s(\otimes^k \cdot \checkmark) = y^{[(th/T) - 1] + 1} - y^{\lambda+1}
\]
assuming that \([(th/T) - 1] < \lambda\), the latter therefore becoming a constraint that we need to add to our optimization problem. One may be tempted to choose the value of \(\delta\) based on the Markov inequality, which gives us
\[
\text{prob}^s(\text{PoWTime} \geq th) \leq T \cdot E^s(\text{noR})/th
\]
But that upper bound \(T \cdot E^s(\text{noR})/th\) depends on the parameters \(s, r, d\); for example, the analytical expression for \(E^s(\text{noR})\) in (26) is dependent on \(\lambda\) and so dependent on \(r\) as well. The representation in (29) also maintains that expression \(y^{[(th/T) - 1] + 1} - y^{\lambda+1}\) is in \([0, 1]\), i.e. a proper probability. Since \(y = (1 - 2^{-d})^s\) is in \([0, 1]\), this is already guaranteed if \([(th/T) - 1] + 1 \leq \lambda + 1\), i.e. if \([(th/T) - 1] \leq \lambda\). But we already added that constraint to our model.

Similarly to our analysis of \(\text{prob}^s(\text{PoWTime} > th)\), we get
\[
\text{prob}^s(\text{PoWTime} < th') = 1 - (1 - 2^{-d})^{s \cdot \left(\lfloor (th'/T) - 1\rfloor + 1\right)} = 1 - y^{\lfloor (th'/T) - 1\rfloor + 1}
\]
which needs \(0 < \lfloor (th'/T) - 1\rfloor\) as additional constraint.

To derive an analytical expression for \(\text{prob}^s(\text{disputes within } \mu)\), each miner can perform \(\lfloor \mu/T \rfloor\) hashes within \(\mu\) seconds. Let us set
\[
w = (1 - 2^{-d})^{\lfloor \mu/T \rfloor + 1}
\]
The probability that a given miner finds PoW within \(\mu\) seconds is
\[
\sum_{k=0}^{\lfloor \mu/T \rfloor} (1 - 2^{-d})^k \cdot 2^{-d} \cdot \frac{1 - (1 - 2^{-d})^{\lfloor \mu/T \rfloor + 1}}{1 - (1 - 2^{-d})} = 1 - w
\]
\[ s_t \leq s \leq s_u, \quad d_t \leq d \leq d_u, \quad r_t \leq r \leq r_u, \quad \lambda = \lfloor 2^r/s \rfloor \]

\[ y = (1 - 2^{-d})^s, \quad w = (1 - 2^{-d})^{[\mu/T] + 1}, \quad 0 \leq [\mu/T] \]

\[ \kappa \geq y^{\lambda+1}, \quad [(th/T) - 1] < \lambda \quad 0 < [(th'/T) - 1] \]

\[ E^s(\text{noR}) = \frac{1 - y^{\lambda+1} - (\lambda + 1) \cdot (1 - y) \cdot y^{\lambda+1}}{1 - y} \]

\[ \tau_u \geq T \cdot E^s(\text{noR}) \geq \tau_1 \quad \delta_1 \geq 1 - y^{\lfloor (th'/T) - 1 \rfloor + 1} \]

\[ \delta \geq y^{\lfloor (th/T) - 1 \rfloor + 1} - y^{\lambda+1} \]

\[ \delta_2 \geq 1 + (s - 1) \cdot w^s - s \cdot w^{s-1} \]  

Figure 15: Arithmetic version of set of constraints \( C \) from Figure 14, with additional soundness constraints for this representation. 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 (36).

Therefore, the probability that no miner finds PoW within \( \mu \) seconds is

\[ \text{prob}^s(0 \text{ PoW within } \mu) = (1 - (1 - w))^s = w^s \]  

(33)

The probability that exactly one miner finds PoW within \( \mu \) seconds is

\[ \text{prob}^s(1 \text{ PoW within } \mu) = s \cdot w^{s-1} \cdot (1 - w) \]  

(34)

Thus, the probability that more than one miner finds PoW within \( \mu \) seconds is

\[ \text{prob}^s(\text{disputes within } \mu) = 1 - \text{prob}^s(0 \text{ PoW within } \mu) - \text{prob}^s(1 \text{ PoW within } \mu) \]

\[ = 1 - w^s - s \cdot w^{s-1} \cdot (1 - w) \]

\[ = 1 - w^s - s \cdot w^{s-1} + s \cdot w^{s-1} \cdot w \]

\[ = 1 + (s - 1) \cdot w^s - s \cdot w^{s-1} \]  

(35)

Figure 15 shows the set of constraints \( C \) from Figure 14 with analytical expressions and their additional constraints, e.g., \( 0 \leq [\mu/T] \) for the analytical representation of \( \text{prob}^s(\text{disputes within } \mu) \).

5.4 Robust Design Security

Our model 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, needed 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 they may simply break down. Robust optimization treats such uncertainty as 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 \( c > 0 \) many subsequent mining races is then seen to be \((l/s)^c\). We can therefore bound this with a constant \( \delta_3 \) as in Figure 15.
We model uncertainty in the number of miners available by an integer constant \( u \) as follows: if \( s \) miners are deployed, then we assume that at least \( s - u \) 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 \( u \) can model aspects such as denial of service attacks or a combination of such attacks with faults: \( u = 3 \), 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 us results that are robust in that they hedge against the fact that values \( d' \) satisfying

\[
|d - d'| \leq u_d
\]

may be the actually running level of difficulty. This enables us to understand a design if we are unsure about which level of difficulty will be deployed or if we want some flexibility in dynamically adjusting the value of \( d \) in the running system. The corresponding robust optimization problem for cost minimization is seen in Figure 16. It adds to the constraints we already consider further requirements on constants \( l, c \), and \( \delta_3 \) as well as the constraint

\[
l^c \leq \delta_3 \cdot s^c
\]

Robustness of analysis is achieved by a change of the objective function from \( \text{Cost}(s, r, d) \) to

\[
\text{Cost}^{u_s,u_d}(s, r, d) = \max_{s - u \leq s' \leq s, |d - d'| \leq u_d} \text{Cost}(s', r, d')
\]

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 (37) for a triple \((s, r, d)\) are only considered for optimization if all triples \((s', r, d')\) used in (37) are feasible – realized with predicate \( \text{feasible}^{u_s,u_d} \): robust optimization guarantees [1] that the feasibility of solutions is invariant under the specified strict uncertainty (here \( u_s \) and \( u_d \)).

\[
\min \{ \text{Cost}^{u_s,u_d}(s, r, d) \mid \text{feasible}^{u_s,u_d}(s, r, d) \}
\]

subject to the set of constraints \( \mathcal{C} \) from Figure 15 together with

\[
\begin{align*}
4 &= l < s & c &= 6 & 0.001 &= \delta_3 \\
l^c &\leq s^c \cdot \delta_3 & u_s &= 5 & u_d &= 3
\end{align*}
\]

Figure 16: Robust cost optimization for the constraints from Figure 15, where up to \( u_s = 5 \) miners may be non-functioning or mis-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 \( \text{feasible}^{u_s,u_d}(s, r, d) \) characterizes robustly feasible triples; it is true iff all triples \((s', r, d')\) with \( s - u \leq s' \leq s \) and \( |d - d'| \leq u_d \) are feasible.

We refer to the publication [31] 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.

### 5.5 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. An 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 is able to 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 \( \text{mayPerform}_{pk_i}(r, \text{mine}) \) already incorporates the White List \( L \) as a filter. Moreover, 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 is becomes more likely for a node from the smaller set \( PK \) to be selected by Cryptographic Sortition. Conversely, a longer White List \( L \) will increase the size of \( PK \), and then \( n_w/|PK| \) becomes smaller – meaning that nodes in \( PK \) become less likely to be selected by Cryptographic Sortition.

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. It turns out that this percentage is meaningful in the context of our Cryptographic Sortition as node selection mechanism. Indeed, let \( N \) be the number of nodes of the total network, and so \( n_w << N \) 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 for 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. So if \( s \) many nodes are selected for mining as determined by function \( \text{mayPerform}_{pk_i}(r, \text{mine}) \) discussed in a previous section, then we know that the expected value of \( s \) is \( n_w \). Moreover, if we set \( l = \lceil q \cdot s \rceil \), then the percentage \( l/s \) is as close to \( q \) as it can be given that \( l \) and \( s \) are integral.

So 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, a minimal value for reasonable security.

Then an adversary can compromise one in four nodes. Since the four selected miners are determined by a random process, 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, he needs to compromise 5,000 nodes in a network of 10,000 nodes, but he will 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. And the reasoning about the probability of winning \( k \) 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.
6 Automated Run-Time Stabilization of Network

6.1 Machine Learning Engines in Permissioned Blockchains

Given a budget, an engine optimizes for stability by adjusting internal parameters of a Permissioned Blockchain at run-time. Optimization decisions taken by an engine are based on a continually evolving machine-learning model.

**Model** We define the set of all possible states the system can be in as $B$ (i.e. the last $k$ blocks from the chain including operational meta-data concerning those blocks). The set of all possible run-time parameters an engine can control is defined as $C$. With that an engine can be defined as the function

$$\text{engine}: B \rightarrow C$$

mapping from Blockchain state to control parameters. We define such engines by appeal to a set $S$ of neural network parameters and a set $A$ of actions. Three steps determine the mapping:

- Feature preprocessing $\text{preprocess}: B \rightarrow S$, transforming $B$ to a set of neural network parameters $S$
- Neural network approximation of optimal policy function $\pi: S \rightarrow A$
- Action validation and mapping to control parameters: $\text{validate}: A \rightarrow C$

Overall, an engine can be formally described as a composite function:

$$\text{engine}(b) = (\text{validate} \circ \pi \circ \text{preprocess})(b)$$

6.2 Environment Agent Model for PPoKW

**State** We represent state $S$ using $k$ triplets each holding the mining time, difficulty, and approximated hash rate (at the time of mining block $B^k$) of the top $k$ blocks of the chain.

$$S = (\text{hash rate}, \text{mining time}, \text{difficulty})^k$$

**Reward** Agent behavior follows rewards associated with $(s,a)$ pairs for $s$ in $S$ and $a$ in $A$. Following our optimization objective (i.e. a maximum mining time of $\text{target}$ seconds), we define

$$\text{robust}_\text{target} = 0.8 \ast \text{target}$$

(41)

To determine rewards. We assign positive rewards if the agent is close to the $\text{robust}_\text{target}$, and decreasing (even negative) rewards the further an agent diverges from $\text{robust}_\text{target}$:

$$\text{reward} = -m \ast (\text{robust}_\text{target} - \text{mining time}_k)^2 + 1$$

The scale of the target can be adjusted using parameter $m$.

**Terminal state** Function $\text{terminal}: S \rightarrow \{\text{True, False}\}$ returns True if state $s \in S$ is terminal:

$$\text{terminal}(S) = \begin{cases} 
\text{True}, & \text{if } \text{mining time}_k < \text{robust}_\text{target} - \text{threshold} \\
\text{True}, & \text{if } \text{mining time}_k > \text{robust}_\text{target} + \text{threshold} \\
\text{False}, & \text{otherwise}
\end{cases}$$
**Action Space**  Run-time parameter adjustments are expressed as a set of actions $A$, affecting both the level of difficulty $d$ and the number of miners currently mining:

$$A = \{d\_\text{increase}, d\_\text{decrease}, \text{miner}\_\text{start}, \text{miner}\_\text{stop}\}$$

**Scalability**  The model presents a minimalistic approach concerning the proposed elements in $A$ and $S$. It has been chosen to represent a framework that scales well to more complex scenarios: Additional run-time adjustable parameters can be added to $A$, additional features can be added to $S$, and reward assignment can be adjusted by modifying $r$.

### 6.3 Q-learning

![Figure 17: Markov Decision Process with transition probabilities and rewards](image)

**Formalism**  Reinforcement learning problems can be formalized as Markov Decision Processes. A Markov Decision Process consists of a set of states $S$, a set of actions $A$, rules for state transitions, and rules for rewards (Figure 17). One run through the Markov Decision Process (from initial state to terminal state) is represented by a finite sequence of states, actions and rewards. Individual runs are often called episodes:

$$s_0, a_0, r_1, s_1, \ldots, a_{n-1}, r_n, s_n \quad \text{(with terminal}(s_n) = \text{True})$$

(42)

**Discounted Future Reward**  Looking at one episode and starting at time $t$, the total reward from $t$ to the end of the episode is defined as

$$R_t = r_t + R_{t+1}$$

(43)

Since our Markov Decision Process is stochastic, the further we look into the future the less confident we can be about upcoming rewards. The concept of Discounted Future Reward accounts for that by using parameter $\gamma$ (with $0 \leq \gamma \leq 1$) to discount expected rewards.

$$R_t = r_t + \gamma R_{t+1}$$

(44)
An agent would then choose a strategy that maximizes $R$, i.e. at any time $t$ choose action $a$ that provides the highest discounted future reward. Parameter $\gamma$ plays a crucial role in the behavior of our algorithm: Small values of $\gamma$ favor immediate rewards, large values encourage long-term optimization. But how would an agent observing state $s$ know which action $a$ to take?

Function $Q(s, a)$ represents the discounted future reward when an agent performs action $a$ in state $s$ and continues to choose the best possible action in each state for the remainder of the episode. Assuming existence of $Q(s, a)$, an agent can easily decide which action to take in each state: The one that maximizes the discounted future reward. This is formalized in policy function $\pi$ that (assuming existence of $Q(s, a)$) decides which action to take in which state:

$$\pi(s) = \text{argmax}_a Q(s, a)$$  \hspace{1cm} (45)

**Bellmann equation** The Bellman equation defines the $Q$-function assuming that we already know the discounted future reward for each $(state, action)$ pair:

$$Q(s, a) = r + \gamma \cdot \max_{a'} Q(s', a')$$  \hspace{1cm} (46)

The discounted future reward for taking action $a$ in state $s$ is defined as the sum of immediate reward $r$ and maximum discounted future reward of the following state $s'$. $Q$-learning uses the Bellman equation to iteratively approximate the ideal $Q$-function $Q^*$. Although initial future rewards are random, updating those values with actually experienced rewards has been proven to converge [49]. The details of how the iterative approximation is carried out in practice highly depend on the way in which we choose to represent our $Q$-function.

### 6.4 Adaptive Neural Network Engines

How can the $Q$-function be represented? A simple representation would be to have a $Q$-table, with one row for each possible state the environment can be in ($s \in S$), and one column for each possible action ($a \in A$). However this approach does not scale well with the number of features: For some inputs the number of features grows rather quickly (e.g. a single 1024x768 photo would require $256^3 \times 1024 \times 768$ features for three RGB layers), resulting in an astronomically large number of rows in the table. Even if the number of features is still tangible, many of those possible states would rarely be encountered, resulting in a sparse $Q$-table that is unlikely to converge in a person's lifetime.

Neural networks, though, are known to be good at capturing complex hypotheses. The influential DeepMind paper “Playing Atari with Deep Reinforcement Learning” [37] proposes a network architecture that takes state as input and outputs $Q$-values for every possible action. Our neural network that approximates the $Q$-function would therefore have $3 \times k$ input units (each input unit representing one attribute of the top $k$ blocks), a number of hidden layers, and $|A|$ output units. The benefit of this architecture is that it enables a single forward propagation pass to give us $Q$-values for all possible actions at once.

### 6.5 Training Adaptive Neural Network Engines

**Exploration-Exploitation Dilemma** Initially, $Q$-values are random, so an agent can only guess which action might yield a result (positive reward). Over time, as rewards are experienced, an agent might use this experience to chose actions that promise increased rewards. The question is: How should an agent decide when to pick actions at random and when to take actions that promise the highest reward?

A solution to this is called $\epsilon$-greedy exploration. Parameter $\epsilon$ (with $0 < \epsilon < 1$) controls the exploration/exploitation trade-off. Random actions get picked with probability $\epsilon$, otherwise the
path that promises the highest future reward is chosen. In practice, it is common to start with a high value for $\epsilon$ (high probability for random exploration) and decrease the value over time as the approximation of $Q$ gets more and more accurate.

**Training** To find the weights of our neural network (“train” the network), we need to define an error function that compares predicted rewards to actually experienced rewards (“target”). In this sense, training a neural network using RL is not too different from training a neural network using supervised learning. We chose to use a standard squared error loss function

$$loss = (\text{reward} + \gamma \max_{a'} Q(s', a') - Q(s, a))^2$$

(47)

where $\text{reward} + \gamma \max_{a'} Q(s', a')$ is the target and $Q(s, a)$ is the current prediction of our $Q$-network. Again, parameter $\gamma$ controls the discounting of future rewards.

By interacting with the environment using actions, an agent observes state transitions (along with occasional rewards). Those observations can be “bundled” into one object called an experience: $\langle s, a, r, s' \rangle$. To update our neural network using those experiences, we do the following:

1. Forward-propagation pass using $s$ as input to get the current predictions, $Q$-value vector
2. Forward-propagation pass using $s'$ as input to get $Q(s', a')$ predictions (for all $a'$ in $A$)
3. Calculate the target for $a$ as $\text{target}_a = r + \gamma \max_{a'} Q(s', a')$ (which is the sum of immediate reward $r$ and maximum discounted future reward)
4. Define vector $\text{target}$ such that:
   $$\text{target}[a] = \text{target}_a$$
   $$\text{target}[a'] = \text{predictions}[a'] \quad (\forall a' \in A \setminus \{a\})$$
5. Use $\text{target}$ with back-propagation to update $\theta$

**Experience Replay** With reinforcement learning, subsequent experiences are often similar in nature. Usually just a few dimensions of the state change while other dimensions remain unchanged, e.g. an autonomous vehicle advances its position but the street, surrounding vehicles, weather condition and speed limit remain the same (or at least very similar). This poses a challenge, since training neural networks with many similar subsequent samples can result in undesired outcomes, like converging to a local minimum.

Figure 18: Neural network architecture for reinforcement learning
To avoid this, a technique called experience replay can be used. Experiences observed by an agent get stored in a cache called replay memory. For training, random samples are drawn from this replay memory, thereby avoiding subsequent training examples that are too similar. The full $Q$-network training algorithm (including $\epsilon$-greedy exploration and experience replay) can be seen as pseudo-code in Figure 6.5.

Algorithm 5: $q\_learning()$

```
begin
initialize replay memory $D$;
initialize $Q$ with random weights;
$\epsilon = 1.0$;
while iteration < num_iterations do
    $s =\text{observe initial state (environment)}$;
    while not terminal($s$) do
        $a = \max_{a'} Q(s,a')$;
        with probability $\epsilon : a = \text{random action} \in A$;
        $s', reward = \text{carry out (environment, a)}$;
        store $(s,a,r,s')$ in replay memory $D$;
        sample random experiences from replay memory $D$;
        train network using sample;
    end while
    $s = s'$;
end while
decrease $\epsilon$;
iteration += 1
end while
```

Figure 19: Reinforcement learning with experience replay and $\epsilon$-greedy exploitation (adapted from [33])

6.6 Prior Work

C.J.C.H. Watkins introduced $Q$-learning in his Ph.D. thesis at Cambridge University in the year 1989 [48]. This was followed by a convergence proof (Watkins and Dayan) in 1992 [49]. A more recent milestone in the application of $Q$-learning was DeepMind’s achievement of expert human levels at playing a selection of Atari 2600 games, leading to a patent grant for Google [36]. Our work differs from Google’s Deep $Q$-Networks in the substantial way that their network processes video data frames using multiple convolutional layers, whereas our network is optimized for time-series data of Blockchain mining behavior.

6.7 Action Validation

Before an action proposed by the engine (or more precisely, by the neural network) gets propagated through the mining network, a final validation layer validates the action and translates it into a valid control command. Receiving the action $d\_increase$ the validation might e.g. check whether $d$ would cross some upper bound $d_{upper}$ after being increased. The validation layer exists to incorporate static business logic in the engine. Another example would be the business
decision to have at least three miners running at any point in time. If the current machine count equals three, and the engine decides (e.g. due to $\epsilon$-greedy exploration) to issue a `miner_stop` action, the validation layer can prevent this from happening, resulting in an administrative no-op.

During training, the engine will experience that certain sequences of actions have no effect and adjusts its behavior (model) accordingly. Returning to the autonomous vehicle example, the autonomous driving AI will learn that hitting the brakes in a standing car does not cause any change in the state of the environment.

### 6.8 Feature Adaption for Governed Blockchain Engines

The chosen framework of using reinforcement learning with a $Q$-function neural network approximation scales well and enables us to integrate additional features without much effort. In the context of governed Blockchains, additional features are required to express the mining balance between different partners participating in the mining process. Designing these features deserves some attention: Informally speaking, we want features that do not vary too much when, e.g., the number of partners $P$ in a Blockchain consortium changes, but that have a strong signal when power balance among those partners changes.

As an example, from input “partner who owns the miner who won the block” we can model an aggregated feature $\sigma_{\text{winning\_partner}}$ that expresses how “balanced” the winning of mining races is across partners (standard deviation):

$$\sigma_{\text{winning\_partner}} = \sqrt{\frac{\sum_{i=1}^{P} (w_i - \overline{w})^2}{N - 1}}$$

with $w_i$ being the number of blocks won by machines owned by partner $i$.

### 7 Automatic Anomaly Detection for Node White-listing

The design of anomaly detection in the context of Blockchain needs to be informed by the defining question one wants to answer, i.e. does a particular node exhibit malicious behavior. In order to retain the distributed nature of Blockchain – in contrast to having a single authority providing anomaly detection training and inference, answering such a question and acting upon the result is required to be executed in a distributed and resilient way. To achieve the goal of automated node white-listing using anomaly detection on the Blockchain, careful integration of several distinct components is required:

- **Training**: Using available information to find a model that is able to distinguish anomalous from non-anomalous node behavior.
- **Inference**: Using a trained model to predict and identify potentially malicious nodes.
- **Decision**: Acting upon inference results to determine necessary changes to the white list.
- **Distribution**: Finding a secure mechanism to distribute inference, decision and training along with the respective results.

#### 7.1 Detecting Anomalies in the Context of Blockchain

##### 7.1.1 Feature Representation for Training and Inference

For both training and inference, it is crucial to design features that help to indicate the presence or absence of an anomaly. Such features should exhibit strong correlations with one or many
malicious node behaviors (see Section 8 for potential attack vectors). The decision about representation of individual features also needs to be informed by the availability, as well as the reliability, of certain types of information. Potential attributes to consider are:

- Mining time of a block
- Identity of a miner
- Hash rate of a miner
- Number of transactions included in a block
- Number of transactions still pending
- Mining difficulty of a block
- Presence transaction that modifies white list
- Gas cost of smart contracts
- Size of white list
- Information from basic Date and Time types

Available attributes features need to be aggregated into one or many feature vectors. How features are chosen to be represented has vast implications for the usefulness of the machine-learning model after training and the associated predictions one can make.

**Naïve approach**  One could accumulate features based on the current Blockchain state into a single vector of size $n_x$, and use it to predict the presence of an anomaly. One such feature vector might include features such as the mining time of block $n$, the mining time of block $n - 1$, the identity of the miner of block $n$, the identity of the miner of block $n - 1$, and so forth.

Choosing this approach, however, has multiple drawbacks: By aggregating the behaviors of multiple nodes into a single vector one cannot identify that a single node is causing the potential anomaly warning. A second layer of logic would be required to determine (with a level of uncertainty) the node that was most likely to cause the anomaly warning. The only benefit one could see in this approach is its simplicity, since aggregating the required features into a vector is relatively straightforward and does not require complex aggregation logic. If one were only interested in the presence of an anomaly (and not the identity of the node causing the anomaly) this approach might indeed be sufficient.

**Improved approach**  A better alternative is to define features with respect to individual nodes. The value of one such feature expresses characteristics of an individual node and thus allows anomaly predictions to be associated with the respective nodes, thereby enabling subsequent systems to act upon those identities.

An example of such a feature is the share of mining races won by a particular node. Its value is a good indication of malicious behavior: Nodes with un-proportionally large hash rates are able to win a large share of mining races compared to honest nodes which will be more likely to each win an equal share of mining races.

The result of such a feature definition is a dataset $X \in \mathbb{R}^{m \times n_x}$ containing $m$ rows and $n_x$ columns, with $m$ being the number of nodes winning mining races during the last $k$ blocks, and $n_x$ being the number of features used for inference.

The major advantage of this approach is identification, i.e. anomaly detection is able to identify malicious nodes. Increased complexity in feature aggregation may be seen as a drawback. And inference needs to be run on multiple examples, not just one. This drawback is mitigated by computational support for vectorized operations in modern CPUs (e.g. SIMD) and GPUs.
7.1.2 Feature Aggregation for Inference

To collect a dataset for inference, we look at the information present in the most recent $k$ blocks. This allows the anomaly detection model to find (and act upon) potentially anomalous behaviors that occurred recently, and it avoids false positives that happened more than $k$ blocks ago.

Choosing a suitable value for $k$ is crucial for the success of anomaly detection, and parameter $k$ may also be different from the one used for computing set of public keys $PK$. Very small values of $k$ (e.g. $k = 5$) will prevent the model from seeing events that occurred quite recently. Those events might be important for detecting trends or interactions; not including them will result in the model not being able to detect important anomalies.

Very large values of $k$ are not helpful either: Events that happened too long ago should not be included. There is no value in considering those events and they might even interfere with the relevance of more recent events. The importance of $k$ is therefore not to be underestimated and justifies careful tuning. Thus, $k$ is not a hyper-parameter in the classical sense of it influencing how training of the model is exercised. But it strongly influences the content of the dataset after aggregation and should therefore be tuned rather carefully.

7.1.3 Feature Aggregation for Training

Anomaly detection datasets are required to include both anomalous and non-anomalous examples which makes aggregation more involved. A large number of non-anomalous examples will form $X_{train}$, and a (comparably) small number of both anomalous and non-anomalous examples will form $X_{cv}$ and $X_{test}$.

Non-anomalous examples can be collected in large quantities during regular operation of a Blockchain network. It is important, however, that the collected examples (and with them the trained machine learning model) are highly dependent on the system environment: a Blockchain network running on a large cluster of low-powered IoT devices will exhibit different characteristics than a Blockchain network running on professional-grade enterprise servers.

Anomalous examples can be aggregated in two ways: The first way is to manually construct anomalous examples by inserting values that are considered to be strong indicators of malicious node behavior. The second way is to collect anomalous examples during operation of a Blockchain network which is under attack. Ideally one would combine these approaches to have both “real-world” anomalies collected from a running network and “hand-crafted” anomalies to ensure all possible attack scenarios are represented in the final dataset.

Splitting aggregated data into $X_{train}$, $X_{cv}$, and $X_{test}$ needs to be done in a careful way. The unsupervised nature of finding anomaly detection models requires $X_{train}$ to consist of only non-anomalous examples. The anomalous examples are split equally between $X_{cv}$ and $X_{test}$. Before any split can be done, the order of examples needs to be randomized. To determine the exact number of examples in each dataset, one can use the following “best practice” split:

- $X_{cv}$: 20% of non-anomalous examples (up to 10000), 50% of anomalous examples
- $X_{test}$: 20% of non-anomalous examples (up to 10000), 50% of anomalous examples
- $X_{train}$: All remaining non-anomalous examples
7.1.4 Inference

Given the parameters that result from training, the probability of a single example \(x\) being non-anomalous can be determined using the following function:

\[
p_{\text{anomaly}}(x) = \prod_{j=1}^{n_x} p(x_j; \mu_n, \sigma_n^2)
\]  

(49)

Finally, the decision of whether example \(x\) is to be considered an anomaly or not is given by:

\[
anomaly(x) = \begin{cases} 
\text{True}, & \text{if } p_{\text{anomaly}}(x) < \epsilon \\
\text{False}, & \text{if } p_{\text{anomaly}}(x) \geq \epsilon 
\end{cases}
\]  

(50)

7.2 Decisions Space of Detector Nodes

Following inference, a decision has to be taken that translates anomaly predictions into transactions which modify the current state of the white list.

Considering that malicious nodes are expected to be an exception (rather than the norm), the current design provides the possibility to either add or remove a single node with each decision. If no white-listed node is required to be taken off the white list, and if no additional node becomes eligible to be put on the white list, the decision will still be recorded as \(\text{no-op}\). This enables seamless traceability of white-listing decisions and their execution. The set of possible decisions is therefore defined as:

\[
D = \{\text{no-op}\} \cup \{\text{add}_n \mid n \in N\} \cup \{\text{remove}_n \mid n \in N\}
\]  

(51)

where \(N\) is the set of network nodes.

The actual decision is a function of the current state of the white list and the anomaly predictions. If the predictions for some nodes fall below the \(\epsilon\) threshold, they are considered as candidates for removal. Among nodes that are potentially malicious and still present on the white list, the single node with the lowest (worst) probability will be picked for removal. If no node is eligible for removal, the remaining nodes are considered for addition if their probability for being honest nodes is above the \(\epsilon + \zeta\) threshold (provided that they are not already present on the white list). Among those candidates, the one with the highest prediction will be picked for addition (the single node with the highest likelihood to be honest). If there are neither nodes for removal nor nodes for addition a \(\text{no-op}\) transaction will be issued. Figure 7.2 depicts this procedure in pseudo-code. We make some observations:

- Removing potentially malicious nodes takes precedence over adding potentially honest nodes
- The addition of \(\zeta\) avoids unstable behavior of the white list for any node \(n\) with \(p_{\text{anomaly}}(n)\) being close to \(\epsilon\)
- Whenever both \(M \cup W\) and \(H \setminus W\) equal \(\emptyset\), a \(\text{no-op}\) results

7.3 Distributed Inference in the Context of Blockchain

Assuming a model to infer anomaly predictions exists, how can we use this model in a Blockchain network, and how can we distribute the resulting decisions to participating nodes in the network? We propose an approach that is inspired by the consortial approach of PPoKW.

A second seed in each block presents the foundation for nodes to become eligible to participate in PoW. Among the PoW participants, the winning node then uses the trained model for
Algorithm 6: \texttt{computeDecision}(W, \texttt{predictions})

\begin{algorithm}
\begin{algorithmic}
\State $M = \{ n \in N \mid p_{\text{anomaly}}(n) \leq \epsilon \}$;
\State $M_{\text{new}} = M \cup W$;
\If{$M_{\text{new}} \neq \emptyset$}
\State $n^* = \text{argmin}_{n \in M_{\text{new}}} p_{\text{anomaly}}(n)$;
\State \textbf{return} \texttt{remove}$_{n^*}$;
\EndIf
\State $H = \{ n \in N \mid p_{\text{anomaly}}(n) \geq \epsilon + \zeta \}$;
\State $H_{\text{new}} = H \setminus W$;
\If{$H_{\text{new}} \neq \emptyset$}
\State $n^* = \text{argmax}_{n \in M_{\text{new}}} p_{\text{anomaly}}(n)$;
\State \textbf{return} \texttt{add}$_{n^*}$;
\EndIf
\State \textbf{return} \texttt{no-op}.
\end{algorithmic}
\end{algorithm}

Figure 20: Determining a decision based on anomaly detection predictions

inference and determines a white-listing decision based on the inference results. This resulting decision is expected to be issued every time a new block is mined: If the node determines that no modification to the white list is required, a \texttt{no-op} decision has to be issued. A transaction containing the final decision ensures distribution of the result to other nodes.

The presented approach enables anomaly detection to happen in a distributed way. The properties and principles that make Blockchain networks resilient are not violated and the consensual approach ensures low computational overhead for the overall network.

8 Mitigating Potential Attack Vectors

We now discuss some potential vectors through which an adversary or a group of adversaries could attack this system, and how we may mitigate such threats. In this yellow paper, we will do this informally, more formal analyses will be deferred to technical papers.

\textbf{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 $p$ of all network nodes, at any given point in time.

Let us analyze 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 [18], 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 $Q^0$.

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, in this case 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 signing 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 \(\text{mayPerform}_{pk_i}(r, \text{mine})\) for compromised nodes \(pk_i\) to determine which of these nodes are eligible to mine.

However, if he has not yet compromised a node \(pk_j\), then he won’t know its secret key \(sk_j\) and so he cannot evaluate the body of \(\text{mayPerform}_{pk_j}\). Therefore, the attacker cannot know which of the nodes that he has not yet compromised will be able to participate in the next mining race. This means that he has no additional information that may help him 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 he has to compromise a percentage \(p\) of all network nodes in order to guarantee that he compromises an expected percentage of \(p\) of eligible nodes for each mining race.

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. But as discussed already, the adversary will only be able to exploit this with probability \(p\) where \(p\) is the percentage of nodes compromised in the entire network. Also, our machine learning would pick up such behavior and, with high probability, remove that public key from the white list \(L\). Moreover, for highly sensitive applications, we may also mitigate against or even prevent these occasional exploitations by embedding such secret keys securely into devices, using Physical Unclonable Functions (PUFs).

Another threat stems from the ability to propose empty, sparse or low quality transaction sets \(TSet_r\) within blocks \(B^r\). We assume that the system policy, embedded into the Blockchain, would be able to flag up such behavior so that the machine-learning engines could react to this — for example by removing a node from the white list \(L\) or by adjusting system parameters as a corrective response.

Naturally, these control mechanisms themselves, including the machine-learning algorithms used, may be subject to attack and manipulation. Adversarial machine-learning techniques (see e.g. [44]) may be mitigated against, given that most if not all the data for learning would stem from the current Blockchain, which we would trust in our security model, and so data poisoning would be prevented. Other 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.

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. One 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.

9 Practical Reasoning & Implementation

As discussed in Section 2.5, 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 [20]. 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 PPoKW – 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 PPoKW, and how we have leveraged smart contracts in our solution.

9.1 Modifying Geth

We chose to modify the Ethereum Foundation’s go-ethereum (Geth) implementation of the Ethereum protocol in our initial implementation of PPoKW 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 21 and 22, respectively.

![Figure 21: Block Verification Process](image-url)
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 21 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 [50]. 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 21 for sake of brevity.

![Geth's Block Proposal Process Diagram]

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 21, 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.

9.1.1 High-level Overview of Modifications

Block verification process  The modifications to Geth required to implement PPoKW, at a high-level, did not require any changes to the block verification process as outlined in Figure 21. However, the implementation of PPoKW 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 25 and details about the extra validation can be found in Figure 21.
Geth’s modified block proposal process Figure 23 provides a high-level overview of the changes that were made to Geth in order to modify the block proposal process to implement the PPoKW algorithm.

Figure 23: PPoKW 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 9.2. (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 \( B^r \).

PPoKW’s machine learning processes The two sets of machine learning algorithms individually run by separate committees follow the same high-level process as shown in Figure 24.

Figure 24: PPoKW Machine Learning Process

(a) A node determines whether it is part of the anomaly detection committee by running the secret lottery using the seed \( Q^r_a \), deterministically derived from \( Q^r \) and task \( a \). A node determines whether it is part of the parameter modifying committee by running the secret lottery
using the seed $Q_r^p$, again determined by seed $Q^r$ and task $p$. (b) If a node is not part of one of the ML committees, then it will check to see if it is part of the other. If it is part of neither, then it waits for the arrival of the next block. (c) The outcome of running one of the ML algorithms is one of two possible options: a transaction containing updates to the network or an empty transaction signifying that no change is required. (d) A PoW algorithm is then run to seal the transaction $TX^r$ to determine which of the nodes in the committee is the chosen producer for this blockheight. (e) Both the ML algorithm and PoW algorithm can take several seconds to complete, during this time another node in committee could have broadcast the new block $B^r$. If this happens, then the long-running process that is executing is terminated.

9.2 Comparison of block structure

In Ethereum, the header of a block can be passed around without its corresponding block – as defined in the official ÐΞVp2p Wire Protocol [16]. 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 25 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 PPoKW, therefore, requires additions to the structure of the Block Header. Figure 25 shows a comparison of the standard Ethereum block structure and the modifications necessary to support the PPoKW consensus mechanism:

**Block structure additions:** The extended block header contains the three seeds required to carry out PPoKW. These seeds are all deterministically derived from the seed $Q^r$ as a function of the task to be performed: (a) The seed $Q_r^b$ is used for determining the block producing committee members. (b) The seed $Q_r^a$ is used for determining the anomaly detection committee members. (c) The seed $Q_r^p$ is used for determining the Blockchain parameter alteration committee members. 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.

9.3 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.

PPoKW 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.

9.3.1 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 \( B \) 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 [22]. 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.

9.3.2 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 $\text{mayPerform}_{pk_i}(r, \text{mine})$.

A node $pk_j$ can verify whether some other node $pk_i$ from $PK$ was a member of that committee $C_r$ by running $\text{mayPerform}_{pk_i}$ 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.

### 9.3.3 Smart Contract, White-List Code

The following Solidity code fragment shows the current simple implementation of $L$ that fulfills most of the criteria outlined in Section 9.3.2, with the exception of restricting which addresses are able to modify $L$. This will be addressed in future versions of the contract.

Another future enhancement will be to switch from a binary true or false membership to a more sophisticated solution that would be more difficult for an adversary to attack.

```solidity
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);
}

9.4 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. We note that, as an optimization, the eligibility is cached for the interval of $p$ blocks at which configuration changes will be made periodically.

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 $B_r$.

9.4.1 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 (52), calculated in (19) 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 $C_r$.

$$\text{SIG}_{pk_i}(r,Q_{r-1})$$

9.4.2 Seed Generation

Since the seeds $Q_{a}^{-1}$, $Q_{a}^{-1}$ and $Q_{p}^{-1}$ are integral to the PPoKW 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:

$$0.H(\text{SIG}_{pk_i}(Q_{r-1}))$$
9.4.3 Finding the Nonce

PPoKW 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.

9.5 Storage Implementation with IPFS and NuCypherKMS

9.5.1 Controlling Access to Data

The storage network is private, as such access to data is restricted to known nodes. From the perspective of a user of a DApp which is using the storage layer, we rely on user based encryption and DApp specific based access control. The DApp based access control will be specific to the application and can allow for mapping of roles between the DApp and client legacy systems. Encryption of the data uses proxy re encryption to give re encrypt stored data so that it can be decrypted with the user’s private key. The re encryption nodes allow the data owner to specify which other users may have access to a particular data item.

**Key details** The record is encrypted in the client with a random symmetric key $K$ to produce cypher text $C_A$. The key $K$ is then encrypted with the public key of the data owner (typically the patient) to produce $K_c$. $K_c$ and $C_A$ are then stored together on IPFS. A transaction recording this addition is then produced by the client software and submitted to the Ethereum network.

**Adding records to IPFS with re-encryption**

![Diagram of the storage implementation process](image)

Our requirement for a secure history for data is achieved by the immutable addition of this transaction to the Ethereum Blockchain.
Retrieving Records

This IPFS address of a record can be obtained from the XAIN smart contract, and thus the encrypted record can be obtained from IPFS.

This gives \( R \equiv (K_c, S, D) \) as described above. The encryption will then proceed as

1. Retrieval by the record owner
   The owner can use their private key to decrypt the data key \( K_c \), and use this key to decrypt the rest of the data record.

2. Retrieval by a third party
   To decrypt the record a re encryption node supplies a re encryption key \( K_r \) to any applicant allowed to view the data.
   \( K_r \) can be used to decrypt the data key \( K_c \), which can then be used to decrypt the data \( D \).
   A transaction is sent from the re encryption node to the XAIN smart contract to record that access has been granted.
   If required access to a record on IPFS can be revoked by the re encryption nodes after a certain time.

9.5.2 NuCypher KMS an Implementation of Proxy Re-Encryption

NuCypher KMS is a decentralized Key Management System [15] designed to overcome the limitations of using decentralized applications that store and manipulate private, encrypted data. Encryption and cryptographic access control are provided by a decentralized network, leveraging proxy re-encryption. Critically it does not require trusting a service provider. NuCypher KMS enables controlled sharing of sensitive data in both decentralized and centralized contexts, providing security infrastructure for diverse applications.

9.6 Distributed Storage Implementation

IPFS Overview

IPFS [2], the interplanetary file system is a decentralized storage system based on a distributed hash table that provides content based addressing. IPFS allows data to be easily propagated, with no single point of failure. Nodes of the IPFS network do not need to trust each other.

Similar Distributed Storage systems The main competitors to IPFS are Storj and Swarm. Storj is not suitable: it has no concept of a private network and storage is charged for. Swarm makes a better alternative, it is part of the Ethereum stack, and unlike IPFS does not require nodes to stay online. However Swarm is currently not ready for production. To allow XAIN to switch to alternative storage systems as required, its application layer uses a storage abstraction, to allow alternative storage implementations to be plugged in, such as Swarm or IPDB.

IPFS Stack IPFS uses a distributed sloppy hash table and block exchange mechanism to form a massive peer to peer system for storing and distributing blocks quickly and robustly. On top of this, IPFS builds a cryptographically authenticated data structure the Merkle directed acyclic graph (DAG) where links between objects are the cryptographic hashes of the targets. The Merkle DAG is used to model information structures such as file systems.

Because IPFS objects are contained in a Merkle DAG they can be

- retrieved via their 512 bit hash
• integrity checked
• linked to other objects
• cached indefinitely

This gives the IPFS system

• Content Addressing: all content is addressed by a hash of the content
• Tamper resistance: the content hash contains a checksum to prevent tampering or corruption
• De-duplication: objects with the same content are only stored once.

The incentivized block exchange mechanism bitswap is a message based data trading module for IPFS, it manages requesting and sending blocks to and from other peers in the network. Indexing information allows particular content to be found efficiently and IPNS a naming service allows human readable names. The distributed nature of content delivery saves bandwidth and unlike the https protocol prevents DDoS attacks.

**IPFS Network**  XAIN uses the IPFS client software to build a IPFS network that is made private by restricting node membership. This is achieved by requiring nodes to have knowledge of a shared 512 bit key. This requirement is in addition to any firewall rules that would also restrict access to the private network. An abstraction layer interfaces between the XAIN infrastructure and the IPFS API to control storage of the XAIN records.

**IPFS Data Structures**  The XAIN data structures are built upon the underlying IPFS structures. A XAIN record $R$ is defined as the 3 tuple

$$R ≡ (Kc, S, D)$$
where

- $K_c$ is a 256 bit symmetric encryption key further encrypted by a public key
- $S$ is the encryption scheme used (32 char)
- $D$ is an unlimited byte array of encrypted data

Furthermore, we define $D$ as the following:

$$D \equiv (P, F, T, M, B)$$

where

- $P$ is a 160 bit user identifier
- $F$ is a filename (64 char)
- $T$ is the file type (16 bit enum)
- $M$ is an unlimited byte array holding the file meta-data.
- $B$ is an unlimited byte array holding the file contents.

The XAIN smart contract may also store, as appropriate according to the amount of data expected:

**A Permissions data structure**

$$P \equiv (U, R, L)$$

where

- $U$ is a user identifier, likely a public key
- $R$ is the data record as defined above
- $L$ describes the role the user has for that record

**A Data history structure**

$$H \equiv (R, A)$$

where

- $R$ is the data record as defined above
- $A$ is the access or update type

**Data Addition** Data is added to IPFS through client software that forms an IPFS node. The client exposes an API to add and retrieve data.

**Data Deletion** Files can be attached to nodes by a “pinning” process. By unpinning files from a node, they are made available for local removal by the garbage collector. The process to remove data is then:

1. unpin the file from any node it is pinned to
2. make sure that no other nodes request the file before it can be garbage collected, since we have control over the nodes in the network we can ensure this.
Adding a file gives a content hash as an address
6bcc495cf803ee33aa9b62b105143a58c452e38ea39a682e9260fca8a04e50d8

The file can be revived from a different node by using its address
6bcc495cf803ee33aa9b62b105143a58c452e38ea39a682e9260fca8a04e50d8

Figure 28: Data Addition Diagram

10 Summary & Conclusion

In this yellow paper, we set out a vision for Blockchain systems in which consensus mechanisms, cryptography-based access control, anomaly detection, and machine learning of system parameters for run-time adaptation are coordinated and distributed throughout the Blockchain network. The motivation behind this vision is that such synergistic architectures should provide much more resilient and robust Blockchain systems, as they will support self-stabilizing system behavior that can reflect a variety of features and their trade-offs in an optimal manner.

Such features of a Blockchain may be generic, such as the security of the practical immutability of chain data, the rate at which new blocks get created or the overall network cost associated with block creation. But features may also be application-specific, for example, whether the Blockchain itself is compliant with privacy demands such as those stipulated by the GDPR for a system that handles personal data within a purchase and shipment workflow.

We made the vision of this yellow paper concrete through the eXpandable Artificial Intelligence Network (XAIN), a framework for the development of intelligent and adaptive Blockchain platforms. The XAIN framework rests on three pillars:

- a new consensus mechanism, Practical Proof of Kernel Work (PPoKW), a variant of Proof of Work based on Cryptographic Sortition, the latter was introduced for Algorand in [40]
- an adaptation of existing Blockchain and enterprise technology stacks for the XAIN framework, based on the governed Blockchain approach in [32]
- use of machine-learning techniques to provide adaptive and application-aware system resiliency for instantiations of the XAIN framework.

The advantage of PPoKW is that it retains the security benefits offered by Proof of Work, while making the costs associated with Proof of Work scale-invariant in the size of the network – yet at the same level of adversarial resiliency. Cryptographic Sortition can also be applied in the assignment of other system tasks, such as anomaly detection, as it randomly selects an expected number of nodes in the network in a manner that an adversary cannot manipulate. Therefore, observations made about the Blockchain system as well as any system
changes made based on such observations are distributed throughout the network and resilient to adversarial manipulation.

We provided technical details on how the XAIN framework makes use of existing Blockchain and smart-contract technology, and we discussed the layered approach of the XAIN framework to access control of critical system tasks such as mining for PPoKW. The XAIN framework was also shown to be suitable for instantiations that support GDPR compliance, through a judicious combination of key management, white-list management, and anomaly detection.

Furthermore, we presented how a mixed-integer, non-linear optimization model can be used to inform initial and dynamic system configurations and optimal trade-offs of system characteristics. This yellow paper also offered a discussion of pertinent attack vectors for the XAIN framework, and what measures it can take to prevent or mitigate against such attacks.

We defer to future technical papers the reporting of experimental results and the description and evaluation of specific case studies of this PPoKW-based XAIN framework.

References


