Spectrum: Cross-chain interoperability at scale

Spectrum Labs

March 2023

1 Introduction

Following the success of Bitcoin, many blockchain-based cryptocurrencies have been developed and deployed. To meet different requirements in various scenarios, a great number of heterogeneous blockchains have emerged. However, most of the presented blockchain platforms are developed independently, therefore, they are designed for their own use cases and are incompatible with each other. Hence, interoperability between blockchains has become one of the key issues which prevents blockchain technology from wide adoption.

With fair blockchain interoperability users can potentially conduct transactions across different blockchain networks smoothly and without any intermediaries. This guarantees a reduction in the fragmentation of the crypto ecosystem and opens up new horizons and business models. Implementation of the blockchain interoperability protocol is challenging since different blockchains have different security solutions, consensus algorithms and programming languages. An inaccurate solution can potentially increase the possibility of attacks and create management challenges across different connected networks.

The classic cross-chain interoperability solution is a trusted oracle that registers some event on one blockchain and performs the required action on the other. Centralized oracles provide fast and cheap transactions but lack a key feature – decentralization. The liquidity of the protocol built on this approach is custodial which is a centralized approach similar to CeFi when users deposit their funds to an exchange's wallet.

Another common approach involves intermediate network consisting of a fixed number of hand-picked oracles to facilitate data transfer among multiple blockchains. The consensus mechanism in such protocols is usually proof-of-authority or proof-of-stake, hence, the wide range of potential validators are eliminated due to verification procedures or high collateral and network moderation typically carried out by several dozen of rarely alternating nodes. Moreover, a common practice is to store funds transferred between blockchains on some kind of threshold wallets, which are generated by the participants of the intermediate network. This results in all funds being controlled by a fixed group of oracle operators. Such a system is also not truly decentralized.

Regarding the application scenarios, one of the most popular in the existing blockchain interoperability proposals is an atomic token swap. However, atomic

token swapping protocols [1] are not self-inclusive enough to complete the tasks of cross-chain decentralized applications with more complex activities than just token exchanges. The reason is that the atomic swapping process does not have the ability to destroy a certain amount of assets in the source blockchain and re-create the same amount on the target blockchain. Moreover, this process always requires a counterparty who is willing to exchange tokens [2].

True blockchain interoperability requires the users and developers have the ability to access information from one blockchain inside another without any additional efforts from a third party. This is a complicated task, thus, before achieving a successfully interoperable multi-blockchain system, many challenges must be overcome, such as scalability when applying to a large-scale scenario [3].

The motivation of this paper is to describe the Spectrum protocol, which provides an open, truly decentralized, secure and scalable cross-chain interoperability solution. The Spectrum protocol is intended for both end-users and developers, who will be able to implement their applications on top of the protocol to widespread the use of blockchain technology in various business areas.

2 Related Work

Blockchain interoperability is promising but still faces various design challenges. There have been many systematic researches regarding this issue and many famous authors have discussed chain interoperability in general. Blockchain interoperability in the literature is usually classified into categories. Buterin [4] suggested centralized, sidechains/relays, and hash locking. Belchior et al. [5] classified it into cryptocurrency-directed approaches, blockchain engines, and blockchain connectors, Wang [6] proposed to group it into chain-based interoperability, bridge-based interoperability, and dApp-based interoperability.

2.1 Existing Interoperability Solutions

In this paper, we want to emphasize the benefits of the decentralization in the chain interoperability mechanism, so we will not include the systematical-level study of all existing approaches and will briefly discuss the classification proposed by Wang.

2.1.1 Chain-based Interoperability

Chain-based interoperability is aimed at public blockchains and uses atomic swaps as its main mechanism to exchange information between different chains. Following the classification, there are three main approaches to implementing chain-based interoperability: hash locking, trusted notary scheme and sidechain.

Hash Locking is an intermediary method that allows to validate or execute blockchain transactions. Hashed Time Lock Contracts (HTLCs) were originally developed as an alternative to centralized switching and can be thought of as a distributed commitment [7] able to fend off Byzantine adversary. It uses a hash

time-locked system to lock the transaction [8] which is similar to the concept of the cross-chain atomic swap.

From the technical point of view, the hash locking approach has some significant drawbacks, for example, it must lock some assets during its opening phase for an established transaction channel, thereby creating a race condition and, moreover, the possibility of losing assets if a timeout occurs.

Trusted Notary Scheme is usually considered as the simplest way to achieve cross-chain interoperability. The blockchain notary schemes can provide the functionalities of timed proof of existence, whose proof can be used as further proof of ownership [9]. It doesn't require any additional changes in the underlying blockchains and uses a trusted notary to verify the correctness and integrity of information transferred. A notary can be a stand-alone authority or a group of trusted parties that monitor order books of the connected chains and initiating transactions upon the occurrence of some valid events or requests.

Well-known solutions using this technology are, for example, Herdius [10] and Bifrost [11]. In practice, the most appropriate way to achieve interoperability using a notary scheme is to combine it with other methods, as it is done in the Interledger [12] which combines it with a sidechain.

Sidechain is the most promising approach in this category. Sidechain can add new functionalities, namely, security and privacy to the existing blockchains, making possible a tokens synchronization and additional data transfer between chains [13]. The essential feature of the sidechain is that it's design always takes into consideration the structure and the consensus of each connected blockchain, but none of the mainchains are aware of the presence of a sidechain. This is achieved by utilizing a two-way peg scheme [14] which uses a relay routine for a bidirectional hooking. An important consequence of this approach is that sidechains can be designed in a decentralized manner and have their own consensus protocols.

Using a two-way pegs introduces a level of centralization, however, there are solutions which uses a federated two-way pegs where single authority is replaced by a group of trusted individuals selected in a trustworthy manner.

State-of-the-art sidechain platforms are Loom [15], Liquid [16] and Proof-of-Authority (PoA) networks [17]. There also exists a lot of ongoing projects since this technology is innovative and in demand by the blockchain industry.

Summing it up, a practical way to apply chain-based interoperability methods to current mainstream blockchain systems is to combine them together. Most existing solutions are designed primarily to exchange assets, however blockchain technology is much wider in its applications, and it's better to focus on transaction interoperation between different chains in practical implementations and effectively use all these promising approaches.

2.1.2 Bridge-based Interoperability

Bridge-based interoperability aims to create a connection component between homogeneous and heterogeneous blockchains. Solutions in this field are more complex and typically support the extension of smart contracts which allows

developers to design and deploy their own logic thereby expanding the interoperability applications. Bridge-based interoperability can be implemented in two main forms: trusted relay and blockchain engine.

Trusted Relay is a very native approach where trusted parties share transactions between different blockchains. Relay schemes replicate block information of the source blockchain via verifiable smart contracts within a target blockchain to allow the target blockchain to verify the existence of data on the source blockchain without requiring trust in a centralized entity [4]. There are many developing relay schemes: BTC Relay [18], PeaceRelay [19], etc. State-of-the art projects are: Hyperledger Cactus [20], Testimonium [21] and Tesseract [22]. All these solutions support complex use case and are highly usable and reliable, however, still not fully decentralized [6].

Blockchain Engine also provides a relay among the connected blockchains. It is based on a shared infrastructure which support different layers and services including network, consensus, incentive, etc. Requirements of multi-layer supports is essential, thus, most existing blockchain engine-based solutions are still in the stage of proof of concept or under active development. Most significant projects are: Polkadot [23], Cosmos [24], WanChain [25], and ARK [26].

All bridge-based solutions provide convenience for end-users since they don't need to know what happens in the bridge. In general, trusted relays are much more simple and adopted to handle chain interoperability, however, they usually utilize mechanisms similar to the notary schemes which also leads to a certain degree of centralization.

2.1.3 dApp-based Interoperability

Presence of well functioning decentralized applications (dApps) is significant in the blockchain ecosystem, so dApps should be interoperable as well and this is the goal of dApp-based interoperability. Each dApp cannot ensure semantic interoperability, and it's essential to develop the minimum semantic that must be supported by each application to achieve interoperability among dApps. dApp-based blockchain interoperability protocols in the literature are typically classified as: blockchain of blockchains, blockchain adapters and blockchain agnostic protocols.

Blockchain of Blockchains is a platform that allows developers to construct cross-chain dApps where each blockchain functions as an independent one. It is similar to the sidechain idea but differs in implementation. Sidechains are typically aimed at atomic swaps among the homogeneous blockchains where all actions should be coordinated by the mainchain. Blockchain of blockchains solutions typically requires a second layer of blockchain (mainchain) to record the activities that happen on each subchain which can be heterogeneous [6]. There are several projects where blockchain of blockchains concept is applied for different scenarios: Overledger [27], HyperService [28], SMChain [29] and etc.

Blockchain Adapter handles the interoperability by providing an interface for the end-users to runtime selection, smart contracts, etc. Most significant

project in this category are PleBeuS [30] and smart contracts *move* protocol [31].

Blockchain Agnostic Protocol: refers to a single platform allowing multiple blockchains to co-exist, enabling cross-chain or cross-blockchain communication between arbitrarily distributed ledgers. Blockchain agnosticism provides its end-users various options to pick their optimal blockchain and provide the capabilities for cross-chain operations. Several agnostic-based technologies have been described in the literature: ILPv4 [32], Gravity [33], SuSy [34] and etc. All these solutions are flexible and has great potential, although most of them are focused on the general design of the prototype and do not grant backward compatibility.

Although dApp-based blockchain interoperability is very promising, most of the solutions in this category are either in early stages of development or lack a practical implementation with criteria to evaluate their effectiveness and efficiency.

2.1.4 Discussion

All of the interoperability approaches described above have their strengths and weaknesses. However, the chain-based interoperability approaches, especially sidechains, are well-established and benefits from extensive research and improvements in design. Sidechains have two important pros that will help to increase the widespread adoption of blockchain technology in various business areas:

- Having their own consensus mechanisms, sidechains can process transactions efficiently and reduce transaction fees for users.
- Taking into consideration the structure and the consensus of each connected blockchain sidechains allow dApps to expand their ecosystem.

The main cons of the existing sidechain protocols is a *centralization* and *poor security guaranties* of the consensus. The disadvantages of centralization are obvious:

- A system is not sustainable when it depends on a single party.
- If the trustee goes down, unfinished swaps can appear frozen halfway.
- A malicious trustee can censor transactions.
- A malicious trustee can perform a man-in-the-middle attack by sending an inaccurate data.

Almost the same deficiencies exist for a semi-centralized protocols, where only a few dozen individuals act as validators. Such "decentralization" is very conditional as it is difficult to meet the requirements to become a validator, furthermore, malicious validators can easily cooperate to successfully attack.

Thus, we come to the conclusion that the scalable practical implementation of the truly decentralized system with a provably-secure consensus protocol is the main step towards wide practical usage of sidechains and bringing their benefits into cross-chain interoperability.

3 Goals

To overcome the outlined problems of the existing protocols the resulted Spectrum protocol must satisfy the following properties:

1. **Decentralization.** The system should be highly decentralized.
2. **Interoperability.** The system should be able to support a large number of heterogeneous blockchains.
3. **Openness.** The system should allow anyone to participate in consensus permissionlessly. Protocol should be fully open-source and all participants will be encouraged by the incentives system.
4. **Consensus Scalability.** The system should be able to operate normally while maintaining sufficiently large consensus groups consisting of hundreds of active validators on each connected blockchain.
5. **Operational Scalability.** The system should scale linearly with the number of supported blockchains.
6. **Security.** The system should be able to withstand Sybil attacks.
7. **Sustainability.** The system should be able to tolerate faults of particular connected blockchains.
8. **Upgradability.** The system should allow to add new blockchains into list of supported over time.

To achieve our goals we will combine the best practices from the approaches that are already in use in the chain-based interoperability solutions. To eliminate the existing bottlenecks, we will supplement them with own-developed improvements which we will emphasize and describe in details in the following sections.

4 System Model

In this section we will describe the main components and general assumptions which is essential to conceptualize and construct the Spectrum protocol.

4.1 Transaction Ledger

We adopt the definition of transaction ledger from [35]. A protocol Π implements a robust transaction ledger, provided that Π is divided into blocks that determine the order in which transactions are incorporated into the ledger. Each block in this model is assigned to a specific time slot and the ledger must satisfy the following properties:

1. *Persistence.* Once a node of the system proclaims a certain transaction tx as stable, the remaining nodes, if queried, will either report tx in the same position in the ledger or will not report as stable any transaction in conflict to tx. Here the notion of stability is a predicate that is parameterized by a security parameter k, specifically, a transaction is declared stable if and only if it is in a block that is more than k blocks deep in the ledger.

2. *Liveness.* If all honest nodes in the system attempt to include a certain transaction then, after time expires corresponding to u slots (called the transaction confirmation time), all nodes, if queried and responding honestly, will report the transaction as stable.

4.2 Semi-Synchronous Model Preliminaries

We consider the security model in a semi-synchronous setting with simple modifications to account for adversarially-controlled message delays and immediate adaptive corruption.

Time and Slots. We consider a setting where time is divided into discrete units called slots. The ledger associates one time slot with at most one block. Participants are equipped with roughly synchronized clocks. This will permit them to carry out a distributed protocol intending to collectively assign a block to this current slot. In general, each slot sl_r is indexed by an integer $r \in \{1, 2, ..\}$, and we assume that the real time window that corresponds to each slot has the following two properties:

- The current slot is determined by a publicly-known and monotonically increasing function of the current time.

- Each participant has access to the current time. Any discrepancies between parties' local time are insignificant in comparison with the slot duration.

Synchrony. We consider an untrustworthy network environment that allows for adversarial-controlled message delays and immediate adaptive corruption. Namely, we allow the adversary A to selectively delay any messages sent by an honest party for up to $\Delta \subseteq \mathbb{K}$ slots and corrupt parties without delay.

Random Oracle. We assume that a random oracle is available to each node $n \in N$. The random oracle is designed in such a way that it is able to produce uniformly-distributed pseudo-random numbers which correctness must be verifiable for all participants of the protocol.

Security Model. The system is composed of a set of nodes N and each node $n \in N$:

- Is associated with a unique wallet holding a stake of tokens s_n.
- Able to generate key-pairs (PK, SK) without trusted public key infrastructure.
- Is able to sign messages $sign : (SK, m) \rightarrow \sigma$.
- Is able to verify signatures $verify : (\sigma, PK, m) \rightarrow 0|1$.
- Has access to random oracle functionality.
- Has access to key evolving signature functionality.

At any time t a subset $V \subseteq N$ of nodes can be controlled by an adversary and are considered faulty. Byzantine nodes can divert from the protocol and collude to attack the system while the remaining honest nodes follow the protocol. We assume that the total stake of all faulty nodes is less than $1/3$ of the total stake of all nodes.

4.3 External Systems

We also assume multiple independent distributed systems $S_1, \ldots, S_K$ with underlying ledgers $L_1, \ldots, L_K$ as defined in [36]. For each ledger $L_k, k \in K$ there is a process P_k that can influence the state evolution of the underlying ledger L_k by committing a transaction TX_k into it. We extend the model defined in [36] by assuming that all ledgers allow for execution of simple predicates upon validation of transactions: $verify : C \rightarrow 0|1$, where C is a *context* that contains description of state the transaction interacts with. There is also a function $desc : TX_k \rightarrow DESC^{TX_k}$ that maps transaction TX_k to some *description*, e.g. specifying the transaction value, recipient address, etc. For each S_k there is a corresponding functionality unit F_{S_k} that allows any node equipped with the unit to interact with S_k. Each node $n \in N$ is equipped with at least one such functionality unit and at most K functionality units.

5 System Design

This section presents Spectrum protocol design starting from a naive approach based on Practical Byzantine Fault Tolerance (PBFT) [37] and gradually addressing the challenges.

5.1 Strawman Design: PBFTNetwork

For simplicity we begin with a notarization protocol based on PBFT and then iteratively refine it into the Spectrum protocol.

PBFTNetwork assumes that a fixed consensus group of $n = 3f + 1$ nodes has been pre-selected upfront and at most f of these nodes are Byzantine. The PBFT protocol is designed in such a way that there is no need to trust each individual notary, but only two-thirds of the set. This approach has proved its reliability in practice and has been widely used in various blockchain protocols for many years.

At any given moment of time, one of the nodes is the leader who observes the events on the connected blockchains, batch them and initiate a notarization round within the consensus group. All validators verify the proposed batch by checking for relevant updates on the connected chains. Upon successful verification each node signs the batch with a secret key and sends the signature to the leader.

Liveness and safety of the PBFTNetwork is guaranteed under the simplifying assumptions already mentioned above that at most f nodes are Byzantine. However, the assumption of a fixed trusted committee is unrealistic for open decentralized systems. Moreover, as PBFT consensus members authenticate each other via non-transferable symmetric-key MACs, each consensus member has to communicate with others directly, what results in the $O(n^2)$ communication complexity. Quadratic communication complexity imposes a hard limit on the scalability of the system. Such a design is not suitable for building a multi-chain system, since the workload of each validator grows linearly with each added chain.

In the subsequent sections, we address these limitations in four steps:

1. **Opening the Consensus Group.** We introduce a lottery-based mechanism to *select the consensus group dynamically.*

2. **Replacing MACs by Digital Signatures.** We replace MACs by digital signatures to make authentication transferable and thus opening the door for *sparser communication patterns* that can help reduce communication complexity.

3. **Scalable Collective Signature Aggregation.** We utilize Byzantine-tolerant aggregation protocol that allows for *quick aggregation of cryptographic signatures* and reduces communication complexity to $O(\log n)$.

4. **Eliminating Validator Bottleneck.** We shard consensus groups into local committees by the type of chain that each node is able to handle to *improve system scalability.*

5.2 Opening the Consensus Group

Spectrum is an open-membership protocol, so PBFTNetwork's assumption on a closed consensus group is not valid. Sybil attacks can break any protocol with security thresholds and an appropriate dynamic selection of the consensus group becomes crucial for preserving network's liveness and safety. Election of consensus group members should be performed in a random and trustless way

to ensure that a sufficient fraction (at most f out of $3f + 1$) of members are honest.

Similar selection mechanics is required in most blockchain protocols. Bitcoin [38] and many its successors are using Proof-of-Work (PoW) consensus, which, in essence, is a robust mechanism that facilitates randomized selection of a leader who is eligible to produce a new block. Later, PoW approach was adapted into a Proof-of-Membership mechanism [39]This mechanism allows once in a while to select a new consensus group which then executes the PBFT consensus protocol.

A primary consideration regarding PoW-based consensus mechanisms is the amount of energy required to operate such systems. A natural alternative to PoW is a mechanism based on the concept of Proof-of-Stake (PoS) [40]. Rather than investing computational resources in order to participate in the leader selection process, participants of a PoS system instead run a process that randomly selects one of them proportionally to the stake. Pure PoS mechanism to solve the PBFT problem was firstly used in [41] to select both consensus group members and PBFT rounds leaders and to introduce randomness into this process, a verifiable Random Function (VRF) has been applied.

5.2.1 Verifiable Random Function

A Verifiable Random Function (VRF) [42] is a reliable way to introduce randomness into a protocol. By definition, a function $\mathcal{F}$ can be attributed to the VRF family if the following methods are defined for the $\mathcal{F}$:

- Gen: $Gen(1^k) \rightarrow (PK, SK)$, where PK is the public key and SK is the secret key;
- Prove: $Prove(x, SK) \rightarrow \pi$, where x is an input and $\pi := \Pi(x, SK)$ is the proof, associated with x and mixed with some random value y, sampled from $\{0, 1\}^{l_{VRF}}$.
- Verify: $Verify(x, \pi, PK) \rightarrow 0|1$, where the output is 1 if and only if $\pi \equiv \Pi(x, SK)$.

The most secure implementations of VRF nowadays are Elliptic Curve Verifiable Random Functions (ECVRFs). Basically, ECVRF is a cryptographic-based VRF that satisfies the uniqueness, collision resistance, and full pseudorandomness properties [43]. The security of ECVRF follows from the decisional Diffie-Hellman assumption in the random oracle model, thus ECVRF is a good source of randomness for a blockchain protocol. Using ECVRF is also cheap and fast, since single ECVRF evaluation is approximately 100 microseconds on x86-64 for a specific curves used in hash functions. Moreover, there is a great UC-extension for batch verification proposed by [44] which make it even faster.

5.2.2 Lottery

Our lottery mechanism is based on ECVRF as a source of randomness and is generally inspired by [45]. The lottery is designed to achieve two main purposes:

select a consensus group dynamically, select a slot leader.

Lottery Function. The main selection logic is implemented in the lottery function. The lottery function $\mathcal{F}_L$ compares a random number y derived from the generated VRF random proof π with publicly known threshold value T. It evaluates to 1 if and only if $y < T$, i.e. $\mathcal{F}_L(state, f, \pi) \rightarrow 0|1$ where *state* is a blockchain state snapshot.

The threshold value is calculated according to the formula $T = 2^{l_{VRF}} \cdot \phi(\alpha, f)$ where $\alpha = s/\sum_{l=0}^{l=M} s_l$ is a relative stake. Consequently, the probability of winning is calculated as $p(\alpha, f) = 1 - (1 - f)^{\alpha}$ Winning probability depends on the participants' relative stake and is adjusted by the free parameter f. This is where the PoS concept comes into play: the bigger the stake, the higher the chance of winning the lottery.

Consensus Group Lottery. The Spectrum protocol initially is running by the manually selected opening consensus group $\{PK_i\}_{i=1}^{M}$ of the predefined size M. Stakeholders interact with each other and with locally installed ideal functionalities $\mathcal{F}_{LB}$, $\mathcal{F}_{VRF}, \mathcal{F}_L$ over a sequence of $L = E \cdot R$ slots $S = \{sl_1, ..., sl_L\}$ consisting of E epochs with R slots each. Let's clarify what the mentioned above pre-defined primitives are needed for:

1. *Ideal Leaky Beacon* $\mathcal{F}_{LB}$: is used to sample an epoch random seed from the blockchain.

2. *Ideal Verifiable Random Function* $\mathcal{F}_{VRF}$: is used as a source of randomness.

3. *Lottery Function* $\mathcal{F}_L$: checks if the protocol participant is a lottery winner (by lottery, we mean either the *consensus group lottery* or the *leader lottery* giving the ability to start a batch notarization round, the lottery function in both cases remains the same, only the arguments matter).

More extended formal definition of $\mathcal{F}_{LB}$ and $\mathcal{F}_{VRF}$ can be found in the original Ouroboros Praos paper [45].

Consensus group is constantly rotated each epoch $e_j > 1$. Any verified protocol participant can try to become a temporal member of consensus group. Participant is verified if his verification key tuple is published and stored in the blockchain for a reliable period of time equals to U_f slots.

At the end of the epoch $e_j > 1$ every verified PK_i requests a new epoch seed η_j from the $\mathcal{F}_{LB}$. When every PK_i evaluates $\mathcal{F}_{VRF}$ and passes the received proof π to the $\mathcal{F}_L$ to reveal the result of the consensus group lottery. To calculate an appropriate threshold T_i^j, $\mathcal{F}_L$ should be parametrized with the same stake distribution which was in the last block used by $\mathcal{F}_{LB}$ to sample the new η_j. Argument of the winning probability function p is $f = M/N$, where M is a number of new consensus group members to select and N is the total number of the verified stakeholders.

Leader Lottery. Once a new consensus group is determined, the lottery process does not stop, but this time the leader of the group should be determined.

During an epoch, for each slot $sl_l \in S$, each member of the consensus group PK_i separately evaluates $\mathcal{F}_{VRF}$ with his own input $x = \eta_j||sl_l||nonce$. In response, he receives the associated random proof π. If $\mathcal{F}_L(state, f, \pi) \rightarrow 1$ then PK_i is a slot leader and he can propose a batch which should be notarized by at least two-thirds of committee members.

The parameter f that regulates the probability of winning is different from the one used in the consensus group lottery. Here, it is the pre-defined value determines how many slots will have at least one selected leader, it is called an active slots coefficient.

The lottery mechanism described in this subsection is fast, secure, and adaptive, since the pre-defined parameters can be changed via the voting process. The same primitives are used to achieve different goals, namely, select a consensus group dynamically and select a slot leader.

Regarding the security it is important to note, that participants use their public VRF-keys for VRF functionality evaluation in the consensus group lottery and secret VRF-keys in the leader lottery. This way, slot leaders don't become publicly known in advance. An attacker can't see who is a slot leader until he initializes batch notarization, thus an attacker can't know who specifically to attack in order to try to subvert a certain slot. Opening consensus group members on the over hand should be known ahead of time for the synchronization. There are hundreds of consensus members in every epoch, so denial of service attacks are difficult to succeed. Grinding attacks are limited because an adversary can't arbitrarily control η_j values, all he can try to do is to make as many forks as possible to estimate the most advantageous, but according to the analysis [45] this advantage doesn't change the security properties of the entire protocol.

5.3 Replacing MACs by Digital Signatures

The main issue with MACs is that any node capable of validating MAC is also capable of generating new messages with valid MACs as the secret key used for MAC generation is also necessary for validation. Digital signatures, on the other hand, use asymmetric protocols for signature generation and signature verification powered by public-key cryptography. A valid secure digital signature for the message can only be generated with the knowledge of the secret key (non-forgery requirement), and verified with the corresponding public key (correctness requirement), and the secret key never leaves the signer's node. The authenticity of the message from the network node can be verified by any party knowing the node public key. Moreover, given the full history of communication, the malicious actor is still not able to forge the new message with valid signature of the node. This gives a way finer control over the set of permissions and provides a strong authentication method.

Spectrum utilizes the specific subset of signatures based on so-called sigma-protocols. The benefits of these protocols are numerous, including the possibility of proving complex logical statements inside the scheme, provable zero-knowledge, and use of standardized and well-established crypto-primitives,

namely conventional cryptographic hash functions and standard elliptic curves with hard discrete logarithm problem. This means the high level of support in the existing chains without modification of the core opcodes or writing supplementary on-chain routines.

5.4 Scalable Collective Signature Aggregation

5.4.1 Problem Statement and Review

In this section we describe our approach to the following problem. The naive approach to writing the consensus values on the blockchain in a verifiable way would be simply write the resulting values together with the signatures from every node which successfully participated in the consensus protocol. Spectrum consensus groups can contain thousands of nodes. If one takes Schnorr signature scheme [46] with 256-bit keys, every signature is 64 bytes long. That means thousands of kilobytes of data needed to be written on the blockchain and consuming valuable storage space, not speaking on the computational efforts from the blockchain validating node to actually verify all these signatures. Therefore, in these circumstances, the signature aggregation method is mandatory.

The aggregation allows one to write a single shorter signature instead of the list of signatures while preserving similar security level. There are few signature aggregation schemes for the sigma-protocol based signatures, such as CoSi [47] and MuSig [48]. These protocols perform extremely well if all the keys of the predefined set of co-signers are included in the resulting signature generation. In this case instead of having thousands of separate signatures one has only one of the size of single Schnorr signature. But this is not the case with many realistic situations with large consensus groups (such as Spectrum). It would be too optimistic to assume that all the nodes are always online, and every single node is following the protocol honestly to every letter. One needs the mechanism to process these failures. Whereas CoSi proposes the method to process such failures, it comes at cost of significant increase in the size of the resulting signature. Our scheme relies on the similar ideas, however we tend to provide better scaling with faulty nodes and more compact constructions than the original CoSi.

In short, we construct a compact aggregated signature scheme with potential node failures based on standard cryptographic primitives. It must have constant small size in the absence of failures and provide reasonably small space and computational overheads in the presence of failures. The signing protocol must be performed in a distributed fashion providing defence from the malicious co-signers.

5.4.2 General Overview

We start with the MuSig scheme and modify it to the meet the criteria listed above. We assume the Discrete Logarithm group to be the subgroup of the elliptic curve as usual. That is, elliptic curve is defined over finite field, we

consider subgroup of its points with coordinates in this field of prime order with fixed generator g and identity element being the point at infinity if the curve is written in the form $y^2 = f(x)$, f is the third degree polynomial. Nothing prevents one from using another group with hard discrete logarithm problem. We use multiplicative notation for the group operation, and the group elements except for generator are written in capital letters. The secret keys are the integers modulo group order, we will denote them by lowercase letters. H is the cryptographic hash function. When we write $H(A, B)$, we assume that there is a deterministic way of serializing the tuple (A, B), and this serialization is used as an argument for H. The public key corresponding to the private key x is the group element $X = g^x$.

Any interactive sigma-protocol consists of three stages in strict order: commitment (when one or more group elements are sent from prover to verifier), challenge (when the random number is sent from verifier to prover), response (when one or more numbers calculated from the previous stages and the secret key are sent from prover to verifier). This triple constitutes the Proof-of-Knowledge of the secret key. To turn the interactive protocol into a non-interactive one, Fiat-Shamir heuristic is used, where the challenge is replaced by the hash value of all the preceding public data.

The takeaways from this setting, which are important for the understanding of our construction are the following:

- In case of n nodes one must have n commitments to aggregate and the list should not be changed till the end of the protocol.

- As the commitments from different nodes come at potentially different time, there can be an attack on this stage. Say, one node does not pick the commitment based on random, but rather calculates it based on the commitments received from the other nodes. This kind of attack is known as k-list attack, as to forge the upcoming signatures the malicious node solves the k-list problem, which is quite possible with a sufficient amount of data. To exclude this possibility one needs all the nodes to "commit to Schnorr commitment" beforehand. One can use hash function with no homomorphic properties for that purpose.

- All the steps are strictly sequential. Hence, every stage must complete with the full aggregation of individual contributions. There does not seem to be a simple way to perform it fully asynchronously.

- Instead of the last step (response) it is sufficient to provide the proof of knowledge for the response. This brings no additional value to the conventional signatures, but it helps with the processing of the node failures during the execution. Namely, the consensus group may demonstrate that somebody in the group knew the discrete logarithms of the commitments not accounted for in the response stage. Therefore, the group as a whole could compute the full response if the failure had not occurred.

- There must be a way to count the failures above, such that the signature verifier could decide whether it tolerates this number or not.

5.4.3 Aggregation Rounds and Structures

Here we list the overall structure of aggregation to give a grasp on the overall process. The detailed explanation is presented below:

Round 1. Collect Commitments for Schnorr commitments. Structure: list of hashes of elliptic curve points. Distribute all the hashes after the aggregation.

Round 2. Collect and aggregate Schnorr commitments. Structures: list of signatures (proofs of discrete logarithms for the commitments) together with Schnorr commitments. Distribute among all the nodes. Upon receiving every node verifies that the hashes of the points are those provided on round 1, and verifies the proofs of discrete logarithms. The commitments with the checks passed are aggregated to get the overall commitment. It is used to compute the challenge and the individual responses in the sigma–protocol.

Round 3. Collect and aggregate the responses. Structure: list of individual responses. Upon receiving every individual response is verified. The responses which passed the verification are added together. If the response is invalid or missing, the corresponding discrete logarithm proof from round 2 is appended to the output.

Output. Aggregate signature (Y, z) together with the set

$$\{(Y_i, DlogProof(Y_i)\},$$

where i runs over the set of nodes which have not provided valid responses.

5.4.4 Signature Generation

The signature generation algorithm is as follows:

1. Each signer computes $a_i \leftarrow (X_1, X_2, \ldots, X_n; X_i)$ and the aggregate public key $\tilde{X} \leftarrow \prod_i X_i^{a_i}$.
2. Each signer generates a pair $Y_i = g^{y_i}$ to commit to, commitment $t_i \leftarrow H(Y_i)$ and the signature σ_i of some predefined message with secret key y_i.
3. The commitments t_i are aggregated in the list L_1.
4. After every participating co–signer received L_1, the tuples (Y_i, σ_i) are aggregated in the list L_2.

5. Upon receiving the tuple (Y_i, σ_i), verify $t_i = H(Y_i)$, and verify that σ_i is a valid signature corresponding to Y_i. The failed records are excluded from L_2, the next steps and communication round.
6. Every node computes the aggregate commitment $Y = \prod_i Y_i$ using all the valid records in L_2.
7. Every node computes the challenge $c \leftarrow H(\tilde{X}, Y, m)$ and the responses $z_i \leftarrow y_i + ca_i x_i$.
8. The responses z_i are aggregated into list L_3.
9. Initialize $z \leftarrow 0$ and empty set $R \leftarrow \{\}$.
10. Upon receiving the response z_i, verify that $g^{z_i} = Y_i X_i^{a_i c}$.
11. If this is the case, set $z \leftarrow z + z_i$. Otherwise, insert corresponding entry from L_2 in R as (i, Y_i, σ_i).
12. Output the triple (Y, z, R).

5.4.5 Signature Verification

The signature verification is carried out as follows:

1. Compute $a_i \leftarrow H(X_1, X_2, \ldots, X_n; X_i)$.
2. Compute $\tilde{X} \leftarrow \prod_i X_i^{a_i}$.
3. Compute $X' = \prod_{i \notin R.0} X_i^{a_i}$.
4. Compute $Y' = \prod_{i \in R.0} Y_i$.
5. Compute $c \leftarrow H(\tilde{X}, Y, m)$.
6. Verify $g^z = X'^c Y Y'^{-1}$.
7. Verify all of $\sigma_i \in R.2$ with respect to $Y_i \in R.1$.
8. Compare $(n - k)$ (where k is the size of R) with the required threshold value.

5.5 Instantiation of Signature Aggregation

We instantitate our signature aggregation protocol on top of Handel [49], a Byzantine-tolerant aggregation protocol that allows for the quick aggregation of cryptographic signatures over a WAN. Handel has polylogarithmic time, communication and processing complexity.

Our signature aggregation protocol involves aggregation of three lists: $L1$, $L2$ and $L3$. As long as Handel requires that the partial aggregation function satisfies both commutativity and associativity conditions we have to replace

lists with sets. We instantiate each of three aggregation rounds on top of Handel round. Because of parallel nature of Handel we have to run multicasting between chained rounds of aggregation in order to consistently aggregate. The resulted construction consists of three Handel rounds and two multicasting rounds in between.

5.6 Eliminating Validator Bottleneck

So far, each member of the consensus group had to track changes on all connected chains in order to participate in consensus properly. However, this approach reduces the number of possible consensus participants and limits the scalability of the system. Therefore, for the optimal design of our consensus protocol, we will use the following observations:

> **Observation 1:** Events coming from independent systems S_k are not serialized.
>
> **Observation 2:** Outbound transactions on independent systems S_k can be independently signed.

Utilizing those properties, we now introduce committee sharding. We modify the protocol in a way such that at each epoch e_n, K distinct committees consisting of nodes equipped with functionality unit F_{L_k} relevant to a specific connected system S_k are selected via the consensus group lottery. All primitives used in the lottery are equal for different committees, however, lotteries are independent.

We denote one such committee shard as V_n^k, which uniquely maps to S_k. Then, complete mapping of committees to chains at epoch e_n can be represented as a set of tuples committee-chain $\{(V_n^k, S_k)\}$. Throughout epoch e_n all events and on-chain transactions in S_k are handled exclusively by V_n^k. Nodes in V_n^k maintain a robust local ledger L_k^{local} of notarized batches of events observed in S_k.

5.6.1 Syncing Shards

Each committee V_n^k forms the notarized batches of events and adds them into their local ledgers L_k. All these batches should be periodically synced and added to a block of the main super ledger L^+ in order for the system to be able to compute a cross-chain state transition. To facilitate this process, batches of the notarized events should be broadcast to other committees. The main actors at this stage are:

1. *Local leader*: local committee leader.
2. *Relayer*: any protocol participant that broadcasts notarized batches to the local leader and to other committees' members. Every local leader can be a relayer at the same time.

3. *General leader*: one of the local leaders who added a block consisted of collected notarized batches and other internal transactions to the L^+.

There is no separate lottery for the general leadership and any local leader is able to publish his block to L^+, thus, he can choose from two main strategies:

1. *Wait*: malicious strategy where local leader waits for broadcasts from other committees members and doesn't broadcast his own batch to eliminate competitors for adding a block.

2. *Broadcast and wait*: fair strategy where local leader immediately broadcasts his batch, waits for broadcasts from other committees' and then competes honestly for adding a block.

There should be a motivation for an individual local leader to choose the fair strategy instead of keeping his batch for too long and there also should be a motivation for every committee member to act as a relayer. This is achieved through the design of the incentive system.

5.6.2 Incentives

There are three types of the incentive for the Spectrum protocol participants: $\{R_b, R_d, R_m\}$, where R_b is a guaranteed reward for adding a notarized batch to the block, R_d is given for broadcasting a batch to the general leader and R_m is given personally to the general leader who will finally add the block. Delivery reward R_d is given if and only if a delivery was made within a predetermined period of time Δt_d.

Reward amounts are initially configured in such a ratio that if $R_d = 0$ there is no prior strategy for local leaders, they will either wait for other batches or broadcast their batches with equal probability. At the same time, all other committee members are motivated to act as a relayers to receive an extra reward, since the notarized batch can be firstly generated by any member of the committee. All the rewards except R_m are shared equally between all committees members whose signatures are included in the finally added block.

As a result, the syncing shards flow looks as follows:

1. After notarization, a committee member holding the notarized batch which contains the local notarization time, sends it to his local leader and to other known committees members.

2. All committees members who receive notarized batches from other committees also send them to the local leader.

3. The local leader collects the received notarized batches.

4. When waiting time approaches Δt_d, the local leader forms and broadcasts a block consisting of all external collected batches and batches from the local L_k that have not yet been added to L^+.

5. After block is reliably settled in the L^+, all associated participants can claim their rewards.

We also introduce another type of authority incentive that increases the chances of participants to be selected in the consensus group lottery. When calculating the stake distribution which is needed to parametrise the lottery function, all stakes are weighed depending on the actions of their holders in the previous epoch, i.e. $s_i = A_m \cdot s_i^{real}$, where A_m is an authority multiplier. If some authority was a member of the previous committee and participated in adding of $2/3$ of the blocks produced in the considered period of time, then his actual stake s_i^{real} is multiplied by $A_m = 2$. Multiplier A_m decreases linearly to 0, which is the case where member was passive during the entire epoch.

With this mechanism, we solve the following problems:

- Members are motivated to be focused on cooperation with other committees so that their participation is reflected in each block added in the L^+.
- Inactive and dishonest members are automatically excluded from the next epoch committee.
- Participants are motivated to stay active throughout the entire epoch so that their chances of being selected in the committee don't decrease due to an authority multiplier $A_m < 1$, otherwise, in order to even the odds with new lottery participants, they will either have to increase their real stake or skip the lottery until the next one.

5.6.3 Forks and Integrity

Protocol flow implies that there can be a several local leaders in every connected S_k committee, which leads to forks. This type of fork is a normal part of the protocol lifecycle, however, total possible number of the normal forks in our protocol is greater than in other blockchains, since any of the local leaders can append their blocks to L^+. The chance of occurring a malicious forks produced by an adversary is minimized due to the lottery and the incentive mechanism design. In addition, the task for an adversary becomes more difficult by virtue of the interaction between the protocol participants during the syncing shards process.

For the above reasons, the main rules for resolving forks are simple and are followed by members of all committees when validating a proposed blocks:

1. *Max valid*: choose the longest appropriate chain given a set of valid chains that are available in the network.
2. *Max stake*: if the max valid rule doesn't resolve a slot battle, then the valid chain chooses according to the real stake size of the battled leaders, the maximum stake is the winner. Stake distribution is picked from the actual blockchain snapshot for the current committee.

However, a large number of forks still significantly affect properties that maintain the integrity of the L^+:

1. *Latency*: the number of elapsed slots required for a transaction to appear in a block on the L^+.
2. *Finality*: the number of elapsed slots required for a transaction to become settled and immutable.

The latency of the protocol is good enough due to the short duration of the slots, while the finality, as a result of the functional features of our protocol, depends on the connected S_k integrity properties.

Most ledgers do not guarantee instant finality of transaction, that means that any (or all) transactions may not be applied to the corresponding S_k ledgers in the end. Different blockchains has different finality parameters, and the Spectrum finality time U_f should be greater than all of them. Thus, the U_f should be set with a margin and, therefore, using the number of slots Δsl that have passed in the Spectrum network, developers should be able to receive information about the number of blocks that have passed in any connected blockchain during this period of time. The duration of block creation in each S_k is different, but the average values are preserved for a certain period of time $\Delta T >> d_s$, where d_s is the duration of Spectrum's slot. Thus, after each ΔT time interval, Spectrum network will update the set of constants: $(\{d_k, U_k\}_{r=1}^{K})$, where d_k is a block duration in the S_k and r_k is the default reliable number of confirmations in the S_k and K is the total number of the connected systems.

Using the data above, each Spectrum's Δsl can be associated with the delta of blocks that have passed in any connected blockchain: $\{\lfloor \Delta sl \cdot d_s \,/\, d_k) \rfloor\}_{k=1}^{K}$. When forming transaction, developers can specify a reliability factor r^*. This factor will be compared with the ratio of the number of blocks passed on the associated S_k to the default reliable number of confirmations r_k of this system.

The ability to access this information is important for tracking the status of value carrying units in the Spectrum's global state. The aspects of the implementation of our ledger is described in detail in the next section.

5.7 Decentralized On-Chain Asset Management

In order to lift on-chain assets to cross-chain level, Spectrum has to take control over them. Thus, all assets that Spectrum operates on are stored in on-chain *vaults* which are ruled by the consensus.

Each vault corresponding to the connected system S_k stores an epoch number n, an aggregated public key αPK_n^k of the current validator set V_n^k and is guarded with a smart-contract capable of performing an aggregated signature verification $verify : (\sigma_n^k, \alpha PK_n^k, m_n^k) \rightarrow 0|1$.

5.7.1 Rotating Authorized Committees in Vaults

As explained before, committees in Spectrum are constantly rotated. Thus, vaults have to be updated accordingly. The transition is performed with the

help of "retiring" committee, that must call $changeEpoch : (\sigma_n^k, \alpha PK_{n+1}^k)$ on the vault contract, where αPK_{n+1}^k is the aggregated public key of the next commitee.

5.8 Ledger

Spectrum's global state includes a pool of value carrying units called *cells*. A *Cell* encodes monetary value (e.g., fungible or non-fungible tokens) travelling inside the system and across its boards.

$$\begin{aligned} \text{TxId} &= \text{H}(\text{Tx}) \\ \text{CellId} &= \text{H}(\text{TxId} \times \text{I}) \end{aligned}$$

Each cell has a unique identifier derived from ID of the transaction that produced the cell and its index in the transaction outputs. The identifier remains stable even when cell is modified as we explain below.

$$\begin{aligned} \text{Value} &= \texttt{u64} \\ \text{ChainId} &= \texttt{u64} \\ \text{Version} &= \texttt{u64} \\ \text{ProgressPoint} &= \text{ChainId} \times \texttt{u64} \\ \text{ActiveCell} &= \text{CellId} \times \text{Address} \times \text{Value} \times \text{Version} \\ \text{BridgeInputs} &= [\texttt{u64}] \\ \text{Destination} &= \text{ChainId} \times \text{BridgeInputs} \\ \text{TermCell} &= \text{CellId} \times \text{Value} \times \text{Destination} \\ \text{Cell} &= \text{ActiveCell} \uplus \text{TermCell} \end{aligned}$$

We distinguish two essential types of cells depending on the state of the value they encode.

5.8.1 Active cells

Active Cell is a value travelling between owners inside the system. An Active Cell can be modified while preserving its original stable identifier. With each mutation version of the cell is incremented which is initialized with `0` when the cell is created. This opens the door for smooth management of shared cells (e.g., stablecoin bank or liquidity pool).

5.8.2 Authenticators, Addresses and Ownership

$$\begin{aligned} \text{Authenticator} &= \text{ProveDlog} \uplus \text{Script} \\ \text{Address} &= \text{H}(\text{Authenticator}) \end{aligned}$$

Each active cell has an exclusive owner identified by an address. Address is derived from an authenticator by applying collision resistant hash function to it. To prove ownership of a cell a party must supply an authenticator whose hash matches the owning address. An authenticator can either be a public key or a script. Once authenticated an owner can freely move value locked within the cell by either mutation or elimination of it.

5.8.3 Terminal cells

Terminal cells encode value to be exported into an external system. In contrast to active cells, terminal cells are immutable and value from them cannot be moved within the system anymore.

5.8.4 Transactions and Effects

$$
\begin{aligned}
\text{Imported} &= \text{ActiveCell} \\
\text{Exported} &= \text{CellId} \\
\text{Revoked} &= \text{CellId} \\
\text{Progressed} &= \text{ProgressPoint} \\
\text{Eff} &= \text{Imported} \uplus \text{Exported} \uplus \text{Revoked} \uplus \text{Progressed}
\end{aligned}
$$

Global pool of cells is modified by atomic state modifiers called *Effects* and *Transactions*.

Effects are state transitions imported from external systems exclusively by local committees. Below we list possible effects:

1. Import of value. A deposit into one of Spectrum's on-chain vaults which results in creation of a new cell.

2. Export of value. An outbound transaction that transfers value from Spectrum's on-chain vault to user address on particular blockchain.

3. Revocation of previously imported value due to roll-back on the source chain.

4. Signalisation that external system reached particular progress point.

$$
\begin{aligned}
\text{CellRef} &= \text{CellId} \times \text{Version} \\
\text{Inputs} &= \text{CellRef} \times [\text{CellId} \uplus \text{CellRef}] \\
\text{RefInputs} &= [\text{Cell}] \\
\text{EvaluatedOutputs} &= [\text{Cell}] \\
\text{Tx} &= \text{Inputs} \times \text{RefInputs} \times \text{EvaluatedOutputs}
\end{aligned}
$$

In contrast to effects, transactions are state transitions triggered by Spectrum users. A transaction accepts cells that it wants to mutate or eliminate as inputs and outputs new cells or upgraded versions of mutated cells. Therefore, scope of transaction is restricted to its inputs and outputs.

Transactions: Referencing inputs. Transaction can reference cells to use as inputs either by cell ref (fully qualified reference) or only by stable identifier. In the latter case, a concrete version of the cell with the given stable identifier will be resolved in the runtime of the transaction. Importantly, each transaction must have at least one fully qualified input, this guarantees that each transaction is unique.

Transactions: Programmability. Some outputs may be computed in the runtime of a transaction as a result of script(s) execution. It is also possible to include pre-evaluated outputs into transaction in order to save on on-chain

computations. This design allows dApp developers to choose the amount of on-chain computations of their apps.

5.8.5 Dealing with finality of imported value

Beacause Spectrum is a cross-chain system, monetary value there is usually imported from an external system (e.g. Cardano or Ergo). Since most of the cryptocurrencies don't provide instant finality of transactions, on-chain transaction that once imported value into Spectrum's on-chain vault may be rolled-back. There are two ways of preventing "dangling" value inside Spectrum. On the one end of spectrum is a conservative approach: wait for settlement on the source chain (e.g. 120 blocks in Ergo) before import to be 100% sure the transaction will not be rolled back. On the other end is a reactive approach: import value immediately and revert locally transactions that depend on that piece of value in the case of rollback. Conservative approach offers simplicity and is cheaper to execute, while reactive one allows to work with imported value inside spectrum with minimal delays.

Observation: Probability of a rollback at a certain height decreases exponentially with square root scaling in the exponent as chain extends [45].

Based on this observation we choose a hybrid approach. Value is imported with a small delay D^c which is configured for each chain and is sufficient to keep probability of rollback low. If rollback happens after the import all transactions directly or transitively depending on the dangling value are reverted.

As long as outbound transactions can not be reverted it is of paramount importance to wait for complete settlement of the imported value before allowing to export it. Each cell is associated with a set of dependencies called *ancors* represented as unique identifier of a chain and a height which the chain is required to reach in order for the anchor to be deemed as *unancored*. Active ancors leak from cells in inputs into created cells in outputs. It is impossible for a terminal cell to be exported until all ancors it depends on are reached.

5.9 Protocol Flow

Let's summarize all of the above and describe the full flow of the Spectrum protocol. Protocol is running by a set of manually selected opening consensus groups $\{V_1^k\}_{k=1}^K$ for K connected distributed systems $\{S_k\}_{k=1}^K$. Each group consists of at least M stakeholders interacting with each other and with the ideal functionalities $\mathcal{F}_{LB}$, $\mathcal{F}_{VRF}$, $\mathcal{F}_L$, $\mathcal{F}_{KES}$, $\mathcal{F}_{SIG}$ over a sequence of $L = E \cdot R$ slots $S = \{sl_1, ..., sl_L\}$ consisting of E epochs with R slots each. The above not previously mentioned functionality $\mathcal{F}_{KES}$ is a key evolving signature scheme that is used by the current leader to sign a new block. Functionality $\mathcal{F}_{SIG}$ implements the presented aggregated signature scheme logic. Also, each protocol participant maintains at least one functionality unit F_{S_k} that allows him to interact with the connected S_k.

5.9.1 Bootstrapping

The system is bootstrapped in a trusted way. All members of $\{V_1^k\}_{k=1}^K$ committees perform the following procedure:

1. On-chain vaults are initialized with an aggregated public key aPK_1^k of the initial committee.

2. All committee V_1^k members i.e. $\forall PK_i \in V_1^k$ must generate the tuple of verification keys $(v_i^{vrf}, v_i^{kes}, v_i^{dsig})$, using the ideal functionalities $\mathcal{F}_{VRF}$, $\mathcal{F}_{KES}$, $\mathcal{F}_{DSIG}$ instances running on their machines.

3. Full set of the verification keys tuples $V_{init} = \{(PK_i, v_i^{vrf}, v_i^{kes}, v_i^{dsig})\}_{i=1}^M$ with the initial stakes $\{s_i\}_{i=1}^M$ must be stored in the blockchain and acknowledged by all members of the initial consensus group (meaning members of all K committees and a full set is $\{V_{init\_k}\}_{k=1}^K$).

4. Functionality $\mathcal{F}_{LB}$, parameterized with the confirmed $\{V_{init\_k}\}_{k=1}^K$ is evaluated independently by every participant to sample an initial random seed value $\eta \leftarrow \{0,1\}^\lambda$.

5. Finally, all approved stakeholders should agree on the genesis block $B_0 = \big(\{V_{init\_k}\}_{k=1}^K, \{stakes\_k\}_{k=1}^K, \eta\big)$.

5.9.2 Normal Flow

Once the system is bootstrapped, the Spectrum protocol will run in a normal flow:

1. **Registration**. Any Spectrum stakeholder can register to become a committee member of his local system S_k. To get a chance to be included in the set of validators V_n^k of the epoch e_n participant PK_i should register in the lottery during the epoch e_{n-2} by publishing his verification tuple $(v_i^{vrf}, v_i^{kes}, v_i^{dsig})$ into the L^+. Once number of slots corresponding to the Spectrum's finality time U_f has elapsed, the participant is considered as verified.

2. **Consensus Group Lottery**. At the end of the epoch $e_{n-1} > 2$ every verified PK_i receives new epoch seed η_n from the $\mathcal{F}_{LB}$. When every PK_i evaluates $\mathcal{F}_{VRF}$ with the input, which includes new η_n and passes the received proof π_i to the $\mathcal{F}_L$. Function $\mathcal{F}_L$ is parameterized with the S^k lottery parameters and uses the same stake distribution which was in the last block used by $\mathcal{F}_{LB}$ to calculate the threshold. Also, all participants must multiply all stakes by the corresponding multipliers, calculated from the activity of participants in the previous epoch. If successful, i.e. $\mathcal{F}_L$ returns 1, then PK_i is a member of V_n^k. Functionality $\mathcal{F}_{LB}$ is parameterized with the history, including blocks with release times up to $-U_f$ from the actual slot. Therefore, even in case of a rollback, the currently selected members of the consensus group remain legitimate.

3. **Committee key aggregation**. Once the new committee is selected, nodes in the V_n^k aggregate their individual public keys PK_i into a joint one aPK_n^k, which is needed to sign the batch applying transactions with the external events: inbound value transfers, outbound value transfers, boxes eliminations.

4. **Committee transition**. Nodes in the V_{n-1}^k publish cross-chain message $m_n^k : (aPK_n^k, \sigma_{n-1}^k)$, where σ_{n-1}^k is an aggregated signature such that $verify : (\sigma_{n-1}^k, aPK_{n-1}^k, m_n^k) = 1$. Finally, vaults are updated such that $vault^k\{(e_{n-1}, aPK_{n-1}^k)\} \coloneqq (e_n, aPK_n^k)$.

5. **Chain extension**.

   5.1. Every online V_n^k member collects existing chains from L^+ and verifying that for every chain, every block, produced up to U_f blocks before contains correct data about the corresponding slot sl' leader PK'. To verify a valid slot leader, responses from the $\mathcal{F}_{VRF}$ and $\mathcal{F}_L$ with the relevant inputs must equal 1. Leader PK' must be also a member of the legitimate committee. All forks are resolved by the rules of the longest chain and the largest stake in the corresponding priority.

   5.2. During the epoch, for every slot sl every committee V_n^k member PK_i separately evaluates $\mathcal{F}_{VRF}$ with his own input $x = \eta_n||sl||nonce$İf successful, $\mathcal{F}_L$ returns 1 and PK_i is the slot sl leader. Leader evaluates $\mathcal{F}_{VRF}$ one more time with the input $x' = \eta_n||sl||test$. The associated proofs π_i and ρ_i are included in the block, which will be added to the L^+. Random proof ρ_i will be used by $\mathcal{F}_{LB}$ to sample a random seeds for next epochs.

   5.3. All committee V_n^k members observe events in their systems S_k and in the $L+$ mempool. If PK_i is a slot sl leader, then he is able to propose a batch b^* of events observed in S_k, which should be notarized by other members of the V_n^k with an aggregated signature using $\mathcal{F}_{SIG}$ and then added to the local ledger L_k.

   5.4. Notarized batch b^* can first be formed by any member of the V_n^k. The batch must be immediately sent to the leader who initiated its notarization and to the members of other committees. After the leader receives enough batches he forms a block B^* consisting of all external collected batches and batches from the local L_k that have not yet been added to L^+. He includes all the leadership proofs, signs it with $\mathcal{F}_{KES}$, and broadcasts it to all committees.

   5.5. After the finality time U_f is passed since B^* settlement in the L^+, all members of all committees that participated in the formation of the block B^* can claim their rewards.

In this way, the Spectrum protocol reaches consensus and implements the cross-chain interoperability. The solution is fairly decentralized, fast and scalable, thus, it can be used in a large number of applications and scenarios.

6 Applications

6.1 Decentralized Cross-Chain Oracle

The system is capable of providing a notarized set of events observed in supported external system be it a blockchain or general data source(s). Cross-Chain Oracle is simple yet opens the door for interoperability for all dApps on Layer1.

6.2 Custodial Asset Management

In custodial mode of operation the system is capable of managing user assets which are stored on corresponding blockchains in vaults which were defined previously.

Natively Cross-Chain Applications Decentralized custodial management in conjunction with a computational layer can be highly beneficial for expanding the capabilities of the system. This moves us beyond simple bridges to what we call Natively Cross-Chain Applications (NCCAs).

NCCAs are applications that are deployed in cross-chain network and are capable of interacting with other blockchains without the need of external oracles or bridges. Compared to single-chain dApps, NCCAs unlocks an additional functionality by taking advantage of multiple chains simultaneously. They make it possible to aggregate fragmented liquidity on different chains into one chain or a coordinated pool of assets and improve the user experience by enabling the localization and customization of parameters and feature sets of the same application on different chains. These unique advantages make them the future of web3 dApps.

References

[1] Dr Miraz and David Donald. *Atomic Cross-chain Swaps: Development, Trajectory and Potential of Non-monetary Digital Token Swap Facilities*. Jan. 2019. DOI: 10.33166/AETiC.2019.01.005.

[2] Stefan Schulte et al. *Towards Blockchain Interoperability*. 2019.

[3] Soohyeong Kim, Yongseok Kwon, and Sunghyun Cho. *A Survey of Scalability Solutions on Blockchain*. Oct. 2018. DOI: 10.1109/ICTC.2018.8539529.

[4] Vitalik Buterin. *Chain Interoperability*. 2016. URL: https://allquantor.at/blockchainbib/pdf/vitalik2016chain.pdf.

[5] Rafael Belchior et al. *A Survey on Blockchain Interoperability: Past, Present, and Future Trends*. 2021. arXiv: 2005.14282 [cs.DC].

[6] Gang Wang. *SoK: Exploring Blockchains Interoperability*. Cryptology ePrint Archive, Paper 2021/537. https://eprint.iacr.org/2021/537. 2021. URL: https://eprint.iacr.org/2021/537.

[7] Randhir Kumar and Rakesh Tripathi. *Content-Based Transaction Access From Distributed Ledger of Blockchain Using Average Hash Technique*. Jan. 2021. DOI: 10.4018/978-1-7998-3295-9.ch003.

[8] Babu Pillai, Kamanashis Biswas, and Vallipuram Muthukkumarasamy. *Blockchain Interoperable Digital Objects*. June 2019. DOI: 10.1007/978-3-030-23404-1_6.

[9] Damiano Di Francesco Maesa and Paolo Mori. *Blockchain 3.0 applications survey*. 2020. DOI: https://doi.org/10.1016/j.jpdc.2019.12.019. URL: https://www.sciencedirect.com/science/article/pii/S0743731519308664.

[10] D. Balazs. *Herdius whitepaper*. 2017. URL: https://herdius.com/whitepaper/HerdiusTechnicalPaper.pdf.

[11] Eder John Scheid et al. *Bifröst: a Modular Blockchain Interoperability API*. Oct. 2019. DOI: 10.1109/LCN44214.2019.8990860.

[12] S. Thomas and E. Schwartz. *A protocol for interledger payments*. 2015. URL: https://interledger.org/interledger.pdf.

[13] Reza M. Parizi et al. *Integrating Privacy Enhancing Techniques into Blockchains Using Sidechains*. 2019. DOI: 10.1109/CCECE.2019.8861821.

[14] Amritraj Singh et al. *Sidechain technologies in blockchain networks: An examination and state-of-the-art review*. 2020. DOI: https://doi.org/10.1016/j.jnca.2019.102471. URL: https://www.sciencedirect.com/science/article/pii/S1084804519303315.

[15] *Intro to loom network — loom sdk*. 2019. URL: https://loomx.io/developers/en/intro-to-loom.html.

[16] Jonas David Nick. *Liquid: A Bitcoin Sidechain*. 2020.

[17] *Poa-network-whitepaper*. 2018. URL: https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper.

[18] J. Chow. *Btc relay*. 2016. URL: http://btcrelay.org/.

[19] N. Rush L. Luu and N. Lin. *Peacerelay: Connecting the many ethereum blockchains*. 2019.

[20] *Hyperledger cactus whitepaper*. 2020. URL: https://github.com/hyperledger/cactus.

[21] Philipp Frauenthaler et al. *Testimonium: A Cost-Efficient Blockchain Relay*. Feb. 2020.

[22] Iddo Bentov et al. *Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware*. Cryptology ePrint Archive, Paper 2017/1153. https://eprint.iacr.org/2017/1153. 2017. URL: https://eprint.iacr.org/2017/1153.

[23] Jeff Burdges et al. *Overview of Polkadot and its Design Considerations*. Cryptology ePrint Archive, Paper 2020/641. https://eprint.iacr.org/2020/641. 2020. URL: https://eprint.iacr.org/2020/641.

[24] J. Kwon and E. Buchman. *Cosmos whitepaper*. 2019. URL: https://v1.cosmos.network/resources/whitepaper.

[25] *Wanchain: Building super financial markets for the new digital economy*. 2017. URL: https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf.

[26] *Ark ecosystem whitepaper*. 2019. URL: https://ark.io/Whitepaper.pdf.

[27] *Quant overledger whitepaper*. 2018. URL: https://uploads-ssl.webflow.com/6006946fee85fda61f666256/60211c93f1cc59419c779c42_Quant_Overledger_Whitepaper_Sep_2019.pdf.

[28] Zhuotao Liu et al. *HyperService: Interoperability and Programmability Across Heterogeneous Blockchains*. London, United Kingdom, 2019. DOI: 10.1145/3319535.3355503. URL: https://doi.org/10.1145/3319535.3355503.

[29] Gang Wang et al. *SMChain: A Scalable Blockchain Protocol for Secure Metering Systems in Distributed Industrial Plants*. Cryptology ePrint Archive, Paper 2019/1401. https://eprint.iacr.org/2019/1401. 2019. DOI: 10.1145/3302505.3310086. URL: https://eprint.iacr.org/2019/1401.

[30] Eder J. Scheid et al. *PleBeuS: a Policy-based Blockchain Selection Framework*. 2020. DOI: 10.1109/NOMS47738.2020.9110386.

[31] Enrique Fynn, Alysson Bessani, and Fernando Pedone. *Smart Contracts on the Move*. June 2020. DOI: 10.1109/DSN48063.2020.00040.

[32] *Interledger protocol v4*. 2020. URL: https://interledger.org/rfcs/0027-interledger-protocol-4/.

[33] Aleksei Pupyshev et al. *Gravity: a blockchain-agnostic cross-chain communication and data oracles protocol*. July 2020.

[34] Aleksei Pupyshev et al. *SuSy: a blockchain-agnostic cross-chain asset transfer gateway protocol based on Gravity*. Aug. 2020.

[35] Aggelos Kiayias et al. *Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol*. Cryptology ePrint Archive, Paper 2016/889. https://eprint.iacr.org/2016/889. 2016. URL: https://eprint.iacr.org/2016/889.

[36] Alexei Zamyatin et al. *SoK: Communication Across Distributed Ledgers*. Cryptology ePrint Archive, Paper 2019/1128. https://eprint.iacr.org/2019/1128. 2019. URL: https://eprint.iacr.org/2019/1128.

[37] Miguel Castro. *Practical Byzantine Fault Tolerance*. Apr. 2001.

[38] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. May 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[39] Eleftherios Kokoris-Kogias et al. *Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing*. 2016. arXiv: 1602.06997 [cs.CR].

[40] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012.

[41] Yossi Gilad et al. *Algorand: Scaling Byzantine Agreements for Cryptocurrencies*. Cryptology ePrint Archive, Paper 2017/454. https://eprint.iacr.org/2017/454. 2017. URL: https://eprint.iacr.org/2017/454.

[42] Silvio Micali, Salil Vadhan, and Michael Rabin. *Verifiable Random Functions*. USA, 1999.

[43] Moni Naor and Asaf Ziv. *Primary-Secondary-Resolver Membership Proof Systems*. Cryptology ePrint Archive, Paper 2014/905. https://eprint.iacr.org/2014/905. 2014. URL: https://eprint.iacr.org/2014/905.

[44] Christian Badertscher et al. *On UC-Secure Range Extension and Batch Verification for ECVRF*. Cryptology ePrint Archive, Paper 2022/1045. https://eprint.iacr.org/2022/1045. 2022. URL: https://eprint.iacr.org/2022/1045.

[45] Bernardo David et al. *Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol*. Cryptology ePrint Archive, Paper 2017/573. https://eprint.iacr.org/2017/573. 2017. URL: https://eprint.iacr.org/2017/573.

[46] Claus Schnorr. *Efficient signature generation by smart cards*. Jan. 1991. DOI: 10.1007/BF00196725.

[47] Ewa Syta et al. *Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning*. May 2016. DOI: 10.1109/SP.2016.38.

[48] K. Itakura. *A public-key cryptosystem suitable for digital multisignatures*. 1983.

[49] Olivier Bégassat et al. *Handel: Practical Multi-Signature Aggregation for Large Byzantine Committees*. 2019. arXiv: 1906.05132 [cs.DC].