A Distributed Post-Quantum PoW Consensus Algorithm for Collaborative Blockchain Networks

***Abstract*—Present day Proof-of-Work (PoW) blockchains are competitive blockchains, implying they rely upon miners competing with each other to mine blocks and gain rewards. This leads to a huge carbon footprint because out of k miners competing to mine a new block, only one miner succeeded and was rewarded. In this work, we propose a post-quantum PoW consensus algorithm that seeks to do away with competitiveness and allow more collaboration. Precisely, we create a post-quantum consensus algorithm on the lattice-based Shortest Vector Problem (SVP). We further distribute a sieving algorithm to solve the approximate version of SVP and detail a consensus model wherein we leverage this distributed version to reuse the computation of all k miners. All the k miners contribute to the solution, and the incentive model is designed so no miner strays from honest behavior.**

***Index Terms*—Distributed-collaborative blockchain, Shortest vector problem, Proof-of-Work, Lattice, Post-quantum consensus**

I. Introduction

In the world of conventional financial settings, usually three parties are involved: the initiator of the transaction, the receiver of the transaction, and a third party to mediate the transfer. A great example of this third-party entity is conventional *banks*, which mediate money transfers through a credit/debit card. In this case, the recipient requires the *bank* to verify whether the transaction indeed succeeded or failed. This example is what a *centralized* financial system looks like a single entity authorizing and validating transactions between an initiator and a receiver. Such centralized systems introduce *single points of failure* in the modern financial world. Should the *bank*'s communication methodology (in modern times, usually a website or an app) get compromised, all financial transactions going through that *bank* are inherently threatened.

Blockchains are an attempt to move away from these *centralized* systems. A blockchain builds a *decentralized financial system* [1]. A blockchain completely removes any central party because now all transactions are recorded on a public (primarily anonymous) ledger. A transaction is validated based on the consensus of the entire network. A blockchain can come in two main flavors depending on the restriction on who can use the blockchain: *permissioned* or *permissionless*. A permissioned blockchain requires authorization before doing transactions on the blockchain (usually, private blockchains come under this category). Permissionless/public blockchains, on the other hand, allow anyone to perform transactions. Once a transaction is recorded on the ledger, it is impossible to mutate its details. The blockchain maintains a record of all transactions done to date. Moreover, any blockchain user can query the ledger and understand what transactions took place at what time without the involvement of a third-party (thereby achieving complete transparency). The transaction information is, however, limited to the transaction itself (i.e., a transaction does not leak information related to the identity of the users participating in a transaction). Since a blockchain is completely decentralized, it implies that we need to find an alternative method of validating transactions (in centralized systems, this role was played by a third-party, like a *bank*). Blockchains use the method of *mining a block*. This means users (or *miners*) need to put in some *work (computation)* to validate a block. A miner has three main tasks: (i) verify the transaction, (ii) create a new block and record the transaction there, and (iii) integrate the block in the blockchain.

A. *Proof-of-Work consensus*

A miner usually spends a lot of computation power to do the tasks mentioned above. Usually, a cryptographic puzzle, which is an NP-Hard problem, must be solved to complete these tasks. Whenever a transaction (or a list of transactions) is (are) submitted to the network for validation, several potential miners start competing with each other over a *computationally hard* cryptographic problem. The miner who solves the cryptographic problem first wins the race. The transaction (or the list of transactions, as the case may be) is then recorded on the ledger, and the miner receives some portion of the reward as a financial incentive to encourage to execute the mining process. As the eagle-eyed reader would have observed by now, this miners' race is actually a question of who uses better hardware. The stronger the processor you utilize (maybe a hardware accelerator), the more chances you win the race. This consensus algorithm is called **Proof-of-Work (PoW)** algorithm [2], which we will be dealing with in the rest of the paper. Such a consensus algorithm follows a simple principle: **solve a cryptographic problem that is difficult to find the solution for; but once a solution is found, it is straightforward to verify if the solution is a correct one**.

In modern blockchains employing PoW consensus, SHA-256 is one of the most popular cryptographic primitives used. As an example, Bitcoin (the first cryptocurrency) also uses SHA-256 for its PoW consensus [3]. Generally, in a Bitcoin network, we denote the data in a block henceforth as `data`. A miner starts by guessing a nonce and computing the value of `SHA256(data||nonce)`. This process is repeated until a SHA256 hash is found with a given number of 0s in the prefix of the hash output. The number of 0s varies to change the difficulty of the cryptographic problem. Formally, we can state this problem as: Given a hash function $f : X \to Y$, and $\mathbf{y} \in Y$ such that $\mathbf{y}$ has a prefix-string of k zeros, find $\mathbf{x} \in X$ such that $f(\mathbf{x}) = \mathbf{y}$. To successfully mine a block, the miners race against each other to solve this partial collision problem in SHA-256.

B. Issues with existing PoW consensus

We now briefly mention the problems with conventional PoW consensus: (i) resource consumption, (ii) 51% attack, and (iii) double-spending. Regarding (i) resource consumption, we note that since the cryptographic puzzle involved is solving the partial-collision problem on the SHA-256 hash function, the amount of work put in by the miners is huge. Coupled with the fact that there are a lot of miners in a network, this puzzle explodes the electricity requirement to mine a single block. The carbon footprint arising from these computations is of a great magnitude [4]. Likewise, regarding (ii), we note that the PoW consensus is a majority consensus. This means if 51% of the network agrees with a certain transaction, it is approved [5]. It directly implies that if an adversary controls 51% of the network, the adversary can mutate the blockchain to any end they like. Moreover, the adversaries can also halt transactions between users of the blockchain as a result of the 51% attack [6]. Finally, regarding (iii), most blockchains are usually used for cryptocurrencies. Cryptocurrencies, like any other currency, must not be spent twice. However, it turns out that some attack vectors can spend one coin twice. Basically, due to the time gap between the transaction and its validation, should a malicious user initiate another transaction, it would mean that the same currency is now under validation in two different places (with no way for the two miners involved to know if the same currency is being used in their transactions).

C. Contributions

In this work, we make the following contributions over a conventional PoW consensus algorithm:

- Explore post-quantum PoW consensus algorithm.
- Explore a distributed collaborative blockchain architecture to mitigate energy-related issues with conventional PoW, implying we do not waste any miner's computation in our blockchain architecture.

D. Paper Organization

The rest of the paper is structured as follows. Section II-A describe the basics of different lattices. Section III describe the proposed distributed Post-quantum-based PoW consensus algorithm. Section IV provides the implementation of the proposed distributed PoW algorithm. Section V concludes the paper.

II. Background

A. Lattices

Integer lattices are discrete subsets of the n-dimensional Euclidean space $\mathbb{R}^n$. Formally, given a set of m vectors in $\mathbb{R}^n$: $\{\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}, \cdots, \mathbf{b_m}\}$ arranged as columns of a matrix $\mathbf{B} = [\mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}, \cdots, \mathbf{b_m}]$ and linearly independent such that $\mathbf{B} \in \mathbb{R}^{n\times m}$, the integer lattice $\mathcal{L}(\mathbf{B}) \subseteq \mathbb{R}^n$ is given by $\mathcal{L}(\mathbf{B}) = \{\sum_{i=1}^{m} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$, or equivalently, $\mathcal{L}(\mathbf{B}) = \{\mathbf{B}\mathbf{x} : x \in \mathbb{Z}^m\}$ for the usual matrix-vector multiplication. The matrix $\mathbf{B}$ is defined as the basis of the lattice. The underlying assumption is that $\mathbf{B}$ has rational coordinates and can be converted to an integer matrix by scaling with a suitable factor. Without loss of generality, integer matrices are thus considered as the basis matrices throughout the paper. The integers m and n are the *rank* and *dimension* of the lattice, and if $m = n$, the lattice is called *full rank*.

Definition 1 (Half-open parallelepiped)**.** Given a basis matrix $\mathbf{B} \in \mathbb{Z}^{n\times m}$, the half-open parallelepiped for a lattice $\mathcal{L}(\mathbf{B})$ is given as $\mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} : 0 \le x_i < 1\}$.

$P(B)$ intuitively describes the smallest *unit* of lattice that is repeated over the entire space rendering the lattice its periodic structure.

Lemma 1. An arbitrarily defined matrix $\mathbf{B} \in \mathbb{Z}^{n\times m}$ is a basis of for the corresponding lattice $\mathcal{L}(\mathbf{B})$ iff $\mathcal{P}(\mathbf{B})$ contains no other lattice point except the origin.

Definition 2 (Equivalent bases)**.** Two distinct bases $\mathbf{B}, \mathbf{B}' \in \mathbb{Z}^{n\times m}$, are equivalent iff there exists a matrix $\mathbf{U} \in \mathbb{Z}^{m\times m}$ and $det(\mathbf{U}) = \pm 1$ (where $\mathbf{U}$ is termed as *unimodular*) and $\mathbf{B}' = \mathbf{BU}$. Elementary integral column operations on $\mathbf{B}$ to achieve $\mathbf{B}^{'}$ are given as [7]:

- Swap two columns.
- Multiply a column with -1.
- Add a column to the scaled result of another column where only integral scalars are allowed.

Definition 3 (Norm)**.** Given a finite n-dimensional vector $\mathbf{x}$, the l_p norm for $p \ge 1$ is given as $\| \mathbf{x} \|_p = (x_1^p + x_2^p + \cdots + x_n^p)^{1/p}$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$. The value of p can be infinite. l_∞ is then given as the *max-norm*: $\| \mathbf{x} \|_\infty = \mathbf{max}\{x_1, x_2, ..., x_n\}$.

The value of p can be infinite. l_∞ is then defined as the *max-norm*: $\| \mathbf{x} \|_\infty = \mathbf{Max}\{x_1, x_2, ..., x_n\}$.

Definition 4 (Open ball)**.** A n-dimensional open ball is given as $\mathscr{B}(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{R}^n : \| \mathbf{x} - \mathbf{y} \| < r\}$ for radius r and center $\mathbf{x}$.

Definition 5 (Successive minima)**.** Successive minima for a lattice $\mathcal{L}(\mathbf{B})$, norm $p \ge 1$ and arbitrary basis $\mathbf{B} \in \mathbb{Z}^{n\times m}$, are given as $\lambda_i(\mathbf{B}) = \mathbf{Min}\{r : dim(\mathcal{L}(\mathbf{B}) \cap \mathscr{B}(\mathbf{x}, r)) \ge i\}$.

Informally, the value of the i-th successive minimum represents the minimum length r such that a ball of radius r centered about a point $\mathbf{x}$ contains at least i independent vectors. Several upper bounds on $\lambda_i(\mathbf{B})$ can be expressed in the form of the determinant of the lattice, formally known as the Minkowski's theorem.

Theorem 1 (Minkowski's theorem)**.** For any lattice $\mathcal{L}(\mathbf{B})$,

$$\{\prod_{i=1}^{n} \lambda_i(\mathcal{L}(\mathbf{B}))\}^{1/n} \le \sqrt{\gamma} det(\mathcal{L}(\mathbf{B}))^{\frac{1}{n}}$$

where γ is the approximation factor and $\gamma \leq n$. Minkowski's theorem helps to upper bound $\lambda_i(\mathbf{B})$. This provides a convenient mathematical tool to upper bound λ_1 and λ_n used in some computation problems such as SVP and SIVP.

B. Technical background on computational problems on lattices

Computational problems can be defined in their search, optimization, or decision version. Search version requires *searching* for the solution amongst all possibilities, the optimization version requires a solution that optimizes a certain criterion, and the decision version requires outputting a binary decision on whether a specified property is met. For the usual meaning of complexity classes P and NP, reducing problem A to a problem B implies converting any instance of A to some instance of B. This means given access to an oracle of B, A can be solved thus stating that A is not harder than B. Polynomial-time reductions for decisions problems are such that for problems A and B, A is reduced to B in polynomial time iff there exists a polynomial-time bounded function f and $x \in A$ iff $f(x) \in B$.

Definition 6 (Minimum distance)**.** The **minimum distance** of a lattice $\mathcal{L}(\mathbf{B})$ can be expressed as:

$$d_{min}(\mathcal{L}) = \underset{\mathbf{v} \in \mathcal{L} \setminus \{\mathbf{0}\}}{\mathbf{Min}} \parallel \mathbf{v} \parallel$$

Definition 7 (Shortest vector problem (SVP))**.** For any lattice $\mathcal{L}(\mathbf{B})$ with basis $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$, SVP is to find a non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ so that $\parallel \mathbf{v} \parallel = d_{min}(\mathcal{L})$.

Definition 8 (Approximate SVP)**.** For any lattice $\mathcal{L}(\mathbf{B})$ with basis $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ and an approximation factor $\gamma \geq 1$, approximate versions of SVP are given as:

- **Search SVP$_\gamma$:** Find a non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\parallel \mathbf{v} \parallel_2 \leq \gamma \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$
- **Optimization SVP$_\gamma$ (OptSVP$_\gamma$):** Find d such that $d \leq \lambda_1 \cdot (\mathcal{L}(\mathbf{B})) < \gamma \cdot d$.
- **Decisional SVP$_\gamma$ (GapSVP$_\gamma$):** Given a positive rational number r, determine whether $\lambda_1 \cdot (\mathcal{L}(\mathbf{B})) \leq r$ or $\lambda_1(\mathcal{L}(\mathbf{B})) > \gamma \cdot r$

Definition 9 (Voronoi cell)**.** A Voronoi cell for a lattice point $\mathbf{x} \in \mathcal{L}(\mathbf{B})$ is given as $V(\mathbf{x}, \mathcal{L}) = \{\mathbf{z} \in \mathbf{span}(\mathcal{L}) : \forall \mathbf{y} \in \mathcal{L}(\mathbf{B}), \parallel \mathbf{z} - \mathbf{x} \parallel_2 \leq \parallel \mathbf{z} - \mathbf{y} \parallel_2\}$, The main property of Voronoi cells:

$$\text{sphere of radius } \frac{\lambda_1}{2} \subset V(\mathbf{x}, \mathcal{L}) \subset \text{sphere of radius } \rho$$

Definition 10 (Closest vector problem (CVP))**.** For any lattice $\mathcal{L}(\mathbf{B})$ with basis $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{u} \in span(\mathbf{B})$, CVP is to find $\mathbf{v} \in \mathcal{L}$ such that $\parallel \mathbf{u} - \mathbf{v} \parallel = d_{min}(\mathcal{L})$.

Definition 11 (Approximate CVP)**.** Given a lattice basis $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$, a target vector $\mathbf{t} \in \mathbb{R}^n$, and an approximation factor γ, approximate versions of CVP are given as:

- **Search CVP$_\gamma$:** Find a vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\parallel \mathbf{t} - \mathbf{v} \parallel_2 \leq \gamma \cdot \mathbf{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$, where $\mathbf{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B}))$ is the distance of $\mathbf{t}$ from $\mathcal{L}(\mathbf{B})$.
- **Optimization CVP$_\gamma$ (OptCVP$_\gamma$):** Find d such that $d \leq \mathbf{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) < \gamma \cdot d$.
- **Decisional CVP$_\gamma$ (GapCVP$_\gamma$):** Given a positive rational number r, distinguish between $\mathbf{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) \leq r$ and $\mathbf{dist}(\mathbf{t}, \mathcal{L}(\mathbf{B})) > \gamma \cdot r$.

Definition 12 (Negligible function)**.** A function $\epsilon(k) : \mathbb{N} \to \mathbb{R}$ is said to be negligible if, for every integer $v > 0$, there exists an integer u such that $\epsilon(k) \leq \frac{1}{k^v}$ holds $\forall\, k \geq u$.

III. A Distributed Post-Quantum PoW Consensus Algorithm

In the previous section, we introduced the various computational problems related to lattices. We also introduced how the various problems relate to one another and how they can be converted into one another. We now introduce a post-quantum PoW consensus algorithm that can be used to power the blockchains of tomorrow. Consequently, we talk about how to *distribute* the post-quantum PoW algorithm into multiple smaller problems that can be distributed among several minors in the network.

A. A post-quantum PoW consensus

In the previous subsections, we noted the different computational problems on lattices and the inter-conversions between them. Ideally, it is possible to convert any of the computational problems into a PoW consensus algorithm. We take the simplest one: search version of the *shortest vector problem*. Concretely, the following operations happen when a new block has to be mined:

- The network initializes a lattice $\mathcal{L}(\mathbf{B})$ for some basis $\mathbf{B}$.
- The network initializes an approximation factor γ.
- Many miners compete to solve the search version of the approximation shortest vector problem.
- The first miner to find a non-zero vector $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ such that $\parallel \mathbf{v} \parallel_2 <= \gamma . \lambda_1(\mathcal{L}(\mathbf{B}))$ wins the race and reaps the reward.

Depending on the dimension of the basis $\mathbf{B}$ and its skewness, the hardness of this search version of SVP can be controlled by the network. It is straightforward to note here that once the solution has been found, it is very easy to verify. We note here that from the background given between the relations between the various lattice based computational problems, this consensus can very easily be converted in any and all of the computational problems we have described earlier.

B. Sieving algorithms for search version of SVP

We begin by noting a sieving algorithm for the search version of SVP. Developed by Micciancio *et. al.* [8], it is the foundation of most sieving algorithms used to solve the search version of the shortest vector problem. Several other approaches also exist in literature. (1) LLL-type approximation algorithms, for instance, deal with very small dimensions of lattices (mostly < 15). Therefore, it is not very suitable to our uses since <15 dimensioned lattices are not very *hard* to solve. (2) Enumeration based algorithms also exist wherein *small* vectors are enumerated with an ellipsoid around the origin.

However, drawing of this ellipsoid is not always feasible. All in all, we stick to sieving methods for our goals.

Algorithm 1: GaussSieve

```
Input: B: Input basis, $c$: number of collisions
1  set K = 0
2  stack = NULL
3  list = NULL
4  /*Iterate unless the number of collisions reach $c$*/
5  while $K < c$ do
6      if stack is NOT empty then
7          set $v$ = stack.pop()
8      end
9      else
10         sample $v$ from a gaussian distribution
11     end
12 end
13 // GaussReduce as used in [8]
14 set $v$ = GaussReduce($v$,list,stack)
15 if $v$ is zero-vector then
16     /* a collision has occurred. Increment K */
17     K = K + 1
18 end
19 else
20     /* Found a new vector */
21     list.append($v$)
22 end
```

The actual sieving algorithm is described in algorithm 1. Herein, c denotes the parameter that measures the *goodness* of the algorithm: how many times have we encountered the NULL vector while running the algorithm. Ideally, as the vectors in the list **list** become shorter and shorter, the norm of the vectors approaches 0. In this case, the number of times the NULL vector is obtained as result of *GaussReduce*() is captured by K. When K reaches c, we assume that **list** has sufficiently small vectors that interest us. From a miner's point of view, the value of c has to be chosen by the miner. Note that the choice of c is directly influenced by the choice of γ by the network. Lower the γ, longer the duration for which the algorithm needs to be run, and thus higher the value of c.

C. Distributing the sieving algorithm

Why the sieving algorithm can be distributed?: Note that in algorithm 1, is the stack is empty, a new vector is sampled from the Gaussian distribution and operated upon. Now if we had a second list with mutually gauss-reduced vectors, we could merge the two lists into one through the same procedure as described in algorithm 2.

Distributable list merger algorithm: We now present the algorithm that distributes algorithm 1.

As evident, we take two lists **list** and **second_list** and merge them by taking one vector from the latter, gauss-reducing it with respect to the vectors in the former list, and then appending to to **list**. In the end of this procedure, we have a list of mutually reduced gauss vectors coming from both the lists.

Algorithm 2: ListMerger

```
Input: B: Input basis, $c$: number of collisions, list,
       second_list
1  set K = 0
2  stack = NULL
3  /*Iterate unless the number of collisions reach $c$*/
4  while $K < c$ do
5      if stack is NOT empty then
6          set $v$ = stack.pop()
7      end
8      else
9          set $v$ = second_list.pop() // change in
           algorithm
10     end
11 end
12 // GaussReduce as used in [8]
13 set $v$ = GaussReduce($v$,list,stack)
14 if $v$ is zero-vector then
15     /* a collision has occurred. Increment K */
16     K = K + 1
17 end
18 else
19     /* Found a new vector */
20     list.append($v$)
21 end
```

D. End-to-End workflow

In this section, we present an end-to-end workflow of the entire consensus algorithm.

- The network initiates a cryptographic problem P based on the search version of the shortest vector problem. The network also initialises a parameter k mentioning the number of miners that will take part.
- k random miners from the mining pool are assigned to P.
- The $\frac{k-1}{2}$ miners participate in generating lists.
- The $\frac{k-1}{2} - 1$ miners participate in list mergers
- The remaining miners validate the authenticity of the lists by verifying if the generated vectors in the list actually belong to the lattice B initialized by the network.

When the value of k increases, the final generated list will be large enough to give us a vector short enough to solve the approximate shortest vector problem.

E. Incentive model and other practical issues

In this section, we briefly mention the incentive model to ensure everything works fine.

Incentive 1*: Can a list-generation or a list-merger miner lie?:* The miners tasked with validating the authenticity of the lists will be validating the lists generated by other miners in the system. From [9], given a vector v, it is easy to see if the vector actually belongs to the lattice basis **B** by doing a parallelepiped reduction. Concretely, the following steps are followed:

- Iterate over the lists generated by other miners in the system
- For a lattice basis **B** given by the network, do a parallelepiped reduction of each vector in the list wrt **B**. If any vector fails the test, report the miner from whom the offending list originated.
- Any miner who is flagged by the validation miner is penalized in the network.

This ensures that all miners who are either generating a list or merging two lists will not lie.

Incentive 2*: Can the validation miner lie?:* The job of the validation miner is to ensure the lists generated from other miners are sound and correct. If the validation miner lies about the correctness of the lists, then they can no longer control the correctness of the vectors in the list, and in turn can no longer control the correctness of the solution to the shortest vector problem. If the verification of the final solution of the shortest vector problem fails, the validation miner shall be penalized. Moreover, the penalization of any other miner done by this particular validation miner shall be revoked (since the validation miner is compromised, we assume the flags raised by them are also compromised thereby revoking all such flags).

Incentive 3*: Man-in-the-middle attacks:* Since the lists are sent over an insecure network, how can we prevent man-in-the-middle attacks? One straightforward way is to simply encrypt all communication. But that would require additional overhead which we do not want to incur. Hence, we resort to signatures.

Since the group of k miners chosen for a certain problem is public information, we conclude that we do not need to encrypt communication. A miner receiving a list from another miner can simply check signatures to ensure if the source of the list is actually another miner from among the k miners chosen. This can be done solely by signatures. In our case, we prefer post-quantum BLISS signature scheme [10].

Incentive 4*: Sending lists between miners:* We cannot use the blockchain itself to send the lists between miners, since the length of the lists can get very large. Instead, we leverage Interplanetary File System (IPFS) to share files while recording IPFS transactions on the blockchain.

IV. Implementation details

Currently, the work has been verified in a controlled simulation. We used a MacBook Pro with 2.6 GHz 6-Core Intel Core i7 processor and 16 GB 2667 MHz DDR4 RAM. MacOS version was v11.6. For generating the bases, we used the bases of dimension > 40 from SVP challenge [11]. The challenge bases are decently skewed, implying they have large stretch along one or two dimensions, and almost negligible stretch along other dimensions. The gaussian sampler was implemented using the sagemath library. Results are summarized in table I.

TABLE I: Experimental results

Phase	**Time taken**
List generation	1.5 hours
List merger (two lists)	3 minutes
Generating final solution to SVP	1 minute

As evident in the table I, most of the time spent is in the list generation phase, where beginning with vectors with very large norms, the algorithm tries to gauss-reduce the vectors with increasingly smaller and smaller norms.

V. Conclusions and future directions

In this work, we migrated away from a competitive blockchain architecture towards a more *collaborative*, post-quantum blockchain architecture. Precisely, instead of requiring the miners to compete with one another to solve a partial collision in SHA-256 hash function problem, we force the miners to collaborate on a post-quantum consensus algorithm. We design a distributed version of the sieving algorithm so that several nodes in the network can work together on one problem. Finally, we present the incentive model that helps prevent any miner from forestalling the work of the entire group.

Future work directions involve testing the consensus on a scaled architecture with (preferably) thousands of nodes in the network. In that case, we will need an almost fair job allocator to prevent adversaries from controlling all k miners related to one problem. One way to do this would be to log the hash power of the systems involved (based on the systems' historical work done) and allocate k miners based on a good mixer of the computational hash power.

References

[1] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—a systematic review," *PloS one*, vol. 11, no. 10, p. e0163477, 2016.

[2] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[3] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[4] R. Brown, "Bitcoin's wild ride renews worries about its massive carbon footprint," https://www.cnbc.com/2021/02/05/bitcoin-btc-surge-renews-worries-about-its-massive-carbon-footprint.html, 2021.

[5] M. A. AlAhmad and I. F. Alshaikhli, "Broad view of cryptographic hash functions," *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 4, p. 239, 2013.

[6] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[7] D. Micciancio, "Introduction to lattices," *CSE 206A: Lattice algorithms and applications*, 2012.

[8] D. Micciancio and P. Voulgaris, "Faster exponential time algorithms for the shortest vector problem," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2010, pp. 1468–1480.

[9] D. Micciancio and S. Goldwasser, *Complexity of lattice problems: a cryptographic perspective*. Springer Science & Business Media, 2002, vol. 671.

[10] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Annual Cryptology Conference*. Springer, 2013, pp. 40–56.

[11] S. challenge, "SVP challenge," https://www.latticechallenge.org/svp-challenge/.