Grant Proposal: An Ethereum Light Client on Axelar

Shresth Agrawal[1,2], Dionysis Zindros[1,3], Dimitris Karakostas[1,4], and Apostolos Tzinas[1,5]

[1] Common Prefix
[2] Technische Universität München
[3] Stanford University
[4] University of Edinburgh
[5] National Technical University of Athens

May 1, 2023
Last update: January 8, 2024

Abstract. Axelar network is a decentralized interoperability layer that connects a large set of blockchains. Axelar is based on the Cosmos SDK and defines a set of Tendermint proof-of-stake validators, who can relay events from supported chains to the Axelar Cosmos side. In particular, Axelar leverages multiple validation models for events happening on the supported EVM chains. Such events can be reported via IBC light clients, when possible, or attestations from Axelar's proof-of-stake validators. Axelar validators are expected to run full nodes of the EVM chains to gather event information. However, this operation is typically expensive, such that the entry and maintenance costs for validators increase, along with possibly leading some validators to use centralized RPC providers. This proposal aims to reduce operational costs and increase Axelar's level of decentralization. Specifically, we advocate for an on-chain light client to run as an Axelar module that trustlessly consumes source data. Such a light client checks the veracity of cross-chain claims to ensure the relevant events have been included in the canonical source chain, safeguarding against untrustworthy EVM RPC providers. Focusing on Ethereum, we discuss and contrast several possible flavors of light or superlight clients based on *Simple Payment Verification* (SPV), *zero knowledge*, or *refereed games*. Tailoring the solution to the particularities of Axelar, we recommend SPV as the most suitable choice. We present an architecture blueprint and estimate the workforce allocation as well as the cost needed for the effort. Lastly, we put forth a theoretical model to support the need for such design changes. Theoretically, our proposed changes enable the system to endure temporary dishonest advantages among Axelar validators and restore safety following the re-establishment of an honest supermajority.

1 Introduction

Blockchain technology has revolutionized the way we create trustless and decentralized applications, offering immense composability and the ability to build complex primitives. However, the blockchain landscape is far from homogeneous, with multiple ecosystems that are based on their own consensus protocols and offer varying security and operation features.

Axelar[6] enables interoperability between diverse blockchain ecosystems. Axelar is based on the Cosmos SDK[7] and aims to interlink various blockchain ecosystems, such as Cosmos, Ethereum, Bitcoin, Polygon, and others. Notably, it has been identified as one of only two possible solutions that are sufficiently secure across the stack to be used by the leading decentralized exchange, Uniswap [12].

Ethereum, in particular, is one of the most widely used blockchains, hosting a multitude of decentralized applications spanning various sectors, including finance, gaming,

[6] `https://axelar.network`
[7] `https://cosmos.network`

and governance [3, 13]. Therefore, establishing a seamless and secure connection to Ethereum is of paramount importance for Axelar. This proposal focuses on building a trustless connection between Axelar and Ethereum.

1.1 Current Construction

Currently, Axelar implements a Tendermint-based delegated proof-of-stake consensus mechanism [2, 10]. In particular, validators lock stake in the Axelar chain and receive stake delegations from Axelar users. The top 75 validators, based on the aggregate (self and delegated) stake, are chosen to participate in Axelar's consensus mechanism.

Axelar adopts a modular architecture to connect with different chains. Each connector module consists of two essential components. The first component verifies source chain data (e.g., Ethereum) into Axelar. The second component generates threshold signatures, which can be verified on the source chain. In this report, we focus exclusively on the former, that is verification of source chain data that are bridged into Axelar.

Currently, connectors utilize an on-chain voting mechanism within Axelar to verify transactions that occur on the source chain. The validators who participate in a connector attestation are called *attestors*. To determine the voting power of an attestor, Axelar employs quadratic voting. Briefly, the voting power of an attestor is the square root of their total stake. This mechanism aims to ensure a fair distribution of influence among attestors, based on their stake. Attestors are required to run a full node of the source chain and have access to the full node's RPC (Remote Procedure Call) interface. This enables them to verify the finalized transactions on the source chain, before voting to bridge them on Axelar.

To bridge data from the source chain to Axelar, a user interacts with an Axelar smart contract on the source chain. Subsequently, the user accesses the connector module on Axelar and initiates a voting poll, which is viewed by attestors. For each poll, an attestor decides whether to vote for or against. To make an informed decision, the attestor queries the source chain's full node RPC and checks if the transaction under question has been finalized on the source chain. If a poll receives sufficient attestations, it is accepted, otherwise it is rejected.

This voting process forms the basis of verifying the source chain data into Axelar. Nonetheless, for the connector construction to function securely, both the source chain and Axelar are assumed safe and live. In addition, the (quadratic) voting power distribution among the attestors is assumed to have an honest majority.

1.2 Problem Statement

The current construction of Axelar relies on attestors running the full node of the source chain to verify transactions and vote in an informed manner. As the Axelar network expands its support for additional chains, attestors will be required to run an increasing number of full nodes to verify transactions from these source chains.

However, running a full node, or many, imposes significant costs on attestors.[8] In order to maintain a diverse and robust ecosystem, comprising a multitude of participants, it is important to develop cost-effective and efficient solutions.[9] Such solutions could reduce

[8] Indicatively, running a full Ethereum node requires 4+ CPU cores, 16+ GB RAM, at least 1 TB SSD, and 25 Mbps of stable connection (source: `https://www.quicknode.com/guides/infrastructure/node-setup/ethereum-full-node-vs-archive-node`).

[9] In practice, systems that impose significant costs for running a full node often demonstrate centralization tendencies, where participants tend to rely on third-party service providers to run and maintain

the entry and maintenance costs for new validators, also possibly leading to more value being gained for users.

To address this challenge, we propose a construction that enables users and attestors to verify the consensus of the source chain within the Axelar execution layer. This is accomplished through light and super-light client constructions of the source chain.

In this report, we explore several constructions for light and super-light clients tailored specifically for Ethereum. We then propose a construction that best aligns with Axelar's vision and requirements, aiming to drive decentralization, enhance security, and improve scalability. Our construction makes use of Ethereum's sync committee and guarantees bridge safety, assuming at least one Axelar attestor is honest.

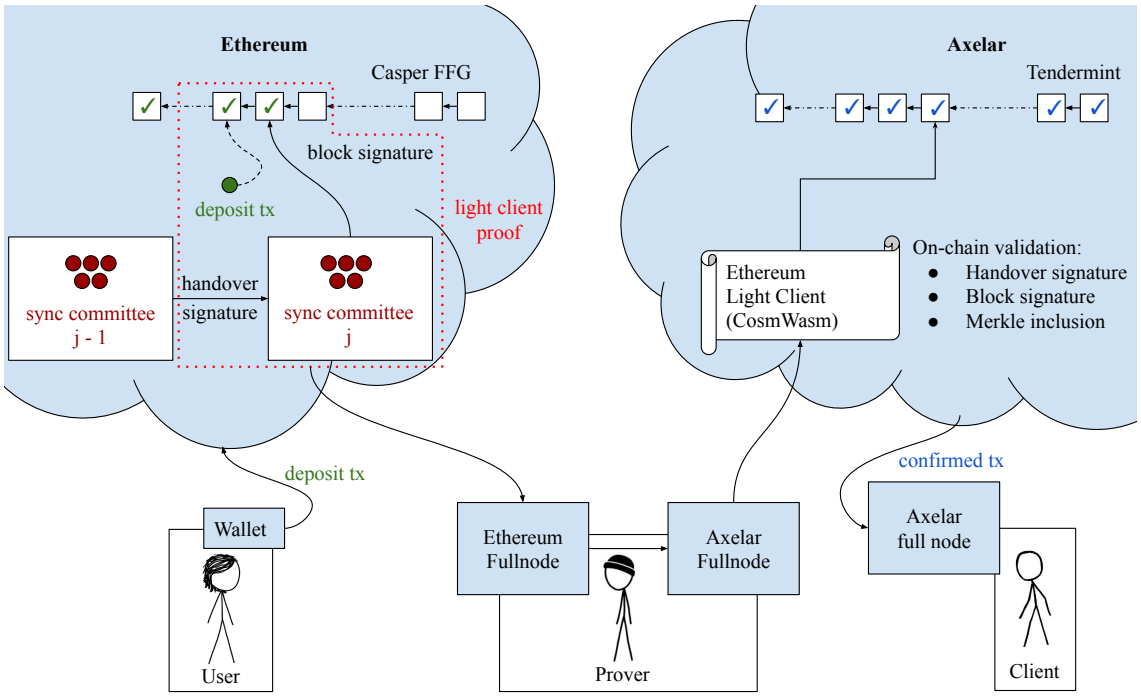

Fig. 1: We recommend implementing an on chain light client using sync committee. The light client will be implemented as a go module that will be deployed on the Axelar network. The light client will be responsible for verifying the handover signatures, block signatures and Merkle inclusion proofs.

1.3 Preliminaries

Bridge

A *bridge* is an interoperability protocol between two ledger protocols. The purpose of the bridge is to relay events or information that take place on the source side to the destination side [9, 15]. In particular, the parties that maintain the bridge transmit *cross-chain* transactions, such that a transaction on the source side is represented by an "image" on

the full nodes on their behalf. Due to economies of scale, a large portion of these networks tends to cluster around a small number of such providers; e.g., at times, 50% of Ethereum's transactions ran through one such provider, Infura [8]. Therefore, proactively addressing such hazards can lead to a healthier, more diverse and robust ecosystem.

the destination side. There are two core properties that a secure bridge should guarantee: safety and liveness.[10]

Bridge Safety. Bridge safety mandates that a transaction appears on the destination side *only if* it has first appeared on the source side, albeit with some delay. Intuitively, safety ensures that *bad things don't happen*, that is it's impossible to find a transaction paying out on the destination without its "pre-image" having appeared on the source side. In essence, if safety is guaranteed, an adversary cannot create money on the destination side without having paid on the source side.

Bridge Liveness. Bridge liveness ensures that a transaction which appears on the source side will eventually make it to the destination side. Intuitively, this guarantees that *good things happen*, that is whenever an honest party attempts to cross the bridge, it will successfully do so.

Light Clients

A *light client* is a client that wishes to synchronize with the rest of the network, but has limited resources available in terms of communication, computation, and storage. One example of a limited resource computer is the on-chain smart contract infrastructure of Axelar, where computation and storage are expensive. The light client begins its lifecycle holding the genesis block and synchronizes to the current tip of the canonical chain once in a while. Our job when building a light client is, given a block that the light client has already downloaded sometime in the past, to allow the client to synchronize with the most recent chain tip.

Ethereum Sync Committee

One useful ingredient of the Ethereum ecosystem that enables the construction of efficient light clients is the so-called *sync committee* [6]. This feature was introduced in the system's "Altair" hard fork and is specifically tailored for use of light clients, as we will discuss in the alternatives of Section 2.

The sync committee consists of 512 of Ethereum validators. They are randomly selected, from the set of all validators, every sync committee period (approx. 1 day). Each honest validator in the sync committee is continually online and signs the header of each block during the sync committee period.

The sync committee of period X is decided one periods in advance, that is the beginning of period $X-1$. As headers of the current period X containing the root of the next sync committe $X+1$ are signed by the current sync committee X, the signatures of the current sync committee act as a handover mechanism from the current committee to the next. A prover can provide signatures from 2/3 of the current committee on some header of current period as a proof of validity of the next committee. Note that there can be multiple headers in the current period which can have more than 2/3 sync committee signatures on them. The prover can choose any of these headers with their respective signatures as a proof.

We demonstrate the usage of the sync committee with the following example. Assume that a client C holds the block header of some slot N, that is part of the sync committee period X. When C wants to authenticate the header of a block at slot N', which is part of the period $X+1$, it proceeds as follows. First, C validates the sync committee of period $X+1$. To do so, it verifies that the Merkle root of the committee was published in a block

[10] Appendix 5.1 offers a formal definition of these properties.

header of period X; if so, C updates its sync committee for $X + 1$ as the one defined in the Merkle tree. At this point, C can validate every block header for each slot of period $X + 1$, by obtaining the corresponding signatures of the sync committee of that period.

Using this iterative process, the light client can validate all future blocks, starting from an initial trusted point. The cost of validating a block depends on the size of the committee, the aggregate signature, and the Merkle path; in Ethereum this is estimated to approx. 25KB [6].

There are two main points of discussion around the sync committee. First, the committee should be honest. In essence, the sampling process, via which the sync committee's members are chosen from the set of all Ethereum validators, should be secure, s.t. the probability that $\frac{2}{3}$ of the committee members are adversarial is negligible. Observe that, if the committee is malicious, then it can convince a light client of a fraudulent Ethereum state. Second, slashing is currently not implemented in the sync committee. Therefore, a committee member can double-sign, i.e., sign conflicting blocks, without direct counterincentives. Whether committee members are incentivized to behave in such manner is outside the scope of this document, although it is an interesting research question.

2 On-Chain Light Client

We now present an array of solutions for the on-chain light client. For each solution we give a high-level overview, highlighting its mechanics as well as possible shortcomings. Each alternative offers different features and relies on different assumptions.

We note that all solutions require that at least one Axelar attestor is honest, in order to guarantee liveness. Additionally, all solutions, except the first, rely on Ethereum's sync committee.

SPV Light Client (sampling validators)

The first solution makes use of Ethereum's validators. In particular, we assume that there exists a mechanism for sampling some of the signatures that were produced by the validators on each block. To verify an Ethereum block, the light client checks: i) that the sampling process was done correctly (e.g., following a verifiable pseudorandom process); ii) that the provided signatures correspond to active Ethereum validators; iii) that enough signatures have been provided to guarantee the block's correctness.

An advantage of this solution is the non-reliance on the sync committee. In particular, the only assumption that is needed is that Ethereum is safe and live (which is already assumed) and that the probability of sampling adversarial signatures is negligible.

However, it also presents various shortcomings. First, Ethereum's consensus relies on Casper FFG [4] and LMD-GHOST. This is particularly complicated and it is unclear how a secure signature sampling process can be implemented. Second, it is unclear how the light client can retrieve the active Ethereum validators at any point in time. In particular, assuming that the light client starts from a trusted block header, it is unclear how to obtain each following period's validators, in a succinct and efficient manner.

SPV Light Client (Sync committee)

This solution makes direct use of Ethereum's sync committee. In particular, each sync committee is recorded on Axelar. When an attestor wants to bridge a transaction from Ethereum to Axelar, it accompanies it with a proof of inclusion in a block that has been validated by the corresponding sync committee. Following, when an Axelar full nodes

wants to validate an Ethereum transaction that has been bridged to Axelar, it first identifies (on Axelar's chain) the sync committee and that corresponds to the transaction and then validates the proof w.r.t. it.

This scheme assumes that, at the bridge's onset, the sync committee is recorded on Axelar correctly. In other words, the first sync committee that is recorded on Axelar should be correct. Following, each update to the sync committee is accompanied by a proof of a handover, that is an Ethereum block that has been signed by the previous committee. Note that, even if Axelar's security is compromised, the adversary cannot produce a proof of a handover to an invalid sync committee (since this depends on the sync committee's security). Therefore, the honest Axelar nodes will always hold a valid list of sync committees (albeit this could possibly be outdated, if liveness is violated).

The major advantage of this solution is that it is straightforward to implement. In particular, the sync committee has already been implemented in Ethereum and there exists a large body of community projects and tools that already implement light client based on sync committee, such as Kevlar[11] and Helios[12].

There are some disadvantages with this solution though. First, it relies on the security of Ethereum's sync committee. Second, the storage requirements increase linearly over time, since each sync committee (which is updated approx. every day) is recorded on Axelar's chain. This can result in significant storage overhead, since the mechanism requires approx. 25KB per committee [6].

Refereed Light Client

This mechanism makes use of Ethereum's sync committee in a manner similar as the previous solution, but aiming to reduce the storage requirements.

In brief, the idea is to (optimistically) assume that an attestor provides honest data about the updated sync committees and enable a dispute resolution mechanism in case the attestor is malicious [1]. Specifically, we assume that the bridge is updated every x (sync committee) periods. In the previous solution, each committee is recorded on Axelar's chain. Here, there exists an Axelar attestor who collects the x committees, each corresponding to the sync committees for the corresponding periods, and publishes a commitment to them on Axelar altogether. If the attestor is honest, then the commitment should be correct. Therefore, the other Axelar nodes can "fast-forward" these x committees.

Nonetheless, the attestor might be malicious and submit an incorrect commitment. To cover this possibility, there exists a contest period, during which another attestor can challenge the first attestor's submission. The challenger provides an alternative commitment, at which point the dispute needs to be resolved. This is done by opening both commitments and identifying the point of divergence between the two committee lists. At this point, both attestors are required to submit proof of correct transition, that is to reveal the (sync committee) keys that correspond to the last agreed committee along with (this committee's) signatures on the keys that correspond to the first committee of disagreement. Since the sync committee is presumed honest, one of the two lists will be revealed as fraudulent, since it should be impossible to provide the necessary signatures that validate the transition under question.

We propose two manners in which the commitment can be implemented.

Linear In this case, the commitment is a list of hashes, with each hash corresponding to a sync committee. The main benefit of this solution is that disputes can be resolved easily,

[11] `https://github.com/lightclients/kevlar`

[12] `https://github.com/a16z/helios`

since the point of disagreement can be identified directly by comparing the two lists element by element. Once the point of disagreement is identified the attestors need to open the last agreed committee, the committee of disagreement and the handover signatures from the last agreed committee to the committee of disagreement. However, the asymptotic complexity of this solution is again linear on the number of committees, the proof size is significantly smaller than the previous solution as only committee hashes have to be submitted instead of complete sync committee public keys.

Bisection Here, the commitment is a Merkle tree, the leaves of which correspond to the hash of each sync committee. Initially the attestor publishes the root of the tree. In case of a dispute, the challenger provides an alternative root and initiates an interactive dispute resolution protocol. In each round the attestors open the children of the disputed node. Then the on chain protocol checks if the children are correct (i.e. the hash of the children matches the hash of the node) and compares left child and then the right child. If the left child is already different for the two attestors the next disputed node is set to the left child, otherwise to the right child. This process is recursively repeated until we reach the leaf of the tree which is also the first point of disagreement. At this point the attestors can follow the same protocol as the linear case to resolve the dispute.

The advantage of this solution is that the storage complexity is constant in the optimistic case (i.e. the attestors are honest and no dispute resolution is required) where as it falls back to poly logarithmic complexity in the case of dispute resolution. [1, 11].

Finally, we note that the advantage of the refereed light client solution only manifests if the bridge is synchronized infrequently and dispute resolutions are rare. If, for example, the bridge is synchronized every day (that is, $x = 1$), then the benefit is marginal even in the optimistic case. In addition, implementing a dispute resolution mechanism securely adds complexity. Finally, the length of the contest period affects the bridge's latency, since it should be large enough s.t. an honest party can challenge an invalid commitment, but small enough to avoid increasing latency significantly.

SNARK Light Client

The final solution is based on zero-knowledge, namely SNARK proofs (Succinct Non-interactive ARguments of Knowledge). The goal of this solution is to reduce the proof of validity of each bridged Ethereum transaction.

The setup of this solution is similar to the sync committee-based solution above. Specifically, each committee hash is published on Axelar, along with a proof of handover for the committee transitions.

In particular, when an attestor wants to bridge an Ethereum transaction to Axelar, it produces a zero-knowledge proof. This proof replaces the aggregate public key and signatures, which were used before for validation. Instead, it proves that there exists a quorum of at least $\frac{2}{3}$ sync committee members that have signed the transaction under question.

The main advantage of this solution is that SNARKs can be very efficient in terms of proof sizes. For example, a Groth16 [7] proof consists of only 2 group elements, that is approx. 209 bytes. In fact we can generate a constant size proof for increasing length of sync periods. But this benefit can not be realised in the current bridge construction because the bridge is continuously syncronised and therefore the proof data submitted to the chain will grow linearly with the number of sync periods.

One main disadvantage of this solution is implementation complexity and cost. Securely implementing ZK-SNARKs can be particularly challenging, even more so since they

have been developed fairly recently. Additionally, the cost of generating a SNARK can be particularly high. For example, according to [14], proving consensus (of 128 validators) on *one* Cosmos block takes 18 seconds on 32 instances of Amazon AWS `c5.24xlarge`. (We are unable to independently reproduce this finding because the authors of zkBridge have not disclosed their code.) At Cosmos' block rate of 1 block per second, that would require 18×32 continuously operating `c5.24xlarge` instances, costing annually $12,967,488 on Amazon AWS (annual pricing, `us-east-1` region, June 2023), or $1,749,600 on Hetzner's equivalents. Scaling this proportionally for Ethereum yields an estimate of $583,333 annually.

Comparison

Table 1 summarizes the comparison of the alternatives presented above. Note that, in all cases, Ethereum is assumed safe and live, as well as, in order to guarantee bridge liveness, we assume that at least one Axelar attestor is honest (in some cases, this assumption is needed also for bridge safety).

	SPV		**Refereed**		**SNARK**
	Sampling	**Sync Committee**	**Linear**	**Bisection**	
Proof Size (Annual)	$365 \cdot 25$ KB	$365 \cdot 25$ KB	$\frac{365}{x} \cdot (25\text{KB} + x \cdot 32\text{B})$	$\frac{365}{x} \cdot (25\text{KB} + 32\text{B})$	$365 \cdot (209 + 32)$ B
Non-interactive	✓	✓	✗	✗	✓
Doesn't rely on Sync Committee	✓	✗	✗	✗	✗
Extra Safety Assumptions	-	-	Exists honest Axelar attestor	Exists honest Axelar attestor	-
Implementation Complexity	High	Low	Medium	Medium	High
Cost of one prover annually	$600	$600	$600	$600	>$500,000

Table 1: Comparison of alternative solutions. All solutions assume that: i) Ethereum is secure (safe and live); ii) to guarantee bridge liveness, there exists an honest Axelar attestor. In the refereed case, we assume that the bridge is updated every x days (where 1 sync committee period is equal to approx. 1 day). Also, in the refereed case we compute the best-case scenario, where no disputes occur. In the SNARK case we assume the usage of Groth16 proofs. Our recommendation is highlighted in gray.

Our recommendation

We recommend implementing the second option, that is an SPV client that relies on the sync committee. From the above comparison, it is evident that the first option (SPV client with Ethereum validator sampling) is rather experimental and requires significant research work before proceeding with implementation.

Due to the high traffic that the bridge is expected to observe, it will be updated at least on a daily basis, so the main benefit of the refereed constructions would be lost.

Finally, the exceptional cost of creating SNARK proofs at this point in time makes it prohibitively expensive as an option. Nonetheless, we will design the bridge in a modular way, such that a SNARK solution can be easily incorporated in the future, when proof creation becomes more cost efficient.

3 Construction & Implementation Details

In this section, we discuss the construction and implementation details of the suggested SPV style sync committee based light client. Note that the details of the contruction are not final and are subject to change. The client will be implemented as a CosmWasm module. On a high level, the new construction would work as follows:

1. A user initiates the bridge transfer by sending a transaction to a designated smart contract on the Ethereum (source) chain, similar to the existing construction.
2. A (possibly different) user then submits a request on the Axelar network with the corresponding Ethereum transaction's hash.
3. Axelar attestors then query the full node to check if the Ethereum transaction is finalized; if so, they generate a light client proof with two components: (i) a consensus verification proof; (ii) an execution verification proof.
4. The attestor submits this proof on the Axelar (destination) chain.
5. The proof is verified on-chain by the Axelar full nodes; if the proof is valid, the transaction is executed on the Axelar chain.

3.1 Consensus Verification

This module is responsible for keeping track of the latest finalized block and the latest sync committee. To verify the finalized block and the sync committee we use the aggregate sync committee signatures as a proof. Now we will discuss how we can obtain this proof using existing beacon chain APIs and how these proofs can be verified on chain.

- **Update Latest Sync Committee.** This proof validates the sync committee handover, that is updating the committee from one period to the next. It can be generated using Ethereum's beacon chain API,[13] specifically the endpoint `/eth/v1/beacon/light_client/updates`. The proof's verification involves checking the BLS aggregate signature of the sync committee and the Merkle inclusion proof of the next sync committee to the signed header.
- **Update Latest Finalized Block** This proof allows for updating the latest finalized block. It can be generated using the beacon chain API endpoint `/eth/v1/beacon/light_client/finality_update`. Its verification involves checking the BLS aggregate signatures of the latest sync committee on the latest finalized block.

3.2 Execution Verification

This component is responsible for generating and verifying the execution verification proof. We note that the scope of this grant proposal includes only verifying if a transaction is included in a block or a certain event was released by a smart contract.

The execution proof mainly constitutes of a Merkle inclusion proof of the relevant transaction or event to the finalized block header received from the consensus verification component. The proof is generated by fetching the complete block, which contains all the transactions and events, and generating the Merkle inclusion proof. Following, the proof verification is a straightforward Merkle inclusion check.

[13] `https://ethereum.github.io/beacon-APIs`

4 Grant Proposal

This project will be implemented in the course of $4-6$ months. Members of the Common Prefix team will be involved on an as-needed basis, depending on the expertise required. For example, the following people will likely be relevant to the project's implementation:

- **Dionysis Zindros** is a post-doctoral researcher at Stanford University. His research is focused on light and superlight clients, on which he did his Ph.D. dissertation, "Decentralized Blockchain Interoperability." Some of his relevant published works include Non-Interactive Proofs of Proof-of-Work, Proof-of-Work Sidechains, Proof-of-Stake Sidechains, Mining in Logarithmic Space, Light Clients for Lazy Blockchains, Proof of Burn, The Velvet Path to Superlight Blockchain Clients, Compact Storage of Superblocks for NIPoPoW Applications, Smart Contract Derivatives, and A Gas-Efficient Superlight Bitcoin Client in Solidity. He co-authored the "Proofs of Proof of Proof of Stake in Sublinear Complexity" paper. He has published in top peer-reviewed conferences, such as IEEE Security & Privacy, ESORICS, Financial Cryptography (and the Workshop on Trusted Smart Contracts), and Advances in Financial Technologies. Regarding practical software engineering experience, Dionysis has worked on the Product Security team at Twitter and the Incident Response Development team at Google. He has also presented at practical security conferences like Black Hat Europe and Black Hat Asia.
- **Shresth Agrawal** is a developer, researcher, and entrepreneur. He is one of the authors of the "Proofs of Proof of Stake in Sublinear Complexity" paper and was responsible for building the first ever light client for Ethereum, Kevlar. He has experience building efficient and secure algorithms, protocols, and smart contracts for several DeFi protocols. Previously, he worked at ParaSwap, where he was responsible for architecting and developing a large portion of the core aggregation algorithm. He was awarded by the President of India for his research on contagious diseases. Shresth completed his Bachelor's degree from Jacobs University Bremen and is currently pursuing his Master's degree at Technical University Munich. He is interested in Cryptography, Security, Consensus Protocols, Decentralized Finance, and Ethereum.
- **Apostolos Tzinas** is a smart-contracts software engineer. He is pursuing a joint Bachelor's and Master's in Electrical and Computer Engineering at the National Technical University of Athens. Apostolos has extensive front-end software engineering experience working at Maya Insights and NutriDice, where he took on a full-stack software engineering role. He has worked with numerous programming languages and technical stacks. He loves algorithms, and as a high school student, he participated in Informatics Olympiads, such as the Junior Balkan Olympiad in Informatics.
- **Dimitris Lamprinos** is a software engineer based in Thessaloniki. He holds a Bachelor's degree in Computer Science from the Aristotle University of Thessaloniki. Dimitris works on smart contract development and basic consensus development. He has significant experience in developing and scaling web2 applications in Amondo, Geekbot, and Vidpulse. He also has algorithmic cryptocurrency trading experience and has worked on developing and deploying smart contracts to the Ethereum blockchain since the early days of ICO boom.

This project will be executed in multiple phases, described in detail below.

4.1 Phase 1: Architectural Design

During this phase, we will study the Axelar codebase in depth and propose a suitable architecture, consisting of a CosmWasm module that will implement the on-chain Ethereum light client. Our approach will stick to the following principles:

- **Respectful of previous work.** Our proposed architecture will require minimal changes to the codebase, maintaining the existing design choices of the codebase.
- **Extensibility.** The architecture will allow extending the light client to different source chains in the future, beyond Ethereum.
- **Modularitity.** The architecture will be modular, in a way that allows upgrading to a different proof mechanism (e.g., SPV or ZK) and makes the protocol future-proof.

Deliverables. At the conclusion of this phase, we will deliver the following items:

- A detailed design document, that describes the architecture of the light client to be implemented on Axelar. The design document will detail the different components that we will develop. These include the CosmWasm module which will run within the Axelar codebase (acting as the verifier), as well as the modifications needed on the attestor codebase (also part of the full node), so that attestations can be reported to the on-chain verifier.

The document will be delivered in LaTeX PDF format and include pseudocode for the relevant components, so that their role is clear from a theoretical point of view.

4.2 Phase 2: Consensus Validation

During this phase, we will implement the consensus components of the on-chain light client. The implementation will be based on the outcome of Phase 1. Based on our current understanding of the Axelar architecture, during the consensus implementation phase we will implement the following:

- verification of the Ethereum sync committee *handover signatures*, including aggregate signature verification and quorum logic;
- verification of latest finalized block using sync committee signatures and Ethereum finalization logic.

Deliverables. At the conclusion of this phase, we will deliver the following items:

- A Cosmos CosmWasm module, which implements the consensus components of the on-chain Ethereum light client.
- A test suite, which verifies the correctness of the consensus components.
- A component that allows any friendly-but-untrusted party to submit to the on-chain verifier the consensus attestation data (sync committee signatures and block headers), produced via the Ethereum Beacon chain API.

4.3 Phase 3: Execution Validation

This phase consists of implementing the execution logic of the on-chain Ethereum light client, responsible for checking EVM-related data (such as Ethereum transactions and events). In particular, we will implement the verification of Ethereum execution data, given a validated finalized Ethereum block. This includes the verification of transactions and events and checking the validity of the Merkle proofs for transactions and/or events as needed.

Deliverables. At the conclusion of this phase, we will deliver the following items:

– An extension of the CosmWasm module (developed during Phase 2), that implements the execution components of the on-chain Ethereum light client.
– A test suite that verifies the correctness of the execution components of the CosmWasm module.

4.4 Phase 4: Theory, Deployment & Audit

During this phase, we will formalize the light client's construction and prove its security guarantees in a formal and rigorous manner.

We will also deploy the light client on Axelar's testnet and hand over the codebase with documentation to the Axelar team. We expect the Axelar team to conduct an audit of the codebase, before deploying it to the mainnet.

During this phase, we will help the Axelar team with the audit process and fix any issues that arise.

Deliverables. At the conclusion of this phase, we will deliver the following items:

– Testnet deployment of the Ethereum light client to the Axelar network (Lisbon).
– Handover of the codebase to the Axelar team with proper documentation.
– A LaTeX document that formalizes the light client construction and proves its security guarantees.
– Any fixes to the codebase that may be required as a result of an audit by the Axelar team.

References

1. S. Agrawal, J. Neu, E. N. Tas, and D. Zindros. Proofs of proof-of-stake with sublinear complexity, 2023.
2. E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on bft consensus, 2019.
3. V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.
4. V. Buterin and V. Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.
6. Ethereum. Minimal light client, 2023.
7. J. Groth. On the size of pairing-based non-interactive arguments. pages 305–326, 2016.
8. O. Kharif. Key player in ethereum infrastructure infura rejects centralization claim, 2022.
9. P. McCorry, C. Buckland, B. Yee, and D. Song. Sok: Validating bridges as a scaling solution for blockchains. Cryptology ePrint Archive, Paper 2021/1589, 2021. `https://eprint.iacr.org/2021/1589`.
10. A. Network. Axelar network: Connecting applications with blockchain ecosystems, Jan 2021.
11. E. N. Tas, D. Zindros, L. Yang, and D. Tse. Light clients for lazy blockchains, 2022.
12. Uniswap. Bridge assessment report, 2023.
13. G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151:1–32, 2014.
14. T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song. zkBridge: Trustless Cross-chain Bridges Made Practical. In *CCS*, pages 3003–3017. ACM, 2022.
15. A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. A. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. Sok: Communication across distributed ledgers. In *IACR Cryptology ePrint Archive*, 2019.

5 Appendix

5.1 Theoretical Analysis

In this section we present the model for the theoretical analysis of the bridge framework. We also present conjectures that we plan to prove if the grant is approved.

5.2 Preliminaries

Execution model. A ledger protocol Π is a distributed protocol that offers a *read* and *write* functionality. The ledger protocol is accompanied by a *validity language* $\mathbb{V}_\Pi$ containing all possible transactions that are "legal" according to the protocol Π. The *read* functionality returns a *ledger* in $\mathbb{V}_\Pi^*$, whereas the *write* functionality accepts a ledger and a *transaction* in $\mathbb{V}_\Pi$. The *ledger* is a finite sequence of pairs (t, tx) consisting of an integer timestamp t and a transaction tx.

A particular protocol *execution* is one run of an environment $\mathcal{Z}_{\Pi,\mathcal{A}}$ [5] which simulates the execution of Π among multiple honest and adversarial parties. The adversarial parties are controlled by one PPT adversary $\mathcal{A}$. The adversary and honest parties are modelled as Interactive Turing Machines. Out of the n_Π parties maintaining the ledger, t_Π are corrupt and are controlled by the adversary $\mathcal{A}$.

Networking. Time evolves in *synchronous* lock-step *rounds* denoted by the integers $1, 2, \ldots$. All parties have perfectly synchronized clocks, as they know the current round number. Messages which are broadcast by an honest party at a round r are received by all other honest parties at the beginning of the next round $r + 1$. The adversary can reorder messages, potentially different for each honest party, and inject different messages to the network tapes of different honest parties, but cannot drop messages. We work in the *client gossiping model*, where a new message received by any honest party is rebroadcast to all other honest parties; therefore, a party cannot identify the origin of a message. We assume there are no bandwidth constraints. The total execution time is polynomial, and each (honest or adversarial) party is bound to polynomial time per round.

Sequence notation. $x \in A$ denotes that either x is an element of set A or x appears in the sequence A. We use $|A|$ for the length of the sequence A. We write $A[i]$ for the i-th element of A (0-based) and $A[-i]$ for the i-th element of the inverse A (1-based); the element $A[-1]$ is the last element of A. We denote by $A[i{:}j]$ the subarray of A starting from element i (inclusive) and ending in element j exclusive. The notation $A[{:}j]$ means the sequence from the beginning up to j, whereas $A[i{:}]$ means the sequence from i till the end.

Ledgers. We denote ${}^{\Pi}L_r^P$ the result of executing the *read* functionality by party P operating in ledger protocol Π at round r. We will write $a \in S$ for an element a and a sequence S if the element a appears somewhere in the sequence S. For a ledger L we will also use the notation $\mathsf{tx} \in L$ to mean that the transaction tx appears in L with some timestamp, that is, there exists some $t \in \mathbb{N}$ such that $(t, \mathsf{tx}) \in L$. In all practical blockchain-based ledger protocols, the timestamp t associated with a transaction will correspond to the timestamp recorded on the block in which the transaction is confirmed. As such, timestamps will all be in the past and non-decreasing.

A ledger protocol is *safe* if the view of different honest parties is consistent.

Definition 1 (Persistence). *A ledger protocol Π is* persistent *during I if for all honest parties P_1, P_2 at rounds $r_1, r_2 \in I$, with $r_1 < r_2$, we have that ${}^{\Pi}L_{r_1}^{P_1} \preceq {}^{\Pi}L_{r_2}^{P_2}$.*

A ledger protocol is *live* if honest transactions make it to the ledger.

Definition 2 (Liveness). *A ledger protocol Π is* live *with liveness $u \in \mathbb{N}$ during I if, for all honest parties P_1, P_2, whenever P_1 attempts to write a* valid *transaction tx to the ledger at round $r \in I$, then for all $r' \in I$ with $r' \geq r + u$ the transaction is included in ${}^{\Pi}L_{r'}^{P_2}$.*

Definition 3 (Timeliness). *A ledger protocol Π is* timely *with timeliness $\nu \in \mathbb{N}$ during I if, for all honest parties P, and for all rounds r, it holds that:*

1. ${}^{\Pi}L_r^P[-1]$ *has a timestamp in the past, and*
2. ${}^{\Pi}L_r^P$ *has non-decreasing timestamps.*

Additionally, for all rounds $r \in I$*, it holds that the timestamps recorded in* ${}^{\Pi}L_r^P[|\,{}^{\Pi}L_{r-1}^P|:]$ *are after* $r - \nu$*.*

Ledger security is defined as a ledger protocol that is both safe and live.

Definition 4 (Security). *A ledger protocol is* secure *during* I *with liveness* u *if it is:*

1. persistent *during* I*,*
2. live *with liveness* u *during* I*, and*
3. timely *with timeliness* τ *during* I*.*

Bridges. A *bridge* $\Lambda(\Pi_1, \Pi_2)$ is an interoperability protocol between two ledger protocols Π_1 and Π_2. The execution is defined by a *shared* environment $\mathcal{Z}$ between Π_1 and Π_2, but each of Π_1 and Π_2 are simulated internally by the environment as before. The purpose of the bridge is to relay events or information that take place on the source side to the destination side. Without loss of generality, we consider Π_1 to be the *source*, and Π_2 to be the *destination*. If the bridge is bidirectional, our statements can be applied in both directions. For our purposes, the purpose is to move value from one side to the other.

A population of n nodes, among which at most t are adversarial, are responsible for operating the bridge. The bridge transmits certain transactions of interest from Π_1 to Π_2. These are *cross-chain* transactions and are defined by a function ϕ which accompanies the bridge protocol.

Definition 5 (Cross-chain transaction). *Let* ϕ *be an efficiently computable and invertible one-to-one injection* $V_{\Pi_1} \longrightarrow V_{\Pi_2} \cup \{\perp\}$*. A transaction* tx *on the source side is a* cross-chain *transaction if* $\phi(\mathsf{tx}) \neq \perp$*.*

A bridge is secure (Definition 8) if it guarantees two fundamental properties: safety (Definition 6) and liveness (Definition 7).

Definition 6 (Bridge Safety). *A bridge protocol* $\Lambda(\Pi_1, \Pi_2)$ *is* safe *with* safety $u_s \in \mathbb{N}$ *during* I *if, for all honest parties* P_1 *of* Π_1 *and* P_2 *of* Π_2 *and for all rounds* $r_1, r_2 \in I$ *with* $r_2 \geq r_1 + u_s$ *we have that, whenever* $(t, \mathsf{tx}) \in {}^{\Pi_2}L_{r_1}^{P_2}$ *with* $t \in I$ *and* $\phi^{-1}(\mathsf{tx}) \neq \perp$*, then* $\phi^{-1}(\mathsf{tx}) \in {}^{\Pi_1}L_{r_2}^{P_1}$*.*

Definition 7 (Bridge Liveness). *A bridge protocol* $\Lambda(\Pi_1, \Pi_2)$ *is* live *with* liveness $u_\ell \in \mathbb{N}$ *during* I *if, for all honest parties* P_1 *of* Π_1 *and* P_2 *of* Π_2 *and for all rounds* $r_1, r_2 \in I$ *with* $r_2 \geq r_1 + u_\ell$ *we have that, whenever* $(t, \mathsf{tx}) \in {}^{\Pi_1}L_{r_1}^{P_1}$ *with* $t \in I$ *and* $\phi(\mathsf{tx}) \neq \perp$*, then* $\phi(\mathsf{tx}) \in {}^{\Pi_2}L_{r_2}^{P_2}$*.*

Definition 8 (Bridge Security). *A bridge* Λ *is* secure *with safety* $u_s \in \mathbb{N}$ *and liveness* $u_\ell \in \mathbb{N}$ *during* I *if it is safe with safety* u_s *during* I *and live with liveness* u_ℓ *during* I*.*

Conjecture 1 (Bridge security under temporary dishonest majority). We conjecture that a bridge Λ is safe and life under temporary dishonest majority.

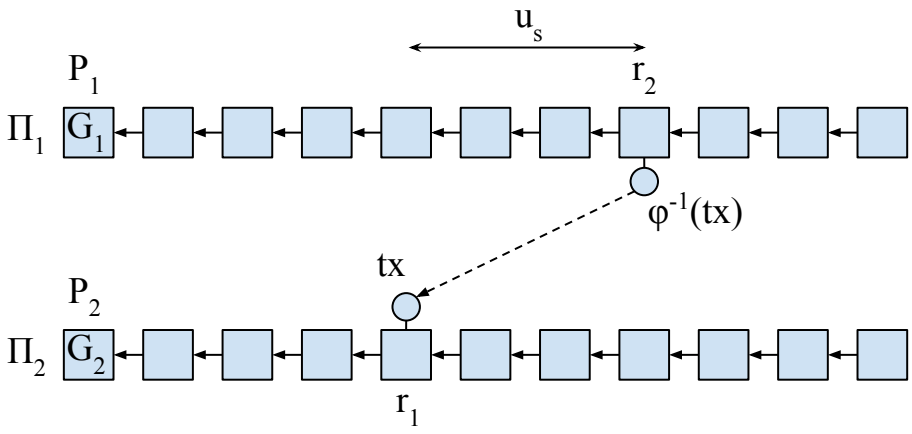

Fig. 2: Bridge safety

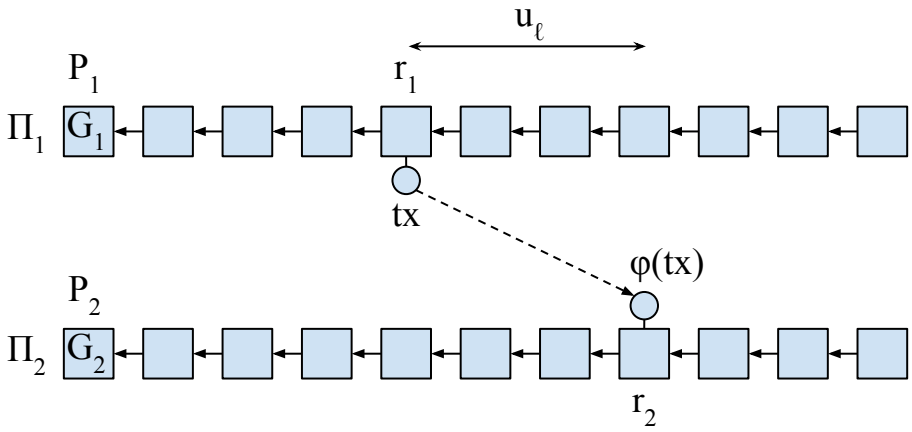

Fig. 3: Bridge liveness

About Common Prefix

Common Prefix is a blockchain research, development, and consulting company consisting of a small number of scientists and engineers specializing in many aspects of blockchain science. We work with industry partners who are looking to advance the state-of-the-art in our field to help them analyze and design simple but rigorous protocols from first principles, with provable security in mind.

Our consulting and audits pertain to theoretical cryptographic protocol analyses as well as the pragmatic auditing of implementations in both core consensus technologies and application layer smart contracts.