24.04.09 Bridge approach and BLS cost

Xun Zhang Bingsheng Zhang
Zhejiang University, CHN
22221024@zju.edu.cn bingsheng@zju.edu.cn

April 09 2024

1 Succinct labs Approach

1.1 Succinct 2022

- **Ethereum PoS Light Clients**. The Altair hard-fork to the Ethereum beacon chain introduced the **sync committee protocol** to allow for very compute-efficient light clients. The sync committee is a group of 512 validators assigned randomly from the full set of Beacon chain validators according to the RANDAO. The sync committee rotates every 256 epochs (roughly once every 27 hours). It's important the group of validators in the sync committee rotates periodically for security reasons.

  An honest validator chosen for sync committee participation is responsible for signing blocks. For each block that the sync committee signs, one member of the sync committee subset is responsible for producing an aggregate BLS signature and bitmap of the participating committee members, to produce a **SyncAggregate** container with this information, which gets gossiped at the p2p layer.

  There are 3 types of updates:

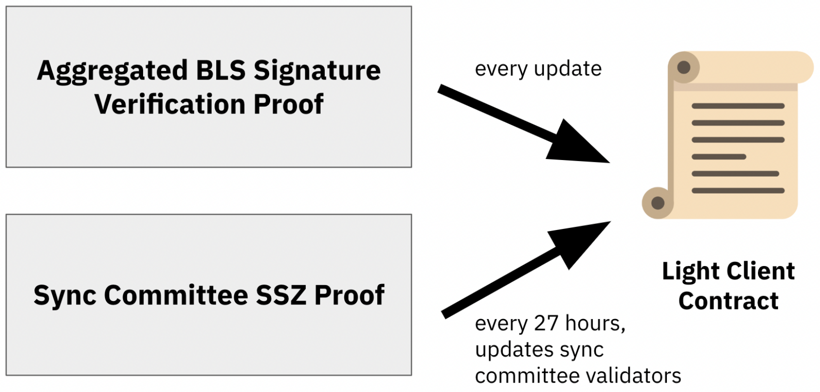
  - **LightClientOptimisticUpdate** contains a block header that has not yet been finalized along with the corresponding SyncAggregate with the aggregate BLS signature and bitmap.
  - **LightClientFinalityUpdate** contains an attested header that the sync committee has signed along with a finalized header that has been finalized according to the Casper FFG finality gadget. The update contains a merkle proof of the finalized header in the attested header.
  - When the next sync committee is known (at least 1 epoch ahead of time) the light client receives a **LightClientUpdate**, which contains an attested header that the sync committee signs, along the next SyncCommittee and a merkle proof that the next sync committee is contained in the attested header.

- **Existing Problem**. Recall that the validator set of the sync committee rotates every 27 hours. On chain, we keep track of a commitment to the set of validators in the mapping **syncCommitteeRootByPeriod**. To update this mapping for the next period, we verify the merkle inclusion proof that the current validator set signs for the commitment for the next validator set.

  Unfortunately, the commitment the validators sign is an **SSZ commitment** (simple serialization, Eth PoS serialization format) that is quite SNARK unfriendly, as it uses the **SHA-256** hash function. It takes 70 million constraints in a Groth16 circuit to compute the serialization of 512 validator BLS public keys to its corresponding SSZ commitment.

  To address this, we use an additional SNARK to provably map an SSZ commitment to a SNARK-friendly **Poseidon** commitment. For each BLS signature verification, we pass in the poseidon commitment of the sync committee validators as public input to ensure that the BLS signature we are verifying is from the correct public keys. The commitment mapping SNARK must only be run every sync committee period (roughly once every 27 hours).

- **ZK-SNARK**. Thus there are two proof task: **Aggregated BLS Signature Verification** and **Sync Committee Commitment Mapping**.


Aggregated BLS Signature Verification Proof
every update
Sync Committee SSZ Proof
every 27 hours, updates sync committee validators
Light Client Contract

  (Succinct labs choose **Circom** and **Groth16** to generate zk-snarks)

  - **Aggregated BLS Signature Verification**. To optimize the gas costs of the on-chain light client, we only store on-chain a SNARK-friendly commitment to the current set of public keys that govern the sync committee (as opposed to storing all 512 public keys in smart contract storage, which would be quite expensive).

    The SNARK-friendly commitment is not normally available in the block headers, so we use the ZK-SNARK described below to map the SSZ commitment that is normally available in the block header to a more SNARK-friendly commitment that uses the **Poseidon** hash function.

– **Sync Committee Commitment Mapping**. This SNARK takes in a signed block header and maps the nextSyncCommitteeRoot field (the SSZ hash of the next validator set) to a SNARK-friendly **Poseidon** hash of the validator set.

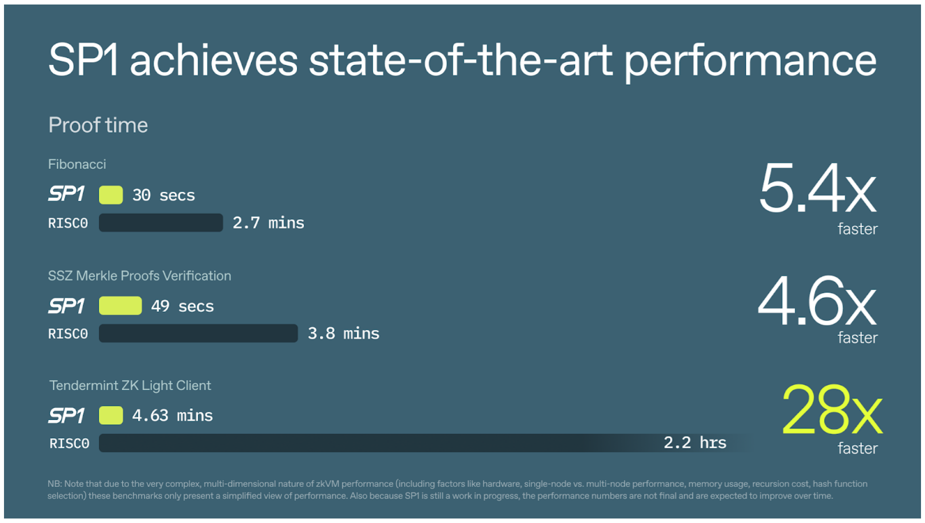
  We store this SNARK-friendly commitment on-chain and use it as an input to the BLS signature verifcation SNARK to ensure that the signature we are verifying is indeed from the sync committee validators. This circuit runs every 27 hours, for every sync committee rotation.

1.2 Succinct 2024

Use SP1 to construct Zk bridge.

”*We're excited to announce Succinct Processor 1 (SP1): our first-generation* ***zero-knowledge virtual machine (zkVM)*** *that verifies the execution of arbitrary Rust (or any LLVM-compiled language) programs. SP1 targets an order of magnitude performance improvement vs. existing zkVMs—its alpha release is already up to 28x faster for certain programs and competitive with circuit-based approaches.*”

”*SP1 achieves state-of-the-art performance on several real-world workloads inspired by common blockchain use-cases like bridging and verifying Merkle proofs. Our performance is a result of multiple design choices that utilize the latest proof system advances, including a cross-table lookup architecture, a customizable "precompile" system that can accelerate almost any performance bottleneck without much additional recursion overhead, and more.* ”



2 ZKIBC Approach

2.1 ZKIBC 1.0

We briefly introduce the ZKIBC 1.0 approache:

Electron labs have built a circom-based library that allows you to generate a zk-snark proof for a batch of Ed25519 signatures.

Ed25519's twisted Edwards curve uses a finite field that is larger than that used by the altbn128 curve (used by zk-snarks). Performing large finite field operations inside a smaller field is difficult because several basic operations such as modulo and multiplication can become very inefficient.

To solve this problem, Electron labs were able to find 2^{85} as a base over which to define our curve operations for twisted Edwards curve. Since the ed25519 prime p = 2^{255} - 19 is a close multiple of 2^{85}, we were able to come up with efficient basic operators such as multiplication and modulo (under 25519 prime) for base 2^{85} numbers.

The proof parameters are following:

- Proof generation time per signature 9.6s (averaged out).
- Number of Signatures per batch/proof $\leq$ 100 (maximum value).
- Time to generate zk-proof for the batch = 16 mins for 100 signature batch.

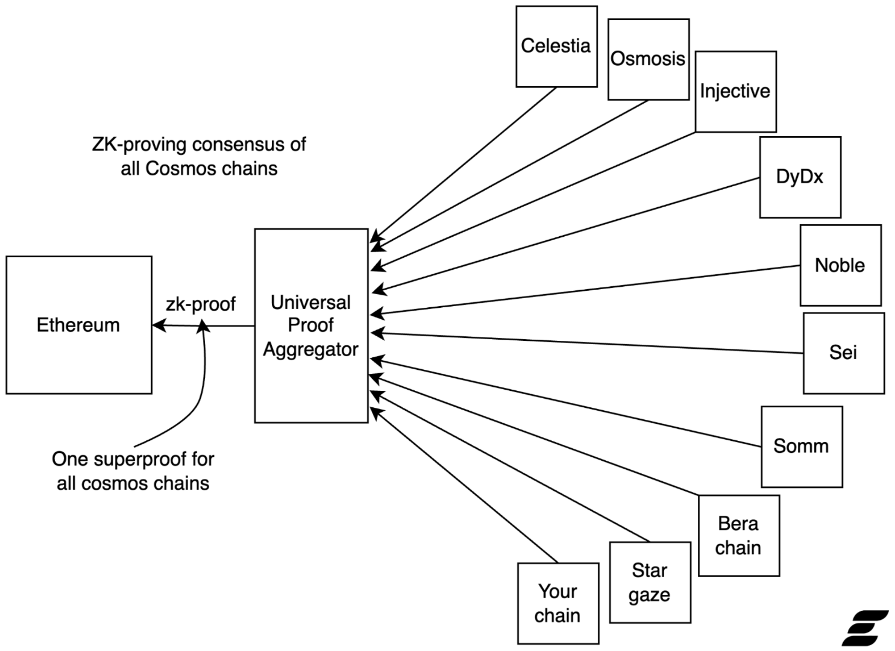
Since the tendermint block production rate is 7 sec and the proof generation time is 8 minutes, need multiple prover machines in parallel to keep up with the block production rate.

Number of parallel machines required = 8 mins *60 / 7 sec = 69 machines.

2.2 ZKIBC 2.0

The Solution of ZKIBC 2.0 is simple – aggregate individual zk-proofs from multiple chains into a single "superproof". This enables the cost of proof verification to be amortised across many chains. The current state of the art ZK-tech already enables us to aggregate a large number of cosmos chains, with reasonable proving costs.

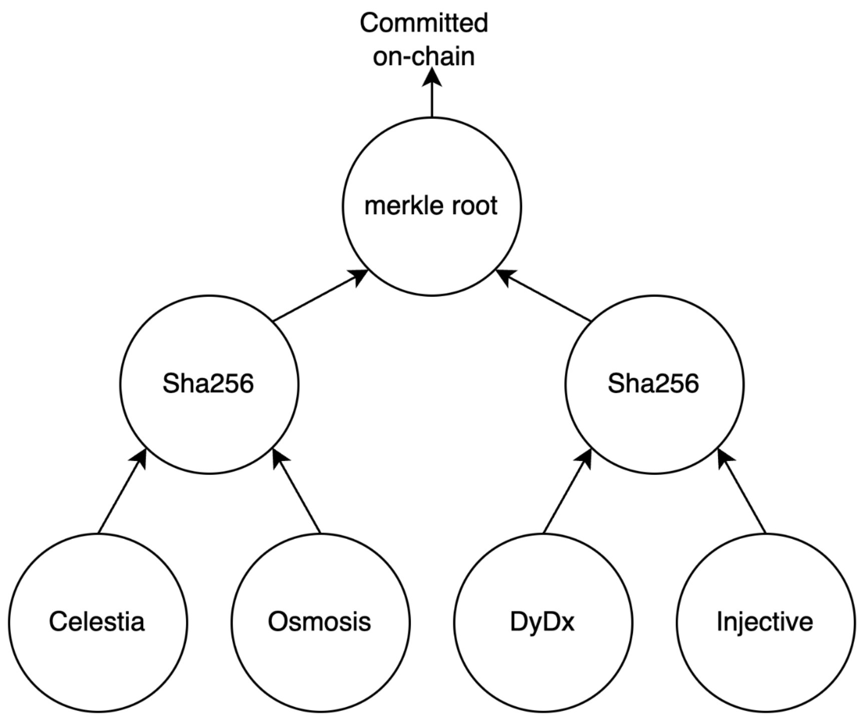
Assuming a proof verification gas cost of 250K, and N Cosmos chains being aggreagted, the cost per chain is 250K/N. For N =10, the cost per chain is 25,000, which is roughly the cost of an Eth transfer. As we add more and more chains into the network, the proof verification cost per chain continues to reduce and tends to zero, till the point the zk-hardware cost dominates.



Celestia
Osmosis
Injective
ZK-proving consensus of all Cosmos chains
DyDx
Noble
zk-proof
Universal Proof Aggregator
Ethereum
Sei
Somm
One superproof for all cosmos chains
Bera chain
Star gaze
Your chain

In ZK-IBC 1.0, we proved a single light header inside the circuit. This enabled us to reduce the information to be posted on-chain to a single hash string + the zk-proof itself.

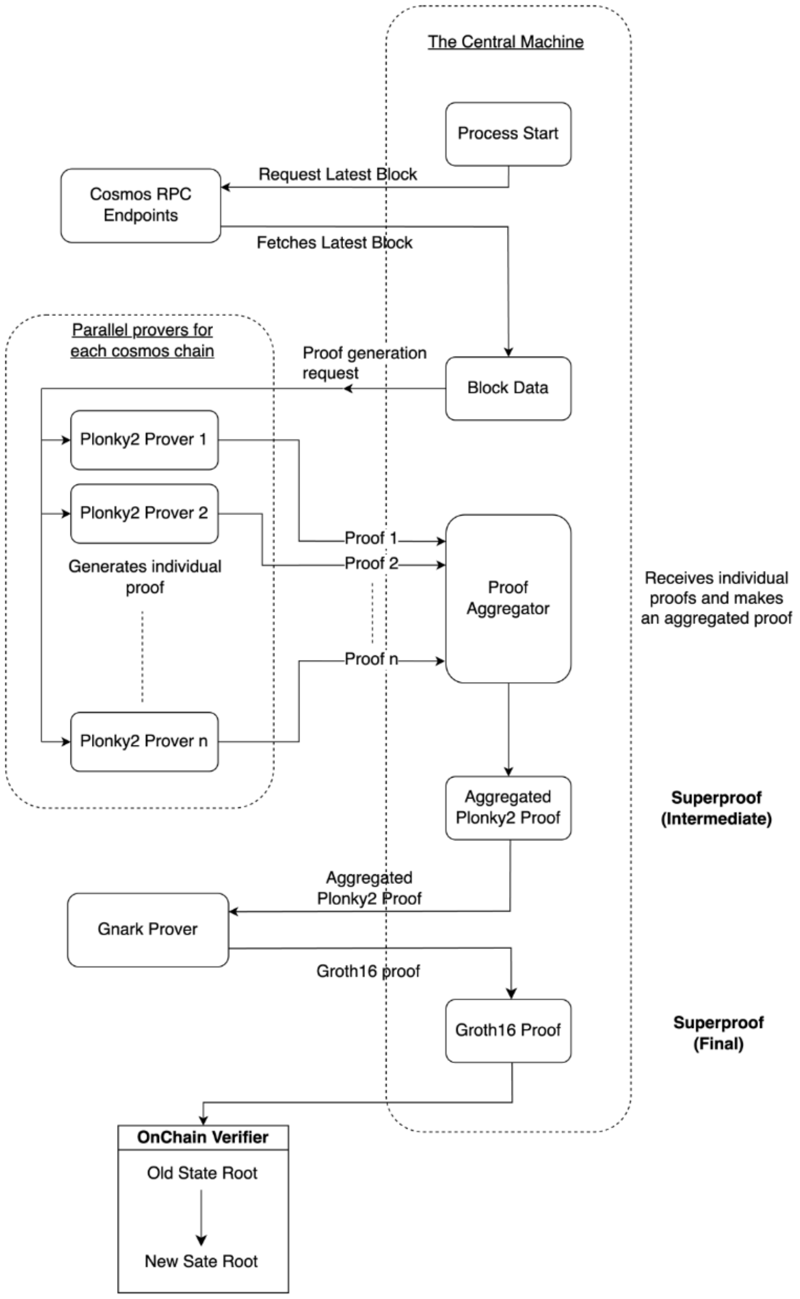
With ZK-IBC 2.0, we are going one step beyond, rather than posting one hash string for each block of a chain, we create a merkle tree out of the individual hash strings from each chain, creating one superhash for all cosmos chains.



one leaf represents one chain as `Sha256(block_hash || height)`

Prover infrastructure. ZKIBC 2.0 has the following proof generation flow:

- Generation of plonky2 proofs for individual chains. These machines run in parallel. Hence, when calculating Total Proving Time, we only need the maximum of the times taken by individual chains' machines.
- Aggregation of individual plonky2 proofs into a single aggregated plonky2 proof.
- Reduction of aggregate plonky2 proof to groth16 proof (the final super-proof) to further reduce the cost of verifying the proof on-chain.


The Central Machine
Process Start
Request Latest Block
Cosmos RPC Endpoints
Fetches Latest Block
Parallel provers for each cosmos chain
Proof generation request
Block Data
Plonky2 Prover 1
Plonky2 Prover 2
Generates individual proof
Plonky2 Prover n
Proof 1
Proof 2
Proof n
Proof Aggregator
Receives individual proofs and makes an aggregated proof
Aggregated Plonky2 Proof
Superproof (Intermediate)
Aggregated Plonky2 Proof
Gnark Prover
Groth16 proof
Groth16 Proof
Superproof (Final)
OnChain Verifier
Old State Root
New Sate Root

3 BLS cost

We also estimated the cost of various operations(mainly signature scheme) on the Cardano.

The aggregate BLS signature schemes have two versions. Single key version has the same key and different messages, with public key over G1. This function returns a list of 10 messages {'msg_1', ..., 'msg_10'}, a public key 'pk', and an aggregate signature 'aggr_sig'.

Multi key version has same message, with public key over G2. This function returns a message 'msg', ten public keys '{pk_1,...,pk_10}', and an aggregate signature 'aggr_sig'.

Operation	Script size	CPU usage	memory usage
BLS Verification(G1)	332	1,463,720,946	5754
BLS Verification(G2)	380	1,342,066,427	5754
AggSig Verification(single key)	777	3,336,910,422	71202
AggSig Verification(multi key)	1704	3,676,891,887	430586
Schnorr Verification(G1)	370	248,136,411	13796
Schnorr Verification(G2)	514	493,212,089	13964

Table 1: Cost of signature verification on BLS12-381