SOKL

The Gas-Free Invisible Blockchain

Secure Gas-Free Scalability for Ethereum Applications

OCT 2024

Contents

Abstract

This document and the associated technologies are under development and subject to change

Blockchain scalability has been the center of attention since 2017 when Ethereum first experienced congestion issues. As new scaling techniques with Layer-1 tokens are consistently announced that have multi-billion dollar valuations, very little traction and marginal innovation, one must ask, what is the end goal of these methods and techniques? Is the goal to launch hype new tokens or deliver value to developers and end users of applications? The reality is that industry innovations should push for one end goal: enabling mass global adoption of blockchain applications while maintaining core principles of decentralization and security. Mass adoption requires scalability both in terms of technical execution but also user experience and cost effectiveness.

Common scaling techniques are listed below:

Monolithic Layer-1 Blockchains

Monolithic Layer 1 blockchains are engaged in an arms race for hardware advancement. These blockchains, designed to handle all aspects of transaction processing within a single layer, have shown effectiveness in improving throughput. However, they are bound to encounter threshold limitations due to the finite capacity of hardware advancements expected over the next two years. As the demand for blockchain applications grows, these inherent limitations will become more pronounced, restricting their scalability potential.

Layer-2 Technologies

Layer-2 solutions typically fall into two camps: ZK Rollups and Optimistic Rollups. These solutions aim to scale blockchain networks by building on top of existing Layer-1 chains, such as Ethereum. These technologies facilitate higher transaction throughput and reduced costs for specific use cases, particularly in decentralized finance (DeFi). However, they also introduce significant trade-offs. Centralized sequencers used in Layer-2 solutions can undermine decentralization, and the need to finalize transactions on the mainnet adds to costs and capacity limitations. Moreover, latency issues for transaction finality and the frequent failure to implement fraud proofs compromise the integrity and efficiency of these systems. We have still not seen initial product specs for fraud proofs and decentralized sequencing become a reality in the Layer-2 ecosystem.

Modularity in Decoupling Consensus from Data Availability

Modularity, which involves decoupling consensus mechanisms from data availability, presents a promising approach to scalability. By separating these functions, blockchains can achieve greater efficiency and flexibility. Nevertheless, this approach introduces increased complexity for both developers and end users, along with user experience (UX) challenges and fragmented liquidity. These factors can hinder the seamless adoption and integration of modular systems.

Modularity with Independent Shards/ Blockchain Modules

Another modular approach involves creating numerous shards or modules that operate independently while sharing a base level of pooled or shared security. This method enhances scalability by distributing the workload across multiple independent units. However, ensuring the security and interoperability of these shards can be challenging, and the complexity of managing a sharded architecture may pose additional hurdles for widespread adoption.

Layer 1 Parallelization

Parallelization in blockchain networks allows transactions to be processed concurrently, leveraging all available network resources. By executing transactions simultaneously across different states, the network ultimately reaches a unified state, agreed upon by the nodes to maintain consensus. This differs from the sequential execution model used by older blockchain systems, where transactions are processed one after the other. These techniques are promising but slow to market and have yet to produce any significant on-chain transactions.

While each of these techniques pushes the envelope of blockchain scalability, they also come with inherent weaknesses. Therefore, we propose that the optimal solution lies in a combination of techniques and architectures. This hybrid approach is designed to achieve one ultimate goal: enabling mass adoption of blockchain technology without compromising on security or decentralization.

"Built Different" – SOKL The Gas-Free, Invisible Blockchain for Mass Adoption

SOKL is a network of Layer-1 blockchains designed to enable mass adoption through six key proven innovations. Ultimately SOKL is a network of Layer 1 blockchains (shards) that are empowered with pooled security properties and a security anchor from the Ethereum Blockchain. SOKL is not a sidechain and SOKL Chains are not sidechains, which have well-documented security gaps. SOKL 's architecture enables its modules to run as independent Layer-1 chains or Layer-2 chains when integrated with ZK rollup capabilities. This architecture has proven to be viable as SOKL has processed over 700 Million on-chain transactions for over 45 Million Unique active wallets in the first 4 years of the network's existence.

Gas-Free Transactions

SOKL introduces a gas-free transaction model, achieved through the following mechanisms:

- **Unified Validation at Protocol Consensus Level:** Unified Validation means that all validators work collectively as a team. Rather than competing for fees with each block, they work as a unit and equally share the fees and inflation with a payout on a monthly basis. They not only produce and validate as a team, but they monitor each other as a team ensuring that only honest and performant validator nodes split the proceeds on a monthly basis. This allows for a more efficient and streamlined consensus process.
- **Dynamic Chain Fees:** Instead of traditional gas fees, SOKL uses dynamic chain fees that maintain healthy network economics without imposing the UX issues associated with dynamic gas fees on end users. This creates a Web2-like experience where developers, applications, and businesses pay infrastructure subscription fees, and not the end users.
- **sFUEL:** An alternative to gas, sFUEL operates similarly but incurs no economic cost to the end user. The scarcity and intelligent distribution of sFuel prevent DDoS and spam attacks as would-be attackers will quickly run out of sFUEL before they can effectively stall the network.

Pooled Security Shards

SOKL leverages modularity at the chain level to create pooled security shards:

- **Independent Shard Operation:** Each shard operates independently from an operational perspective but shares security across a single validator network.
- **Shared Security without Shared Capacity:** Each shard is in essence its own Layer-1 blockchain. The congestion or utilization of one shard has zero performance or price impact of another shard. The nodes running the shards are selected as a dynamically shifting subset across the larger pool of the network nodes.
- **Elastic Supply**: Shards can easily be created as more capacity is required in the network.
- **Appchains or HubChains**: Shards can act as individual AppChains or shared chains called HubChains that are utilized concurrently by many different applications.

Root Security Anchor

SOKL anchors its security into the Ethereum network through core principles of shared security:

- **Root Staked Token:** The root staked token is created and staked on the anchor chain (Ethereum), securing the network.
- **Shard Functionality Anchored into Ethereum Security:** This ensures shard security and network operations that prevent collusion are managed on the anchor chain, not the SOKL validator network.

Optimized Ethereum Virtual Machine

SOKL runs a hyper-advanced C++ Ethereum Virtual Machine (EVM):

- **Enhanced Throughput:** This enables far faster throughput of consensus within an EVM environment. Many high-speed consensus algorithms are limited by standard gETH EVM forks, but SOKL 's advanced EVM circumvents these limitations.

Shard/Chain Interoperability

SOKL Shards can all message and/or bridge near instantly with one another with zero fees.

SOKL IMA Bridging is the world's first live BLS threshold encrypted bridge with near instant and free bridging between chains. This dynamic connects the SOKL Network and abstracts the complexity of sharding away from end users via gas free bridging, custom smart contracts, and the use of a UX optimized bridge UI that makes it simple for users to engage across the ecosystem without needing to understand the complexities of sharding, chains, liquidity, token mapping, etc.

By integrating these innovations, SOKL aims to provide a scalable, secure, and user-friendly blockchain platform that supports mass adoption without compromising on decentralization or security.

SOKL Manager

SOKL Root Security Anchor is composed of smart contracts that run on the Ethereum mainnet and are collectively referred to as SOKL Manager. SOKL Manager operates critical functions of the SOKL Network, such as validator registration, chain creation/destruction, super node selection, super node rotation, staking, bounties, withdrawals, slashing, and more.

Super Node Creation

To be added as a super node to the system, a prospective super node must run the SOKL daemon which will evaluate the prospective node to ensure that it is upholding network hardware requirements[1]. If the prospective super node passes this verification step, the daemon will permit it to submit a request to join the network to the SOKL Manager. This request will contain both the required network deposit as well as node metadata collected by the daemon (e.g. IP address, port, public key,..). After the request has been committed to Ethereum, the prospective super node will either be added to the system as 'full node' or a 'fractional node'[2]. Full nodes will have all of their resources utilized for a single SOKL Chain while fractional nodes will participate in multiple SOKL Chains (multi-tenancy).

After a super node is created, it will have a large group[3] of peer super nodes in the network randomly assigned to it; peers regularly audit super node downtime and latency at predetermined periods (e.g. five minutes) and submit these batched metrics to the SOKL Manager once for every network epoch where they are used to determine the super node's bounty reward.

Super Node Destruction

When exiting the network, super nodes must first declare their exit and wait a finalization period[4]. After this finalization period (e.g. two days), the node will be inactive and able to withdraw its initial stake from the network.

In the case that a user is unable to wait for the finalization period and exits their node immediately from the network, it will be classified as a non-conforming (dead) node by SLA nodes, and the bounty for the super node will not be paid. It will then be scheduled to be cycled out of the chain.

SOKL Chain Creation

When a SOKL Chain is created, a new chain owner selects the chain's configuration and requests the creation of a SOKL Chain. Upon completion, the chain owner must pay into the SOKL Chain Pricing Paymaster on the SOKL Europa Hub before the end of the current epoch. A chain that is not paid in on time is eligible for deletion.

To maximize resource efficiency, chain owners are provided with the option of choosing the size of their SOKL Chains – small, medium, or large. A minimum of 16 nodes are randomly assigned to validate each chain with each node using a designated portion of resources depending on the chain size.

A small chain will use 1/128 of node resources. A medium chain will use 1/16 of node resources and a large node will use 1/1 or all of a node's resources (aside from the resources running the SOKL daemon).

> *Presently, all resources in the network are of equal value and the cost for consuming these resources is based upon the size of the chain. As the network matures, the cost of network resources can conceivably be calculated dynamically to account for current network conditions, system load, and other variable factors. Additionally, the chain creation process can also allow for users to customize*

[1] These hardware requirements are subject to change so as to lower barriers to entry by parties interested in running nodes.
[2] The ratio of full nodes to fractional nodes will be determined by market demand - the actual algorithm will be specified in the future.
[3] The number of peer nodes is targeted to be 24, but this is subject to change.
[4] Period during which a node's peers are decommissioned and new nodes are appointed to the SOKL Chains of the exiting node.

additional aspects of their chains including specifying the number of nodes, number of signers, consensus algorithms, and other consensus/chain-based variables.

After a creation request has been received by the SOKL Manager, a new SOKL Chain will be created and its respective endpoint returned to the chain owner. If there are insufficient resources in the network to support the creation of the desired SOKL Chain, the transaction will be canceled and the chain initiator will be notified.

Unified Validation, Random Selection, and Frequent Node Rotation

The SOKL Network is a Proof of Stake operating environment. Security and validity of transactions in a Proof of Stake network primarily depend on the performance, architecture, and behavior of the validator nodes. To ensure the validation layer is operating properly, a network requires a large number of validator nodes. A small number of nodes in a network is inherently risky and fragile.

To build and operate a secure and robust network, the underpinnings of the infrastructure depend both on a) the random selection of chain validator sets and b) the rotation of nodes in and out of chains on a frequent basis. Without randomness and rotation, there is an increasing risk of bribery of and/or collusion between validators, vastly reducing the security and integrity of the chains within a network.

The validator nodes in the SOKL Network operate and organize themselves in an independent manner. There is no central node controller that chooses which chains a node should work on. Instead, the nodes are collaboratively working on algorithms as a community to enhance randomness and rotation to drive which chains to work on. Each node acts similarly to a drone. A node is singularly communicating with the SOKL Manager mainnet in parallel with other nodes in the network in a decentralized but coordinated and unified manner.

A final requirement with a Proof-of-Stake network is a proper incentive structure that addresses both penalties and rewards. With respect to the former, every validator node is required to have a significant value staked into a network. Staking is an enforcer of good behavior in that if a validator is determined to have colluded or gone byzantine, the validator will forfeit its stake and will be removed from the network.

For a bad actor to coerce or bribe the validators of a SOKL chain with this type of a pooled validation model – one that employs random selection and frequent node rotation – the bad actor would have to effectively bribe 2/3 of the overall network. At current and as the network grows, it will be exceedingly difficult to accomplish this when considering the size of the current network. SOKL 's network design is based on these core principles and is directly aligned with stopping and eliminating attacks while preserving the integrity of transactions within each chain in the network.

SOKL Chain Destruction

Destruction of a SOKL Chain occurs when a chain owner's payments for network resources is greater than three (3) months past due or when a chain owner has flagged their SOKL Chain for deletion. If chain payments are not paid as per SOKL Chain Pricing, the chain owner will be notified of their chain's pending deletion and given the opportunity to pay the debt owed to the network and pre-pay for additional time.

The destruction process will transfer any crypto assets originating from Ethereum to their owners on the mainnet, remove all nodes from the SOKL Chain, hard reset their storage and memory, and remove the SOKL Chain from the SOKL Manager before rewarding the submitter who commissioned the destruction of the chain.

Note: The reward earned by the submitter for destroying a chain is slightly greater than the cost of the transaction and serves as an incentivized garbage-collection mechanism for network resources.

Validator Rewards and Bounty Issuance

Proposed: The creation of SOKL Chains is independent from the network's core delegations. This enables greater profitability for validators while also ensuring that delegations are not being reduced for validator incentivization. The network has a rent system that every chain must follow in order to keep their chain active. The benefit of such a system is that validators are paid the chain fees monthly and can claim it as part of their incentivization package.

An inflation (issuance) event also takes place each month whereby new SOKL tokens are created via a contract on the Ethereum mainnet, the result of which gets pushed into the bounty pool for payout to validators. By way of example, if there are a thousand validator nodes in the network and they all perform well, they will each participate in the monthly proceeds from the bounty pool – this bounty pool consists of a portion of the chain token stakes plus the inflation amount.

SOKL Chain Operations

SOKL Consensus

The SOKL Network uses a variant of the Asynchronous Binary Byzantine Agreement (ABBA) protocol as its consensus model. The fundamental benefit of the ABBA protocol is that it is designed to exhibit robustness in the case of node downtime where each latent and/or down node is regarded as a slow link. The ABBA-based consensus layer replaces the default consensus model in the EVM (Ethereum Virtual Machine) that is included as a container within each node. It is this EVM that gets instantiated within each node that manages and maintains each SOKL chain. SOKL Chains are validated via a set of nodes which engage in block creation and commitment through the use of this asynchronous, leaderless, and provably secure variant of the ABBA protocol.

The SOKL Network consensus algorithm relies heavily on academic research so as to make the case for a mathematically provable approach to consensus-based chain validation and creation. Even at this evolved state of the industry, there is a split in the blockchain world between production-first systems and academic research. In the former category are Proof of Work networks like Bitcoin and Ethereum 1.0 and Proof of Stake systems like Cosmos and Tendermint. In the academic world, one finds ABBA-based approaches and a number of other mathematically provable algorithms.

The design of the SOKL Network bridges this gap between real world trail and mathematical research. SOKL is the first network to make use of binary consensus in a large-scale commercially available production system. This is not to imply or infer that other production algorithms are flawed, but rather to make the point that the design of SOKL Consensus is guided by the rigor to make use of algorithms that are mathematically provable.

So long as >⅔ of the total node validator set are online, they will continue creating and committing new blocks to the chain. This protocol is a multi-phase process illustrated by the diagram below and detailed in the following sections.

Threshold Signatures

Our protocol uses threshold signatures for supermajority voting. Upon creation of a SOKL Chain, Boneh Lynn Shacham (BLS) private key shares PKS[I] are created using joint-Feldman Distributed Key Generation (DKG)

and issued to each node. For each PKS[I], there exists a verifiable public key PK[I] which is stored and made publicly available on the SOKL Manager for signature verification purposes. BLS threshold signatures are implemented as described in Boldyreva with elliptic curve (altBN256) and group pairing (optimal-Ate) implemented in Ethereum Constantinople release.

Threshold Encrypted Messaging

Network Security Assumptions

The protocol assumes that the network is asynchronous with eventual delivery guarantee, meaning that all nodes are assumed to be connected by a reliable communications link – links can be arbitrarily slow, but will eventually deliver messages.

This asynchronous model is similar to Bitcoin and Ethereum blockchains and reflects the state of the modern Internet, where temporary network splits are normal but eventually resolve. The eventual delivery guarantee is achieved in practice by the sending node making multiple attempts with exponential backoff to transfer the message to the receiving node, until the transfer is successful.

Pending Transactions Queue

Each node maintains pending transactions[5] queue. The first node to receive a transaction into that queue will attempt to propagate it to its peers via dedicated outgoing message queues for each. To schedule a message for delivery to a particular peer, it is placed into the corresponding outgoing queue. Each of these outgoing queues is serviced by a separate thread, allowing messages to be delivered in parallel so that failure of a particular peer to accept messages will not affect receipt of messages by other peers.

Block Proposal

After the previous consensus round has been completed, each node's TIP_ID will increment by 1 and immediately cause them to create a block proposal.

To create a block proposal, a node will:

1. Examine its pending transaction queue.
2. If the total size of transactions in the pending queue is less than or equal to the MAX_BLOCK_SIZE, the node will fill in a block proposal by taking all transactions from the queue.
3. In the case that the total size of transactions in the pending queue exceeds MAX_BLOCK_SIZE, the node will fill in a block proposal of MAX_BLOCK_SIZE by taking pending transactions from the queue in order of oldest to newest received.
4. The node will assemble block proposals with transactions which are ordered by SHA-256 Merkle root from smallest value to largest value.
5. In the case that the pending queue is empty, the node will wait for BEACON_TIME, and then, if the queue is still empty, make an empty block proposal containing no transactions.

   Note: nodes do not remove transactions from the pending queue at the time of proposal. The reason for this is that at the proposal time there is no guarantee that the proposal will be accepted.

[5] Each user transaction is assumed to be an Ethereum-compatible transaction, represented as a sequence of bytes.

Data Availability

Once a node creates a block proposal it will communicate it to other nodes using the data availability protocol described below. The data availability protocol guarantees that the message is transferred to the supermajority of nodes.

The five-step protocol is described below:

1. The sending node A sends both the block proposal and the hashes of the transactions which compose the proposal P to all of its peers.
2. Upon receipt, each peer will reconstruct P from hashes by matching hashes to transactions in its pending queue. For transactions not found in the pending queue, the peer will send a request to the sending node A. The sending node A will then send the bodies of these transactions to the receiving node, allowing for the peer to reconstruct the block proposal and add the proposal to its proposal storage database PD.
3. The peer then sends a receipt back to A that contains a threshold signature share for P.
4. Node A will wait until it collects signature shares from a supermajority (>⅔) of nodes (including itself). A will then create a supermajority signature S. This signature serves as a receipt that a supermajority of nodes are in possession of P.
5. A will then broadcast this supermajority signature S to each of the other nodes in the network.

   Note: Each node is in possession of BLS private key share PKS[I]. Initial generation of key shares is performed using Joint-Feldman Distributed Key Generation (DKG) algorithm which occurs at the creation of the SOKL Chain and whenever nodes are shuffled.

In further consensus steps, a data availability receipt is required by all nodes voting for proposal P whereby they must include supermajority signature S in their vote; honest nodes will ignore all votes that do not include the supermajority signature S. This protocol guarantees data availability, meaning that any proposal P which wins consensus will be available to any honest nodes.

Pluggable Binary Byzantine Agreement

The consensus described below uses an Asynchronous Binary Byzantine Agreement (ABBA) protocol. SOKL currently uses a variant of ABBA derived from Mostefaoui et al. Any other ABBA protocol P can be used, as long as it satisfies the following properties:

- Network model: P assumes asynchronous network messaging model described above.
- Byzantine nodes: P assumes less than one third of Byzantine nodes.
- Initial vote: P assumes that each node makes an initial vote yes(1) or no(0)
- Consensus vote: P terminates with a consensus vote of either yes or no, where if the consensus vote is yes, it is guaranteed that at least one honest node voted yes.

  Note: An ABBA protocol typically outputs a random number COMMON_COIN as a byproduct of its operation. We use this COMMON_COIN as a random number source.

Consensus Round

Immediately after the proposal phase completes, each node A who has received supermajority signature S for their proposal P will vote for Asynchronous Byzantine Binary Agreements (ABBAs) in a consensus round R. The protocol is as follows:

1. For each R, nodes will execute N instances of ABBA.
2. Each ABBA[i] corresponds to a vote on block proposal from the node i.
3. Each ABBA[i] completes with a consensus vote of yes or no.

4. Once all ABBA[i] complete, there is a vote vector v[i], which includes yes or no for each proposal.
5. If there is only one yes vote, the corresponding block proposal P is committed to the SOKL Chain.
6. If there are multiple yes votes, P is pseudo randomly picked from the yes-voted proposals using pseudo random number R. The winning proposal indexes the remainder of division of R by N_WIN, where N_WIN is the total number of yes proposals.
7. The random number R is the sum of all ABBA COMMON_COINs.
8. In the rare case where all votes are no, an empty block is committed to the blockchain. The probability of an all-no vote is very small and decreases as N increases.

Finalizing Winning Block Proposal

Once consensus completes with winning block proposal P on any node A, the node will execute the following algorithm to finalize the proposal and commit it to the chain:

1. Node A will check if it has received the winning proposal P.
2. If node A has not received the proposal, it will request it from its peer nodes for download.
3. Node A will then sign a signature share S for P and send it to all other nodes.
4. Node A will then wait to receive signature shares from a supermajority of nodes, including itself.
5. Once node A has received a supermajority of signature shares, it will combine them into a threshold signature.
6. Node A will then commit the P to the blockchain together with the threshold signature S.

Blocks which are committed to the SOKL Chain contain a block header and block body. The block body is a concatenated transactions array of all transactions in the block and the block header is a JSON object which includes the following:

Name	**Data Type**	**Description**
BLOCK_ID	integer	ID of the current block, starting from 0 and incremented by 1.
BLOCK_PROPOSER	integer	ID of the node that proposed the block.
PREVIOUS_BLOCK_HASH	string	SHA-256 Merkle root of the previous block.
CURRENT_BLOCK_HASH	string	SHA-256 Merkle root of the current block.
TRANSACTION_COUNT	integer	Count of transactions in the current block.
TRANSACTION_SIZES	integer[]	An array of transaction sizes in the current block.
CURRENT_BLOCK_PROPOSER_SIG	string	ECDSA signature of the proposer of the current block.
CURRENT_BLOCK_TSIG	integer	BLS supermajority threshold signature of the current block.

Table 1. SOKL Block Header Format

SOKL Super Nodes and Nodes

Each SOKL Chain is composed of a collective of randomly appointed nodes which run the SOKL daemon and run SOKL consensus. Unlike other protocols, SOKL nodes are not restricted to a one-to-one mapping between participating nodes in the network. This is made possible through the containerized node architecture deployed on each node in the SOKL Network which allows each node to run multiple SOKL Chains simultaneously.

Nodes within a SOKL Node are referred to either as super nodes or nodes. Each node participates in independent SOKL Chains. Below is a diagram of a container running on a SOKL node.

This containerized architecture was selected as a means to bring enterprise grade performance and optionality to decentralized application developers on par with centralized systems, which offer elasticity, configurability, and modularity. Containers are divided into five main components that ship with a dockerized Linux OS - allowing for each node to be hosted in an OS-agnostic manner. Each container is encapsulated within one of the following services.

SOKL Admin Service

The SOKL Admin Service serves as the human-facing interface for nodes with the SOKL Manager (located on the Ethereum Mainnet). Functionality shipping with this interface includes the ability for nodes to see which SOKL Chains they are participating in as well as the ability to deposit, withdraw, stake, and claim[6] SOKL tokens. Because nodes within Super Nodess are appointed randomly to participate in SOKL Chains, there is no interface for being able to join / leave SOKL Chains within the network.

Node Monitoring Service

The NMS (Node Monitoring Service) is run on each SOKL Node and facilitates the performance tracking of each of that node's peer nodes. Performance tracking is measured in both uptime and latency through a regular process which pings each peer node and logs these measurements to a local database. At the end of each SOKL Chain epoch, these metrics will be averaged and submitted to the SOKL Manager which will use them to determine the payout to each node.

Node Orchestration Service

The NOS (Node Orchestration Service) orchestrates node computation and storage resources to instantiate nodes using a dynamically created node image consisting of the SOKL daemon (SOKL d), the catchup agent for syncing a SOKL Chain, and the transfer agent for interchain messaging. This service also performs respawning of failed nodes as well as deallocation of resources to nodes who have been decommissioned.

SOKL Chain Pricing

SOKL has a sustainable decentralized economic model in place to ensure validator incentives are both sustainable and attractive while also balancing the price of resources compared to pay-per-transaction blockchains.

Chain pricing is the equivalent of DSaaS – or Decentralized Software as a Service – program where network resources are available for rent with a stable and consistent payment.

[6] SOKL tokens available to be claimed are those which are regularly issued to nodes based upon their average uptime / latency for a network epoch.

Chain pricing operates as follows:

- SOKL Chains have a flat rate rental fee that is dependent on the network utilization. Network utilization is calculated based on the Number of SOKL Chains / (Total Number of Nodes in the Network / 2)
- SOKL Chain fees and distribution ratios may be modified through on-chain voting as described in the governance section of the whitepaper
- SOKL Chains may be prepaid for up to 24 months at the current price which protects chain owners from short-term price fluctuations
- SOKL Chains may run up to three (3) months in-debt before their chain and all associated state and information is eligible for removal from the network

Validators benefit from SOKL Chain pricing by:

- Being able to withdraw monthly rent each epoch after payment is due
- Being able to withdraw SKL tokens directly on the SOKL Europa Hub, allowing them to save on gas fees that would otherwise cut into their profit margin

SOKL Chain pricing enables a sophisticated economic model for a decentralized network that allows all the constituents to fairly balance operational costs and resource usage. To learn more about SOKL Chain pricing, visit the SOKL Forum post.

Attacks and Faults

To account for network downtime, SOKL has integrated a series of contingency strategies for performing fault recovery on both the node and chain level. These range from an automated agent for performing recovery for a downed node to a security incident response team available to all SOKL Chain operators in the network.

Reboots / Crashes

During a reboot, the rebooting node will become temporarily unavailable—for peer nodes, this will look like a temporarily slow network link. After a reboot, messages destined to the node will be delivered—this protocol allows for a reboot to occur without disrupting the operation of consensus.

In the case of a hard crash where a node loses consensus state due to a hardware failure or software bug that prevents the node from being online, its peers will continue attempting to send messages to it until their outgoing messages queues overflow—causing them to drop older messages. To mitigate the effects of this, messages older than one hour are targeted to be dropped from message queues.

While a node is undergoing a hard crash, it is counted as a Byzantine node for each consensus round—allowing for <⅓ of nodes to be experiencing hard crashes simultaneously. In the case where >⅓ nodes experience a hard crash, consensus will stall, causing the blockchain to possibly lose its liveness.

Such a catastrophic failure will be detected through the absence of new block commits for a set time period. At this point, a failure recovery protocol utilizing the Ethereum main chain for coordination will be executed. Nodes will stop their consensus operation, sync their blockchains, and agree on time to restart consensus.

Catchup Agent

A separate Catchup Agent running on each node is responsible for ensuring that node's blockchain and block proposal database are synced with the network. The catchup engine is continuously making random sync connections to other nodes whereby any node discovering that they have a smaller TIP_ID than their peer will

download the missing blocks, verify supermajority threshold signatures on the received blocks, and commit them to its chain.

When the node comes online from a hard crash, it will immediately start this catchup procedure while simultaneously participating in the consensus for new blocks by accepting block proposals and voting according to consensus mechanism but without issuing its own block proposals. The reason for this is that each block proposal requires the hash of the previous block, and a node will only issue its own block proposal for a particular block id once it has finished the catch up procedure.

With such an agent running on each node, nodes having experienced a hard crash will be able to easily rejoin in block proposal after re-syncing their chains.

Security Incident Response

Security is a foremost requirement for all decentralized systems, but despite progress in cryptography and computer science, most security experts agree that perfect security is unattainable. With this in mind, architects need to concentrate on raising the bar as much as possible for the amount of resources and money required to break the system.

Since the SOKL architecture is based on SOKL Chains, a security compromise of SOKL could involve the compromise of a particular SOKL Chain. For example, a significant number of nodes could be affected by a computer virus due to a bug in the Linux kernel. In such a case, the default procedure is as follows:

1. SOKL Chain owners suspecting of a security compromise will issue a request to the Ethereum SOKL Manager contract to temporarily suspend the chain.
2. The SOKL Manager will mark the SOKL Chain as suspended.
3. Uncompromised nodes will receive a notification from the SOKL Manager to freeze their operation.
4. Clients of the SOKL Chain will be notified of the suspension and have their requests to the SOKL Chain rejected.

SOKL Hubs

SOKL Hubs are SOKL Chains that are shared chains amongst many applications.

The SOKL Network Hubs offer both key services to other SOKL Chains while also acting as a shared compute resource for similar type of applications. The Nebula Hub as an example is the home to over 50 games at the time of this writing. Hubs allow for key network resources to be deployed in a scalable manner so that many chains can take advantage of the resource without requiring a deployment on every chain.

Rather than exchanges or marketplaces living on every single chain – which would result in fragmented liquidity – liquidity pools, fiat onramps and offramps, marketplaces, and more are available to the SOKL Hub allowing both dApps and other SOKL Chains to leverage these key resources.

SOKL Hubs also represent shared compute environments for dApps to build directly on. While the SOKL can continuously add new chains, the vast majority of applications require only a fraction of the resources existing SOKL Chains can provide. Based on that, these dApps choose to build directly on one of the SOKL Hubs.

A multi-chain architecture is a powerful solution for increasing network capacity and scalability but cross-chain interactions can create significant friction including delayed responses and high fees. This cross-chain friction is solved within the SOKL Network via its modular design, its use of advanced cryptography and consensus algorithms, and its zero-gas design.

Extensions

With this network architecture and protocol, a number of extensions can easily be added to allow for greater functionality and utility of the network. The first two which have been built include enhanced Filestorage within each node and a mechanism for relaying and executing messages between SOKL Chains.

SOKL Block Storage and SOKL FileStorage

To expand potential use-cases, SOKL has modified the existing EVM to allow for much larger file storing capabilities. The changes enabling this included the increase of blocksizes (allowing for more data to be included in each block) as well as direct access to each node's file system from a fileStorage precompiled smart contract.

Consumers in the network are also able to split files into 1MB "chunks" and submit them to the fileStorage smart contract to be stored on each node's filesystem in a contiguous manner. Files in the network can also be deleted in a bespoke fashion to ensure that the network can reallocate resources due to state bloat from additional storage capabilities.

The SOKL Network also offers secure node-based file storage via the SOKL Filestorage network service. Each SOKL Chain after version 3.1 will have configurable storage capacity based on the needs of the chain owner.

Developers can make use of this storage within their smart contracts on a SOKL Chain and all write operations to this storage are managed via the SOKL consensus model. Chain owners can elect to persist this data on its own as separate standalone decentralized file storage should they wish to do so. This one-stop digital storage capability eliminates the friction that results from mixing in third-party off-chain solutions. An example of how this storage might be used is to securely store NFT-based images, which is one of the bigger challenges with NFT-based applications.

SOKL Interchain Messaging Agent (IMA Bridge)

Messaging to and from SOKL Chains and the Ethereum Mainnet – and between SOKL Chains and SOKL Hubs – is addressed via the SOKL Interchain Agent (or SOKL IMA Bridge). The benefits of the IMA bridge design include high security, fast transfer speed, preservation of custody, cost efficiency, and a high degree of customization. The IMA Bridge accomplishes this by making use of BLS cryptography, Ethereum mainnet smart contracts, Proof of Stake consensus, and other unique aspects of its design. This bridge allows developers and users to safely and economically transfer digital assets between the Ethereum mainnet and any SOKL Chain or between SOKL Chains and Hubs. These digital assets can include ETH, ERC-20, ERC-721, and ERC1155 tokens as well as general messaging data.

Network bridges are essential components between connected blockchain solutions as they allow for the secure transfer of digital assets between networks and other chains within a network. These bridges serve as transfer agents for these assets. Instead of transferring the actual assets, however, they serve as a combination of lockboxes and proxy providers to provide verified transfer of digital equivalents.

On a basic level, a bridge lets a user send an arbitrary message from one chain to another chain in such a way as to allow the contents of that message to become functional in that second chain without risk of double spend or separate concurrent activity on both chains. This message passing implies a locking or unlocking of the contents of the messages (i.e. the tokens) in one chain and the provisioning of a synthetic asset on the other chain.

The SOKL IMA Bridge works in a simple, transparent, and secure manner. Here is the basic transactional flow.

- Token holders signal their intention to transfer tokens and other digital assets from Ethereum to a SOKL Chain (or from the SOKL Chain to Ethereum).
- When sufficient validators for that SOKL Chain see this transfer signal on chain and validate it as well-intentioned and true, the transfer initiates on the Ethereum mainnet (or SOKL Network if in reverse). The assets are secured and placed into a deposit box on the Ethereum mainnet via SOKL contracts on the mainnet.
- After a set number of blocks (10) have passed on the Ethereum mainnet (so as to ensure transactions are fully recognized and valid), a transfer request gets sent to a proxy called the SOKL IMA Agent which in turns calls the SOKL TokenManager operating within the SOKL node network to process this transfer within the respective SOKL Chain. As opposed to days or even hours, this transfer happens quickly, buffered only by mainnet block creation.

A transfer in the other direction, going from a SOKL Chain to Ethereum, happens in a similar manner, although in the case of exiting SOKL into the Ethereum mainnet, the tokens are burned on the SOKL Chain and unlocked on the mainnet. In the case of tokens that are minted on the SOKL Chain and then transferred to Ethereum – a minting operation on Ethereum takes place as part of the transfer in such a way as to preserve the ownership of the minted token.

The use of public keys makes it possible for independent SOKL Chains to verify that a block has been signed and committed on another SOKL Chain, allowing for the execution of smart contracts as well as the transfer of digital assets across SOKL Chains. This mechanism is facilitated through a series of smart contracts located on the Ethereum mainnet, each SOKL Chain, as well as an agent running on each node which is responsible for facilitating these interchain messages.

SOKL Chains each have an inbox and outbox. Messages which are being sent to other chains are kept in the outbox until they are picked up by a randomly-appointed agent who will then send the message to the appropriate recipient chain's inbox as well as any additional metadata[7] which that chain requires to validate that the transaction was included in the sending chain's blockchain. Once this has been proven on the recipient blockchain, the transaction will be forwarded to the destination address / smart contract through an on-chain messaging proxy.

In the case of value transfer from a parent blockchain (ex: Ethereum mainnet), a *DepositBox* is used as a money caching mechanism and two-way peg whereby vouchers are issued against this pooled value on each SOKL Chain and exchanged freely amongst participants in the same manner as transactions. When value is exchanged between SOKL Chains, the value is first destroyed on the sending chain before creating it on the receiving chain to eliminate the possibility of double-spend attacks. This process is also adhered to for redemption transactions sent to the Ethereum mainnet which free locked capital in the deposit box.

Governance

SOKL is a fully decentralized project that operates under the control of all community and SKL token holders. No entity or institution has control or authority over the network. All network decisions are made via group consensus. Developers work to create improvement proposals for the network. Any improvement proposal that impacts network economics must be approved by a vote. The mechanism for voting itself is flexible and subject

[7] Agents within the network are required to send the committed block in which the transaction was included to the receiving chain so that it is able to verify that the transaction was included in the block and that the block has a valid BLS signature for the current node validator set.

to change based on proposals and voting. All code changes must be accepted, updated, and run independently by validator operators.

Current voting mechanics and proposals can be viewed here:

SOKL Token - SKL

The SOKL token (SKL) is a hybrid use token which represents the right to work in the network as a validator or access a share of its resources by deploying and renting an SOKL Chain for a period of time as a developer.

Users pay SOKL in a subscription model to rent these resources (computation, storage, bandwidth) for a predetermined amount of time in the form of a SOKL Chain.

Validators stake SOKL into the network and then gain the right to run nodes and earn both fees and tokens via inflation.

SKL Token Economics can be accessed via the SOKL Network website: https://SOKL .space/network#token

Summary

The SOKL Network uses a unique “Built Different” model to achieve mass adoption level scaling for blockchain. SOKL ’s mission as a decentralized project is to bring cost-effective, gas-free scalability to developers and users of blockchain applications globally.

While many scalability techniques and architectures promise scaling and security, SOKL ’s Built Different model has been proven to work in real-world applications across gaming, AI, DePin, social, DeFi, and more.

SOKL Terminology

Term	Description
ABBA	A randomized agreement protocol which guarantees validity, agreement, and probabilistic termination.
ABFT	A type of Byzantine Fault Tolerance which accounts for the possibility that some messages between honest participants are being delayed or not being delivered to their intended recipients.
Catchup Agent	The agent shipping with each node responsible for syncing the node with the current 'tip' of the SOKL Chain. This enables nodes which have faulted to come back online and download / verify the blocks they missed as well was the syncing of new nodes who have been shuffled into an existing SOKL Chain.
DepositBox	The smart contract which facilitates the locking and unlocking of capital on both SOKL Chains and the Ethereum mainnet. This mechanism is foundational to the implementation of a two-way peg for value transfer.
Messaging Agent	An agent which runs on each node and listens to connected SOKL Chains to facilitate interchain messaging. These agents are appointed to listen to incoming messaging at random times to mitigate the possibility of censorship.
Parent Blockchain	The blockchain from which value is drawn by way of a one or two-way peg.
Peer Node	Nodes in the network which are monitoring the uptime and latency of a specified node. Peer nodes report these metrics to the SOKL Manager at regularly scheduled network epochs. These metrics are used to determine the reward for the specified node.

SOKL Chain	A Proof-of-Stake based chain that uses a randomly selected and frequently rotated set of network nodes. These nodes collectively, but in a decentralized and independent manner, accept user transactions, run SOKL consensus, and store identical copies of the SOKL Chain. A typical implementation will run a network node in a virtualization container such as a Docker container, so a single physical server can provide support to multiple SOKL Chains.
SOKL Manager	This contract (located on Ethereum) manages the orchestration of all entities within the network, inclusive of SOKL Chain creation / destruction, node creation / destruction, withdrawals, and bounties.
SOKL Super Node	Super Nodes in the SOKL Network which are responsible for the orchestration of resources for the creation and destruction of nodes, the monitoring of other nodes in the network, and interfacing with the SOKL Manager.
Two-Way Peg	Allows the transfer of one digital asset from one blockchain to a secondary blockchain and vice-versa. The transfer locks the digital asset on one blockchain while unlocking the equivalent amount on a secondary blockchain. The original digital asset can be unlocked when the equivalent amount of tokens on the secondary blockchain are locked, again.
Node	Nodes are allocated under Super Nodes to SOKL chain and run network validation, data availability, and storage.

SOKL Protocol Parameters

Name	**Description**
BEACON_TIME	The time between empty block creation. If no one is submitting transactions to the blockchain, empty beacon blocks will be created. Beacon blocks are used to detect normal operation of the blockchain.
COMMON_COIN	Introduced by Rabin [7], this is a distributed object that delivers the same sequence of random bits b1, b2, . . . , br, . . . to each process.
MAX_BLOCK_SIZE	The maximum size of the block body in bytes. Currently, we use 8MB and may consider self-adjusting block size to target a particular average block commit time in the future.
N_WIN	The total number of proposals which have been agreed to by nodes as valid for commitment and finalization. This number is used in modulo division to decide the index of the winning proposal.
TIP_ID	The ID of the latest block (or, “the tip”) in the blockchain.

References

1. Achour Mostefaoui , Hamouma Moumen , Michel Raynal, Signature-free asynchronous byzantine consensus with t < n/3 and o(n2) messages, Proceedings of the 2014 ACM symposium on Principles of distributed computing, July 15-18, 2014, Paris, France
2. Alexandra Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In Yvo Desmedt, editor, Public Key Cryptography - PKC 2003, volume 2567 of LNCS, pages 31–46. Springer, 2003.
3. P. Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In FOCS'87, pages 427–437, 1987.
4. Beuchat, J.L., Díaz, J.E.G., Mitsunari, S., Okamoto, E., Rodríguez-Henríquez, F., Teruya, T.: High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In: Joye, M., Miyaji, A., Otsuka, A. (eds.) Pairing 2010. LNCS, vol. 6487, pp. 21–39. Springer, Heidelberg (2010)
5. Boneh, D., Shacham, H., Lynn, B.: Short Signatures from the Weil Pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001)
6. Christian Cachin , Klaus Kursawe , Victor Shoup, Random oracles in constantipole: practical asynchronous Byzantine agreement using cryptography (extended abstract), Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, p.123-132, July 16-19, 2000, Portland, Oregon, USA [doi>10.1145/343477.343531]
7. M. O. Rabin. Randomized Byzantine Generals. In 34th Annual Symposium on Foundations of Computer Science, Palo Alto California, 3-5 November 1993, pages 403–409. IEEE Computer Society, 1983.