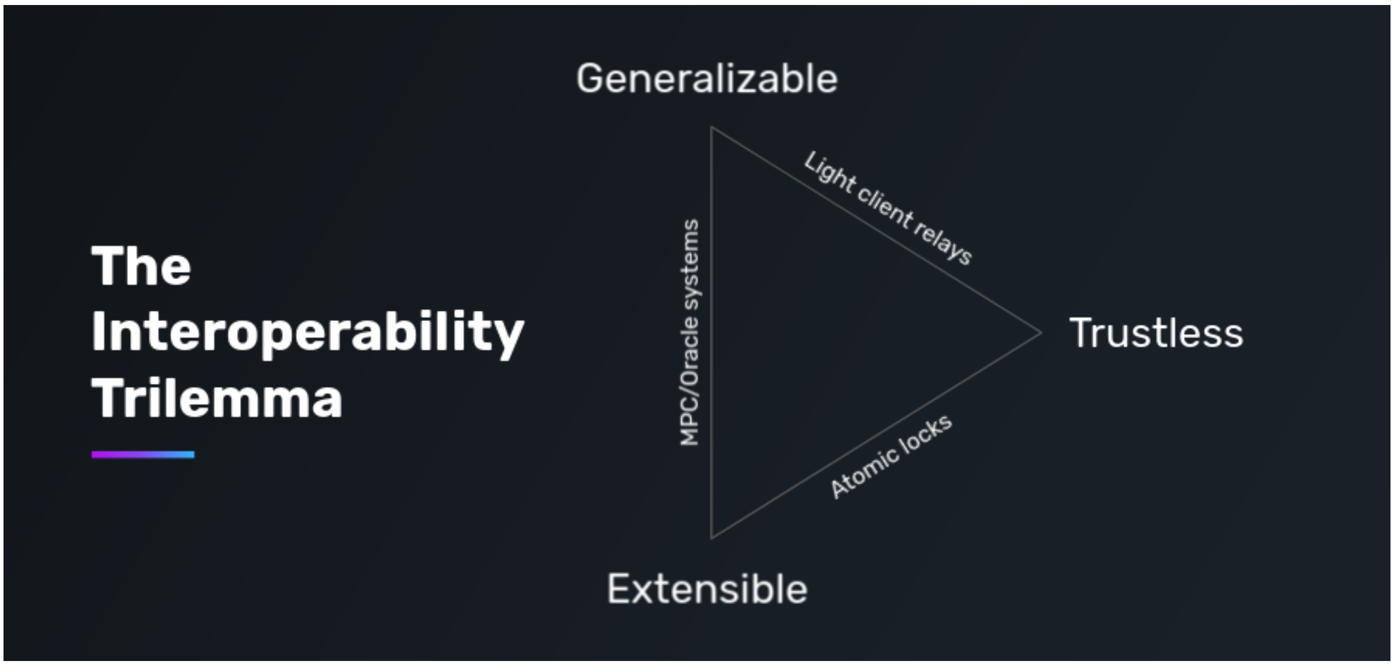
Assignment 6

Question 1. Interoperability

Question 1.1

> Explain each aspect of the interoperability trilemma. Provide an example of a bridge protocol explaining which trade-offs on the trilemma the bridge makes.

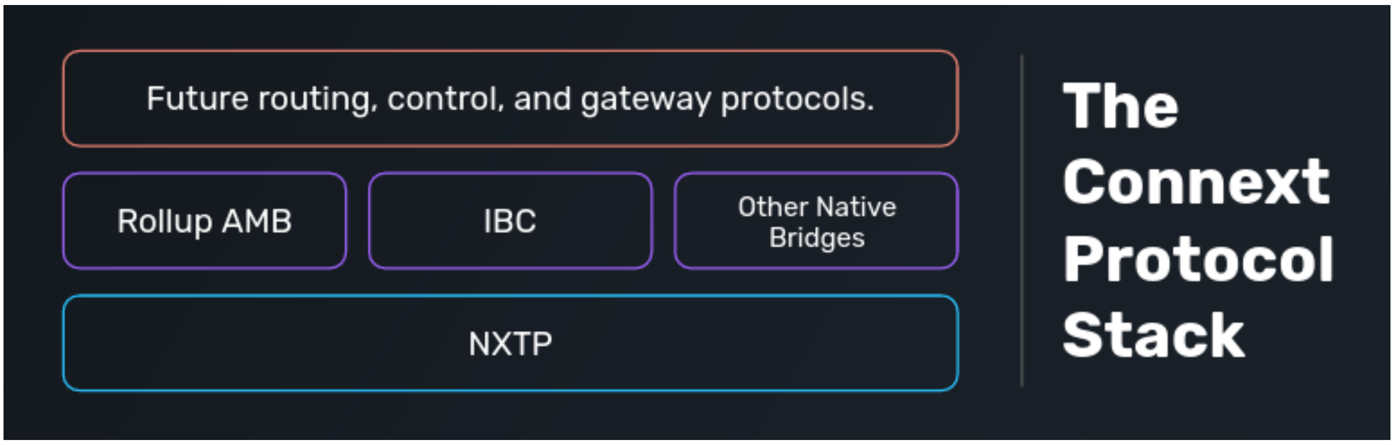


- Generalizable: It means the protocol's ability to handle arbitrary cross-domain data. If the bridge accomplishes the property, it enjoys great speed improvement while network effect due to the effective complexity which is `O(1)`. If a single transaction could be broadcasted thanks to the bridge, it could give full access to multiple blockchains that are compatiable to each other. However, the implementation of the property could lead to unintended but harmful consequences to the ecosystem.
- Trustlessness: It would maximizes the security of the blockchain to the underlying consensus level. In one way, the property could be achieved by providing economical disadvantages to vallidator to be corrupted.
- Extensibleness: It provides the seamless extension to any other blockchain domains regardless of different security levels.

According to the article, it claims that any interoperability protocols on the market cannot serve all of the properties stated above but are forced to choose two at the expense of remaining one. The article elucidates the point by giving a example of IBC and NEAR Rainbow Bridge. The aforementioned bridges have a strong upside for `Generalizblity` since they are able to transmit arbitrary data using their own relayer mechanism. In addition, they are fit at `trustless` property since they gives a trust to relayer when transmitting arbitrary data while no lockup periods are required either. However, the bridges are failed at `Extensibleness` considering the status quo that instantiating and deploying new light client is required for both source and destination chains. Developers need to deploy *custom built* for each type of domain. This design could lead to another latency layers to be synchronized with data and doing validation for a proof.

Question 1.2

> Are there any projects that focus on solving trilemma similarly like Ethereum solves the scalability issue? If yes describe how it solves the problem.


Future routing, control, and gateway protocols.
Rollup AMB
IBC
Other Native Bridges
NXTP
The Connext Protocol Stack

- Context Protocol is working on this; the protocol has been working on building locally verified system - which is called NXTP - that aims to be usable on any domain but to support a security level of underlying domain chains. In addition, to guarantee generalizeability, it supports `natively verified protocols` on top of NXTP (as a Layer 2 concept) for seamless connection with other domains. For example, IBC and other general native bridges would be involved as Layer 2.

Question 2. Ultra light clients

Question 2.1

> Describe a specification for a light client based on zero-knowledge proofs. You should explain at least how to get the client synced up to the current state of the blockchain. Preferably go as far as explain how transaction inclusion proofs are generated too.

- To begin with, the goal is to build a light client that uses ZKP technologies to synchronize data from the blockchain and verify whether the transaction is included in the current state or not.
- When a light client is initialized by connecting itself to the full-RPC node, the server needs the following:
  1. zk-proofs of committee transition from epoch `i` to `j` . The proof validates whether the suggested block is consistent without any prone error, and also checks the transaction signature. When it comes to Mina protocol, it uses a recursive zk-snark proof so that it validates the current block and previous blocks again and again and again...
  2. The merkle roots of recipient, state, and transactions. The given input is used to validate the current state.
  3. After verifying two parameters given above, the client attempts to store the status to historical merkle roots. The transactions, states, and recipients has their own merkle roots. Flyclient, which implements a non-interactive approach that enables any arbitrary clients to render zk-proofs to be used even after verficiation for reducing proof-generation computation, uses Merkle Mountain Root. The client saves transaction root that adopts MMR; only saves to the merkle root.

> Within every epoch, all blocks are connected to each other in the form of a MMR and the MMR root is included in every block header. By verifying the MMR commitment, it become possible for light client to validate the block, without having to perform expensive BLS signature verification. Reference

- The light client does not have any persistent connection to full RPC nodes, which implies that periodical synchronization process is required for the client. A light client could raise a request that queries the specific block range to the latest one. During the querying process, a zk-snark proof is generated; it could be also generated in a recursive zk-snark proof to show that all blocks are up to date without any worongdoings or errors involved.

Question 2.2

> What is the relevance of light clients for bridge applications? How does it affect relayers?

- When it comes to bridge application, the light client plays an important role in guaranteeing safe transactions over-the-chains. When a user locks own token on Ethereum to use it on Harmony, it is required that the light-client from Ethereum side which is deployed on Harmony side to validate the lock transaction on Ethereum side. The similiar action happens in reverse sides too, which implies that the light-client from Harmony side is required to be deployed to verify lock and unlock transactions happens in Ethereum side.
- Relayer also plays an important role in "relaying" the block headers and proofs. They can guarantee trustless valid delivery without placing another trust in third party since they does continuously forward data to other nodes and vice versa. The mentioned role were also played in non-zk approach. In zk-approach, Relayer also does pushing the status quo of specific chain to other chains through deployed smart contract. The status from other chains provided by a light client is also went to Relayer. The light client's job is to verify the validity of messages using block headers and message proofs received from Relayer.

Question 2.3

> Suppose code from Plumo is updated and was working in production on Celo. What would be the main difficulty in porting Plumo over to Harmony?

- I think that the implementation of validator set and usage of BLS signatures in Harmony protocol is similar to Celo protocol, which renders me to think that no major obstacles found for porting Plumo over to Harmony. They also share PoS (Proof-of-Stake) mechanism to choose validators, and uses PBFT algorithm.
- However, as stated on the link, Harmony protocol team would require to put much effort to translate the circuit of Plumo deployed at Celo protocol to fit to the Harmony's block structure. Specifically, sharding mechanism working on Harmony protocol for 250 validators should be supported on newly constructed circuits. They also need to reimplement cryptographic gardgets like Keccak Gardget, new curve arithmetic, and an aggregated BLS signature verifications in circuits with PLONK and vice versa. In current status quo, Celo's PLONK does not support recursive zk-Snark proof so that it is needed to be reimplement recursive method to make sure proof size to be controlled.

Question 3

Question 3.1.a

> Check out Horizon repository. Briefly explain how the bridge process works (mention all necessary steps). a) Comment the code for:
>
> - harmony light client
> - ethereum light client
> - token locker on harmony
> - token locker on ethereum
> - test contract
>
> Provide commented code in your submission.

How does Horizon Bridge work?

Ethereum to Harmony for Asset Transfer (which is on docs)

- User locks ERC20 on Ethereum by transferring to bridge smart contract and obtains the hash of this transaction from blockchain
- User sends the hash to EProver and receives proof-of-lock
- User sends the proof-of-lock to bridge smart contract on Harmony
- Bridge smart contract on Harmony invokes ELC and EVerifier to verify the proof-of-lock and mints HRC20 (equivalent amount)
- Note
  - ELC: Ethereum Light Client Smart Contract on Harmony
  - EVerifier: Ethereum Verifier Smart Contract on Harmony
  - EProver: Ethereum Prover is an Ethereum full node or a client that has to a full node

Harmony to Ethereum for Asset Redemption (which is on docs)

- User burns HRC20 on Harmony using Bridge smart contract and obtains the hash of this transaction from blockchain.
- User sends the hash to HProver and receives proof-of-burn.
- User sends the proof-of-burn to bridge smart contract on Ethereum.
- Bridge smart contract on Ethereum invokes HLC and HVerifier to verify the proof-of-burn and unlocks ERC20 (equivalent amount)
- Note
  - Harmony Light Client, HLC is a smart contract on Ethereum
  - Harmony Verifier, HVerifier, is a smart contract on Ethereum
  - Harmony Prover, HProver, is a Harmony full node or a client that has access to a full node
  - Harmony Relayer does relaying every checkpoint block header information to HLC.
  - checkpoint blocks: 1 block every x blocks, where $1 \leq x \leq 16384$, 16384 is the #blocks per epoch

Asset Bridging Phase

1. Users transfer assets to a source chain bridge contract, which generates a transaction hash.
2. The hash is checked by a prover contract and generates a proof of asset lock.
3. The user then sends the proof of lock to the bridge contract on the designated chain.
4. The proof of lock runs the `validateAndExecuteProof` function in the token locker smart contract with `TokenLockerOnHarmony.sol` and `TokenLockerOnEthereum.sol`.
5. If the evidence is valid, the token mining/unlocking is executed.
6. There is a validation function call in the light client. For example, `isValidCheckPoint` is a function in `HarmonyLightClient.sol`. The client uses `FlyClient` method that includes block history committment in the header. The function returns whether it is `true` or `false` in `epochMmrRoots[epoch][mmrRoot]` after validating the given checkpoint.
7. On the Ethereum side, relayers call `submitCheckpoint` to submit it, and the checkpoint includes `mmrRoot` that accumulates block history on the Harmony protocol. In addition, the function called `addBlockHeader` does the work for verification on all of the relayed block header is correct and represents the longest chain.

Comment on Codes

- Harmony Light Client
- Ethereum Light Client
- TokenLockerOnHarmony
- TokenLockerOnEthereum
- bridge.hmy.js

Question 3.1.b

> Why HarmonyLightClient has **bytes32 mmrRoot** field and EthereumLightClient does not? (You will need to think of blockchain architecture to answer this)

- The difference is at consensus level. HarmonyLightClient takes the `FlyClient` approach while EthereumLightClient takes `SPV` approach.
- Ethereum is based on Proof-of-work which means that `mmrRoot` is required for `EthereumLightClient` to include transaction validation. Each block is required to have a `mmrRoot` verification and proof-of-work verification, thus the replayer sends block headers to the smart contract. This verification is expensive so that difficult to initiate with the gas limit to finalize the block considering the amount of blocks (more than 700 blocks). Harmony protocol uses proof-of-stake so that the block is verified with validating BLS signatures. Thus, we can think that Ethereum requires more resources so that Ethereum needs tracking every single header with the needs of verifying `Ethash`.
- For the case of Harmony side, BLS signatures - which is included in an `mmrRoot` in each block header - could leverage its strength to verify a signature one block per every epoch (which is around 18 hours, 32768 blocks, and Horizon executes the maximum 16384 blocks in one update). The relayer only neds to send one block per every epoch, since the BLS signatures makes the protocol to be available with combining multiple messages with different set of signers to be involved in a single signature.
- Ethereum requires a fork version if to include such signature commitment in the block header so that it is forced to utilize `SPV` proofs to retrieve all block headers and genesis hash.

Question 3.2

> What are checkpoint blocks? Do they differ from epoch blocks and how? Why are they used?

- I think checkpoint blocks represents block range in one epoch. For example, it could represent 1 to 16384 blocks, which is the maxmium of block amount. As I stated above, HarmonyLightClient does not uses SPV-like client style to verify all of the block headers but to utilizes the way of checkpoint blocks to reduce cost, storage usage and time. Harmony Light Client takes BLS multi signauture approaches to combine multiple blocks in a single checkpoint block with a epoch.
- Harmony protocol can adjust its block amount in a single checkpoint block. On Harmony protocol, a epoch represents the fixed 32768 blocks with taking around 18 hours, Meanwhile, the size of single checkpoint could be accommodating to optimize gas fee and transaction speed.

Question 3.3

> *[**Infrastructure only]*** Horizon still doesn't use zk-proofs in order to speed up light clients. What changes would you need to make to the code in order to apply initial state sync through zk-snarks? Provide pseudo code of improved version of light client.

- Relayer calls prover periodically to render a checkpoint proof using C1 circuits and update `mmrRoot`.
- The parameter for `initialize` state function is `mmrRoot` of genesis block with `zkrProof` and assigns a `MmrRoot` variable to new `mmrRoot` of the latest block of the recent epoch.

```
// NewHarmonyLightClient.sol
function initialize(bytess32 memory genesisMmrRoot, Proof memory zkrProof) external
initializer {
  ...
  require(verifyProof(zkrProof.input, genesisMmrRoot), "given recursive proof is
invalid!");
  MmrRoot = genesisMmrRoot;
  MmrRoot = zkrProof.roots["mmrRoot"];
}

function submitCheckpoint(bytes memory rlpHeader, Proof memory zkCheckPointProof)
external onlyRelayers whenNotPaused {
  // ...
  require(verifyProof(zkCheckPointProof, MmrRoot), "given recursive proof is
invalid!");
  MmrRoot = zkCheckpointProof.roots["mmrRoot"];
}
```

Question 4

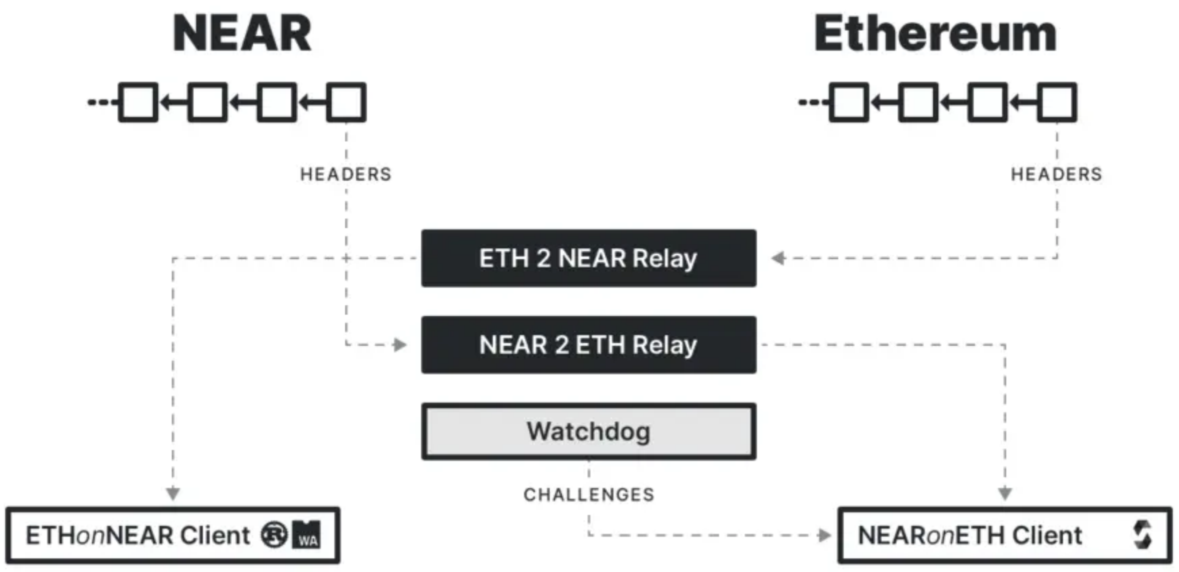
Question 4.1

> ***[Infrastructure only]*** Comment code implemented in [NearBridge.sol](). Note that they're using fraud proofs and not zk proofs.

- [NearBridge.sol]()

Question 4.2

> Explain the differences between Rainbow bridge and Horizon bridge. Which approach would you take when building your own bridge (describe technology stack you would use)?



- Rainbow bridge takes precedence over Horizon bridge in the following points.
  - Trustlessness
    - Rainbow bridge has a challenger for monitoring the network and economic incentive for users to challenge a possible invalidate NEAR block headers. EIP665 states that the need of "EIP-665: Precompile for Ed25519 signature verification" implementation because it reduces the latency of cross-bridge interactions between NEAR-ETH. [Link]() If the proposal is accepted, NEAR protocol does not allow to exponentially increase the minimum gas price of Ethereum blocks by more than 2x with every block for more than four hours. If so, the constraints could be lifted up.
    - However, for Horizon approach, it relies on a trusted setup done by Harmony relayer to add the checkpoint to the Ethereum contract for the light client.
  - Checkpoint size is adaptable
    - Rainbow bridge might use the functionality to only publish new block headers during status updates, which is more flexible and light, because every NEAR header has a root of the merkle tree computed from all headers before it. Meanwhile, for Horizon approach, it uses a MMT tre with checkpoint block and size could be adaptable.
- Horizon bridge takes precedence over Rainbow bridge in the following points.

  - NEAR need a single signature for each validator which renders more economically inefficient approach. However, Harmony protocol utilizes BLS multi-signature to check the signature of validator sets that verifies blocks in a single epoch with one signature. This approach saves the gas cost when verifying signature for Horizon bridge.
  - Rainbow bridge stores a week of Ethereum block header in NEAR blockchain for storage usage that mandaters user to burn token in Ethereum and min token in NEAR after getting locked the token in Ethreum for 7 days. It illicits bad user experience in using Rainbow bridge.
- My opinion
  - Due of the inefficiency of NEAR to Ethereum interactions, the watchdog service is working on Ethereum to challenge block headers from NEAR transactions. I believe that Horizon bridge approach is much modular so that able to extend and scale in generic form outside ERC20 tokens.
  - However, if possible to combine both Rainbow bridge, Horizon bridge and Plumo, it would be best - implementing the Rainbow bridge architecture with utilizing recursive zksnark verification by introducing Celo's Plumo approach.

Question 4.3

> **[Bonus]** Explain how merkle mountain ranges work and how they can be used in order to do block inclusion proofs. (You can check FlyClient for a light client implementation that uses MMR).

- The Merkle Mountain Range, like the Merkle Tree, is an append-only data structure that still be able to be express in a flat list type. As soon as there are two children, a new parent will be computed. Each parent has a concatenated hash of their two sibilings, which is similiar to a merkle proof such that supplying the node and siblings. Given a node's location and the size of the list, we can quickly discover these siblings thanks to the MMR's list nature. The MMR is easy way to add nodes inside the tree, which leads to have multiple binary trees.
- The MMR is a better approach than binary Merkle tree in block inclusion proof since the MMR structure can be expressed by array without amending the tree height. In addition, the MMR is append only so that the value of node does not change but the hgistory of the MMR roots is stored inside thre tree forever. Think Git stores all of the commit changes and its history internally!
- Thus, I believe that the MMR structure is efficient appending approach for the prover side while efficient block verification is available for the verifier. In addition, the full node is able to prove that the transaction belongs to the longest chain by providing an MMR proof, in addition to the proof that shows the transaction inclusion status in the block.

Question 5 Thinking in ZK

Question 5.1

> If you have a chance to meet with the people who built the above protocols what questions would you ask them?

- Horizon and Rainbow bridge are fairly domain specialized. Is it realistic or planned to adapt the protocol to accommodate Ethereum's layer 2 protocols?
- For Plumo, what is the biggest obstacles for the protocol to build a recursive zk-snark block proving ways for Ethereum? Any changes required to happen for the implementation compared to existing Ethereum blockchain and Ethereum 2.0?
- Could Horizon bridge develop their existing implementation by eradicating a trusted setup done by

Harmony protocol relayer?