Zephyr: A Cost-Effective, Zero-Knowledge Light Client for Enhanced Blockchain Interoperability

Xiangan He
hexh@bc.edu
Boston College
MA, USA

ABSTRACT

Blockchains are siloed by nature. A longtime limitation of blockchain technology is that individual cryptocurrencies are bound to their own chains. Users and applications need to transfer arbitrary data and funds across blockchains. At its peak in 2021, cross-chain bridges custody over $20B in various cryptocurrencies, serving users' needs to engage in DeFi, gaming, etc. Plagued by a lack of ability to do so in a decentralized way, applications force its users to rely on centralized exchanges to achieve cross-blockchain interoperability. User flow thus becomes significantly more complicated, and developers are prevented from defining and building applications that are uniquely enabled by on-chain bridges. Furthermore due to hacks of over $2B, a loss of faith in natively on-chain solutions force users to look to off-chain solutions like Coinbase.

This paper addresses the issue of "*cross-blockchain interoperability*" by presenting Zephyr, a trustless and efficient Ethereum light client built on top of zero-knowledge proofs (ZK proofs). The light client is an essential component for achieving the implementation of a bridge that guarantees strong security without extra trust assumptions on centralized middlemen, while having significantly reduced on-chain verification cost[11]. The key contribution of this work is to present metrics for evaluating how the zero-knowledge light client enables blockchain bridge performance above that of what was previously possible. Performance evaluation under different implementations is made possible with the help of tools created from existing libraries.

In particular, we use tools like circom and snarkJS[14] to study the tradeoffs between latency (of completing the cross-blockchain message passing) and economical efficiency. In our preliminary study, we demonstrated that implementing Zephyr using the Plonky2 and Groth16 proof systems is 320x cheaper in computational cost than previous light client protocols (i.e. Bitcoin's SPV[12]) with moderate performance loss (roughly 2x less throughput)[1]. As a future work, we plan to construct and evaluate performance of additional off-chain components to the bridge protocol, as well as investigating the robustness of such a protocol.

1. INTRODUCTION

Previously state-of-the-art solutions which ushered a dawn of cross-chain native applications have three primary disadvantages: (1) they're centralized, which is counterproductive to creating a decentralized platform, (2) they're prone to getting hacked [2], and (3) they deal only with tokens, not arbitrary data. For example, in some (previously-hacked) bridges [3], messages are transferred when at least a quorum of 13 of 19 validators agree on the state and execution of the blockchain the message originates from. This means that users and applications must trust that a majority of the validators do not collude to forge or censor them.

Bridges opting for more decentralization have adopted 'optimistic' security models. Rather than 'pessimistically' verifying each message natively, such bridges rely on local verification by participants, with the opportunity to flag it in the system. This tradeoff allows these bridges to become computationally lighter, but makes them vulnerable in security to the failure of message verification by participants. Bogus messages are only caught when they're challenged, and as a result a notably popular implementation was hacked for $200M USD [4].

The alternative to trusting the validators' view of the chain state is verifying the source blockchain on the target blockchain directly. Doing so would essentially embed a blockchain in another, which is economically infeasible. Each opcode executed on chain costs money; this cost, which is measured in gas, would be exorbitently high due to computational intensity. An approximation is verifying only the consensus of the source chain. In our example implementation, the source chain becomes Ethereum which boils trusts assumptions down to the economic security of the chain.

2. A ZERO-KNOWLEDGE LIGHT CLIENT

A program that performs the alternative verification described above is called a light client. Running a light client directly on-chain (as a smart contract) is still infeasible due to the high cost of verification [13], but it's possible to use zero-knowledge proofs to perform succinct, verifiable computation [5][6]. By performing the expensive verification off-chain, we would then only need to verify the correctness of the light client's execution on-chain. Originally costing roughly 64 million gas for one block header, the expense of the computation is reduced to less than 300,000. The following diagram illustrates the lifecycle of a message being passed and verified in this way.

First, the user sends a message m. On-chain, m is hashed & inserted into a merkle tree, producing a new accumulated root M. Next, the Zephyr light client prover generates zero-knowledge proofs that the accumulator root M is included within a final block's state root S. Proving inclusion of root

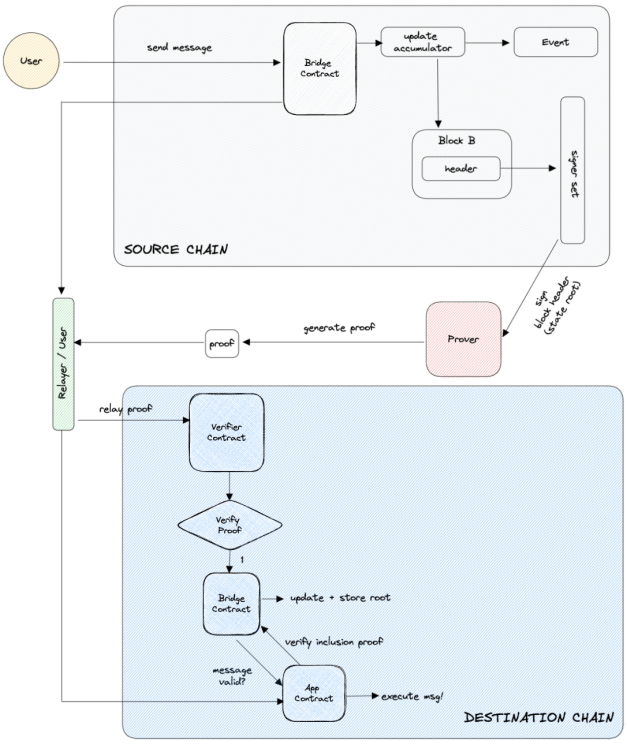

Figure 1: Message lifecycle

M within a finalized block could be performed in the same circuit that implements the light client, so block finality and message root checks can be verified in a single transaction. Then, some Relayer submits & verifies the zero-knowledge proof for root M on the destination chain. Verified root M is stored on-chain. Alongside that, some off-chain infrastructure observes messages such as m being sent, re-constructs the state of the Merkle tree, and generates (and serves) inclusion proofs that individual messages are included within verified roots such as M. Lastly, user submits & verifies on-chain the Merkle inclusion proof of single message m within root M, and executes the effects of the message.

3. CHOICE OF A PROOF SYSTEM

There are a variety of proof systems at-hand to choose from to build the light client prover with, but it's important to choose the proof system that satisfies the tradeoffs most appropriate for this system. We consider a few of these critical tradeoffs:

- **Proof generation latency**: is critical to the overall usability and success of a cross-chain messaging protocol. Users are ultimately highly sensitive to the latency of moving messages across different ecosystems. This is particularly relevant for natively cross-chain applications, which rely heavily on rapid state access.

- **Setup type**: setups for proof systems are often categorized into three broad buckets: trusted, universal, and transparent. We'd like to minimize the need for trusted setups.

- **Proof size**: the size and number of the artifact(s) that are relayed to the target chain(s) can vary based on proof systems (e.g. choice of polynomial commitment scheme).

- **Verification cost**: will depend on what precompiles are needed to verify the ZKPs. We'd like to choose a proof system that can operate with a minimal and widely available set of precompiles.

- **Recursion**: support for recursion is useful on a few different levels: (1) batch messaging, (2) smaller circuit sizes, and (3) enable proof aggregation.

- **Verification time**: the time to verify a given proof should be small in absolute terms and in relation to the size of the proving circuit.

Industry standard light clients (e.g. Helios) currently use the Plonky2 proof system. To reduce the size of the resulting proof (and thus make it economically feasible to verify on the target chain), we can wrap the Plonky2 proof in a Groth16 proof, which is then submitted to a Solidity contract that verifies it [1]. One cannot simply use Groth16 to do the whole thing because the proof would be too large. The Groth16 proof would attest to the fact that if the Plonky2 proof were submitted to a verifier, it would be valid. Rather than ferrying individual messages across chains, this design would bundle messages into a single 32-byte Merkle root which can contain a large number of individual messages. This way bridges can unlock much lower gas costs per message.

In work conducted parallel to the writing of this paper, Tiancheng Xie et al. use a cryptographic mechanism called Virgo and improve upon it to create deVirgo[11]; results show that computational performance is slightly worse when we adopt our alternative approach. Wrapping a Plonky2 proof in a Groth16 proof and using inclusion proofs use only 20k gas more. Proof generation speeds for the Plonky2 prover implemented in circom (30 seconds) are 13 times faster than that of Virgo's (400 seconds). Yet, we still lag moderately behind that of deVirgo's (2 times slower; 30 seconds versus 13). Additionally, the following table demonstrates early benchmarks on gas costs of and time required for proof verification under a few different cryptosystems.

	GROTH16	FFLONK	PLONK
Gas Costs	230K	203K	300K
Proving Time	1.4s	10s	6.6s

Table 1: Proving times determined with a SHA-256 Prover

Comparing FFLONK with Groth16 and Plonk in the above table, for example, it can verify proofs on chain 27K gas cheaper that Groth16 and 30 percent cheaper than regular Plonk; the drawback is that proof generation speed is about 10 times slower than Groth16 [9]. Given these tradeoffs, preliminary results shows promise for iterative improvements of Zephyr: it is possible to explore different proving methods that combine faster message proving times and low gas costs.

4. COST OF MESSAGES AND FINALITY

Calculating the root produced by the merkle inclusion proof costs 9,942 gas. In addition to loading a stored root

and performing an equality check, the total for verifying a proof is under 13k gas per message. In this experiment, each proof was a total 1024 bytes, and at 16 gas per byte of nonzero calldata, it turns out that 16,384 gas is required to pass the proof via calldata. This brings the final total to around 30k gas for passing & evaluating a merkle inclusion proof. The total gas cost, including the Groth16 verification, comes out to be around 240k.[1]

With regards to finality guarantees, we are interested in messages that originate from finalized blocks. On Ethereum, where the light client prototype was implemented, validators take turns in slots (every 12 seconds) to propose blocks, and every 32 slots, a checkpoint is taken. These 32 slot periods are called epochs. A block is justified if it's in the most recently completed epoch. A block is finalized if it's behind a justified epoch. This means in the best case, a block will be finalized in 64 slots (13 minutes). Once a block is finalized, we can rely on its contents and therefore consider messages contained within this block to be valid — these are the same semantics bridges today abide by [15]. Thus there are two steps required for message passing:

- Check finality of blocks
- Prove that a message is contained in a final block

Similar to the Helios light client, we will start with a trusted block, called the weak subjectivity checkpoint. This checkpoint is stored as a block hash, and is periodically updated (whenever a new final block is verified by the light client, it becomes the checkpoint). The checkpoint is stored on-chain. The checkpoint block's header includes the composition of the sync committee, a set of 512 validators that sign every block within a 256 epoch period (27 hours), after which a new sync committee is elected. The only bookkeeping needed for tracking the validity of a block is tracking this sync committee. We implement a BLS signature verifier circuit in Plonky2 to allow verifying the sync committee signatures. Assuming we have the most recent sync committee (whose public keys are provided as private input to the circuit), the BLS verification is sufficient to determine if a block is valid (but not necessarily final). The members of the sync committee are rewarded for signing valid block headers but are not currently slashed for invalid or malicious actions. We implement this model for this prototype, and plan to later evaluate the switch over to full validator set tracking.

5. FUTURE WORKS & CONSIDERATIONS

For future works, we intend to analyze the robustness of the finality guarantees. The existing light client design relies on the sync committee's signatures, and thus far we've assumed that those are always valid with reasonable uptime. This, however, is not always the case. Light clients grab sync committee attestations and apply them to their local view of the chain. Unfortunately, there are no protocol constraints on what this signature can be. This allows sync committee members to coordinate to cause some degenerate scenarios:

- Zephyr may receive many semantically invalid attestations, with valid signatures.
- Zephyr may receive a valid proof, with semantically invalid data.
- Zephyr may receive a valid proof, but have the data withheld.

Sync committee penalties are also insufficient (capped at 0.1 ETH per validator for the 27 hour period). It is cheap (around $82,000 or 0.3% of stake) for a committee to not produce any valid attestation during a period. Malicious committees can ensure that only invalid attestations are available.

6. REFERENCES

[1] Tenderly. "Ethereum Development Platform." Tenderly, https://tenderly.co/.

[2] Kharif, Olga. "Crypto Bridge Hacks Have Stolen $1 Billion in Little over a Year." Bloomberg.com, Bloomberg, 30 Mar. 2022, https://www.bloomberg.com/news/articles/2022-03-30/crypto-bridge-hacks-reach-over-1-billion-in-little-over-a-year.

[3] "Wormhole Development Book." Introduction - Wormhole Development Book, https://book.wormhole.com/.

[4] "Explained: The Nomad Hack (August 2022)." Halborn, https://www.halborn.com/blog/post/explained-the-nomad-hack-august-2022.

[5] "Building Helios: Fully Trustless Access to Ethereum." a16z Crypto, 8 Nov. 2022, https://a16zcrypto.com/building-helios-ethereum-light-client/.

[6] Wang, Wenqi, et al. "Efficient Cross-Chain Transaction Processing on Blockchains." MDPI, Multidisciplinary Digital Publishing Institute, 27 Apr. 2022, https://www.mdpi.com/2076-3417/12/9/4434.

[7] Stone, Drew. "Trustless, Privacy-Preserving Blockchain Bridges." Arxiv, 9 Feb. 2021, https://arxiv.org/pdf/2102.04660.pdf.

[8] McCorry, Patrick, et al. SoK: Validating Bridges as a Scaling Solution for Blockchains - IACR. https://eprint.iacr.org/2021/1589.pdf.

[9] Walton-Pocock, Thomas. "Plonk Benchmarks − 2.5x Faster than Groth16 on MiMC." Medium, Aztec Network, 29 Jan. 2020, https://medium.com/aztec-protocol/plonk-benchmarks-2-5x-faster-than-groth16-on-mimc-9e1009f96dfe.

[10] Wood, Gavin. Ethereum: A Secure Decentralized Generalized Transaction Ledger. https://ethereum.github.io/yellowpaper/paper.pdf.

[11] Tiancheng Xie, et al. 2022. "In Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022".

[12] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review (2008).

[13] ETH-NEAR Rainbow Bridge – NEAR Protocol. https://near.org/blog/eth-near-rainbow-bridge/ (2022).

[14] Circom Documentation. https://docs.circom.io/ (2023).

[15] Wormhole Documentation. https://docs.wormhole.com/ (2023).