# PStake Audit

## Executive Summary

pStake is a liquid staking solution which helps in unlocking the liquidity of the staked assets. pStake solution is built up using ERC20 contracts over Ethereum Blockchain. pStake is thoroughly focused on security and this document describes the different aspects of the smart contracts and mechanisms which will help in auditing the smart contracts.

The audit for the Smart contracts is divided into 3 phases:

* PHASE 1: The first phase includes 3 main smart contracts which help in building up the liquid stacking solution.
* PHASE 2: The second phase includes the Farming Yield Mechanism which helps staking the assets from other contracts.
* PHASE 3: The third phase includes the Redemption Pool Mechanism which helps in unbinding the tokens fast.

Apart from this pStake is also looking for the Audit of Bridge component which helps the Cosmos Blockchain interact with the Ethereum clients. The audit for Bridge is also divided into 3 phases:

* PHASE 1: The first phase includes core bridge components used for monitoring Cosmos and Ethereum Blockchains..
* PHASE 2: The second phase includes the Multi Validator implementation
* PHASE 3: The third phase includes the insurance pool and slashing algorithm for validators.

Overall, the contracts and bridge includes clean and modular code with focus on security and access control.

## Scope

The main specification which are to be audited are listed below:

* Inconsistency between the specification and Implementation
* Flaws in design, logic or access control
* Arithmetic Overflows and Underflows
* Function Visibility
* Compiler Warnings
* Limits on ByteCode and Gas Usage
* Reentrancy, Code Injection and Denial of Service Attacks
* Call stack Depth
* Time Manipulation, Unbounded Loops and
* External Calls and Validation of inputs for public/external functions.
* Authentication mechanism
* Race Conditions and other known attacks
* Failure States, Speed Bumps and Circuit Breakers

As this is a finance based application, high risk areas are of primary concern and open for suggestions. On a general outlook, we have defined the high risk areas which need to be addressed at the point of audit. Any vulnerabilities related to high risk areas has to be properly categorized and reported to pStake.

### High Risk Areas

* External and Public Functions
* Assembly and Low Level Calls
* Super user privileges
* Timing or network congestion
* Payments and Withdrawals

### Out of Scope

This audit is only for the smart contracts, the following listed below are not part of the scope:

* All the components which do not belong to pStake.
* All Smart Contracts belonging to Open Zeppelin Library.

### Depth of Scope

The goal of the smart contract audit is to meticulously go through all the smart contracts and identify security flaws and vulnerabilities.

Depth of the Audit Includes:

* Overall Analysis of Smart Contract Code and Documentation
* Code Review, including third party interaction, library structure, functionality and cryptography
* Automated and Manual analysis of application, nested components, input fields, actions and all the requests.
* Bug Scanning, both at binary and source code level. Potential deviations from coding guidelines and security practices.
* Scanning Result Verification, false positives and false negatives which might affect the application.

## Tools used by pStake

Truffle, Remix, Ganache, Mythrill, Geth, Metamask, Surya

## Files Included for Smart Contract Audit

Phase 1:

* LiquidStaking.sol
* UTokens.sol
* STokens.sol

Phase 2:

* LiquidStaking.sol
* UTokens.sol
* STokens.sol
* FarmingLP.sol

Phase 3:

* LiquidStaking.sol
* UTokens.sol
* STokens.sol
* RedemptionPool.sol
* FarmingLP.sol

## Deliverables

The following deliverables are expected from the audit team:

* Executive summary of Audit
* Project approach
  + Rules of Engagement
  + Description of security audit methodology
  + Scope description in detail
* Vulnerability analysis, findings and recommendations
* Workflow of security audit
* Further information on findings and detailed recommendations
* Conclusion
* Summary recommendations and further steps

### Glossary

For the purpose of audit we would like to adopt the OWASP risk rating methodology. We would like to get the information about the following factors:

* **Likelihood**: Likelihood of a security vulnerability to be encountered or exploited in the wild.
* **Impact**: Technical and Business related consequences of the exploit.
* **Severity**: This is derived on the bases of Likelihood and Impact. (Low, Medium, High, Critical)

## Smart Contracts System Overview

The audit has to be carried out in 3 phases, the functionality, contracts and capabilities for the same are described below:

### Phase 1

#### UTokens Contract

This contract is used for minting and burning ustkTokens which are pegged with the blockchains for which the liquid stacking mechanism is enabled. The tokens are not staked yet, they act as a representation of assets on other Blockchain with Ethereum. This contract includes the ERC20, Safemath and Ownable Contract from the Openzepplin Library. The details about the roles, access rights and visibility of functions is submitted below

##### 

##### Description from Surya

Contracts Description Table

| Contract | Type | Bases | | |

|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|

| └ | **\*\*Function Name\*\*** | **\*\*Visibility\*\*** | **\*\*Mutability\*\*** | **\*\*Modifiers\*\*** |

||||||

| **\*\*UTokens\*\*** | Implementation | ERC20Upgradeable, IUTokens, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | mint | Public ❗️ | 🛑 | whenNotPaused |

| └ | burn | Public ❗️ | 🛑 | whenNotPaused |

| └ | setSTokenContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | setLiquidStakingContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*IUTokens\*\*** | Interface | IERC20Upgradeable |||

| └ | mint | External ❗️ | 🛑 |NO❗️ |

| └ | burn | External ❗️ | 🛑 |NO❗️ |

| └ | setLiquidStakingContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | setSTokenContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

Legend

| Symbol | Meaning |

|:--------:|-----------|

| 🛑 | Function can modify state |

| 💵 | Function is payable |

##### UML Diagram

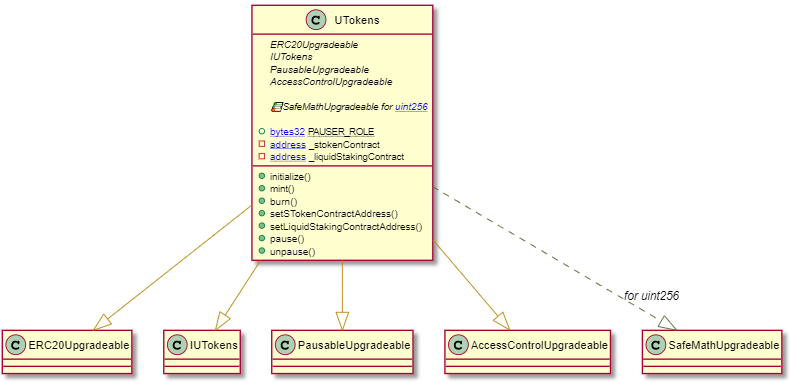
Variables and Functions which are used with the Utokens Contract are defined below with the uml diagram.

Variables:

* \_sTokenContract: Stores the address for the Stokens Contract.
* \_liquidStakingContract: Stores the address for the Liquid Staking Contract

Functions:

* \_constructor(): Set the constructor with an ERC20 contract.
* mint(): Mints the provided token amount for the address supplied.
* burn (): Burns the provided token amount for the address supplied.
* setSTokenContractAddress(): Sets the address for Stoken contract, only called by the owner.
* setLiquidStakingContractAddress(): Sets the address for the Liquid Staking contract, only called by the owner.



#### STokens Contract

This contract is used for minting and burning stkTokens which are pegged with the ustkTokens for which the liquid stacking mechanism is enabled. This contract includes the ERC20, Safemath and Ownable Contract from the Openzepplin Library. The tokens are considered staked and the corresponding asset on other Blockchain is staked as per the amount of tokens in this contract for a specific address.

The details about the roles, access rights and visibility of functions is submitted below

##### Description from Surya

Contracts Description Table

| Contract | Type | Bases | | |

|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|

| └ | **\*\*Function Name\*\*** | **\*\*Visibility\*\*** | **\*\*Mutability\*\*** | **\*\*Modifiers\*\*** |

||||||

| **\*\*STokens\*\*** | Implementation | ERC20Upgradeable, ISTokens, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | setUTokensContract | Public ❗️ | 🛑 | whenNotPaused |

| └ | setRewardRate | Public ❗️ | 🛑 | whenNotPaused |

| └ | getRewardRate | Public ❗️ | | whenNotPaused |

| └ | getStakedBlock | Public ❗️ | | whenNotPaused |

| └ | mint | Public ❗️ | 🛑 | whenNotPaused |

| └ | burn | Public ❗️ | 🛑 | whenNotPaused |

| └ | \_calculateRewards | Internal 🔒 | 🛑 | |

| └ | calculatePendingRewards | Public ❗️ | | whenNotPaused |

| └ | calculateRewards | Public ❗️ | 🛑 | whenNotPaused |

| └ | \_beforeTokenTransfer | Internal 🔒 | 🛑 | |

| └ | setLiquidStakingContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*UTokens\*\*** | Implementation | ERC20Upgradeable, IUTokens, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | mint | Public ❗️ | 🛑 | whenNotPaused |

| └ | burn | Public ❗️ | 🛑 | whenNotPaused |

| └ | setSTokenContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | setLiquidStakingContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*IUTokens\*\*** | Interface | IERC20Upgradeable |||

| └ | mint | External ❗️ | 🛑 |NO❗️ |

| └ | burn | External ❗️ | 🛑 |NO❗️ |

| └ | setLiquidStakingContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | setSTokenContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*ISTokens\*\*** | Interface | IERC20Upgradeable |||

| └ | mint | External ❗️ | 🛑 |NO❗️ |

| └ | burn | External ❗️ | 🛑 |NO❗️ |

| └ | setRewardRate | External ❗️ | 🛑 |NO❗️ |

| └ | getStakedBlock | External ❗️ | |NO❗️ |

| └ | getRewardRate | External ❗️ | |NO❗️ |

| └ | calculatePendingRewards | External ❗️ | |NO❗️ |

| └ | calculateRewards | External ❗️ | 🛑 |NO❗️ |

| └ | setLiquidStakingContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | setUTokensContract | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

Legend

| Symbol | Meaning |

|:--------:|-----------|

| 🛑 | Function can modify state |

| 💵 | Function is payable |

##### UML Diagram

Variables and Functions which are used with the Stokens Contract are defined below with the uml diagram.

Variables:

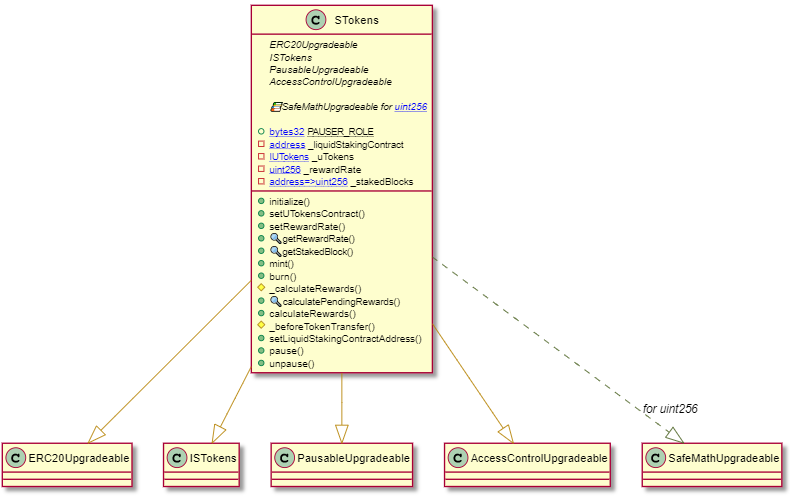
* \_liquidStakingContract: Stores the address for the Liquid Staking Contract.
* \_uTokensContract: Stores the address for the UTokens Contract
* \_rewardRate: Stores the rate of reward awarded for staking
* \_stakedBlocks: Store the block number for the user when staking is initiated

Functions:

* \_constructor(): Set the constructor with an ERC20 contract.
* setUTokenContractAddress(): Sets the address for the Utoken contract, only called by the owner.
* setRewardRate(): Sets the reward rate for the contract, only called by the owner.
* getRewardRate(): Gets the reward rate for the contract.
* getStakedBlock(): Gets the stacked Block number for the specific address.
* mint(): Mints the provided token amount for the address supplied.
* burn (): Burns the provided token amount for the address supplied.
* \_calculateRewards(): Private Function to calculate the rewards for a user and mint to the user address.
* calculatePendingRewards(): Calculate the pending rewards for a user.
* calculateRewards(): Public Function to calculate the rewards and mint for the user.
* \_beforeTokenTransfer(): Private Function to run the rewards calculation before any token transfer.
* setLiquidStakingContractAddress(): Sets the address for the Liquid Staking contract, only called by the owner.

Interface

* Only mint function is used from Utokens Contract which is added with this contract.



#### Liquid Staking Contract

This contract is used for staking, unstaking and withdrawal of tokens.This is a head contract which interacts with both UTokens and STokens contracts. This contract includes the Safemath and Ownable Contract from the Openzepplin Library. This contract includes the public functions where the users are going to interact. The minting and burning of tokens is handled by this contract.

The details about the roles, access rights and visibility of functions is submitted below:

##### Description from Surya

Contracts Description Table

| Contract | Type | Bases | | |

|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|

| └ | **\*\*Function Name\*\*** | **\*\*Visibility\*\*** | **\*\*Mutability\*\*** | **\*\*Modifiers\*\*** |

||||||

| **\*\*LiquidStaking\*\*** | Implementation | ILiquidStaking, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | setUTokensContract | Public ❗️ | 🛑 | whenNotPaused |

| └ | setSTokensContract | Public ❗️ | 🛑 | whenNotPaused |

| └ | generateUTokens | Public ❗️ | 🛑 | whenNotPaused |

| └ | withdrawUTokens | Public ❗️ | 🛑 | whenNotPaused |

| └ | stake | Public ❗️ | 🛑 | whenNotPaused |

| └ | unStake | Public ❗️ | 🛑 | whenNotPaused |

| └ | withdrawUnstakedTokens | Public ❗️ | 🛑 | whenNotPaused |

| └ | getTotalUnbondedTokens | Public ❗️ | | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*UTokens\*\*** | Implementation | ERC20Upgradeable, IUTokens, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | mint | Public ❗️ | 🛑 | whenNotPaused |

| └ | burn | Public ❗️ | 🛑 | whenNotPaused |

| └ | setSTokenContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | setLiquidStakingContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*IUTokens\*\*** | Interface | IERC20Upgradeable |||

| └ | mint | External ❗️ | 🛑 |NO❗️ |

| └ | burn | External ❗️ | 🛑 |NO❗️ |

| └ | setLiquidStakingContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | setSTokenContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*STokens\*\*** | Implementation | ERC20Upgradeable, ISTokens, PausableUpgradeable, AccessControlUpgradeable |||

| └ | initialize | Public ❗️ | 🛑 | initializer |

| └ | setUTokensContract | Public ❗️ | 🛑 | whenNotPaused |

| └ | setRewardRate | Public ❗️ | 🛑 | whenNotPaused |

| └ | getRewardRate | Public ❗️ | | whenNotPaused |

| └ | getStakedBlock | Public ❗️ | | whenNotPaused |

| └ | mint | Public ❗️ | 🛑 | whenNotPaused |

| └ | burn | Public ❗️ | 🛑 | whenNotPaused |

| └ | \_calculateRewards | Internal 🔒 | 🛑 | |

| └ | calculatePendingRewards | Public ❗️ | | whenNotPaused |

| └ | calculateRewards | Public ❗️ | 🛑 | whenNotPaused |

| └ | \_beforeTokenTransfer | Internal 🔒 | 🛑 | |

| └ | setLiquidStakingContractAddress | Public ❗️ | 🛑 | whenNotPaused |

| └ | pause | Public ❗️ | 🛑 |NO❗️ |

| └ | unpause | Public ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*ISTokens\*\*** | Interface | IERC20Upgradeable |||

| └ | mint | External ❗️ | 🛑 |NO❗️ |

| └ | burn | External ❗️ | 🛑 |NO❗️ |

| └ | setRewardRate | External ❗️ | 🛑 |NO❗️ |

| └ | getStakedBlock | External ❗️ | |NO❗️ |

| └ | getRewardRate | External ❗️ | |NO❗️ |

| └ | calculatePendingRewards | External ❗️ | |NO❗️ |

| └ | calculateRewards | External ❗️ | 🛑 |NO❗️ |

| └ | setLiquidStakingContractAddress | External ❗️ | 🛑 |NO❗️ |

| └ | setUTokensContract | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

||||||

| **\*\*ILiquidStaking\*\*** | Interface | |||

| └ | generateUTokens | External ❗️ | 🛑 |NO❗️ |

| └ | withdrawUTokens | External ❗️ | 🛑 |NO❗️ |

| └ | stake | External ❗️ | 🛑 |NO❗️ |

| └ | unStake | External ❗️ | 🛑 |NO❗️ |

| └ | withdrawUnstakedTokens | External ❗️ | 🛑 |NO❗️ |

| └ | getTotalUnbondedTokens | External ❗️ | |NO❗️ |

| └ | setUTokensContract | External ❗️ | 🛑 |NO❗️ |

| └ | setSTokensContract | External ❗️ | 🛑 |NO❗️ |

| └ | pause | External ❗️ | 🛑 |NO❗️ |

| └ | unpause | External ❗️ | 🛑 |NO❗️ |

Legend

| Symbol | Meaning |

|:--------:|-----------|

| 🛑 | Function can modify state |

| 💵 | Function is payable |

##### UML Diagram

Variables and Functions which are used with the Liquid Staking Contract are defined below with the uml diagram.

Variables:

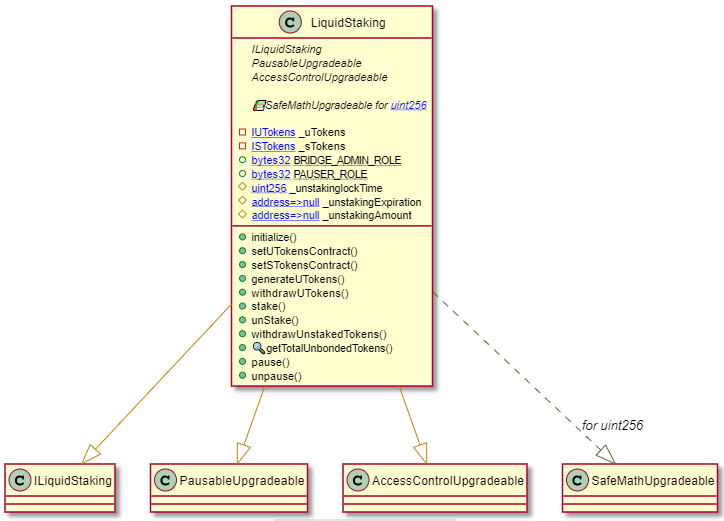
* \_sTokens: Stores the address for the STokens Contract.
* \_uTokens: Stores the address for the UTokens Contract
* \_unstakingLockTime: Stores the time period for unstaking requests. (Default = 21 days)
* \_unstakingExpiration: Mapping to handle the unstaking lock period.
* \_unstakingAmount: Mapping to handle the unstaking amount.

Functions:

* \_constructor(): Set the UToken and SToken contract addresses.
* setUTokenContractAddress(): Sets the address for the Utoken contract, only called by the owner.
* setSTokenContractAddress(): Sets the address for the Stoken contract, only called by the owner.
* generateUTokens(): Mint new UTokens, only called by the owner.
* withdrawUTokens(): Burns the UTokens and the corresponding pegged token from other blockchain are given back to the user.
* stake(): Burns the UTokens for the amount and address supplied. Mints the Stokens for the amount and address supplied.
* unstake(): Burns the STokens for the amount and address supplied. Locks the UTokens with \_unstakingExpiration and \_unstakingAmount.
* withdrawUnstakedTokens(): Mints the UTokens where the lock period has expired for the user as per the \_unstakingExpiration and \_unstakingAmount.
* getTotalUnbondedTokens(): Return the total tokens where the locking period has expired as per \_unstakingExpiration.

Interface

* Only mint, burn and balanceOf functions are used from Utokens Contract which are added with this contract.
* Only mint, burn and balanceOf functions are used from Stokens Contract which are added with this contract.



## PStake Bridge System Overview Phase 1

### Introduction

pBridge, is a proprietary bridge mechanism connecting Ethereum and Cosmos, enabling seamless transfer of value pertaining to pSTAKE application. This bridge runs in the background, continuously listening to the events relevant to pSTAKE, and executes routines to programmatically create and send transactions at both chains.

### Seamless Transfer of Value across Blockchains

pBridge makes sure that transactions originating at pSTAKE application end, pertaining to staking, unstaking, redeeming rewards, etc. are instantaneously executed at the protocol end of cosmos, namely, MsgSend, Delegate, Undelegate, Withdraw Delegation Reward etc. This enables seamless execution of PoS protocol transactions of cosmos, remote controlled from Ethereum smart contracts. The bridge also performs creation and consumption of 1:1 pegged representative tokens of ATOM, namely ustkATOM. ATOM transactions to pSTAKE cosmos wallet address triggers automatic minting of the corresponding ustkATOM to the designated Ethereum address of the user. Conversely, a withdrawal request triggers the burning of ustkATOM and ATOMs transferred to the user’s address from the pSTAKE Cosmos Wallet.

### Transaction Times

The inter-chain transactions times are fairly fast, but it strictly follows the blockchain principles. Which means that only after the transaction gets confirmed at one chain, does it progress to the second chain. Hence the transaction time will be roughly equal to the sum total of average transaction confirmation times for Ethereum and Cosmos.

### Decentralization and Fault Tolerance

The bridge mechanism is also quasi-decentralized for now, in the context that it will be run by multiple validators. This ensures downtime fault tolerance, with the routine robust enough to handle downtime of a third of the nodes, thereby providing near absolute uptime. This multi validator approach also contains a secret sharing mechanism for creating and sending transactions to both chains. This involves a secret being divided into multiple shares, with participants in the secret sharing scheme needing a majority of the parts to be able to reconstruct the secret, and in extension, a blockchain transaction. (*Currently the multi-validator is not implemented and not included in phase 1*)

### Tech-Spec

#### Software Requirements

Running the bridge would require the following prerequisite components to be setup:

#### Ethereum Blockchain Client

Access to a dedicated Ethereum client is a requirement to maintain high bandwidth availability of blockchain data. A locally running Geth node is recommended. The endpoint parameters need to be updated in the config file of the bridge.

#### Cosmos Blockchain Client

Access to a dedicated Cosmos client is a requirement as well, to maintain high bandwidth availability of blockchain data. A locally running Gaia client is recommended. The endpoint parameters need to be updated in the config file of the bridge.

#### Mongo DB

The bridge collects the relevant events and stores in the db, from where the messages are batched into a single cosmos transaction per block creation. A local implementation of MongoDB is recommended.

#### Hardware Requirements

Minimal

* 1 GB RAM
* 500 GB HDD
* 1.4 GHz CPU

Recommended

* 2 GB RAM
* 1 TB HDD
* 2.0 GHz x2 CPU

Operating System

* Linux/Windows/MacOS(x86)
* Recommended
* Linux(x86\_64)