The Meshwork Ledger, its Consensus and Reward Mechanisms

Mayank Raikwar*, Danilo Gligoroski*
*Norwegian University of Science and Technology (NTNU) Trondheim, Norway
Email: {mayank.raikwar,danilog}@ntnu.no

***Abstract*—We propose a new blockchain ledger concept called "Meshwork ledger" and a corresponding consensus algorithm. Meshwork is a network of coequal client nodes that contribute to the endorsement of the transactions by providing digital signatures to a validator node that collects them in an aggregate signature scheme. The essential sustainability component of the ledger is the reward mechanism for the meshwork client nodes. The prime design objective is the coequality of all client nodes, meaning there is no advantage for getting rewards if the client is an early adopter, if the client has collected a significant stake of rewards or if the client just joined the meshwork.**

The consensus algorithm is based on aggregate multi-signatures. A joint aggregate signature on a block of transactions is constructed with the signatures collected from the mesh client nodes in a multi-signature scheme. A signature provided on the block by a client node is acknowledged as approval. The core idea of the consensus is to race for the maximum number of signatures (approvals) on a block from the mesh clients, in order to append the block in the blockchain. The race in the consensus is among particular types of nodes called validator nodes that try to collect a maximum number of approvals (signatures) from the mesh clients. Clients that participated in the aggregate signature organized by the winner validator node get some reward as a small share of the total transaction fee. These reward transactions are performed in an off-chain manner, using a commit-chain.

Compared with other blockchain consensus algorithms, the meshwork consensus algorithm is faster, significantly more energy-efficient and scalable.

I. Introduction

Blockchain technology has evolved rapidly in the past decade. As a paradigm, it offers many unique features such as a distributed and trusted ledger of transactions without any need for a trusted third party, immutability of the ledger, pseudo-anonymous and anonymous transactions, smart and secure contracts, to name a few. The peers participating in the blockchain maintain the ledger, and each peer has a local copy of the ledger. These peers collectively adhere to some predetermined set of rules to ensure the consistency of the ledger. The mechanism to determine this set of rules is the core of the blockchain and is known as *consensus mechanism*. The first consensus mechanism proposed in the use of blockchain is Proof of Work (PoW) in Bitcoin [1].

Many consensus mechanisms have been introduced after Bitcoin PoW with their unique functionality. For example, there are numerous mechanisms that use different and complex compositions of cryptographic hash functions: QuarkCoin [2] using a chain of six hash functions, DASH [3] with eleven hash functions denoted as X11, and Verge [4] with even 17 hash functions (X17). The motives for their proposals were to offer PoW consensus protocols that will be fair for all users of the blockchain (not just only to early adopters, nor just to the powerful hardware miners). A general property of those consensus protocols is that they suffer from huge energy and computational power waste.

However, we can say that the short (slightly more than one decade) history of blockchain and cryptocurrencies [5], is a history of failed attempts to construct a sustainable blockchain that will address the issue of fairness by preventing the appearance of powerful hardware miners that can mine the blocks with hash computing rates that are several orders of magnitude higher than the ordinary users that would use just CPUs (and maybe GPUs).

The energy waste problem of PoW was addressed in several other consensus mechanisms such as Proof of Stake (PoS) [6], Proof of Stake Velocity (PoSV) [7], in mechanisms that use verifiable random functions such as in Algorand [8] and in the Secure Proof of Stake (SPoS) [9], in mechanisms that use trusted random functions such as in Proof of Luck (POL) [10], and in Proof of Elapsed Time (POET) [11]. Some of the proposed consensus protocols, to boost their energy efficiency, have also proposed the use of special nodes (master nodes in DASH [3] or validator nodes in Libra [12]).

Needless to say, the solutions addressing the energy problem of the original PoW consensus, did not offer solution for the fairness problem, since the owners of big stakes of coins would have significant advantage to get newly minted coins.

Since our Meshwork ledger heavily relies on aggregate multi-signatures, next, we describe a few works done on multi-signatures and their application to the blockchain. Boneh at el. [13] constructed a multi-signature scheme using BLS signatures, which provides public-key aggregation and security against rogue public-key attack. The scheme supports a fast verification of transactions and reduces the blockchain size. Algorand [8] presented a forward-secure multi-signature consensus scheme Pixel [14]. Pixel signatures reduce the storage, bandwidth, and verification costs in the blockchain. They integrated the Pixel signature with Algorand blockchain and showed a substantial reduction in the block size and the verification time. Another work of a multi-signature scheme in the blockchain is Decisional Diffie-Hellman based construction of multi-signatures [15].

The use of aggregate signature in consensus algorithms got

much attention from academia and industry. Theta blockchain ledger protocol [16] adapted the concept of aggregate signatures in the BFT consensus algorithm and introduced a multi-level BFT consensus mechanism. They proposed an aggregated signature gossip protocol to reduce the communication complexity of the consensus. However, we notice that Theta blockchain ledger might suffer from a high signature verification cost when the number of guardian nodes in the system increases as more number of pairing operations are needed. PCHAIN [17] also introduced PDBFT2.0 [18] to reduce the communication complexity and hardware requirements in blockchain consensus using the aggregate signatures, but it also suffers from high signature verification cost. The other related works in this domain are [19], [20], [21], [22], [23], [24]. Another interesting related work is the new Schnorr multi-signature scheme with deterministic signing [25].

A. Our Contribution

We propose a new permissioned public blockchain ledger concept called "Meshwork ledger" with two design goals:

1) Its consensus protocol is fair to all ledger members i.e.; there exists a coequality for all client nodes: there is no advantage for getting rewards if the client is an early adopter, if the client has collected a significant stake of rewards, or if the client has just joined the meshwork;
2) Its consensus protocol is energy efficient, and there is no advantage if the client has powerful hardware or a single CPU or MCU just capable to produce digital signatures.

The sustainability of the "Meshwork ledger" is guaranteed by its reward mechanism, which is also designed to be fair to all meshwork client nodes. To achieve the design goals for the consensus algorithm, we use aggregate multi-signatures. The idea is to construct a joint aggregate signature on a block of transactions with the signatures collected from the mesh client nodes in a multi-signature scheme. One single signature on the block of transactions produced by a single client node is acknowledged as approval. Then, the main novel idea in our consensus proposal is to race for the maximum number of signatures (approvals) on a block of transactions from the mesh clients. The race in the consensus is among particular types of nodes called validator nodes that try to collect a maximum number of approvals (signatures) from the mesh clients. Clients that participated in the winning aggregate signature get some reward as a small and equal share of the total transaction fee. These reward transactions are nano-valued transaction. As there are many of these transactions, the transactions can be performed in off-chain manner. We reviewed all the possible off-chain transaction solutions and concluded that the best way to perform all these nano-value transaction for our model in a cost-effective manner is commit-chain [26].

Last but not least, we also provide complexity and security analysis of our consensus protocol. In the security analysis, we enlighten possible attack scenarios in the protocol and the prevention mechanism followed in the consensus protocol. We also conducted a few experiments for the robustness of the system.

II. Preliminaries

1) Bilinear Maps: $\mathbb{G}_1$ and $\mathbb{G}_2$ are two multiplicative groups of prime order q with generators g_1, g_2 correspondingly. A map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ has the following properties:

- Bilinearity: $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p : e(u^a, v^b) = e(u, v)^{ab}$;
- Non-degeneracy: $e(g_1, g_2) \neq 1_{\mathbb{G}_t}$.

We say the pair $(\mathbb{G}_1, \mathbb{G}_2)$ is a pair of bilinear groups iff the group operations in $\mathbb{G}_1$ and $\mathbb{G}_2$, and the bilinear map e are efficiently computable.

2) Multi-Signature Scheme: For individual transaction/block signing and verification we use the BLS signature scheme [27], and for the multi-signature scheme, we use the aggregate signature scheme proposed by Boneh et al [28]. In our case, all the signers sign the same message in the aggregate signature scheme while ensuring security. BLS signature scheme requires bilinear map e described as above alongwith a full-domain hash function for signing process $H : \{0,1\}^* \to \mathbb{G}_1$. BLS signature scheme works as follows:

- **KeyGen:** For a user, choose random $sk \xleftarrow{\$} \mathbb{Z}_q$, compute $pk \leftarrow g_2^{sk} \in \mathbb{G}_2$, the user's keypair is (pk, sk).
- **Sign**(sk, M)**:** For a user, given secret key sk and a message $M \in \{0,1\}^*$, signature on M is $\sigma \leftarrow H(M)^{sk} \in \mathbb{G}_1$.
- **Verify**(pk, M, σ)**:** Given a user's public key pk, a message M, accept the signature σ, if $e(\sigma, g_2) = e(H(M), pk)$.

Signature Aggregation: Given n signatures $\sigma_1, \sigma_1, \ldots, \sigma_n$ on message M by n users, the procedure for signature aggregation of n signatures works as: $\sigma \leftarrow \prod_{i=1}^{n} \sigma_i$. The aggregate signature is $\sigma \in \mathbb{G}_1$.

Aggregate Verification: To verify the aggregate signature σ, given the original message M and the n public keys $pk_1, pk_2, \ldots, pk_n$ for all n users, the verifier checks if:

$$e(\sigma, g_2) \stackrel{?}{=} e(H(M), pk_1)e(H(M), pk_2)\ldots e(H(M), pk_n)$$
$$\stackrel{?}{=} e(H(M), \prod_{i=1}^{n} pk_i) \stackrel{?}{=} e(H(M), apk)$$

If the equation holds, the verifier "Accept" the signature, else "Reject". In the above equation, $apk \in \mathbb{G}_2$ and stands for *aggregate public key*.

This simple aggregation scheme suffers from **rogue public-key attack** where an attacker takes the public key pk of an honest user Alice, and constructs his public key pk^* as $g_2^{\alpha} \cdot (pk)^{-1}$ (where $\alpha \xleftarrow{\$} \mathbb{Z}_q$). Then, given a message $M \in \{0,1\}^*$, attacker presents an aggregate signature $\sigma := H(M)^{\alpha} \in \mathbb{G}_1$ claiming that he and Alice, both has signed the message M. However in reality, Alice did not sign the message M but the aggregate verification holds as

$$e(\sigma, g_2) = e(H(M)^{\alpha}, g_2) = e(H(M), g_2^{\alpha})$$
$$= e(H(M), pk \cdot pk^*)$$

Few defense mechanisms [29], [30] against this attack were proposed which require each user to prove the possession of the corresponding secret key (PoP). We also apply the PoP in Meshwork ledger for each party.

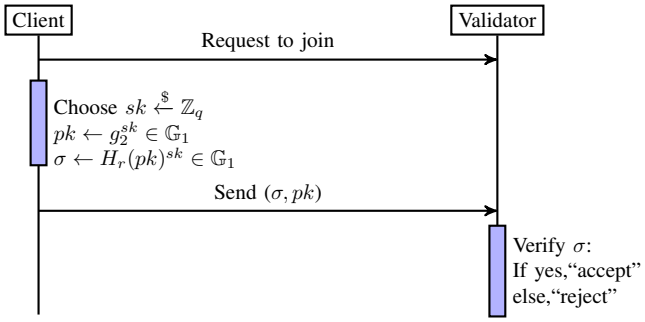

Fig. 1. Client Registration Protocol

Why BLS Multi-Signature

- BLS Multi-signatures are non-interactive and easy to follow in nature, therefore, it is easier to perform the signature and key aggregation without any additional round.
- BLS signatures do not rely on random number generator and BLS signatures are single curve points, so its size is two times shorter than Schnorr or ECDSA signature [31].

III. Meshwork Ledger

A. Entities in the Ledger

In Meshwork ledger model, we have two primary entities client and validator node participating in the blockchain:

1) Client Node: Client nodes are the ones that invest a tiny computational power to sign a block, and for that gets rewarded. A client node can have a few connections with different validator nodes. For registration, client nodes have to go through strong authentication checks to prevent the Sybil attack [32], followed by key registration protocol using PoP with a validator node to prevent the rogue-key attack as depicted in Figure 1. Client nodes also execute transactions with other client and validator nodes in the system.

Client Registration Protocol It is an interactive protocol between a client node and a validator node as depicted in Figure 1. In this process, a client node registers itself by proving the knowledge of its secret key to a validator node by signing its public key. To completely prevent rogue public-key attack, the registration protocol uses a different hash function $H_r : \{0,1\}^* \rightarrow \mathbb{G}_2$ for creating signatures over the public key. This hash function can be constructed from hash function H using domain separation. Thus, the client generates a signature using H_r on its public key, and if verified, the validator stores the user's public key. The client node also has a public identifier associated with it, which is a hash of its public key. The specific hash function is global for all client nodes. Thus, rather than sending long public keys, these short identifiers are used for communication.

Note: The Client Registration Protocol is an important component for the overall security of the Meshwork ledger, but it is not an exclusive and non-replaceable component. Namely, our meshwork can use some pre-existing client registration protocols based on a national digital identity [33].

2) Validator Node: Validator nodes are the major players for reaching the consensus in the distributed ledger. Each validator node maintains its ledger of blocks. Validator nodes join the system according to the objective participation criteria, and these nodes have a stake in bootstrapping the blockchain system. Therefore, these nodes have to lock up some minimum amount of stake in the system for a period of time. Each validator node has a pair of public-private keys. The validator nodes publish their public key along with the Proof of Possession (PoP) of the corresponding secret key. Here, the PoP scheme is similar to the PoP used in Figure 1. Validator nodes can also perform transactions in the network. In a consensus round, validator nodes perform signature aggregation on the block, thus validator nodes invest resources towards achieving the consensus. Accordingly, each validator node should be equipped with enough computational power and storage space. There are fewer validator nodes than the client nodes, and each validator node maintains several client node connections. A validator node also has its publicly available list of connected client nodes along with their corresponding public keys and public Identifiers.

Validator Registration Protocol: To join the pool of validator nodes in the system, a new node has to pass a strong authentication check and also give proof about having enough stake and enough storage and computational power.

B. Assumptions in the Meshwork ledger

- The network is partially synchronous, which means the message transmission between two directly linked nodes arrives within a specified period.
- The majority of the validator nodes are honest for the security against byzantine fault and the local clock of each validator node is loosely synchronized.
- The blockchain should be account-based and smart-contract enabled e.g., Ethereum [34].

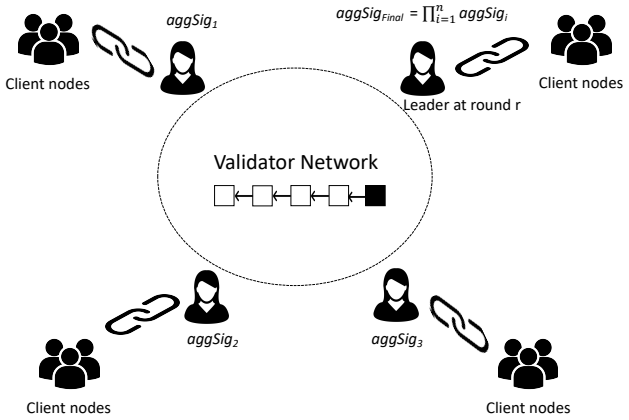

Fig. 2. Overview of Consensus at round r

Bootstrapping the System During the deployment of the system, a common genesis block is given to all the validator nodes. This genesis block specifies a number s denoting the minimum number of signatures required from each validator node during a consensus round. The value s is updated from time to time, depending on several factors including the number of participating entities in the system.

C. Consensus Algorithm

The consensus algorithm involves the active participation of validator nodes as well as client nodes of blockchain. The core idea of the algorithm is to collect the maximum number of signatures on a block from the nodes participating in consensus round r as depicted in Figure 2. The taxonomy of the symbols used in Meshwork ledger is listed in Table I. There are n validator nodes $V_1, V_2, \ldots, V_n$ and each validator node V_i has some client nodes $C_{i1}, C_{i2}, \ldots, C_{ij}$ (where $j \gg n$) connected to it. The detailed algorithm is as follows:

Symbols	Definition
$\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$	Multiplicative groups of order q
g_1, g_2	Generator of group $\mathbb{G}_1/\mathbb{G}_2$ respectively
e	Bilinear map
(PK, SK)	Public and secret key
H, H_r	Hash functions for signature and client key registration
V	Validator node
C	Client node
σ	A BLS signature
s	Number of required signatures from a validator node
$aggSig$	Aggregate signature
$aggPK$	Aggregate public key
B_k	k^{th} Block in the blockchain
τ_{block}	Time to wait to receive the new proposed block
τ_{agg}	Time to wait to receive all the aggregate signatures
$Tolerance$	Upper bound how many times a client node can send the same signature to multiple validator nodes

TABLE I
LIST OF SYMBOLS IN THE MODEL

- In a consensus round r, a validator node V_k (where $k \in [1, n]$) is elected as the leader of the round. The leader node (*block proposer*) collects different transactions, writes the recent value of s and prepares a block B_k. After the block preparation, block proposer V_k sends the block B_k to all validator nodes. Leader V_k is excluded from the task of contacting its client nodes in round r.
- After receiving the block B_k, each validator node V_i checks the minimum number of signatures s required for the block B_k from its side. The validator node V_i randomly selects a client node set $\{C_{im}\}$ (where $m \leq j$ but $m \geq s$). The validator node signs the block and creates a signature σ_{V_i}. The main goal of each validator node V_i is to collect at least s signatures from its connected client nodes.
- Each validator node V_i collects the signatures $\sigma_{i1}, \sigma_{i2}, \ldots, \sigma_{im}$ on block B_k from its selected client nodes $C_{i1}, C_{i2}, \ldots, C_{im}$. Then V_i verifies all the individual signatures $\sigma_{i1}, \sigma_{i2}, \ldots, \sigma_{im}$, and further creates aggregate signature $aggSig_i$ from the verified signatures including its own signature σ_{V_i}, and aggregate public key $aggPK_i$ from $PK_{i1}, PK_{i2}, \ldots, PK_{im}$ and PK_{V_i}. Then V_i sends $aggSig_i$, $aggPK_i$ and a list of client public key Identifiers L_i $(ID_{i1}, ID_{i2}, \ldots, ID_{im})$ to the block proposer V_k.
- Each validator node V_i also gives a *Proof of Inclusion* $P_{C_{il}}$ to each of its client nodes C_{il}. This proof corresponds that the client signature σ_{il} (for $l = 1 \ldots m$) has been included in the aggregated signature $aggSig_i$. This proof is a signature over the signature σ_{il} which is $P_{C_{il}} = Sign(sk_{V_i}, \sigma_{il})$ and verifiable using the public key pk_{V_i} of V_i.
- The leader V_k collects the received aggregate signatures $aggSig_1, aggSig_2, \ldots, aggSig_n$, along with aggregate public keys and identifiers from all the validator nodes in a *First come, First served* basis. In the beginning, the leader resets a variable $TotalAggSig$ that keeps track of the number of unique aggregated signatures for that round. The leader also checks the following conditions:
  1) The received aggregate signatures $aggSig_i$ (for $i = 1 \ldots n$, $i \neq k$) are valid.
  2) For each validator node V_i, check whether the total number of public key identifiers is $s_i \geq s$. If so, it updates the variable $TotalAggSig$:
  $$TotalAggSig \leftarrow TotalAggSig + Unique(s_i)$$
  where $Unique(s_i) \subseteq s_i$ is a subset of s_i that have not submitted its signatures to multiple validator nodes.
- The leader is also keeping track of how many aggregate signatures it collected so far:
  1) There should be at least $\frac{2}{3}$ of $(n-1)s$ unique signatures in all the aggregate signatures received by V_k i.e.,
  $$TotalAggSig \geq \tfrac{2}{3}(n-1)s$$
  2) Determine a list $\mathcal{D}$ of clients that submitted two or more signatures to two or more validator nodes.
- If all the above conditions satisfy, the leader V_k constructs final aggregate signature $aggSig$ from signatures $aggSig_1, aggSig_2, \ldots, aggSig_n$ and final aggregate public key $aggPK$ from public keys $aggPK_1, aggPK_2, \ldots, aggPK_n$ (where $n \neq k$). Finally, V_k constructs the final block $B_{round} = (B_k, aggSig, aggPK)$ by appending the final aggregate signature $aggSig$, final aggregate public key $aggPK$ to the proposed block B_k.
- Leader V_k also gives a *Proof of Inclusion* P_i (where $P_i = Sign(sk_{V_k}, aggSig_i)$) of the aggregate signature $aggSig_i$ to node V_i which confirms that the $aggSig_i$ is included in the final aggregate signature $aggSig$.
- Leader V_k attaches the block B_{round} to its blockchain and broadcasts the block B_{round} in the blockchain network. It also sends the list $\mathcal{D}$ to all the validator nodes. After receiving the block B_{round}, each node V_i verifies the block by verifying the final signature $aggSig$ using $aggPK$. If the block verifies, the node V_i attaches the block B_{round} to its blockchain. Each validator node also checks the received list $\mathcal{D}$ and keeps a record for clients that sent the same signatures to multiple validator nodes.

D. Reward System

One of the primary goals of this consensus algorithm is to consume significantly less amount of computational power. The total reward in each consensus round is the sum of the transaction fees associated with the transactions of the block. Then the reward is dispersed among the nodes who participated in the consensus on that block. The validator nodes invest more

resources to get the signatures from their client nodes, and to aggregate the client signatures, so the validator nodes get more reward than the client nodes.

In each round, the leader node makes reward transactions to the validator nodes, which are recorded in the blockchain. These reward transactions are further distributed by each validator node to its client nodes that participated in the signing. Hence each validator node creates many client node transactions, and so the total client node transactions created by all the validator nodes are many in numbers.

Let say the total reward in a consensus round r is X, then the distribution of reward X will be as follows:

- Each validator node V_L and its client nodes $C_{L1}, \ldots, C_{Lm}$ get the accumulated share of reward as $j = \frac{X}{n}$
- This share j is further distributed among a validator node V_L and its client nodes $C_{L1}, \ldots, C_{Lm}$ in following way:
  1) Each validator node V_L gets share as $t * j$.
  2) Each client node C_{Li} gets share as $\frac{(1-t)*j}{m}$ (considering m client nodes connected to V_L participated in round r)

Here t is the reward distribution parameter.

Note: In the line of our design goal of the Meshwork ledger to be fair for early and late adopters that are meshwork clients as well as for the validator nodes, the incentives for validator nodes are also tweakable with the parameter t. This t is recalculated periodically and involves the energy and communication costs spend by the validator nodes. The process of including all client node reward transactions in the main blockchain is a severe bottleneck for the scalability. To tackle this, we adopted the off-chain solution, commit chain [26]. As the reward transactions for the client nodes are nano-value transactions, and the client nodes might not always be online, then to off-loading the transactions in off-chain, commit-chain fits as the viable option.

IV. Consensus in the Meshwork Ledger

Algorithm 1 illustrates the steps involved in the consensus. In this section, we describe all the consensus algorithm functions as $LeaderElection$, $CreateBlock$, $ClientSelection$, $Sign$ and $Verify$ in detail. It also defines other necessary factors of our Meshwork ledger.

A. Leader Election

In each consensus round, a validator node is chosen as a leader by the function $LeaderElection()$. The leader is responsible for the creation and finalization of the block. There are different mechanisms in different PoS blockchains for leader election. Many of the PoS blockchains [35] are using verifiable random function [36] for electing the leader. For our Meshwork ledger, we follow the leader election using an efficient robust round-robin selection technique [37] with some modification. In short, leader candidates are selected according to their age in a round-robin manner. After the candidates' selection, a set of endorsers give a quorum of confirmations for the leader candidates. The one having the majority of confirmation becomes the leader node. In our model, we follow the same idea with some modifications to prevent the malicious leader or leader crash issue. We define a threshold value t_E for the number of endorsements and we select an expected number of leader candidates based on t_E. The value t_E varies and computed for each round. Hence, in our model, we have three sets of validator nodes after the leader election process:

1) *Leader validator node* is the deterministic leader candidate resulted from the robust round-robin technique.
2) *Backup validator nodes* are the validator nodes which passes the threshold criteria t_E of endorsements.
3) *Remaining validator nodes* are the nodes which are either not selected in the robust round-robin technique or did not pass the threshold t_E .

The leader validator node defined as V_k in Section III-C is responsible for proposing a block and collecting the required number of signatures from other validator nodes. Backup validator nodes $\{V_b\}$ and remaining validator nodes participate in consensus to get the aggregate signature on the proposed block by the main leader node. Backup validator nodes are also responsible for the following things:

- If the leader node crashes or if backup validator nodes do not receive any new block within τ_{block} time, then a backup validator node having the next highest quorum of

Algorithm 1: Consensus Algorithm

Input : $round, s, \{V_1, V_2, \ldots, V_n\}$
Output: B_{round}, $\mathcal{D}$

$V_k \leftarrow LeaderElection(round, V_1, V_2, \ldots, V_n)$;
V_k runs $CreateBlock(tx_1, tx_2, \ldots, tx_u)$ and output B_k;
V_k sends block B_k to other validator nodes and waits for τ_{agg} time to receive the required number of signatures;
V_i actions::;
for *each validator node* V_i *from* $V_1, V_2, \ldots, V_n$*, except* V_k **do**
    run $ClientSelection(C_{i1}, C_{i2}, \ldots, C_{im})$;
    wait for τ_{block} time to receive new block B_k;
    if *receive* B_k **then**
        $(aggSig_i, aggPK_i, \{ID_j\}_i) \leftarrow Sign(B_k, s)$;
        send $(aggSig_i, aggPK_i, \{ID_j\}_i)$ to node V_k;
    end
end
V_k actions::;
$TotalAggSig \leftarrow 0$;
for *tuples* $(aggSig_i, aggPK_i, \{ID_j\}_i)$ *received from the validator nodes* V_i **do**
    · $Verify(aggSig_i, aggPK_i, B_k)$;
    · Check if $s_i \geq s$, where s_i is the number of client identifiers from V_i;
    · Determine the number of unique signatures coming from V_i, $Unique(s_i)$;
    · $TotalAggSig \leftarrow TotalAggSig + Unique(s_i)$;
    if $TotalAggSig \geq \frac{2}{3}(n-1)s$ **then**
        exit for;
    end
end
V_k prepares the final block $B_{round} = (B_k, aggSig, aggPK)$, appends it to its blockchain and sends to the other validator nodes;
V_k also sends list of double signees $\mathcal{D}$ to all V_i;

confirmations becomes the leader, announces itself as a leader, and executes the consensus protocol.
- Backup validator nodes continuously monitor the behavior of the leader node, and the leader is caught if it performs malicious activities. For example, if a leader node is malicious, it can give different blocks to different validator nodes but it will be caught by the backup validator nodes and later it will be penalized in the system.

B. Block Proposal and Client Selection

The leader of the consensus round collects the transactions from the client and validator nodes and prepares the block using $CreateBlock()$ function. The leader also appends value s, the minimum number of signatures required from each validator node, in the block. This current s should be the latest value for the next few consensus rounds. The other validator nodes wait for a definite amount of time for the new block to receive. A validator node might not receive the latest block in a specific amount of time due to some network-related problems like network congestion, etc.

After receiving the new block from the leader node, each validator node runs $ClientSelection()$ function. The strategy under this is to select at least s client nodes to sign the block is *Round Robin With a Reset* strategy. That means every validator node keeps track of how many rounds the clients were waiting to sign some block. Validator picks randomly s out of those clients that had the highest waiting time. Once a client is chosen to sign, its waiting time is reset to 0.

C. Block Signing and Verification

The validator nodes announce the new consensus round to its connected client nodes. The selected client nodes by a validator node sign the block and send it to the validator node. A validator node waits for a certain amount of time to receive at least the minimum number of required signatures from its client nodes. Then finally, the validator node verifies all the client signatures and prepares an aggregate signature from those, along with an aggregate public key using $Sign()$ function and sends the aggregate signature and key, along with the client node public identifiers to the leader node.

The leader node receives the different aggregate signatures along with other details from different validator nodes. The leader node first verifies all the aggregate signatures using the function $Verify()$. Further, the leader node checks whether these aggregate signatures qualify the criteria for the minimum number of required signatures for aggregation. If all the verification conditions meet, then the leader prepares the final block by appending other necessary details to the original block.

D. Race Conditions

The race in the Meshwork ledger is among validator nodes that are racing to collect at least s signatures and to be included in the leader signature aggregation that tries to collect at least $\frac{2}{3}(n-1)s$ unique signatures. We find that in comparison with the classical PoW consensus races, our race conditions have significantly less consumed energy.

The majority of 2/3 is a tweakable parameter and can be increased to 3/4 or some other value, but we do not recommend it to be less than 2/3.

E. Safety and Liveness of Consensus

In every consensus algorithm, safety and liveness are essential factors. These things majorly depend on the network synchronicity and the number of honest participants in the consensus. Particularly for our consensus:

- *Safety* in the blockchain context, ensures that the honest participants in consensus should work on the same blockchain. Hence, safety considers the past and take actions based on history. That means if an honest participant accepts a new block in its blockchain, then in the future, this block will always be in the blockchain of other users. In Meshwork ledger, a block will be in the blockchain if it gets on an average at least $\frac{2}{3}s$ signatures from each of the validator (Including its client nodes' signatures) in the blockchain. This argument implicitly points out that there must be at least $\frac{2}{3}$ honest participants during the consensus round in the model and hence, ensures safety in the model.
- *Liveness* ensures that the major participants will be in charge of keeping the system alive; hence it considers the future. That means the validator nodes will always make progress in the blockchain. From the duty of a leader node or backup validator nodes, a new block will always be created and added to the blockchain in a consensus round when at least $\frac{2}{3}$ participants in the round are honest.

F. Coequality of mesh clients

The coequality of every mesh client is ensured by the procedure $ClientSelection(C_{i1}, C_{i2}, \ldots, C_{im})$ which is invoked at line 6 of the algorithm. The core property of this function is that it selects at least s client nodes to sign the block in a *Round Robin With Reset* manner. In such a way, every mesh client that was once selected to contribute in an aggregate signature (and possibly get a reward) will have to wait for several rounds until other mesh clients from that validator node also get its fair share of contribution.

G. Parameters in the Meshwork Ledger

There are a few parameters in Meshwork ledger. Signature and timeout parameters play a vital role in reaching consensus and achieve safety and liveness properties. Details of these parameters are as follows:

- *Signature parameter "s":* The parameter s representing the minimum number of signatures required from each validator is updated from time to time (e.g., Weekly or monthly). This update also depends on the density of the network. If the number of participants (validator and client nodes) increases/decreases in a considerable amount, the parameter s is updated accordingly in a quick manner.
- *Timeout parameters:* In our consensus, we have a few timeout parameters. The parameter τ_{block} defines the time to wait for the proposed block to reach to a validator node in a consensus round. Another timeout parameter τ_{agg} establishes the time to expect by the leader node in a consensus

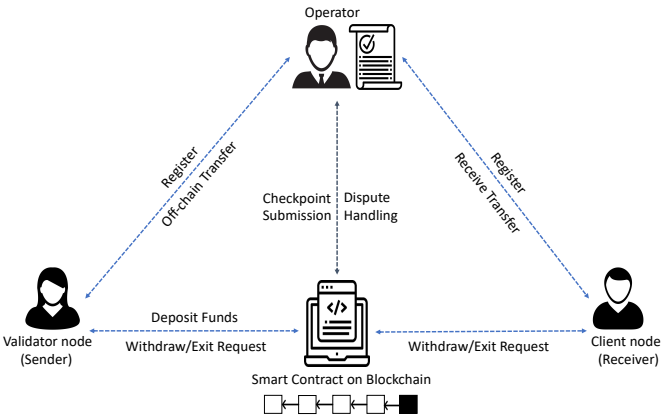

Fig. 3. Overview of Reward Mechanism through Commit-Chain

round to receive all the aggregate signatures from the other validator nodes. Both parameters should be reasonably and carefully decided using the Poisson distribution to assure the liveness of the system.

- *Reward distribution parameter "t":* It is decided in each consensus round based on the average number of client nodes connected to the validator nodes. In general, $0 > t \leq 0.2$. The leader node includes t along with the signature parameter s in the block proposal. Therefore, validator and client nodes can locally estimate the share they might receive after the successful consensus round.

V. Reward Mechanism using Commit Chain

Compared to traditional PoS models, our system has many nano-value reward transactions. Thus, the reward should be distributed cost-effectively, which should incur almost zero transaction fees. Hence we adopted a commit-chain distribution for the reward transfer. The reward transfer mechanism performed by commit-chain NOCUST requires an operator. The execution of the commit-chain protocol is performed in rounds, which are called eons. Each eon of the commit-chain will have many consensus rounds of the main blockchain protocol. Therefore, after completing each eon, the nodes participated in the consensus rounds within that eon will receive the sum of the rewards earned in those consensus rounds. All these nano-value reward transactions can be made zero-fee transactions reliably depending on the operator fee schedule. The correct execution of these transactions is enforced by the smart contracts of the main blockchain, but the transactions are performed on the commit chain. Following is the overview of the reward transaction mechanism using commit-chain:

Register All validator and client nodes create an account with the commit-chain operator via off-chain messages, hence register themselves to perform transactions on commit-chain.

Deposit After a consensus round, each validator node locks the amount of reward transaction (Originated from the leader validator node to other validator nodes) in the commit-chain.

Transfer To distribute the reward money to the client nodes, each validator node authorizes itself to the operator to debit its account and credit the client nodes' accounts.

Withdraw/Exit To withdraw the balance from the commit-chain or to exit the commit-chain, the validator or client nodes submit the off-chain request to the operator.

Checkpoint Constant size periodic checkpoints are used by the operator to commit the latest states of all the validator and client node accounts using a smart contract. This checkpoint is the root of the Merkle tree aggregating client and validator nodes state and balances. Each of the checkpoints requires an on-chain transaction.

Challenge/Response Challenge-response dispute mechanism is enforced by commit chain using the smart contract in case of operator misbehave.

The above operations are depicted in Figure 3. Moreover, a single operator for the off-chain solution becomes a central point of failure. Depending on the number of nodes participating in the blockchain, a few commit chains (less than the number of validator nodes) or watchtowers can be deployed to remove the single point of failure, but that will incur an extra cost. Hence in our model, the fairness of the reward mechanism depends on the fairness of the used commit-chain.

VI. Complexity Analysis of the Consensus

Communication Complexity In every consensus protocol, many iterations of communication are required to reach the final consensus and append the block in the blockchain. In Meshwork ledger, each validator node has to send the new proposed block to its connected client nodes. Therefore $O(sn)$ number of messages will be transmitted to propagate the block in the system. The same will apply for receiving the aggregate signatures from the validator nodes. So in total, the communication complexity of the network will be $O(sn)$.

Computational Complexity The signature process requires computation to create the signature and to verify them. On average, each validator node receives s signatures and verify them using $2s$ pairing computations. Nevertheless, in total, the number of pairing computations will be $n*2s = O(sn)$. After receiving all the aggregate signatures, the leader has to perform $2n$ pairing computations for verification. Hence in total, the computational complexity in terms of pairing operations in a consensus round will be $O(sn)$.

VII. Security

A. Security Model:

1) Malicious Validator Node: If a validator node V_m acts maliciously and does not include its client node C_{mj}'s signature in the aggregate signature $aggSig_m$, the client node can raise this issue in the blockchain network using its Proof of Inclusion $P_{C_{mj}}$. Therefore, the verification of $P_{C_{mj}}$ is performed by the other validator nodes. If the proof $P_{C_{mj}}$ is verified and the client node signature (Can be checked by client node public identifier $ID_{C_{mj}}$) is not found in the aggregate signature, then the validator nodes agree on penalizing the validator node V_m.

2) Malicious Client Node: A client node C_m can act maliciously by submitting the same signature σ_{C_m} multiple times in a consensus round to earn more reward. However,

as in a consensus round, each validator node gets a list $\mathcal{D}$ from the leader specifying the public key identifiers of the client nodes that submitted signatures on the same block to validator nodes. The validator nodes at the final stage keep a clean local house by keeping a record for the number of double signatures incidents for its client nodes. If the number of incidents caused by a client node exceeds a $Tolerance$, the client node is permanently blocked in the system.

3) Security against Existential Forgery: In the Meshwork ledger, an adversary (a malicious validator) can try to forge a multi-signature on a block choosing some subset of its client nodes. The forgery of multi-signature can be reduced to the Computational Diffie-Hellman (CDH) Problem [28]. The forgery can be defined as; For n signers, an adversary must forge a multi-signature $\sigma \in \mathbb{G}_1$ on message M under the public keys $pk_1, \ldots, pk_n$. To see how forgery is equivalent to solving CDH problem, the following description justifies the point: Given randomly chosen $g, g^{sk_1}, h(= H(M))$, the adversary can randomly generate $(n-1)$ key pairs $(sk_2, pk_2) \ldots (sk_n, pk_n)$. Then adversary creates a multi-signature σ which should satisfy the the verification equation.

$$e(\sigma, g_2) = \prod_{i=1}^{n} e(h, pk_i) = \prod_{i=1}^{n} e(h^{sk_i}, g_2) = e(\prod_{i=1}^{n} h^{sk_i}, g_2)$$

which means $\sigma = \prod_{i=1}^{n} h^{sk_i}$,

From the above check for signature σ, if it passes that means adversary has computed h^{sk_1}, given randomly chosen g, g^{sk_1}, h which is equivalent to solving CDH problem.

B. Attacks

- Validator-Specific attacks: In many of the PoS based systems, the validator nodes try to collude and earn a bigger reward by increasing their total stake in the system. In our system, as the reward is equally distributed among the validator nodes, so in case of collusion, the validator nodes have to share the joint reward, and that will be less than the reward of any other validator node. Hence, any kind of collusion does not bring any advantage in terms of reward.
- Network-Level attacks: In case of a network partition (Eclipse attack), until the number of received aggregate signatures and client node identifiers satisfy the consensus rules to accept the block, the consensus is reached. The consensus will not be achieved in other scenarios of a network partition, and the system will go in recovery mode.
- Sybil attack: Sybil attack is prevented in the system using strong authentication checks for client and validator nodes.

VIII. Technical Details and Experiments

We conducted experiments to test the robustness and efficiency of the Meshwork ledger. The experiments were carried out on a MacBook Pro system having 2.3 GHz Intel Core i5 processor and 8 GB 2133 MHz memory. So far, we have designed a validator network which is connected with client nodes using Docker containers. Validator nodes perform the BLS signature aggregation. In our experiments, we analyze the cost of signature aggregation and verification in each consensus round by varying the number of nodes participating in the consensus. The signature aggregation cost depends on the number of signatures to be aggregated, hence signature aggregation cost increase linearly with the number of nodes/signatures. Figure 4 depicts the implementation result of signature aggregation cost (in milliseconds). In contrast, the verification cost does not depend on the number of nodes, as we are verifying a single aggregate signature. An aggregated BLS signature verification requires the computation of two bilinear pairing operations. We are using ate pairing scheme in our implementation, and the verification cost of aggregate BLS signature is 3.54 ± 0.07 milliseconds. For storing the data in our account-based blockchain nodes, we use persistent key-value and fast database store leveldb [38] (see also [39]).

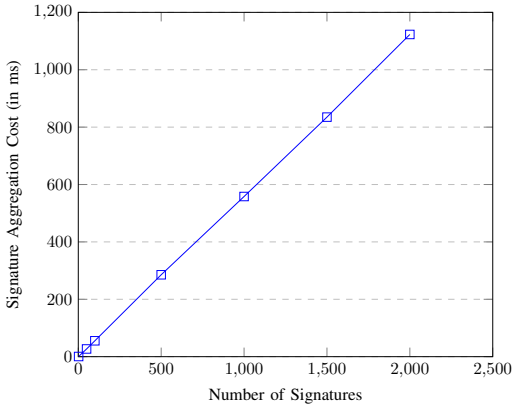

Fig. 4. BLS signature aggregation cost vs Number of signatures

IX. Conclusion

We proposed the "Meshwork ledger" which establishes a network of coequal client nodes that contribute to the endorsement of the transactions by providing digital signatures to a validator node that collects them in an aggregate signature scheme. We also proposed a reward mechanism for the meshwork client nodes. That reward mechanism had a prime design objective to offer coequality for all client nodes. Our goal was to design a blockchain ledger where there is no advantage for getting rewards if the client is an early adopter, if the client has collected a significant stake of rewards or if the client just joined the meshwork.

The pillar design component in our consensus algorithm is the use of the aggregate multi-signatures. The core idea of the consensus is to race for the maximum number of signatures (approvals) on a block from the mesh clients, to append the block in the blockchain. The race in the consensus is among validator nodes that try to collect a maximum number of approvals (signatures) from the mesh clients. The future direction of work for our consensus algorithm is to perform a detailed scalability analysis, to evaluate the performance of consensus in a large network, and to compare it with existing consensus algorithms to advance its adoption in the practical world of implementation.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf," 2009.

[2] V. Buterin, "QuarkCoin: Noble Intentions, Wrong Approach," *Bitcoin Magazine*, Dec 2013, [Online; accessed 3-Jun-2019].

[3] E. Duffield and D. Diaz, "Dash: A payments-focused cryptocurrency," Whitepaper, https://github.com/dashpay/dash/wiki/Whitepaper, 2018, [Online; accessed 3-Jun-2019].

[4] CryptoRekt, "Official Verge Blackpaper 5.0," Blackpaper, https://tinyurl.com/y88sd7ze, Jan 2019, [Online; accessed 14-Jun-2020].

[5] M. Raikwar, D. Gligoroski, and K. Kralevska, "SoK of Used Cryptography in Blockchain," *IEEE Access*, vol. 7, pp. 148 550–148 575, 2019.

[6] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.

[7] L. Ren, "Proof of stake velocity: Building the social currency of the digital age," *Self-published white paper*, 2014.

[8] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, pp. 51–68.

[9] A. Kiayias, I. Konstantinou, A. Russell, B. David, and R. Oliynykov, "A Provably Secure Proof-of-Stake Blockchain Protocol." *IACR Cryptology ePrint Archive*, vol. 2016, p. 889, 2016.

[10] M. Milutinovic, W. He, H. Wu, and M. Kanwal, "Proof of Luck: An Efficient Blockchain Consensus Protocol," in *Proceedings of the 1st Workshop on System Software for Trusted Execution*, ser. SysTEX '16. ACM, 2016, pp. 2:1–2:6.

[11] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On Security Analysis of Proof-of-Elapsed-Time (PoET)," in *Stabilization, Safety, and Security of Distributed Systems*, P. Spirakis and P. Tsigas, Eds. Springer International Publishing, 2017, pp. 282–297.

[12] Libra Association, "The Libra Blockchain," June 2019. [Online]. Available: https://libra.org/en-US/

[13] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology – ASIACRYPT 2018*, T. Peyrin and S. Galbraith, Eds. Cham: Springer International Publishing, 2018, pp. 435–464.

[14] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee, "Pixel: Multi-signatures for consensus," Cryptology ePrint Archive, Report 2019/514, 2019, https://eprint.iacr.org/2019/514.

[15] D.-P. Le, G. Yang, and A. Ghorbani, "Ddh-based multisignatures with public key aggregation." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 771, 2019.

[16] J. Long and R. Wei, "Scalable bft consensus mechanism through aggregated signature gossip," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, pp. 360–367.

[17] "Pchain," 2018. [Online]. Available: https://pchain.org

[18] Graytrain, "PDBFT2.0 — Pchain's Revolutionary Consensus Algorithm For Solving The Trilemma," November 2019. [Online]. Available: https://tinyurl.com/y58v6koa

[19] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple schnorr multi-signatures with applications to bitcoin," *Designs, Codes and Cryptography*, vol. 87, no. 9, pp. 2139–2164, Sep 2019.

[20] G. Fuchsbauer, M. Orrù, and Y. Seurin, "Aggregate cash systems: A cryptographic investigation of mimblewimble," in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 657–689.

[21] Y. Zhao, "Practical aggregate signature from general elliptic curves, and applications to blockchain," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS '19. New York, NY, USA: ACM, 2019, pp. 529–538.

[22] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: a scalable decentralized trust infrastructure for blockchains," *CoRR*, vol. abs/1804.01626, 2018.

[23] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 279–296.

[24] R. El Bansarkhani and J. Sturm, "An efficient lattice-based multisignature scheme with applications to bitcoins," in *Cryptology and Network Security*, S. Foresti and G. Persiano, Eds. Cham: Springer International Publishing, 2016, pp. 140–155.

[25] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille, "Musig-dn: Schnorr multi-signatures with verifiably deterministic nonces," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1717–1731. [Online]. Available: https://doi.org/10.1145/3372297.3417236

[26] R. Khalil, A. Gervais, and G. Felley, "Nocust-a securely scalable commit-chain," Cryptology ePrint Archive, Report 2018/642, Tech. Rep., 2018.

[27] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 514–532.

[28] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Advances in Cryptology — EUROCRYPT 2003*, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 416–432.

[29] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *Advances in Cryptology - EUROCRYPT 2006*, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 465–485.

[30] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," in *Advances in Cryptology - EUROCRYPT 2007*, M. Naor, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 228–245.

[31] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[32] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260.

[33] P. A. Grassi, M. E. Garcia, and J. L. Fenton, "Draft nist special publication 800-63-3 digital identity guidelines," *National Institute of Standards and Technology, Los Altos, CA*, 2017.

[34] V. Buterin *et al.*, "Ethereum: A next-generation smart contract and decentralized application platform," *URL https://github.com/ethereum/wiki/wiki/% 5BEnglish% 5D-White-Paper*, 2014.

[35] W. Li, S. Andreina, J.-M. Bohli, and G. Karame, "Securing proof-of-stake blockchain protocols," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, J. Garcia-Alfaro, G. Navarro-Arribas, H. Hartenstein, and J. Herrera-Joancomartí, Eds. Cham: Springer International Publishing, 2017, pp. 297–315.

[36] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, Oct 1999, pp. 120–130.

[37] M. Ahmed-Rengers and K. Kostiainen, "Don't mine, wait in line: Fair and efficient blockchain consensus with robust round robin," 2020.

[38] A. Dent, *Getting started with LevelDB*. Packt Publishing Ltd, 2013.

[39] M. Raikwar, D. Gligoroski, and G. Velinov, "Trends in development of databases and blockchain," in *2020 Seventh International Conference on Software Defined Systems (SDS)*, 2020, pp. 177–182.