Lido for Polkadot/Kusama technical spec

Abstract

The general idea is to create stDOT and stKSM for Polkadot and Kusama accordingly, representing liquidly staked versions of DOTs and KSMs. The Proposed mechanism will allow users to stake their tokens and get continued rewards and being liquid at the same time, so users won't need to wait such a long unstaking period (28 days for Polkadot and 7 days for Kusama). A set of node operators and other parameters such as fee will be controlled by Lido governance(LDO holders) on Ethereum blockchain, but the initial version will manage via multisig of Lido stakeholders on a particular chain(Polkadot for stDOT and Kusama for stKSM) when cross-chain bridges to Ethereum become available on target chain the system will be upgraded to the governance model.

Since Polkadot and Kusama are the multichain system itself(https://wiki.polkadot.network/), the appropriate RelayChains have contained only low-level functionality and doesn't provide the ability to host custom user's code such as Smart Contract our system will be backed by particular parachain and use XCM (cross-consensus messaging protocol, https://github.com/paritytech/xcm-format) to implement trustless and secure staking scheme across the RelayChain and Parachain.

Below in this document, we will consider the Kusama-based staking scheme, because Kusama has already launched parachains support and it's ready to host application layer projects. Since Kusama is a sister chain of Polkadot and has extremely close technical capabilities and interfaces our scheme will be compatible with Polkadot when it will release parachains support.

Glossary

Polkadot - Large blockchain that implements sharding through parachains and shared security provided by relaychain. (More details: https://polkadot.network/)

Kusama - Sisterchain of Polkadot. Both blockchains have the same architecture, but Kusama has some differences in configuration and also gets new features first. (More details: https://kusama.network/)

Relaychain - The chain that coordinates consensus and communication between parachains (and external chains, via bridges).

Parachain - A blockchain that meets several characteristics that allow it to work within the confines of the Polkadot Host. Also known as "parallelized chain."

Staking - The act of bonding tokens (for Polkadot, DOT) by putting them up as "collateral" for a chance to produce a valid block (and thus obtain a block reward). Validators and nominators stake their DOT in order to secure the network.

Stash account - This account holds funds bonded for staking, but delegates some functions to a Controller. As a result, you may actively participate with a Stash key kept in a cold wallet, meaning it stays offline all the time.

XCM - Polkadot Cross-Consensus Messaging protocol.

HRMP - Horizontal Relay-routed Message Passing, also known as HRMP, is a precursor to the complete XCMP implementation, that mimics the same interface and semantics of XCMP. It is similar to XCMP except for how it stores all messages in the Relay Chain storage, therefore making it more expensive and demanding more resources than XCMP. The plan is to retire HRMP once the implementation of XCMP is complete.

Specification

High-Level overview

High-level components scheme

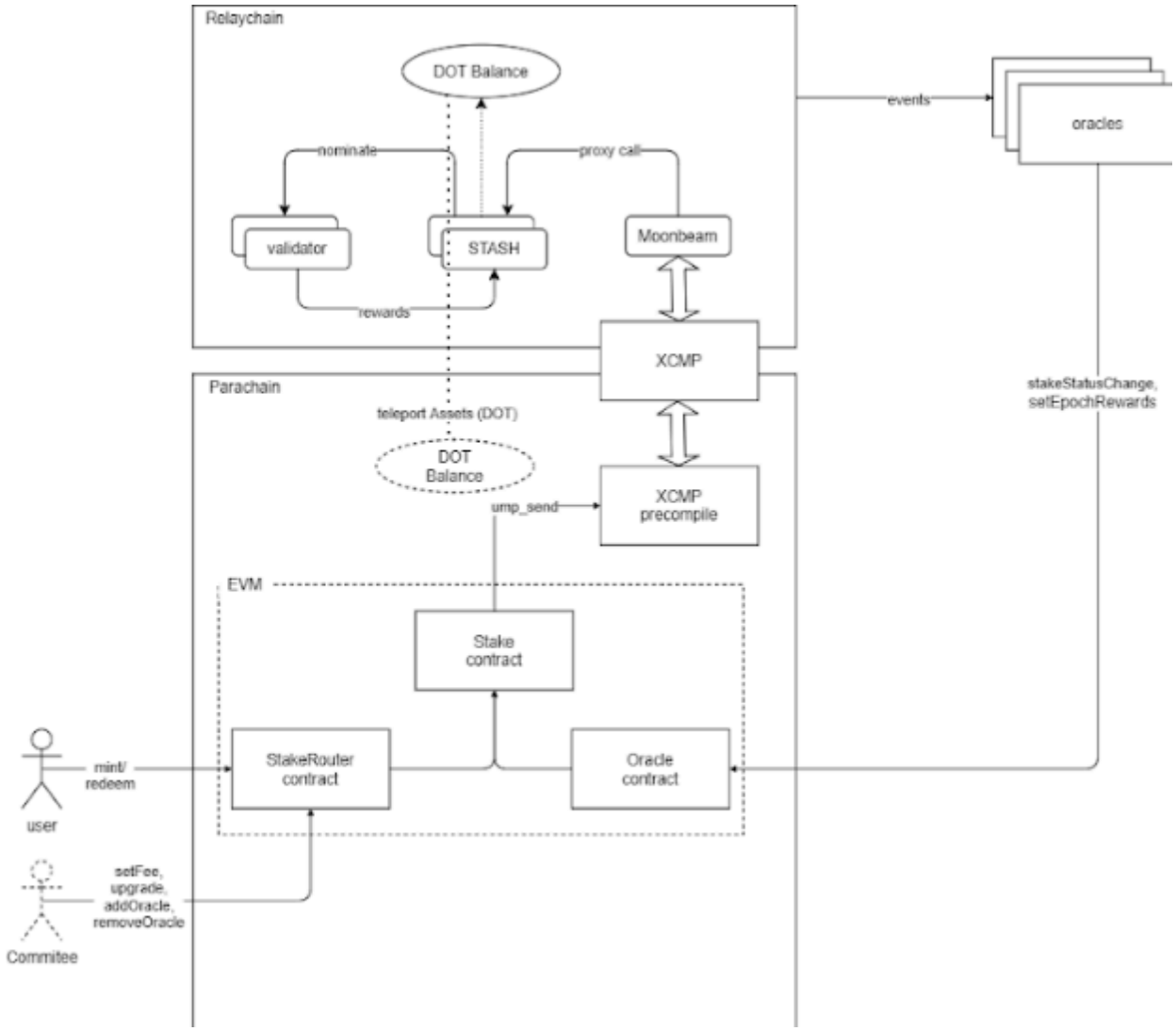

Fig 1. High-level components scheme

Staking through parachain

Polkadot and Kusama blockchains use NPoS consensus algorithm that allows staking for validators, however, staking functionality is available only on relaychain. Relaychain contains only low-level functions such as consensus, staking, native token balances, and that list doesn't include functionality to support custom code that is required for our needs.

In order to described the above issue we need to consider other solutions that fit the whole needs:

- Ability to introduce custom code that handles staking pool logic
- Ability to transfer native tokens in a fully on-chain manner(without any custodial wallets due to known security reasons)
- Ability to manage staking functions from custom code

Here we come to a parachains based solution. Polkadot provides a cross-chain messaging protocol that allows us to send messages across different parachains and relaychain. Using that primitive we theoretically can fit all our needs.

Parachain capabilities

Runtime layer

Each parachain can contain any custom code, so we can introduce our staking logic here. In terms of substrate framework, the code that represents business logic is a “runtime” code. Runtime is a set of pallets - logically isolated pieces of code, e.g code that implements balances and transfers is “balances” pallet, code that handles staking - “staking” pallet. On the runtime layer we get access to the XCM protocol, so here is our first place to inject our code.

Smart contracts

The smart contract is an extremely convenient way to host custom code on the blockchain. The key difference from runtime layer code is the ability to deploy contracts by any user, runtime code cannot be changed by ordinary users, upgrading procedures usually require some governance voting.

Our staking solution ideally should be controlled by Lido DAO, so if we put all logic to the runtime layer we can lose that feature. Some parachains in the Polkadot ecosystem include smart contracts engines, e.g Moonbeam, Acala has EVM, some others also have !ink smart contracts. Using smart contracts as a part of our system we can reach upgradability and customizability in the usual manner like in the Ethereum blockchain.

stKSM / stDOT tokenomics

Overview

stDOT / stKSM is a token that is pegged 1:1 to its native token(DOT, KSM). Basically, users can mint 1 by 1 stKSMs by staking their own KSMs through our system. After minting any holder of stKSMs will get rewards automatically by increasing the stKSMs balance in a rebasing scheme similar to Lido. Users also can unstake(withdraw) their KSM tokens back in two ways:

- By swapping stKSMs to KSMs on AMMs (see how stETH do this)
- By fair unstaking using our protocol, but in this method, the user will have to wait an unstaking period(28 days for DOTs and 7 days for KSMs)

Users should be able to use their stDOT / stKSM in DeFi to get additional APY.

Staking

System aggregates stake and unstake requests during some period “P”. After the period “P” system calculates target staking state for relaychain and sends appropriate transactions to replicate local parachain’s state to relaychain. Under the hood, each replication distributes pooled tokens across several stashes, distributes rewards across users, and schedules required unstaking requests.

Unstaking

Unstaking logic contains two main preferences:

- The system has an independent “unstaking queue” on the smart contract side for pooled tokens; that queue has the same retention period as a native queue.
- The system has an internal target “unstaking” tokens amount that is used to build unstaking transactions per each replication cycle.

So the system always moves relaychain side stashes to target staking state stored on smart contract based on actual data.

Reward distribution

While each replication cycle system calculates rewards amount based on stashes, stake changes and distribute it across all users via rebasing.

Slashing

Slashing as well as rewards are calculated while each replication cycle, each slashing will trigger token rebasing that distribute losses proportionally across stKSM / stDOT holders. Since the system will hold a stake for different validators we expect that temporary slashing losses of a particular validator will be compensated by other ones and it will lead to losses minimization.

Rebasing token

stKSM / stDOT are rebasing tokens that mechanics is used to keeping a 1:1 ratio while minting and ability to distribute reward with O(1) complexity. In two words, the contract tracks only the user’s KSM/DOT shares “S” relatively to the total staked amount “T_staked” and total shares “T_shares” and calculates the user’s balance “B” on the fly using the reverse formula: B = S / T_shares * T_staked.

That approach allows to reach O(1) complexity for reward distribution, but in the general case has an obvious disadvantage: reward amount doesn’t depend on staking duration(Alice who staked a year ago will get the same reward as Bob staked an hour ago). However, since we distribute rewards for each replication cycle and assume that staking request per during one replication period is negligibly smaller than the total staked amount we avoid that disadvantage.

Governance

Multisig

We plan to use multisig of Lido stakeholders during the first iteration of launching. The reason for multisig usage is the non-readiness of Ethereum - Parachain bridges which are required as part of the multi-chain governance system. However, we expect working bridges very soon.

Lido DAO

The system will be controlled by Lido DAO hosted on Ethereum blockchain through a multi-chain adapter for our system. The multi-chain will include Ethereum <-> Parachain bridge as well as smart contracts on Ethereum and Parachain sides. So LDO token holders will control all parameters of the system and replace a multisig based configuration scheme.

Stake management

Overview

Nomination

The system uses several stashes(see fig. 2) on the relaychain side for holding staked tokens and nominations. Stashes are fully controlled by the smart contract on the parachain side, so stash doesn't have any private key for direct control and can be managed only programmatically, that reached by using a protocol based on a combination of XCM's sovereign accounts, derivative accounts, and proxy accounts.

Requirement of several stashes based on the stash limitation: single stash can nominate up to N validators so when our system wants to host more than N validators we have to use a scheme with multiple stashes.

The list of validators for nomination will be controlled by Lido governance, so it will continuously be filtered and updated to maximize stability and APY.

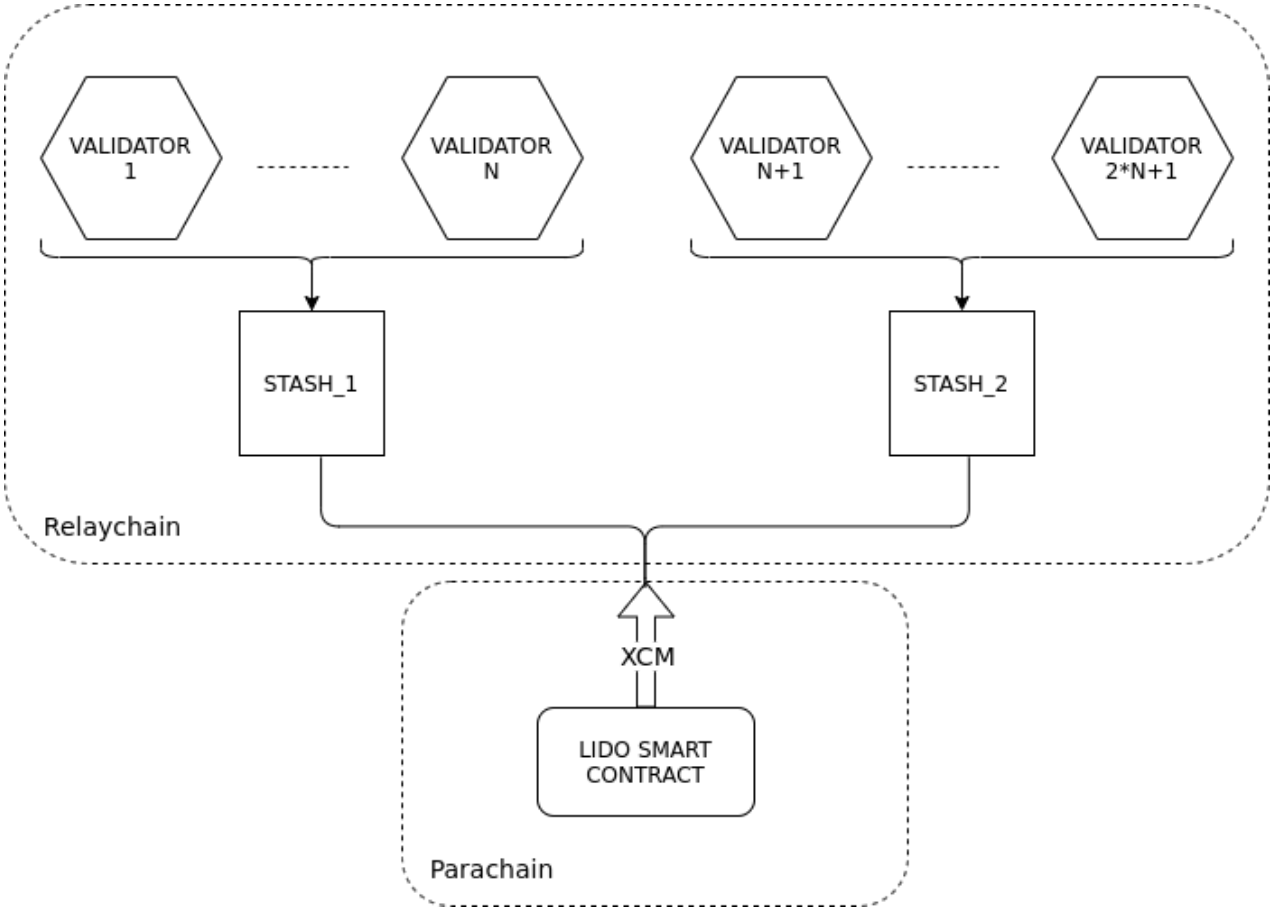

Fig 2. Lido stashes scheme

Fees flow

Node operators(validators) take fees using the standard NPoS mechanism: they can set the percent that they want to take as fees from the validation reward. The remaining reward will be distributed across stKSM / stDOT holders and as a Lido service fee.

Cross-chain XCM staking

Overview

Native token bridge

To be able to operate with the native token(KSM / DOT) on the parachain side we need to have some token bridge that allows users to move their DOT/KSM from relaychain to parachain and back.

Polkadot and Kusama have embedded token teleporting mechanics(https://github.com/paritytech/xcm-format), but unfortunately, that feature is allowed only for “common good” parachains. So we need to go through another way.

Our scheme is based on the XCM::ReserveTransferAsset message (see fig. 2), so that scheme allows holding user’s funds on the parachain’s sovereign account and sends appropriate XCM notification to parachain when a user makes a deposit, so we handles this message on parachain and mint same amount of vKSMs or vDOTs inside parachain. The withdrawal scheme uses XCM::Transact to transfer back user’s funds from the sovereign account to a user’s relaychain account and just burns withdrawn amounts of vKSMs / vDOTs.

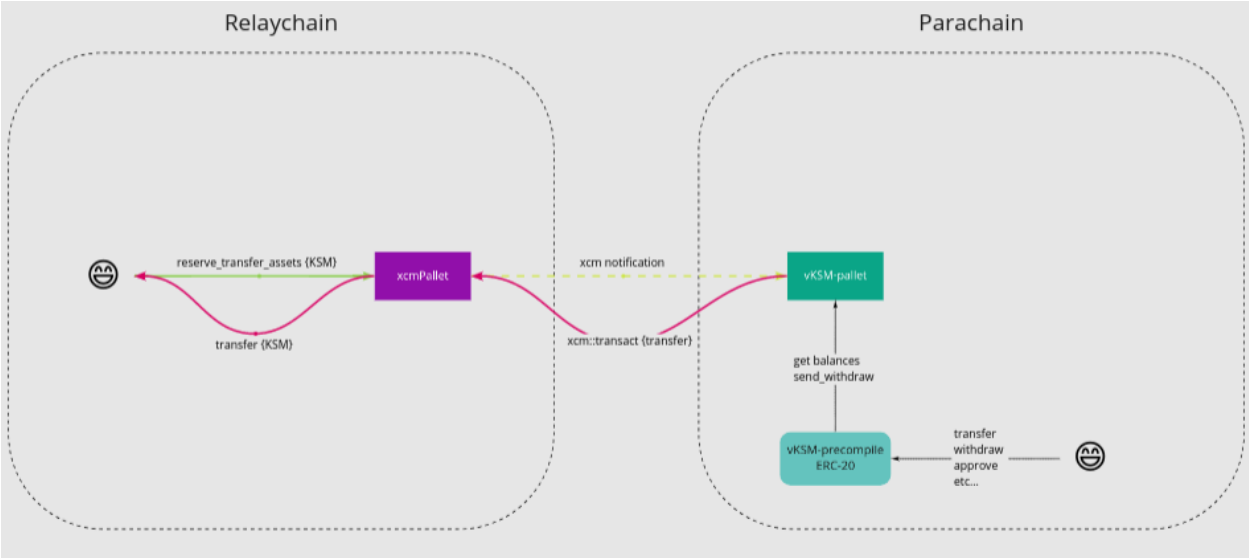

Fig 3. Native token bridge scheme

Cross-chain linked sovereign accounts

The system requires an ability to control stash accounts on relaychain from parachain based smart contract. To provide this functionality we introduced the model of relaychain <-> parachain account linking. This scheme is built on top of XCM::Transact messages and anonymous proxy accounts(https://wiki.polkadot.network/docs/learn-proxies). In detail, on the parachain side, we introduce a custom pallet that allows creating anonymous proxy by request and link that proxy with local parachain’s accounts, then we allow to send arbitrary calls to relaychain on behalf of created proxy by crafting message like: XCM::Transact { proxy::proxy{ “real”: <PROXY_ACCOUNT_ID>, “call”: <SCALE_ENCODED_RAW_CALL>} } (see fig. 4)

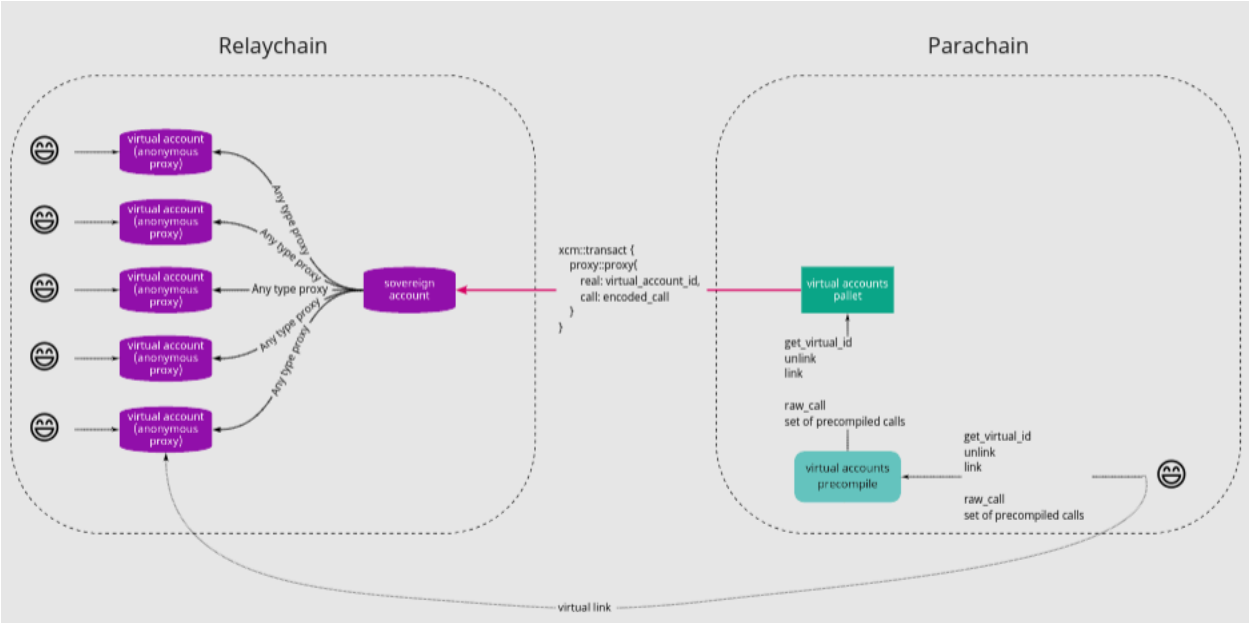

Fig 4. Relaychain account linking scheme

Integration with EVM layer

To be maximally flexible main application logic will be implemented on the EVM layer as a set of smart contracts. That requires access to native token bridge and relay chain linked accounts from the EVM layer, to meet these requirements we'll wrap that functionality to the EVM precompiles.