NAME: **PAVAN R KASHYAP**

INTRODUCTION:

The rapid advancement of quantum computing poses significant challenges to the security of traditional blockchain technologies. Conventional Merkle trees and Proof-of-Work algorithms, while robust against classical attacks, become vulnerable to quadrupled brute-force attacks when faced with the potential computing power of quantum machines. To address these concerns and create a more secure and decentralized blockchain network, we propose a novel protocol that leverages the principles of Shortest Vector Problem (SVP) and lattice-based cryptography.

SVP is a fundamental problem in lattice theory, aiming to find the shortest non-zero vector within a lattice, representing a grid of points in multidimensional space.

Our protocol aims to revolutionize the way Merkle trees are generated and verified, while integrating the Proof-of-Work process. In the conventional approach, Merkle trees are constructed by repeatedly applying a hash function to generate parent hashes from child hashes. An attacker seeking to manipulate the tree would need to find a specific sub-tree that results in the same hash as the current parent, which requires extensive brute-forcing.

Similarly, the Proof-of-Work algorithm relies on miners discovering a nonce that, when combined with the Merkle root, produces a result that satisfies the difficulty level set by the network. This process also entails extensive brute-force searching to find the correct nonce.

To fortify the blockchain against quantum attacks, our protocol introduces the concept of Shortest Vector Problem and lattice-based cryptography. By employing these techniques, we make the protocol resistant to the threats posed by quantum computers, thus ensuring the integrity and security of the network.

Beyond security, our protocol addresses the issue of centralization in the blockchain network. Currently, a small group of miners with significant computational resources dominate the system, leading to increasing disparities in the distribution of rewards. To mitigate this, we propose the introduction of a new role called "verifiers."

Verifiers play a crucial role in the protocol by independently verifying the integrity of the generated Merkle tree. Unlike miners, verifiers do not have access to the transactions but are provided with a perfect basis and other necessary information. This separation of roles ensures a more decentralized and transparent system, reducing the control wielded by a select few.

Furthermore, our protocol introduces a unique and forward-thinking concept—computational power as an investment. With the future dominated by quantum computing capabilities, computational power will become an essential asset for individuals and entities. By investing in computational resources and participating as verifiers, individuals can potentially reap long-term rewards and contribute to a more equitable blockchain ecosystem.

Conventionally a Merkel tree is one-dimensional or two, depending on how you inspect it. This approach aims at N-dimensionalizing it i.e changing this tree into a tree that spans over N-dimensions. Every transaction that needs to be validated by the miner is first passed into an f() function that maps every transaction (and its metadata) uniquely into an N-dimensional lattice. NIST suggests that when lattice-based cryptography is used, the total number of dimensions to consider must lie somewhere b/w 500 to 1000, if one has to be truly quantum-resistant. Considering N>=500, we will now first understand the characteristics of the f() function first.

The f() function or the lattice mapping function is not a conventional hash function that maps a variable sized message into a fixed size mapping pool. The f() function takes a variable size transaction and maps it into an N-dimensional q-ary lattice, such that

A) The mapping of the transaction onto the N-dimensional q-ary lattice is not linear/chronological. Every mapping is random but unique.
B) For the given N-dimensional q-ary lattice, whenever that particular transaction is passed into the f() function, it always maps to the same point on the lattice. This indicates that repeatability and integrity will be preserved, since every mapping is unique and repeatable.
C) The f() function is able to uniquely map to all the q+1 points in all the N-dimensions before reusing previously allocated points for new mappings

i.e the lattice uses all the q+1 points (q numbers + 0) in all N dimensions for new mappings before reallocating old mapping spaces.

D) An attacker would have to wait for $(q+1)^N$ mappings to happen before they can remap another transaction to the same lattice point. When N and q are very large (say 500 and 309), then the attacker must wait significantly longer than normal (the block would've been in the chain before the attacker can act), thereby preventing any pre-image attacks.

The most important feature of a hash function is that it uniquely represents a certain message/transaction in a fixed number of bits. It is almost as though the hash function consolidates the unique details of that message into a certain bit stream. No matter how many times the hash function is applied over the same message, the hash does not change (the consolidation stays the same).

The same is the case with our lattice mapping function f(). By uniquely representing every transaction on the lattice, it indirectly consolidates all the unique aspects of the transaction/message and maps it into one point.

Once the miners have validated the transactions first, they spray these transactions onto their N-dimensional lattice via the lattice mapping f() function. Once that is complete, they seek approval to start the tree construction by obtaining the bases to work on. This is where the Proof-of-Verification begins.

The Proof-of-Verification primarily gets its name from the verifiers, new entities that populate the blockchain network. We will delve into them a little later, but for now, we will understand that there are various stages in the Proof-of-Verification mechanism. They are –

A) Tree creation – The miners create the N-dimensional Merkel tree of all the validated transactions
B) Optimized spraying – The miners introduce/spray dummy transactions to align the tree to the difficulty set by the consensus
C) Quorum call broadcasting – The miners seek out a quorum of verifiers who can verify the integrity and validity of the tree
D) Verification process – The verifiers verify the integrity of the tree generated; they then vote for or against the validity of the tree. Whichever

quorum is able to verify the fastest and with a maximum majority, that miner's block is placed onto the blockchain.

E) Reward distribution – Once placed, the miners and the verifiers are rewarded for their good work, depending on the schemes and difficulties of the network.

We will explore each section of this protocol now and highlight the technical and economic aspects of it.

TREE CREATION

The transactions that the miner maps onto the lattice are immutable. Once they have been mapped, they are fixed wherever they are mapped. If the miner wishes to operate on these immutable points and generate a tree, then the miner will have to fetch basis vectors. These basis vectors are needed to solve the Shortest Vector Problem (SVP) and obtain a certain linear combination of those vectors that map to the closest point from a given point. When the basis vectors are orthogonal, it is exceptionally easy to identify the shortest vector as they are the smallest possible number (integer in our case) in every dimension. However, when basis vectors start approaching each other i.e get closer, they become worse; the process of finding the SVP becomes hard.

Depending on the consensus agreed upon, cosine similarity is used to map perfect basis vectors for the given lattice into bad basis vectors for the miner. The difficulty dictates how close all the given vectors will be. Cosine similarity is applied in different dimensions and random basis vectors are generated, that follow the network stated difficulty. Once that is done, these basis vectors are assigned/allocated to every miner when the miner requests to start the tree generation. Once allocated, these basis vectors again become immutable for that specific tree generation. The same basis vector must be used for the entirety of that tree creation. Miners cannot meddle with the lattice and choose their own basis vectors; they must use the one that is provided to them.

It can be safely assumed that no miner receives the same basis vector for k times in a row where k >> 1000. We will also assume that after k iterations, the same basis vectors will not appear in the same pattern to a given miner.

This will ensure that the difficulty of finding the SVP is preserved and no miner becomes too comfortable with a certain bad basis.

The N-dimensional tree is N-dimensional only in the distribution of the points. The tree is still a binary tree with two leaf nodes connecting to a parent. All the transactions that the miner selects from the memory pool become the leaf nodes of this tree and they have to be constructed into a binary tree.

If the miner has insufficient number of transactions to generate the Merkel tree, then the miner can introduce a nonce dummy transaction (or an n-transaction) into the list of transactions. A nonce dummy transaction is analogous to a dummy transaction in a conventional Merkel tree, but with a little additional constraint. A dummy transaction is a basically a transaction that involves transfer of no BTC. In our case, it is the same. However, for every dummy transaction the miner initiates in this **phase,** a unique nonce is associated with it just to provide some metadata to the dummy transaction (because a nonce is being provided it is called a nonce dummy transaction or n-transaction). A miner ID can also be introduced as the transaction metadata to signify to the participants in the network that a certain miner has broadcasted an n-transaction for tree generation.

Once the transaction is initiated (metadata created and transaction broadcasted), it has to be passed through the lattice mapping function f(). This function will take the dummy transaction and uniquely map it to a certain position onto the lattice. Whilst all dummy transactions appear alike, it is the introduction of the nonce and the miner ID that uniquely maps it to a given point. Miners must be careful when deciding on how many n-transactions to use i.e they must introduce only as many n-transactions as they need. This is because like conventional transactions, once these n-transactions have been mapped onto the lattice they are immutable. If a miner unnecessarily chooses to introduce several n-transactions, then the miner only complicates the work they must accomplish to generate the tree.

Once the miner has completed spraying the n-transactions, the miner focuses on generating the tree. The quantum resistant algorithm employed for the generation of the tree is SVP (Shortest Vector Problem) as stated previously. The miner does some pre-processing to identify which of the given transaction pairs are close enough. There is no point generating an

SVP for two transactions that are totally far apart from each other (probably in very distinct dimensions).

Once a transaction pair is generated, the SVP for the two transactions is obtained. Let us consider the two transactions X and Y such that they both are relatively close to each other

$X \rightarrow (a_1, a_2, \ldots, a_n)$ $Y \rightarrow (a_1 + k_1, a_2 + k_2, \ldots, a_n + k_n)$

Where k_i is some constant in the i^{th} dimension.

SVP is traditionally calculated for one vector. In our case, we wish to generate a tree structure where the SVP behaves like the root of the two transactions. To account for these two conditions that we must uphold, we decide to generate the mid-point between these two vectors (R) -

$R = (X + Y)/2 = \text{floor}(a_1 + (k_1/2), a_2 + (k_2/2) \ldots, a_n + (k_n/2))$

The floor function ensures that the results are always integers. The result of the average b/w two integers can either be a whole number or a whole number with 0.5 as its fractional part. This indicates that there are only two possible outcomes for R. It would not be ideal to round up a num.5 because the average has not reached the higher integer. Therefore, the floor function is considered most optimal for this approach.

It is true that we should be writing R truly as R mod q, but we realise that the average of two numbers within the q-ary lattice will never exceed q, and therefore mod q is not needed.

R represents an amalgamation of two unique vectors X and Y. Whilst it is possible that individual dimensions of R may be same as some other R' vectors, one thing is definite – Two other transactions Z and W cannot generate an R' such that R' = R.

In simple terms what that means is that R as a vector is unique because it is generated by taking the average of two other unique vectors which will not repeat for q^N times at least.

When generating a binary tree, we usually denote the parent node as the centre node that joins both the child nodes. In our case, R is technically our

centre of X and Y (maybe not mathematically) but theoretically it is the midway between the two vectors.

We have taken two vectors/points and generated a single vector/point of it now. Now, we can apply the SVP algorithm on this vector to obtain the Shortest Vector point (SVp).

The example suggested above implies that the two points have some influence in every dimension. It is not necessary that the influence be uniform in every dimension. Unless we know the constants a_i, k_i and the other transactions in the lattice, we cannot comment about whether X and Y are close to each other. This example is highly generic and is only aimed at highlighting the point on which the SVP algorithm is applied (R).

We will justify why the SVp of R is truly an equivalent for SHA(l || r).

It is true that the parent node must represent the underlying subtree effectively and SHA does so with its hash. By combining the left and right nodes in its subtree the parent represents the entire sub-tree in a compact manner. So does the SVp of R.

We understand that no two SVps are alike. This is primarily because R vector in itself is unique. R can only be generated by a certain X and a certain Y. So, this implies that the SVp of R is indeed uniquely representing the underlying subtree (the transactions or the other SVPs). It is consolidating the information that was used to generate it, the two transactions X and Y in our case, just like a typical hash function at the root.

The SVp is an N-dimensional vector which holds 0 in the dimensions that it is not present in. A traditional hash function used in Merkel tree generation has a fixed length output. So does the SVp that we generate, because it represents the results in all N-dimensions. So, another property of the hash function is upheld by our shortest vector point (SVp).

No two SVps (shortest vector point) are alike because no two transactions generate the same (X+Y)/2. No two points generate the same R because no two points are ever mapped to the same place in the lattice (for a long period of time). The use of the lattice mapping function along with the SVP algorithm

makes the SVps distinct. This ensures that there is no collision resistance, another important property of the hash function.

Possession of the SVp of R does not directly guarantee an attacker the point R itself. The attacker does not know the exact point in the N-dimensional space which is closest to the given SVp, The attacker must try out different possibilities in N-dimensions to truly obtain R, which is a very computationally expensive activity. Even if the attacker were to find out R, it would be next to impossible for an attacker to reverse engineer and identify where X and Y were mapped because R or SVp of R does not directly reveal any information about X or Y.

This indicates that once an SVp is generated, one cannot use the SVp to trace back and recreate the underlying subtree. This is the one-way property of the hash function which is again satisfied by the SVP.

Pre-image attacks may be possible if attacker identifies X and decides to introduce a certain Z such that R is the same. Another case where an attack is possible is when the attacker introduces their own Z and W pair such that it generates the same R as for X and Y.

As discussed previously, for a given X and R, there is only a certain Y that lands it there. The attacker cannot introduce another point Z anywhere in the lattice such that exactly R is generated (unless Z is placed at Y's location itself).

Similarly for the second case, whilst it is true that certain dimensional coefficients of R might be the same for another R' ((Z+W)/2), it is not possible for all coefficients of R to be the same (if they are, then Z and W are basically just X and Y).

This again highlights the fact that the SVp generated is immune to pre-image attacks too.

To reiterate the difference, the SVP is the Shortest Vector Problem, an algorithm that uses lattice mappings to find the shortest vector to a given point, whilst the SVp is the shortest vector point that is generated on applying the SVP algorithm on R. Whilst SVP does not satisfy any properties of hash functions, the SVp generated does follow.

This indicates that we can remove the conventional hash function which will become vulnerable to quantum attacks and replace it with the SVp, which is quantum-resistant and fully classifiable as a hash function that uniquely represents a given sub-tree. What we must understand here is that the use of the SVP along with the lattice mapping function f() that truly introduces the hash function properties in the SVp. Without the mapping function, the SVP algorithm cannot introduce hash characteristics into the SVp of R.

The miner now iteratively generates the SVps for all the subsequent subtrees until the root SVp is obtained. This root SVp is analogous to the root hash in a typical Merkel tree. However, now we have a binary tree that is distributed in N-dimensions. Once the final SVp is computed, it is subsequently stored for use in the next stage. The tree generation activity has been successfully completed. The decision to clear the lattice points on the lattice is up to the miner – if the miner wishes to operate on a fresh lattice for the next stage, they can clear the transaction mappings and only retain the root SVp. If the miners believe that the scattering of the points in the lattice will help them with optimization in the next stage, they can decide to retain it (as immutable entities). The basis vectors continue to stay the same even if the lattice is cleared for reuse.

Once the tree generation is complete, the root SVp gets mapped as an immutable entity onto the lattice. This is the exit criterion for this stage and an entry criterion for the next.

OPTIMIZED SPRAYING

Miners have now generated the basic Merkel tree. We will now need to introduce some computationally expensive operation(s) analogous to the Proof-of-Work i.e we will now have to operate on this Merkel tree and generate a certain result that aligns with the difficulty set by the network.

To do the same, here is the followed approach

A) The network agrees upon a certain difficulty level. This difficulty level suggests that the final Merkel tree generated from the miner must have a non-zero co-efficient in certain given dimensions. These dimensions may be contiguous, may be total extremes (1st and 500th dimension say) or anything in between. The number of dimensions that must have a co-

efficient is again decided by the difficulty level. Certain difficulties can suggest more restrictive rules such as the value of the 5th dimension must have 12 as its final co-efficient and so on.

B) The probability that the tree generated in the previous stage lies in all the dimensions mentioned is very low. This indicates that there must be some modification done to the tree to eventually get them to lie in certain dimensions. To do so, we introduce the concept of miner dummy transactions or m-transactions.

Miner dummy transactions are dummy transactions that the miner introduces in the lattice to steer the direction of the N-dimensional tree to the dimensions set by the difficulty level. These transactions have no nonces associated with them and whenever these transactions are passed into lattice mapping function, they always map to the origin of the lattice. These transactions, unlike the n-transactions are not immutable until they are committed. What that implies is that once an m-transaction is introduced in the lattice, the miner has the full liberty to place this transaction at any point in the lattice. The miner can place this transaction point on the existing points occupied by the SVps or the committed transactions. The miner can place this m-transaction point on different ends of the lattice and obtain the new SVp (where X is the root SVp of the tree generated and Y is the m- transaction point). The miner has full liberty to place this m-transaction at any point on the lattice and calculate the corresponding SVP.

Every miner can introduce any number of m-transactions to steer the tree's direction. However, there is one contention here. If the miner wishes to introduce a new m-transaction, then they must commit the previous m-transaction to a certain point on the lattice i.e deem it immutable. This is essential because every time a new m-transaction is introduced there will be only one previous SVp which will act with it.

If it is the first m-transaction, then the previous SVp will be the root SVp of the tree generated.

If it is the nth m-transaction, then the previous SVp will be the SVp generated by the (n-1)th m-transaction and its previous SVp.

Whilst the miner has the flexibility of introducing any number of dummy transactions, the miner must focus on optimising resources rather than brute-forcing. The miner must make a choice for every m-transaction if they wish to use trial and error to generate better steering results or if they wish to commit it and generate a new m-transaction. A miner is using computational resources extensively if they keep adding new m-transactions. A miner might be delayed if they keep using trial and error methods for mapping lesser m-transactions. This trade-off makes the miners more vigilant and self-aware. Unlike the conventional brute forcing techniques used by Proof-of-work, Proof-of-validation works on trying to obtain better solutions in a more optimal way.

C) The miner continues this iteration of generating new SVps until the final root SVp is generated. The miner has used k m-transactions to steer the course of the tree root SVp to the final root SVp. Whenever root SVp is suggested, please recollect is as the SVp of the transaction tree (Merkel tree). Whenever final root SVp is suggested, please recollect if as the final root of the entire tree (the transaction subtree + k m-transactions). The diagram followed by the explanation below should make this concept clear --

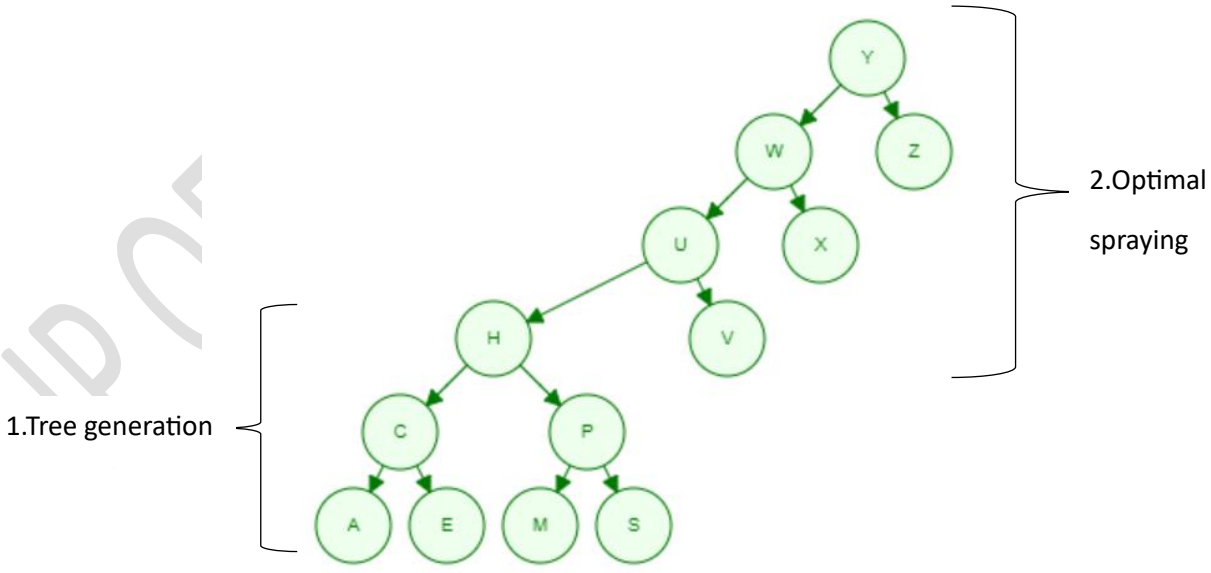

Fig 1: A 2-D representation of the final tree after optimal spraying and tree generation is completed.

Tool used: https://www.cs.usfca.edu/~galles/visualization/BST.html

In Figure 1 A,E,M are the transactions that the miner has selected and mapped into lattice points from the memory pool. S in an n-transaction introduced by the miner to generate the tree. C is the SVp of the average point b/w A and E. P is the Svp of the average point b/w M and S.
H is the root SVp i.e the SVp of the transaction tree. That is the final point that is carried forward to the optimal spraying stage.
V, X and Z are m-transactions. U is the SVp of the average of H and V (where V is committed to a certain point on the lattice). The same holds true for W and then the final root SVp that is generated is Y. This satisfies the difficulty constraints.
Whilst some may argue that the overall tree is heavily skewed, and that it does not resemble a traditional Merkel tree, I would like to remind the reader, that the transaction tree is the entity that is analogous to the traditional Merkel tree and not the entirety of the skewed tree. The transaction tree takes into account closely placed transaction pairs, and generated a tree that is evenly distributed, thereby ensuring that integrity and security of the Merkel tree. The introduction of the skewness is solely to steer the tree to dimensions where its transaction tree root's presence might not have existed at all. The skewness accounts for solving the difficulty and is not associated with the integrity of the transaction tree.

The choice of skewing the tree depends entirely on the miner and the miner's optimization techniques. A miner can use suitable optimization to ensure that the final root is not heavily far off the transaction tree root (lesser number of m-transactions are introduced). A miner may decide to not use any optimization and manually introduce m-transactions at points on the lattice where he/she/they believe(s) the tree will start drifting to the intended dimensions. We will discuss further how heavily skewing the tree will be disadvantageous to the miner, but at this point we will understand that miners must strive to minimise the skewness if they wish to be a greater benefactor of the reward generated when a block is mined.

A valid concern that props up at this stage is, can I reverse engineer where the m-transactions have to be located, based on the constraints of the network?
As a miner I am aware of the transaction tree SVp. This will serve as my transaction X. I know the final SVp based on the constraints provided by the

network. I have to identify where to place my m-transaction, which is basically my transaction Y.

Can I not use the formula floor R = (X+Y)/2], to re-arrange the same equation to Y = 2R – X , and obtain the exact location where Y (the dummy transaction) has to be placed?

No, you cannot follow this reverse engineering approach. There are few reasons why this approach will not necessarily work.

Firstly, the final root SVp is not R, but the SVp of an R. This indicates that I cannot plug in the network constraints as R and reverse engineer Y.

If the miner truly wishes to use this equation, then the miner must use the SVp to generate R first and then go ahead with deciphering Y. As discussed previously, it is exceptionally difficult to obtain R from the SVp of R , because we do not know which point it is closed to, and which dimensions.

Secondly, the network constraints dictate that certain constants have to be present in certain dimensions, and certain coefficients must be present in certain dimensions. This implies that the final root SVp is not one unique solution; there can be several coefficient values that can fit the dimensional variables. This implies that a miner will have to manually try out different SVps to reverse-engineer and identify Y. Some of these SVps may never be reached from the transaction tree root, no matter where an m-transaction is placed. This indicates that the miner cannot easily obtain Y, owing to the added nature of trial and error in this approach of reverse engineering.

Thirdly, the original equation holds a floor function on the RHS side [floor((X+Y)/2]. This implies that I cannot simply re-arrange the terms to generate Y = 2R-X. Every coefficient present in R could be a floored result of X and Y or an integer result of X and Y. There is no way for us to know that when we possess R. If we do not truly know if the R, we possess is the result of a floored action on a co-efficient or a normal integer, how will we use the rearranged equation to exactly recompute Y back? Several trial-and-error actions will need to be carried out (first reverse-engineer Y, then use the X and Y to recompute R, find SVp of R and check if it is same as what you started with, repeat and refine till they match etc.) to accurately obtain the value of Y. That is again computationally expensive.

Lastly, as previously suggested, there may never be a case where a single M-transaction can steer the tree into the intended dimensions. So, reverse engineering Y may yield no results, or results outside the q-ary lattice, if they ever are obtained.
These points clearly indicate the great difficulties associated with reverse-engineering Y. This assures us that no miner will be able to deduce the locations of the m-transactions very quickly. The difficulty posed by the network is still preserved, and there is no work around, as of now.

Whilst reverse engineering may not yield substantial results, it could still be part of the miner's bag of optimization techniques. Miners can use trial and error to understand how different positions of points on the lattice steer the tree differently. They can devise their own mechanisms for optimization as they learn and adapt to the mining process.
This ensures that the miners are simply not doing a monotonous task, like sheer brute forcing in Proof-of-work. The onus shifts from exhausting a linear search space, to understanding and optimising mappings to a lattice space.

D) Once the miner has successfully completed the process of optimal spraying, the entire N-dimensional Merkel tree is complete. The miner must now verify if the N-dimensional Merkel tree generated is valid, if he/she/they wish(es) to put the block generated into the Blockchain.
To do so, the miner must broadcast the results obtained to verifiers, who will verify the validity and the integrity of the generated tree.

Conventionally, in a traditional PoW scheme, the miners verify the root hash in O(1) time complexity. The verifier calculates the hash of the child nodes, generates a hash and then checks if this root hash is satisfying the network constraints to makes a decision. If a substantial majority of verifiers agree with the miner's block validity, then it is placed onto the Blockchain.

However, the same cannot be followed in our scheme. To even verify if the final SVp of R is accurate, I would need to first obtain R and then find the SVp of R, before comparing it with the value provided. This indicates that more computation power has to be used to first find the SVp and then verify the SVp value.

Verifiers receive no incentives in the conventional scheme because the process of verification is extremely easy; should someone fail to verify the result; the miners can always consult another verifier for verification. The same cannot be stated for our case. In our case, verifiers are investing computation power to calculate the SVp for verification. If I do not provide verifiers with any incentives to verify, no one will agree to utilize their computation power for sheer goodwill. If verifiers stop the verification process, then it will disrupt the blockchain network. Therefore, to ensure that the Blockchain network persists, verifiers are considered separate entities in this protocol and incentives are provided to them for verifying N-dimensional Merkel trees.

The entry and exit of a verifier, is pretty much a free market i.e anyone in the blockchain network can decide to become a verifier if they have the appropriate computation power for verification. Once a verifier, the verifier must strive to maintain ethical practices in verification. Should a verifier be caught doing malicious activity, significant punishments will be imposed on that verifier, which can go up to the limit of permanently suspending the entity from the blockchain network.

Entities in the network can either become a verifier or a miner. They cannot vacillate between the two boundaries based on circumstances. They must render their responsibilities ethically while they are active in their role of choice.

The primary responsibility of the verifier, as stated previously is to verify the validity and the integrity of the N-dimensional Merkel tree. To do so, the miner provides every verifier with the SVp of the transaction tree, the dummy transactions' co-ordinates and depth and the final root SVp generated by the miner. The verifier has no access to the transactions that the miner has selected from the memory pool/ does not know what transactions make up the N-dimensional Merkel tree. This ensures that the anonymity of those transactions is upheld.

NAME: **PAVAN R KASHYAP**

QUORUM CALL BROADCASTING

Once the miner has completed the tree generation, the miner initiates a broadcast message to all the verifiers in the network asking them to join the quorum for verification. The quorum is basically a body of N entities, all of whom verify the miner's N-dimensional Merkel tree. The size of the quorum is decided by consensus and dynamically changes as the number of verifiers in the network change. The miner initiates the broadcast call and waits for a certain number of verifiers to join. Once the quorum is complete, based on the difficulty of the basis of the miner, the quorum is provided a certain time duration for operation. The quorum has exactly that time duration to verify and vote accordingly. The quorum's time duration will continue until one of the following actions occur

A) The quorum's time is exhausted and the entirety of the quorum has not voted/responded.
B) The entire quorum has responded/completed the verification much earlier than the quorum timeout.
C) Another quorum has successfully completed the verification of the same/similar block (that block contained a majority of the same transactions that this block contained) in lesser time and with a higher positive majority.

Verifiers are allowed to compute until the timeout is reached. If a verifier completed the computation much earlier than the timeout period, then the verifier can vote, but they cannot exit the quorum until all other verifiers have completed the verification or a timeout is reached. Once a verifier has exited a quorum, the verifier is placed in a cool down period for a certain k minute (depends on the network constraints). The verifier cannot participate in any quorums until the cool down period is complete. Once done, the verifier can then again decide to be part of a quorum, on receiving a broadcasting call for it.

The cool down time and the binding to the quorum timeout ensures that verifiers do not overwork to obtain more rewards. The concept of introducing rewards to verifiers was to ensure that verifiers receive

small but consistent rewards for their activity. If they start getting over ambitious, then our point is defeated.

The time duration allocated for the quorum does not depend on the number of dummy transactions sprayed by the user; it only depends on the difficulty of the basis and the difficulty associated with the dimensions. This highlights that the network does not consider introduction of large number of m-transactions as a worthy measure of difficulty. It highlights a poorly crafted, heavily skewed N-dimensional Merkel tree. That is a consequence of the bad choices made by the miner; that does not indicate that the tree was difficult to generate.

Increasing the time duration for those trees that have large number of m-transactions will jeopardise the working of the blockchain environment by locking verifiers into quorums for long durations. Therefore, it is not considered a valid parameter for deciding the timeout period.

We will now understand the specifics of how a verifier is mapped to a given miner's quorum. An important point to remember is that every verifier can be part of at most one quorum at one time.

The miner broadcasts its difficulty level and calls for a quorum. The miner's details are not revealed, the anonymity of the miner is upheld to prevent collusions between miner and verifiers. It is assumed that only valid, registered miners are capable of calling for a quorum. Any malicious miner that calls a quorum and locks verifiers with the aim of making them unavailable will be severely penalised for doing so. The miner's past rewards will be proportionately distributed to all the verifiers who were locked/compromised. This act will serve as a reminder to miners to engage in only meaningful, moral and ethical quorum calls.

Every verifier is associated with a certain computation index, that highlights the nature/capacity of that verifier's computation power. Verifiers, by default, are made aware of those quorum calls that match their computation index. This ensures that verifiers do not unnecessarily join higher index quorums and jeopardise that quorum's

reward prospects, whilst it also ensures that verifiers do not join lower index quorums and risk the extent of the rewards they receive.

This mapping to equiproportional quorums ensures that the reward the verifier receives is consistent with their peers. It also prevents verifiers from finishing the verification extremely early or leeching the quorum by carrying out the verification extremely slowly (because of lack of computation power).

In general, quorums that have higher computation power receive greater portions of the mining reward, and have lesser timeout periods. Quorums with lower computation power have greater timeout periods but lesser rewards.

Miners, based on their computation index must decide on what type of quorum they wish to call. Do they wish to call a high quorum and risk losing greater proportions of the reward to verifiers? Or do they wish to call a lower quorum and risk losing the block generation to their competitors (because of higher timeouts)?

Whilst verifiers are mapped to their commensurate quorums, verifiers can decide to participate in higher quorums, if they wish to receive higher rewards. They can participate in higher index quorums but they have to stake a certain amount of BTC as a promise that they will not jeopardize the rewards of the quorum by delaying the execution. The amount of stake increases exponentially as one moves up to higher index quorums.

If the verifier is successfully able to perform as well as the entities in the higher quorum, then the verifier not only gets back their stake, and the additional rewards, but they get to participate in the same index quorums at lower stake amounts.

If the verifier fails to verify the tree, then all of the stake is lost to the quorum. If the verifier is believed to be the leeching entity, then apart from the stake, the verifier is penalized to pay for the expenses incurred by the other entities in the quorum. This fear, will ensure that verifiers cautiously decide when they wish to participate in higher index quorums, whilst still having a lower computation index.

A verifier can increase his/her/their computation power to boost the computation index, so that verifiers now participate in higher index quorums, by default.

Although it is discouraged, a verifier can participate in lower index quorums than their own. Practically it does not make sense for verifiers to participate in quorums that provide lesser rewards and take longer times. Once a verifier has scaled their computation power, it indicates that the verifier is ready to handle greater computation load; the verifier must not deprecate their capabilities after scaling up, except for in certain economic situations, which we will discuss towards the end.

A verifier must make the conscious choice of whether they wish to scale or whether they wish to continue reaping whatever benefits they receive of their current computation index.

By default, every quorum has a fixed number of participants at all times. If quorums change dynamically with increase in the number of verifiers, then it will result in the dilution of the rewards received by individual verifiers. The aim is to not scale quorum size as the verifier pool scales/ increases. If rewards in the quorum start getting diluted, then we will eventually reach a point where we come back to the traditional blockchain scheme, where every verifier is receiving a reward that is almost equivalent to nothing.

So, quorum sizes, while can change, do not change with an increase in the number of verifiers. At this point, we will assume that there will never be a dearth in the number of verifiers who wish to verify the trees. The reward will incentivize new verifiers to enter the market, even if the old ones leave.

If the miners do not find their quorum size, then miners can incentivise their verifiers by promising higher rewards to the verifiers for successful verification. This will drive the verifiers to join the quorum and continue verification.

If quorum sizes change and they do not change commensurately with a change in the number of verifiers, what brings about that change?

An increase in the number of verifiers in the network does not directly indicate an increase in the number of valid, consistent verifiers. Valid, consistent verifiers are those set of verifiers who have been consistently supporting the verification process and keeping the network competitive. It highlights their interest in continuing to support the blockchain network. Increase in verifiers does not directly imply increase in the number of valid, consistent verifiers.

The quorum size can dynamically change when the proportion of valid consistent verifiers in every quorum rises by a certain factor k (or a k%). This will indicate that the proportion of valid entities that are actively engaged in keeping the network intact is increasing, and the quorum must now scale to increase the security and the efficiency of the quorum. That is the situation or the condition when the quorum size will increase.

The stake price will decrease as the number of valid consistent verifiers in the network increase. This will act as an incentive, encouraging verifiers to stay for the long-haul and contribute to a more safe and secure blockchain environment.

VERIFICATION PROCESS

Now, that we have discussed and understood the procedures involved in calling a quorum, we will now focus on understanding how the verification process works.

As soon as a verifier is inducted into a quorum, the verifier is provided with the perfect basis vector and the lattice for the mapping. All verifiers, no matter what quorum index they are part of, are provided with the perfect basis. The difference in every quorum index is dictated by the difficulty of the miner's basis, the difficulty of the dimensions, their respective computation powers and their timeouts. Every verifier is provided with the right basis, so that they do not have to struggle with additional difficulties. Reconstructing certain parts of the tree, which is the verifier's job is itself a challenging task. We do not wish to complicate that by making the verifiers struggle.

The miner distributes information to the quorum for verification on a need-to-know basis as a broadcast message. There are five primary steps involved in the verification process that will be explained in detail.

A) M-transaction verification
B) First SVp verification
C) Final root Svp verification
D) Network Constraints verification
E) Voting

M-transaction verification

The miners sequentially number all their m-transactions and store them. As soon as the miner receives the intended quorum, the miner first broadcasts all the stored m-transactions to all the intended quorum entities. These m-transactions do not reveal or indicate anything about the transaction tree, so disclosing it to the verifiers will not cause any privacy/security issues.

The verifiers obtain the m-transactions and pass them through their lattice mapping function f(). As stated previously, all m-transactions always map to the origin (the miner can move them later, they are first mapped to the origin). The verifiers will verify if all the m-transactions the miner has introduced to steer the tree, are in fact m-transactions. This verification is essential to verify if the miner has truly used m-transactions to steer the transaction tree. A miner could have provided n-transactions and regular transactions as m-transactions (deception) to trick the verifiers into believing the final Merkel tree is the transaction tree itself. This has to be verified, and therefore this verification is done.

It can be safely assumed that the probability of an N-dimensional transaction tree satisfying all the network constraints (of dimensions) is very low (~0).

The sequencing of the m-transactions provides transparency to the verifiers of the miner’s steering elements.

If certain broadcasted m-transactions do not map to the origin, then the verifier automatically returns FalseA and alerts the entity that provided it with the basis that the miner is deceptive. If majority of the quorum members follow the same error message, then the miner is penalised for locking these verifiers for a tiny duration. The penalty that the miner has to pay at this stage is less, because the miner has not significantly locked up the verifiers.

If the same activity persists, then the miner can be expelled from the network itself,

The penalty a miner has to pay for duping verifiers increases with every stage i.e the more time the miner locks up the computation power of the verifiers, the greater the penalty the miner must pay, recompensating for the same.

First SVp verification

Once the first stage is complete, the miner now broadcasts the co-ordinates of all the m-transactions introduced to the verifiers. The miner also specifies the depth of each of these m-transactions to indicate which m-transaction has to be selected for tree reconstruction using the bottom-up approach. To take an example, we will consider the previous tree that we discussed.

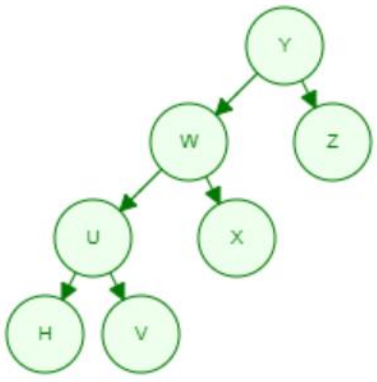

Fig 2: Understanding the depth levels of the m-transactions

To recollect, H is the transaction tree SVp. V, X and Z are the m-transactions accordingly and Y is the final root SVp.

The miner broadcasts the details of the m-transactions as follows-

M – transaction	Co-ordinates	Depth (tree root: 0)
Node V	(0, 13, 2, 9, 5, 0)	3
Node X	(9, 12, 309, 0, 1)	2
Node Z	(0, 0, 11, 15, 15)	1

Table 1: M-transaction metadata table

The verifiers now place all the m-transactions at the appropriate positions and label each of their points accordingly. The first SVp that will be calculated will be with Z, the second with X and so on.

Once the m-transaction metadata is obtained, the miner broadcasts two crucial data parameters, the transaction tree SVp and its parent SVp.

Shortest Vector point (SVp)	Co-ordinates
Transaction tree SVp (Node H)	(1, 1 ,15, 3, 9)
First parent SVp (Node U)	(7,7,0,5,1)

Table 2: SVp metadata broadcasted to the verifiers

The verifier maps the transaction tree SVp onto the lattice. The verifier then calculates the first parent SVp of the final tree, by taking the transaction tree SVp as the first entity in the transaction pair and the m-transaction with the highest depth as the second entity in the transaction pair.

Note that the basis used by the verifier is perfect, so calculation of the first parent SVp (Node U) is accurately obtained. Once it is obtained, the calculated SVp is compared with the SVp obtained. If the two values match, then it indicates that miner has correctly obtained the SVp. It only makes sense for the verifier to continue with the tree construction if and only if the miner has calculated the SVp accurately.

The assumption we are making here is that if this SVp is accurate, then it implies that the transaction tree has been built correctly, and the subsequent final tree built must also be correct. It is a valid assumption because if the miner despite the bad basis, is able to correctly generate one SVp, then it means all other SVps will be accurately generated too.

Since the verifiers have the perfect basis, we believe that the SVp calculated by the verifier is perfect/accurate. If the miner's SVp does not match, then a FalseTerminate message is sent to the managing entity, indicating that the SVp calculation is wrong, and hence the possibility of the entire tree being wrong is high (and so is the transaction tree). If a majority of the verifiers generate this error flag, then the block is immediately discarded. The cool down period is revoked for miners in such cases, as they have helped in protecting the network for attacks on integrity and security. The miner is penalised for this activity, but the punishment depends on the extent of the deviation of the SVp from the original.

If it is significantly deviant, then the miner is significantly punished for wasting verifier's time and resources. If the deviation is small, the miner is punished little, and encouraged to strive for better results.

Whilst some may argue that a very small deviation in the SVp can be tolerated and carried ahead for use, let me remind you that any small changes done to a lower subtree SVp will have a cascading effect on the final root SVp. The final SVp may differ significantly from the true SVp, if a small error is accounted for at a certain depth. This highlights the need for a perfect SVp and a perfect basis, and therefore verifiers are equipped in such a way.

Final SVp verification

The final SVp generated by the miner is disclosed to the verifiers as soon as they join the quorum. This part is mentioned here, because it caters to this particular task. Different verifiers may be doing different stages of the verification process at different times. This implies that the miner cannot broadcast the details to all entities when a specific

verifier requests for it. Similarly, if the miner sends the final SVp to individual verifiers after the First SVp verification, then the miner can reverse engineer and map how many entities have proceeded to the next stage/step.

To ensure that anonymity is preserved, the miner is not directly involved in the quorum at all. The miner publishes all the metadata and stream details to the quorum management entity. The miner is blocked for the quorum time instance too (miner's cooling period). The management entity first broadcasts the final root SVp and asks the verifiers to safely store them first. Then, the procedures stated in the M-transaction verification phase continue.

So, to summarize the idea, we have understood through elaboration that the first activity initiated when a quorum is created is publishing of the SVp, along with the provision of the perfect basis and lattice. Secondly, we understand that the miner never directly interacts with any of the participating verifiers. The miner is also not the entity that receives all the votes from the quorum, it is the management entity.

Once the first verification is complete, the verifiers iteratively start building the tree using the bottom-up approach. Once the last m-transaction at depth 1 has been used to generate the final root SVp, the verifiers compare it with the final SVp that they have stored at the beginning.

If the two SVps align, then it indicates that the tree has been crafted correctly by the miner. This verification is an O(1) time complexity activity and is essential to check before the constraints are checked. This check assures the verifier that the miner has consistently generated correct SVps for the transaction pairs. If the final root SVps do not match, then the verifier votes FalselNotMatch, indicating that the final root hash is not matching.

If the miner is the miscreant, then the verifiers are rewarded by purging the past rewards of the miner. If the verifier is the miscreant (if a majority consensus claims that the miner is correct, and this verifier

suggests FalseNotMatch, then it indicates flaws in the verifier's approach), then the verifier is penalised for trying to disrupt the successful working of the quorum. One possible penalty could include getting blacklisted from participating in any quorums over m days.

Once the verifiers have reconstructed certain parts of the tree and verified the root hash, the last set of tasks get completed very quickly.

Network constraint verification

The primary and most important constraint is checked at this stage. The dimensions of the final SVp are compared with the dimensional requirements specified by the network. If they match, then it indicates that the miner has perfectly generated a valid block that can be mined and placed in the chain. If it fails, then it indicates that whilst the SVps have all been accurate, the miner has not satisfied the primary constraints itself. This is the stage where the miner is most heavily penalized for any malicious activity.

Verifiers spend extensive amount of time and computation power to reach the last stage of the verification process. If they realise that all this effort is futile, then verifiers will never wish to invest their resources into such situations. To prevent verifiers from being disincentivised by faults of the miners, the miners are stripped of their past rewards and they are distributed to the verifiers.

The verifier signals a FalseFinal error flag to the management entity in this case. If a majority signal the same, the management entity halts the quorum from proceeding any further, strips rewards from the miner, allocates it to the verifiers based on the extent of work they have completed and then closes the quorum.

This step of verification is again an O(1) time complexity verification. This indicates that majority of the time is spent by the verifiers in reconstructing the partial tree. Once the partial tree is constructed, the verification process completes quickly.

By increasing the computation power, verifiers can speed up their tree construction procedure, thereby finishing faster. Usually, the higher

quorum indexes have smaller timeout periods, as previously stated. They however, have larger cool down periods. This is to prevent them from becoming over-ambitious and generating magnus amounts of wealth in the same time as it takes for smaller quorum indexes to finish one round.

Voting

On successful verification, the verifier sends a TrueVote message to the verifier indicating that the verification is successful. If a majority of TrueVote is received (need not be 51% always, the majority percentage can be set by consensus to say 70% or 85%), then the management entity checks to identify if no other blocks have been minted before it with the same/higher vote percentage. If there are none in the pool of quorums that are working on similar transactions, then the management entity broadcasts a blocking message indicating that this miner has successfully completed the verification stage. The blocking message suspends the quorums that are verifying same/similar transaction blocks, thereby preventing them from using their resources any further. After that, the miner is intimated about the success. The management entity appends the miner's block onto the chain. Once that is complete, the reward of mining a block is distributing amongst the miners and the verifiers.

The successful miners and verifiers now have to undergo a larger cool down period, to allow for other verifiers and miners to receive rewards too.

MANAGEMENT ENTITY (LOCAL)

To ensure that the quorum functions smoothly, the management entity operates/handles the quorum. The management entity ensures that the verifiers are unaware of the miner's identity or other verifiers. Likewise, it also ensures that the miner is not aware of the verifiers that have decided to participate in the quorum. This ensures that miners and verifiers do not collude together and decide to mine and verify together. The management entity enforces the concept that a certain verifier cannot be part of a certain forum for k times or k time

durations, if it has already been part of it once. This ensures that verifiers do not keep joining the same miner quorums again and again. It prevents verifiers from getting too comfortable with the miners, their working and their rewards. This way, collusion or favouritism is avoided.

Verifiers can join new quorums of the same computation index by following mechanisms decided by the consensus protocol (fastest connector first, auction system etc.).

The miner is a mute entity during the quorum's functioning. The management entity collects logs, error messages and uses those details to appropriately take the necessary actions. Most of the management entity's activities have been previously covered. We will now jump into the concept of reward distribution and then discover computation power as an investment.

REWARD DISTRIBUTION

We have briefly discussed about the concept of rewards so far. Now, we will discuss the specifics on how rewards are distributed.

Traditionally, in the conventional blockchain scheme, when a successful block is mined and added to the network, the miner is rewarded with a certain amount of Bitcoin (BTC). In our protocol, however, because there are two entities involved, the miners and the verifiers, the reward that was previously only reserved for the miner is now distributed amongst the miner and the verifiers who have actively helped the miner mine and place the valid block.

The first question that may arise is, why will miners be willing to use this protocol? The conventional protocol is giving them all of the rewards, why share with verifiers?

The aim of this protocol is to prevent centralization or accumulation of wealth and assets to a selected few. Whilst a miner may be tempted to stick to the traditional protocol, there will be active participation from the verifier end, to adopt to this new protocol. If the miners do not follow the change, then they will face the fate of becoming obsolete.

With Quantum computers becoming a reality, the conventional Merkel trees and PoW mechanisms come at a risk. Relative to losing everything to a malicious entity, miners will agree to concede and share some part of their rewards with verifiers for helping them place the block in the network. It will favour a healthy relationship where miners and verifiers can both benefit mutually. That is significantly more motivating to a miner, than the risks posed by quantum computers on their traditional protocols.

As the verifier's computation index increases, the reward that the verifier receives increases. In other words, if a miner selects a higher index quorum, then a greater proportion of the reward is given to the verifiers and a lesser proportion remains with the miner.

If a miner selects lower index quorums, then whilst the miner retains the greater chunk of the reward, the miner faces a major issue with time. If the same/similar block is mined with greater majority and much faster by another miner, then that miner's management entity broadcasts a blocking message and arrests the functioning of other similar quorums.

The miner and the quorum who have been successful receive the rewards, whilst all the quorums that were halted do not. This implies that it is not always true that verifiers and miners are rewarded.

Miners have to trade-off between time for verification and reward chunk. This motivates miners to optimise their trees and seek out optimal quorums, quorums that are not too slow and quorums that do not take away significant amounts of the miner's chunk.

Verifiers on the other hand realise that if they belong to lower index quorums, then they will have to work slower and longer, and the reward they receive will be less. Therefore, verifiers are motivated to increase their computation power and move up the quorum indexes to receive greater rewards.

The miners and the verifiers are driven by different goals, each of them motivated by opposite factors. This opposite driving force

ensures that miners and verifiers are always competitive and do not collude with each other.

To further keep the competition healthy and intact, two global indexes are computed and publicly displayed. The first index is called the **Miner Power Index (MPI)** and the second index is called the **Verifier Power Index (VPI).**

The Miner Power Index (MPI) highlights the average miner's computation power to the verifiers and other miners. The verifiers can use this MPI to identify if miners are scaling, and scale accordingly to compete with them. The miners can identify if their computation power is lagging w.r.t the average and their peers, and appropriately scale up.

Likewise, the Verifier Power Index (VPI) highlights the average verifier's computation power to the miners and other verifiers. Miners can identify if verifiers are trying to scale up for greater rewards, to scale up themselves. Similarly, other verifiers can look at their current computation index and the average, and decide if they wish to scale further.

One important point to keep in mind is that the VPI can never exceed that of the MPI. Whilst we are minimising the dominance of the miners in this new protocol, their role is important. Verifiers cannot exceed the computation power of the miner, because then verifiers can cause centralization and monopoly in the verifier market. They can start squeezing out more rewards chunks from the miners, even when miners do the most work, which is unfair and disruptive.

Therefore, every verifier has a cap on the limit to which they can grow. This cap can increase when the miners scale up and decrease when the miners scale down. But for now, we will assume that miners do not necessarily scale down, and so the cap rises, but in accordance with the majority of the miners.

Verifiers will have to scale up as miners scale up, because the difficulty of the network will rise as miners get better and better. Similarly, miners will have to scale up when verifiers scale up, if they

wish to retain greater chunks of the rewards for themselves. Verifiers will have to scale down, if miners start preferring lower index quorums (in a way the miner is scaling down). Verifiers will not scale down on purpose, unless triggered by the miner's actions.

This constant tussle for growth will keep the two entities competitive, and prevent domination of a particular group of miners/verifiers in their respective domains.

Let us consider three index quorums k1, k2 and k3 such that k1>>k2>>k3 and identify their respective reward distributions.

Quorum Index	Verifier reward %
K1	80
K2	50
K3	20

Table 3: Quorum indexes and their distributions

K1 has the highest quorum index and therefore the quorum receives 80% of the total reward when a block is mined.

Let us consider the quorum size for each index as N and the total reward received as x BTC.

If, we now calculate the reward received by one verifier in every quorum, it is as follows

Quorum Index	Reward for every verifier	Reward for the miner
K1	0.8x/N	0.2x
K2	0.5x/N	0.5x
K3	0.2x/N	0.8x

Table 4: Rewards to miners and verifiers on successful block mining

The miner obviously receives more rewards than an individual verifier, but it is significantly smaller than what they received previously (which was x). N changes occasionally, so every verifier's reward is more of less the same (it does not dilute).

0.8x/N (K1 quorum reward) is 300% larger than 0.2x/N (K3 forum reward). Scaling up from K3 to K1 can significantly improve the rewards earned by a single verifier.

Does that mean the only motivation for verifiers is to keep growing/scaling up? What about those verifiers who cannot keep growing? Do they become obsolete?

No. As much as growth is essential for surviving in this dynamic market, there is another feature introduced in this protocol that rewards loyal, consistent verifiers.

Verifiers that are consistently performing their duty of verification at a fixed index without scaling, are provided rewards as a gratitude for their persistent efforts at keeping the network secure. These rewards are provided in greater chunks to those verifiers who belong to smaller index quorums, as they invest significant amount of time in doing so. So, whilst the reward they receive from the verifier chunk may be extremely little, the consistent reward is larger. This motivates them to continue their responsibility at the index at which they are situated.

It is true that the market conditions may force loyal, consistent verifiers to scale, if they wish to survive in the market. In such scenarios, the reward size shrinks, but the reward provision is guaranteed.

Where do these rewards come from?

Whatever BTC is lost by the verifiers who staked it when applying to a higher index quorum, are all obtained and pooled up together (this pool holds the BTC from all index quorums—it is a general pool). A global management entity maintains a reputation list of all the consistent performers, their quorum indexes and the consistency period. This table is used to decide which entities have to be given what chunk from the collected pool. Verifiers that have been

consistent for longer periods receive more rewards, and so do verifiers that belong to lower index quorums. This is an incentive for lower index quorums to continue operating; it is of little significance to higher index quorums who receive significant rewards anyway.

Whatever stake is collected from individual quorums on failure, is not used to distribute it within that quorum. It is collected from all quorums and pooled together and then distributed based on the reputation. This makes sense, because the higher index quorums have larger stakes, and distributing it to that quorum's verifiers will imply the rich get richer, which defeats the whole purpose of it.

Philanthropic miners can decide to provide a portion of their rewards for distribution to loyal verifiers. Miners cannot provide this reward to their quorum directly. They can submit this to the global management entity which will use this amount to distribute it to its consistent verifiers accordingly. The global management entity publishes the miners and their contributions, if the miners wish to disclose their identities. The global management entity uses different algorithms to decide how rewards have to be distributed. (top k entities get all rewards, everyone gets a little of the reward et.)

Philanthropic verifiers can also decide to provide some portion of their rewards as a gratitude to optimal miners. This again reaches the global management entity. This entity manages a list of how optimal a particular miner is, over the course of its mining activity. The optimal nature of a miner can be decided based on how many transactions their tree contains, how quickly their quorums finish, how difficult their basis vectors and dimension constraints are etc.

Such miners are rewarded again by the global management entity. Verifiers' identities are however not revealed openly, as this may spark some form of collusion amongst the verifiers themselves.

An important point to note is that when the miners/verifiers receive these rewards, it is done in a discrete fashion by the global management entity. What that implies is that verifiers/miners do not

know which entity has provided the reward, that they are receiving. They simply receive it.

Other participants in the network can decide to distribute their assets to optimal miners and consistent verifiers too, via the global management entity.

DISPARITY REDUCTION AND PREVENTION OF CENTRALIZATION

One of the primary flaws with today's traditional blockchain network is that wealth is being accumulated by a selected few i.e the miners. Miners are becoming richer with every block they mine whilst the other entities in the network are not. This is creating a disparity in the distribution of wealth/assets in the Blockchain network and enforcing centralization/monopoly of a few sophisticated miners.

The proposed protocol is capable of preventing centralization and reducing the disparity in the wealth distribution in the network. We will understand how.

Miners and verifiers both are being rewarded for their contributions. Verifiers have no restrictions for entry/exit into the market. The rewards of the verifiers may not be significantly huge, but they are being evenly distributed across multiple entities over long periods of time. Whilst miners still hold greater portions of wealth, this practice is not uniform. Verifiers could receive more wealth for a time if the miners are opting for larger forums unanimously, and then it may reverse back to miners being wealthy again, if they start opting for lower index forums. The rewards earned by a miner is not uniform like the traditional Blockchain network.

Likewise, verifiers are receiving consistent rewards over long periods of time. So, on a whole whilst verifiers may not receive wealth that is equivalent to miners in the short term, in the long run, the wealth will get distributed more evenly/proportionally that it did in the conventional blockchain. The same can be verified and validated through agent-based simulation, which will highlight how in the long run, more entities are wealthy in the given protocol, than they are in the conventional Blockchain protocol.

The rich do not keep getting richer in this protocol. A free and fair means if provided for every entity in the network to grow and receive rewards for their contribution. Strict punishments to verifiers and miners, ensures that malicious entities are removed and the assets they possessed are distributed in a fair way to loyal entities in the network. Another important aspect that will help in reduction of disparity in wealth distribution is decentralization.

If the network is highly decentralized, then it means no one is dominating the market/holding a monopoly. This will imply that wealth is not being concentrated to a selected few, everyone is competing to obtain the very rewards that are at stake.

So, that brings us to the question, is this protocol promoting decentralization? Is it inducting the most important factor into account, the concept of a free, fair and competitive market, to prove that disparity is minimised?

Yes. We will understand how this protocol enforces decentralization and introduces a new novel concept of investing in Computation power.

As suggested earlier, miners do not wish to indulge in larger index quorums because they will lose significant rewards. They do not wish to indulge in smaller index quorums because they risk getting any rewards, because of the time delay. This implies that miners will concentrate on finding optimal quorums where they are ready to sacrifice a certain reward to the verifiers in exchange for a certain speedup in the verification.

This implies that the number of verifiers who are in the very extreme ends of the computation spectrum are very less. Verifiers wish to align their computation powers with where the miners are ready for a compromise, so the number of verifiers that are in the mid or the tipping point are going to be significantly higher. This distribution is analogous to a Gaussian distribution, that looks something like this

NAME: **PAVAN R KASHYAP**

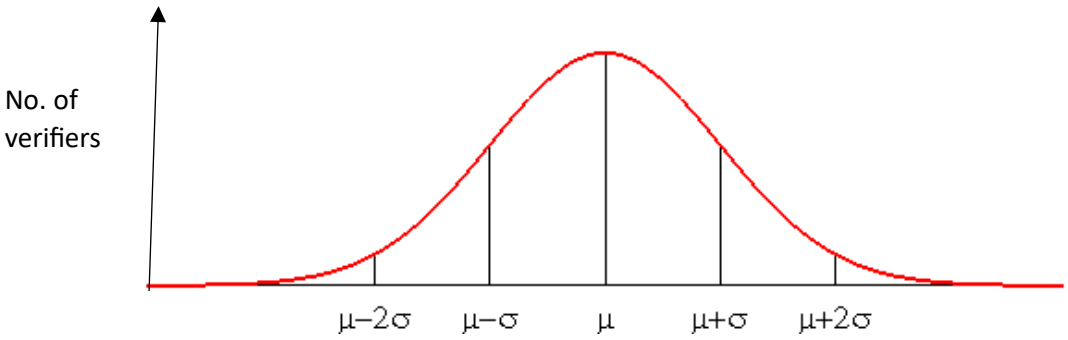

Figure 3 : Gaussian distribution of verifiers at different reward levels

In this Gaussian distribution, as can be seen, majority of the verifiers pool up at the mean of the distribution, which is in fact the optimal/the tipping point where miners are ready to indulge in quorums.

A distribution that is Gaussian indicates that verifiers exist in a spectrum and not in discrete, continuous sections. Verifiers that increase their computation power significantly do not necessarily dominate the verifier's domain because the miners do not prefer their quorums. The same holds true with verifiers that are at the lower ends of the spectrum.

Well, whilst that verifies the fact that lower and higher computation index quorums do not cause centralisation, does that also indicate that the behemoth crowd at the optimal/tipping point/mean is not going to cause centralization?

No, it does not cause centralization. There are a few important points that helps us solidify this claim.

If there is a large number of entities competing in a given space, then it obviously indicates a free and fair market. Competition sparks decentralisation. All those quorums with indexes at the tipping point compete fiercely to ensure that their quorums mine the block first and place it in the chain. Where there is fierce competition, there is no case of centralization.

Similarly, we will now draw some analogies to demand and supply, and understand what happens to the Gaussian curve on certain changes.

When verifiers on the lower ends of the curve scale up in bulk, it increases the concentration of verifiers at the tipping point, thereby making the centre of the curve section bulkier. When there is a significant number of verifiers in that index, miners will realise that the competition at this index is significantly large, and if they have to win over their competitors, they will have to risk some of their rewards and jump to a higher index quorum. This will cause the miners to prioritize verifier quorums that are higher than that of the mean, indirectly indicating that the miners identify a new mean to operate in. The mass scaling of the verifiers towards the mean causes the mean to shift to a new position, thereby generating a new Gaussian curve with a new mean. Those verifiers who are able to scale to the new mean enjoy the benefits of it, until a new mass scaling action comes along.

One point we must note is that the two extreme ends of the Gaussian curve are always fixed. The lower end is at 0BTC and the higher end is at k BTC, which is the overall reward obtained for mining a block.

When verifiers at higher index quorums decide to scale down i.e participate in lower index quorums in bulk, then it again causes the Gaussian curve to bulge. Miners understand that there are too many quorums and verifiers in the current mean space. They believe that they are now compromising on large amounts of their rewards to a large number of people. To minimise this effect, miners start preferring lower index quorums at the cost of time, for greater rewards. This causes the curve to shift to the left, reversing the bulk scale up consequences.

Similarly, miners may believe that the competition in the given index space is huge and if they wish to best their competitors, they must choose higher index quorums at the cost of providing more rewards. When they call out for higher index quorums, because of the scale down, the quorum sizes remain unfilled. There is a demand now for

higher index quorums. Verifiers scale back up to meet this new demand, thereby shifting the curve rightwards.

As observed, as verifiers scale up or scale down, the nature of the curve remains the same. This indicates that on scaling up or scaling down, the curve structure remains the same, and therefore highlights the point that decentralization exists at all times.

When miners decide to scale up, the overall difficulty of the network increases. This causes miners to prefer higher index quorums for faster computation and greater accuracy in results. When verifiers identify that certain quorums are being preferred over others, they start scaling up too, thereby causing the mean to shift rightwards to a new point.

When miners decide to scale down (which is not a normal scenario in conventional blockchain), they do so primarily to avoid losing significant portions of the reward to verifiers (that are currently at the mean). They decide to select lower index quorums for verification. Verifiers also choose to scale down to follow the miner trend, and move towards a new mean.

Again, we understand that activities caused by the miner still result in a Gaussian curve, thereby validating our claim that this proposed approach is decentralised w.r.t verifiers no matter what parameters change.

Is it decentralised for miners too then?

Yes. The primary driving agents that ensure that miners are decentralized are the network constraints on the dimensions, the difficulty of the basis vectors and the nature of the optimal solutions employed.

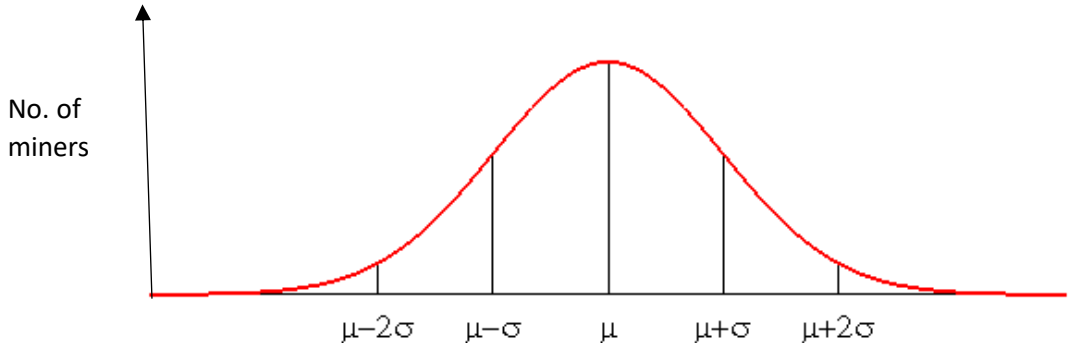

Fig 4: Gaussian distribution of the no. of miners at different optimization levels

The Gaussian distribution highlights a case of miner distribution after a certain period of time since the inception. Certain miners are able to obtain and learn innovative optimization solutions and so therefore generate better trees i.e trees that get approved by quorums faster.

Certain miners continue to remain inefficient and use minimalistic optimization techniques. Such miners run the risk of never getting their tree approved by lower index quorums or losing very significant amounts of reward to the higher index quorums. That motivates miners to improve their optimization ability.

Miners that are have exceptional optimization abilities face greater difficulty in terms of network constraints and basis difficulty. Whilst they can generate shorter trees, they must now focus on generating more accurate trees, as the basis vectors start approaching closer.

This highlights how again, miners are decentralised and they exist in a spectrum, where some are exceptionally non-optimal and face the risk of losing everything, whilst some are exceptionally optimal and again face the risk of losing everything (tree generation is exceptionally difficult, can face penalties for faulty SVps). It also highlights how it is optimization is not a linear quantity that always has to be increased. Striking the right balance between optimization ad difficulty will ensure that again miners do not centralize or gang up at a certain optimization level/difficulty level.

NAME: **PAVAN R KASHYAP**

INVESTMENT IN COMPUTATION POWER

We understand that the future is increased computation power. The day quantum computers become commonplace, the race for even better and faster computers will begin. The race for better computation power is never going to end. It will become a very important asset to possess in the future, just like gold and stocks today.

This paper proposes the concept of investing in computation power as a future asset. The free market for verifiers encourages entities to readily participate and invest in computation power by becoming a verifier. Once a verifier, a verifier receives consistent and long-term rewards for verifying and adapting to the trends in the network.

Analogous to stocks, when the price of the stock rises, many individuals buy it in the hope that they will get greater returns. Similarly, the desire for greater rewards will motivate verifiers to buy computation power/ invest in it or scale up, so they move to higher index quorums.

Analogous to stocks, when the price of a stock falls, many individuals start selling it in the hope that they do not undergo a loss. Similarly, when miners start preferring lower index quorums, verifiers will scale down or sell their computation power to minimise the losses they incur due to electricity charges etc.

This implies that the aim of investing in computation power is not to always keep linearly growing. Possession of significant amounts of computation power does not automatically guarantee significant rewards. Investing in computation power is not linear, it is Gaussian just like we have observed previously.

Those investors who are on the extreme ends of the distribution, no computation power and max. computation power (must still be less than Miner Power Index) will get kicked out of the free market, because it will become impractical for either of them to survive.

Those investors who are able to assess the miners' and verifiers' trends and always scale up/scale down accordingly to stay near the

tipping point/mean/equilibrium will always be guaranteed returns. The equilibrium point is defined as that point where the miners are ready to compromise their rewards and indulge in quorums, and that point that verifiers are capable of scaling and reaching for higher rewards.

This is analogous to stock brokers speculating the market and identifying which stocks to invest in/divest in.

As discussed previously, the Gaussian distribution keeps changing/fluctuating. What is the optimal computation power today may not be the same tomorrow or two days down the line. This implies that verifiers will have to dynamically adjust their computation power (buy or scale up/ sell or scale down) if they wish to stay relevant in the competitive market.

We must realise that when we mean computation power, we cover a vast bracket of innovation schemes employed by different companies. Scaling up using computation power of Company X may not be analogous to scaling up using computation power of Company Y. This implies that verifiers will have to carefully decide which companies provide it with the maximum benefit and choose investments there.

This investment will not only make the verifiers competitive, but it will also make the companies providing this computation power more competitive.

This new concept has great potential for growth and application in many fields. One primary field i.e blockchain is highlighted in this paper.

NAME: **PAVAN R KASHYAP**

CONCLUSION

In conclusion, the proposed protocol presents a promising solution to address centralization and wealth disparity in blockchain networks. By introducing verifiers and a Gaussian distribution for reward allocation, it fosters a competitive and decentralized environment. Verifiers' investment in computation power and scaling based on market trends incentivizes fair participation and rewards, ensuring long-term stability. The protocol's adaptability and reputation-based system encourage consistency and prevent dominance by select entities. Overall, this innovative approach paves the way for a dynamic, decentralized, and sustainable blockchain ecosystem, fostering collaboration between miners and verifiers while promoting a free, fair, and equitable market for all participants.

NAME: **PAVAN R KASHYAP**