Indigo: Synthetic Assets on Cardano

Indigo Labs
info@indigo-labs.io

November 2021, Draft v0.2

Abstract

We are in the midst of a global switch of the essential financial train tracks that we use as a global society to trade and transact at the retail and wholesale level, aided by blockchain technology evolution. In this paper, we showcase Indigo Protocol to assist in that transition for the people. For a majority of the world's billions of people, fair access to the financial tools of this world is all but inaccessible. Accessible for the accredited and wealthy, but not the person in a developing country or without particular credit.

With Indigo Protocol, our mission is to bring the world's financial and equitable assets to the blockchain allowing everyone accesses to them, in a synthetic form, to control their financial destiny. As excellent wealth distribution takes place with money flowing out of the western and developed world and into other countries, Indigo Protocol will be just one tool in decentralized blockchain that will help this long overdue, more accessible, and fair distribution happen seamlessly.

Summary

1 Introduction

This paper presents the Indigo Protocol. Apart from the eUTXO[1]-driven contract designs, other texts like user stories, key concepts, and protocol specifications are platform-agnostic. Compared to the introductory White Paper, this writing can be considered a Yellow Paper and can be used as a high-level protocol specification.

While Indigo Labs is building the protocol on Cardano, the contract designs should adapt to any UTXO blockchains that support scripting and custom user tokens. Moreover, the separation between protocol specifications and contract

designs allows us to innovate ideas on the specifications quickly. We have iteratively designed the protocol. This way, each iteration is well-scoped for the specific additions and optimization that it introduces.

This draft is our attempt to complete the first working protocol. Further innovations will be incorporated in the upcoming versions. Further optimization will be introduced with the growth of the underlying platform.

1.1 User Story

Meet Blue, an avid investor who has recently become interested in synthetic assets. Blue believes that decentralization and the blockchain are potent tools. Blue has learned about the Indigo Protocol and wants to use it instead of the highly centralized, expensive brokerage and exchange accounts.

Violet is a retail day trader located in Nigeria, and she regularly invests in West African, US, Chinese, and other global markets. She is living outside of the countries where the markets trade has a unique set of challenges.

Both Blue and Violet have cost-saving and accessibility incentives to use the Indigo Protocol. Blue is incentivized by the low cost to mint synthetic assets and the availability of decentralized exchange platforms they may use to trade or supply their tokens as liquidity providers to earn trading fees from other users. Violet is incentivized to trade synthetic assets, called iAssets, built on Indigo since she can trade all of her assets on a single platform around the clock.

1.2 Synthetic Assets

Synthetic assets are cryptocurrency derivatives that resemble traditional derivatives (tracking the price of an underlying asset) but are far more composable, accessible, and verifiable as transactions process on a public blockchain. So, for example, we can lock up some collateral to create a synthetic asset called iBTC that has the same value as Bitcoin on the Blockchain without having any real Bitcoin in the first place. This service gives users exposure to various assets without the need to own the underlying asset. These assets can be anything that has value in the real world. With its transparency, efficiency, low barriers to entry, and decentralized traits, the blockchain can smoothly deliver all these assets to anyone with access to the Internet.

Synthetic assets are highly composable in a broader DeFi ecosystem, helping create and maintain massive markets and dApps that rely on them. For example, synthetic assets of fiat like iJPY can be used in "fiat" lending protocols.

1.3 Market & Users

Synthetic assets can create massive markets thanks to the following properties:

- They can track many different types of assets, basically anything with a real-life value. Hence being able to create many markets for all these assets.

- There are no backed assets or custodians involved to introduce inefficiencies to processes.
- There is a low barrier to entry. Anyone with cryptocurrency can lock some up to mint new synthetic assets or buy and trade them on the open market.
- They are highly composable in a wider DeFi ecosystem.

Like most DeFi protocols, Indigo needs cash flows to incentivize different actors in the platform. These can be trade fees from DEXs where Indigo users can provide liquidity with their diverse token types. Indigo users can mint new synthetic assets to create new markets on different AMMs and provide liquidity to them for yield farming.

Users can also use synthetic assets directly on other protocols. For example, iBTC can be used as either collateral or loan money on lending applications. In addition, users can use iETH and other synthetics on specific platforms.

Indigo generally brings many new assets to the whole DeFi ecosystem and more low-barrier-to-entry services to its users.

1.4 How It Works

Anyone can lock up some crypto collateral on the blockchain to mint new synthetic assets that can be held or traded like any other native token. Autonomous Oracles update the real-world price of these assets to the smart contracts that use them. This process sets a market price for trading and calculates the collateral ratio to guarantee that minted synthetic assets are over-collateralized at all times.

The collateral ratio cr at any given time is the ratio of the collateral value over the minted amount value.

$$cr = \frac{p_c * a_c}{p_m * a_m}$$

When cr drops below the minimum threshold, the collateral is liquidated to ensure solvency.

We are designing an extended version of the Stability Pools that Liquity has introduced. This new design still allows a low collateral ratio, more automatic and guaranteed liquidation events, but extra efficiency thanks to Cardano's low transaction cost. We also have to adapt to our protocol's many more token types, but the underlying idea is still there. Users can provide stability by depositing iAsset tokens, which will be used to pay the debt of under-collateralized CDPs in exchange for a share of the claimed collaterals and INDY.

With the minted assets, users can trade to go short, deposit into liquidity pools for yield farming and stay long, or deposit into the stability pool to gain a net profit on liquidation events of other debt positions.

All procedures are transparent, decentralized, and tracked by smart contracts. No one is powerful enough to modify the protocol at will. Like enabling

a new asset to mint, all changes must go through democratic on-chain voting processes.

1.4.1 Going Short

Blue thinks that the price of Bitcoin is overvalued and is going to drop in price. He currently has USD Stablecoin that he collateralizes to mint some iBTC. Blue immediately trades away his iBTC. Once the iBTC price drops, Blue trades back the minted iBTC at a lower price, returns it to get back his collateral at a net profit.

1.4.2 Going Long

Violet thinks that Ethereum's price is undervalued and is going to rise in price. She trades some USD Stablecoin tokens for iETH then deposits them into a liquidity pool for yield farming. Once the iETH price rises enough and Violet wants to cash out, she withdraws her iETH with yield from the liquidity pool, trades back more USD Stablecoin for a net profit.

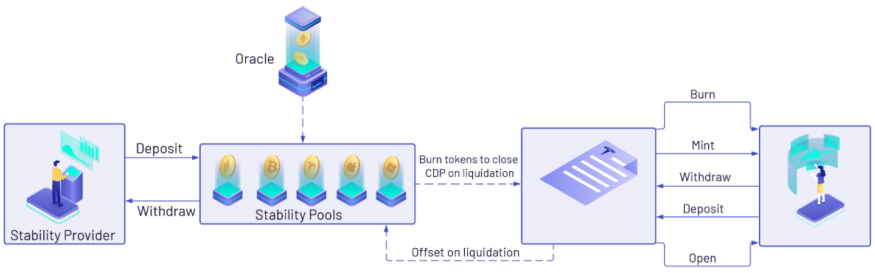
1.4.3 Stability Pool Redemption

Blue thinks that iBTC will rise in price and observes many under-collateralized debt positions that minted it. He trades some USD Stablecoin for iBTC then deposits them into the stability pool. For each liquidation event, Blue's iBTC is burned to offset the loan, for him to get back a lot more stablecoin than his initial investment, thanks to both the price rise and the instant net profit on the collateral.

1.5 Protocol Overview

The below diagrams show different functionality of the application and how they interact with different users of the Indigo Protocol.

1.5.1 Minting & Stability Functionality



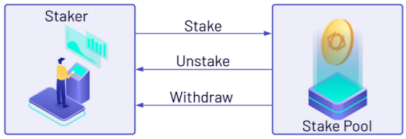
Minting and Stability Pools are key features of the Indigo Protocol. Minting allows any user to open a Collateralized Debt Position in return for an iAsset.

Stability Pools are then used to peg the price of the iAsset to the real-world value.

1.5.2 Providing Liquidity to DEXs

By providing both an iAsset and stablecoin, you can pool your token to provide liquidity to the protocol and participate as a liquidity provider.

1.5.3 Staking INDY

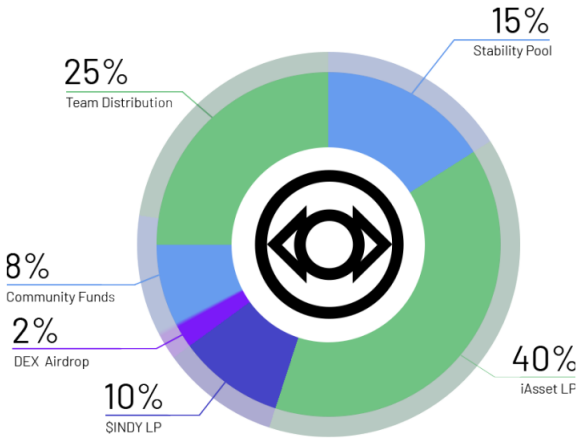


The native asset of the Indigo Protocol, the INDY token, can be staked to vote on active polls and is required as a deposit for making new governance suggestions.

1.6 Tokenomics

INDY is the protocol's governance token. It is mainly used for governance but also used as a reward for stakers in the platform. The total supply of INDY is 35M tokens with a 6 decimal precision. We believe in a fair distribution of the INDY tokens, therefore there will be no pre-sale and private distribution to investors prior to launch.

The INDY tokens have already been minted, all at once, to be distributed to the community as shown below. And by a monetary policy that disallows future minting and burning.



1.6.1 INDY Token Distribution

Percent-based Distribution Schedule						
	Genesis	Y1	Y2	Y3	Y4	Total
Stability Pool	7.5%	3.75%	1.875%	1.875%	0.00%	15%
iAsset LP	0.00%	15.00%	12.00%	8.00%	5.00%	40%
INDY LP	0.00%	3.00%	3.00%	3.00%	1.00%	10%
DEX Airdrop via. Governance Vote	0.00%	2.00%	0.00%	0.00%	0.00%	2%
Community Funds & DAO Treasury	5.00%	0.00%	1.00%	1.00%	1.00%	8%
Team Distribution	7.00%	7.00%	6.00%	5.00%	0.00%	25%
Total	20%	31%	24%	19%	7%	100%

- *Stability Pools*: We reward INDY to stability providers, who deposit iAssets in preparation for guaranteed collateral liquidation. This is critical for both system solvency, user experience, and security against liquidity risks.
- *iAsset LP*: We reward INDY to iAsset LP stakers for their commitment to the iAssets LPs, essential to the whole ecosystem's liquidity. Without them, it's not easy to do useful things with the minted iAssets.
- *INDY LP*: We reward INDY to INDY stakers, who stake to participate in the governance processes of the protocol. We need such incentive for more people to hold INDY, which in turn balances our decentralized system. This prevents having a few whales who run the system.
- *DEX Airdrop*: We need a DEX to expose the iAssets further and create a revenue stream from yield farming for different agents in the protocol. Gifting INDY to DEX users isn't just a nice gesture to be part of a bigger ecosystem; it drives our collaboration with the DEX. The plan is to choose

the concrete DEX after a certain amount of time (1 year after genesis) through democratic governance voting.

- *Community Funds*: This is a small pool of funds that can be spent through a governance proposal to fund future contributors and partners of the protocol mainly.
- *Team Distribution*: This is the reward for the initial development team to develop and maintain the protocol until stability is absolute.

1.7 INDY Token Staking

The INDY token can be staked to be used in the governance processes. INDY Token holders who have staked their position can vote on polls. Voting power is then weighted by the total amount of staked INDY each holder has. Therefore, users with a larger amount of INDY tokens will have more influence on voting. We have done careful tokenomics distribution to prevent whales in the ecosystem who control it. No presale, with live rewards distributed fairly among different protocol agents.

INDY stakers will be rewarded with more INDY. This doesn't just empower dedicated participants, it incentivizes people who lock it down for governance, for the growth of the whole protocol.

1.8 iAsset Tokenomics

iAsset tokens are over-collateralized synthetic tokens that are minted and burnt through the Mint contract. iAsset tokens give users price exposure to real assets, as well as allowing fractional ownership of the underlying asset, with decimal precision of 6. iAssets cannot be created without collateral, which allows the Indigo Protocol to efficiently peg the price of the token through our state-of-the-art liquidation model.

1.8.1 Minting Policy

iAsset tokens can only be minted and burned against a CDP through the CDP smart contract. Therefore, their validation rules are tied to the Mint endpoints being called.

1.9 Governance

The Indigo protocol is fully decentralized in that no one is powerful enough to change it at will. Instead, all changes must go through a democratic governance process. INDY holders can deposit a small sum of Proposal Deposit to open new proposals for other INDY stakers to vote on.

Initially, INDY holders can start the following proposals:

- White list a new synthetic asset type to make it mintable.

- Delist an asset in the event of unstable market conditions, making it no longer mintable.
- White list a new LPToken type to receive INDY reward by staking it.
- Update protocol parameters, like the minimal amount of INDY tokens required to open a new proposal.
- Spend the community fund for new development, Indigo Improvement Proposals, and Bug Bounties.

After a Voting Period, if the amount of staked INDY participants and the ratio of yes over total votes pass both minimum requirements, the proposal passes. However, the protocol will wait for an Effective Delay before making the change to ensure a smooth transition. For example, upon whitelisting a new iAsset, its oracle might need some time to stabilize.

1.9.1 Protocol Parameters

These are the protocol parameters that can be updated through governance.

Name	Type	Description
Minimum Collateral Ratio	Per iAsset	Each asset might have a very different underlying market, volatility property, and stability pool volume hence needing a different minimum collateral ratio.
Quorum	Protocol	The minimum percentage of stake required to pass a proposal.
Threshold	Protocol	The minimum percentage of yes votes required to pass a proposal.
Voting Period	Protocol	The period (in blocks) in which stakers can vote on a proposal.
Effective Delay	Protocol	The period (in blocks) the system waits before executing a passed proposal.
Expiration Period	Protocol	The period (in blocks) that a proposal has to be executed before being invalidated.
Proposal Deposit	Protocol	The number of INDY tokens need to be deposited to open a new proposal.

1.10 Risk Management

We constantly audit the platform and market conditions to inform users with a low collateral ratio that might get liquidated to deposit more collateral. We also provide intuitive tools for the users to monitor and read the market themselves.

We also constantly attack the protocol on both the testnet and mainnet to find vulnerabilities to patch or warn users to avoid risky scenarios if there are any.

For malicious and under-performing risks of oracles and such, we have quick governance solutions to switch oracles or punish bad actors in the platform. This includes falling back to our own Indigo Labs infrastructure, given enough staked votes from the community. We've designed secure incentive formulas for agents to benefit only if they play by the rules, and lose if they don't.

2 Protocol Specifications

2.1 CDP

2.1.1 User Requirements

- ***R1***: Users can open a CDP.

  A user can open a collateral debt position (CDP) by locking ADA into a smart contract and minting an iAsset (iXAU, iOIL, iXAG, etc). The minted iAsset will be sent to the user's wallet or any wallet the user wants. It is not checked in the validator.

  When opening a CDP, the user must associate that CDP with a kind of iAsset in the *iAssets Whitelist* of the protocol. The user can only interact with the iAsset associated with the CDP. The user opening a CDP is the owner of that CDP.

- ***R2***: Users can interact with their own CDP.

  A user can interact with his/her CDP in one of the following ways:

  - Deposit more ADA collateral.
  - Withdraw ADA collateral from the CDP. The ADA will be sent to the user's wallet or any wallet the user wants. It is not checked in the validator.
  - Mint iTSLA. The minted iTSLA will be sent to the user's wallet or any wallet the user wants. It is not checked in the validator.
  - Burn iTSLA from the user's wallet.
  - Close their CDP by burning all minted iAssets and getting back all the ADA collateral. The collateral of closed CDP will be sent to any address the owner wants (The on-chain logic doesn't check where this collateral goes).

- ***R3***: Users can liquidate an *undercollateralized* CDP by burning an equivalent amount of minted iAsset and transferring the collateral to the Stability Pool.

  A CDP is *undercollateralized* if its CR is below the *Minimal Collateral Ratio* (refer to **R6**).

  In a liquidation event, the *Stability Providers* will lose a pro-rata share of their iAssets deposit and gain a pro-rata share of the collateral (ADA).

Let's have an example: A CDP with 1,000 ADA and 500 iXAU is being liquidated. This table contains the deposited balance and share of all iXAU *Stability Provider* before the liquidation events.

	iXAU deposit balance	iXAU share
User A	1000	20%
User B	1500	30%
User C	2500	50%

This table contains the iXAU balance lost, ADA reward gained from the liquidation event, and the final iXAU balance after the liquidation of all iXAU *Stability Provider*.

	iXAU lost	ADA rewarded	Final iXAU balance
User A	500 x 20% = 100	1,000 x 20% = 200	1000 - 100 = 900
User A	500 x 30% = 150	1,000 x 30% = 300	1500 - 150 = 1350
User A	500 x 50% = 250	1,000 x 50% = 500	2500 - 250 = 2250

2.1.2 Protocol Requirements

- ***R4***: A CDP can only be interacted with by its owner (***R1***).
- ***R5***: One user can open multiple CDPs (***R1***).
- ***R6***: All CDPs must maintain the *Minimal Collateral Ratio* at all times. We define the collateral ratio (cr) of a CDP as:

$$cr = \frac{p_c * a_c}{p_m * a_m}$$

  where:

  - p_c is the price of collateral asset.
  - p_m is the price of the minted asset.
  - a_m is the total minted amount of this CDP.
  - a_c is the locked collateral amount of this CDP.

- ***R7***: All CDPs must maintain a positive minted amount at all times.
- ***R8***: An iAsset can only be minted/burned from the CDPs associated with that iAsset.
- ***R9***: The prices of collateral (ADA) and minted iAsset will be queried from a trusted Oracle.
- ***R10***: A CDP can only be liquidated if the Stability Pool contains enough amount of iAsset to cover all the debt generated by the position.

2.2 Stability Pool

2.2.1 User Requirements

- ***R1***: Users can deposit their iAssets to the Stability Pool.
- ***R2***: Users can withdraw their iAssets from the Stability Pool.
- ***R3***: Users can withdraw their rewards (collateral ADA transferred from liquidated CDPs) from the Stability Pool.

2.2.2 Protocol Requirements

- ***R4***: Stability Pool asset must be in iAsset white list.

2.3 Staking

2.3.1 User Requirements

- ***R1***: Users can stake INDY to gain voting ability.
- ***R2***: Users can withdraw their staked INDY tokens.

2.3.2 Protocol Requirements

- ***R3***: User can not withdraw their locked INDY tokens (refer to **Governance - R2**).

2.4 Governance

2.4.1 User Requirements

- ***R1***: Users can create a proposal by depositing Proposal Deposit INDY tokens. There are two kinds of proposals that users can create:
  - Change the *Governance* parameters (*Proposal Deposit, Quorum, Threshold, Voting Period, Effective Delay, Expiration Period*)
  - Add a new iAsset to the *iAsset Whitelist*.
- ***R2***: Users can vote for a proposal during its *Voting Period*. To vote for a proposal, users need to specify the vote option (*yes, no, or abstain*) and vote amounts. The vote amounts cannot exceed their staked INDY amount. A user can only vote for a proposal once, and during the *Voting Period*, their staked *INDY* used for voting will be locked.
- ***R3***: Users can end a proposal after its *Voting Period* to finalize its outcome. The outcome of a proposal can be either *Passed* or *Failed*. If the proposal is *Failed*, the proposal must be removed.
- ***R4***: Users can execute a *Passed* proposal after its *Effective Delay* and before its *Expiration Period* to apply the change to the system.

2.4.2 Protocol Requirements

- ***R5***: Determining the outcome of a proposal after its *Voting Period* (correspond to end action in R3)A proposal is *passed* if it satisfied both conditions below:
  - $\frac{total\_votes}{total\_INDY\_staked} \geq quorum$
  - $\frac{total\_yes\_votes}{total\_votes} \geq threshold$

  Otherwise, the proposal is *failed*.
- ***R6***: Determining where the *Proposal Deposit* goes (correspond to end action in R3).When we end a proposal after its voting period, if the proposal satisfied the condition R5-a, we will return the *Proposal Deposit* to the creator. Otherwise, we transfer the *Proposal Deposit* to the Treasury Fund.
- ***R7***: Changing protocol parameters
  - ***If Voting Period or Proposal Deposit is changed, it will not affect the ongoing proposals***.

    When creating a proposal, the end slot of the proposal is calculated from the *current slot* and the *voting period* of the protocol.
  - ***If Quorum or Threshold is changed, it will affect proposals in Voting Period, won't be applied to passed/failed proposals***.

    When ending a proposal, check if and *quorum*, the *threshold* is passed (and some other conditions) then set proposal status to *Passed* or *Failed*.
  - ***If Effective Delay or Expiration Period is changed, it will be applied to all ongoing proposals***.

    When executing a proposal, check if the proposal's status is Passed and *effective delay* and *expiration period* are passed, then kill the proposal & execute its action.

2.5 Vesting

Users (Team members) can withdraw 8.75 millions INDY tokens corresponding to the Percent-based Distribution Schedule below:

	Genesis	Y1	Y2	Y3	Y4	Total
Team Distribution	7.00%	7.00%	6.00%	5.00%	0.00%	25%

2.6 Liquidity Pool

2.6.1 User Requirements

- ***R1***: Users can earn INDY tokens by staking INDY/iAsset LP Tokens in the *LP Tokens Whitelist* of the protocol in the Liquidity Pool contract.
- ***R2***: Users can add a new kind of INDY/iAsset LP Token to the *LP Tokens Whitelist* via *Governance*.
- ***R3***: INDY/iAsset LPTokens are earned by adding liquidity to the appropriate INDY/iAsset-[Other tokens] pools. Each pool has a unique LP Token associated with it.
- ***R4***: Reward distribution schedule for liquidity pool as below:

	Genesis	Y1	Y2	Y3	Y4	Total
iAsset LP	0.00%	15.00%	12.00%	8.00%	5.00%	40%

Note: Percent on total supply 35M INDY.

2.6.2 Protocol Requirements

Reward Distribution Mechanism

- ***R5***: During each period mentioned above, INDY tokens are distributed uniformly throughout every second.
- ***R6***: Let *#indy* be the amount of INDY tokens distributed in the second t. The amount of INDY tokens user x gained in the second t is:

$$\#indy \times \frac{total\_weighted\_LPTokens_x}{total\_weighted\_LPTokens}$$

  where:

  - $total\_weighted\_LPTokens_x$ is the sum of *weights* all LP Tokens staked by user x.
  - $total\_weighted\_LPTokens$ is the sum of *weights* all LP Tokens staked in the Liquidity Pool contract.

- ***R7***: The weight of a kind of LP Token is determined as follow:
  - INDY-[Other tokens] LP Token's weight = 300.
  - *iAsset*-[Other tokens] LP Token's weight = 100 if *iAsset* is added into *iAsset Whitelist* in the first year.
  - *iAsset*-[Other tokens] LP Token's weight = 30 if *iAsset* is added into *iAsset Whitelist* after the first year.
- ***R8***: Each time a user stake/un-stake some LP Tokens, they will receive the reward unlocked from the last time they stake/un-stake some LP Tokens.

3 Smart Contract Design

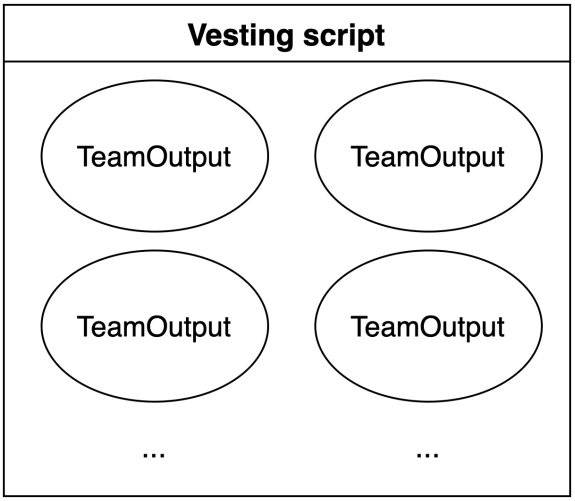
We choose the Cardano Blockchain for its research-first approach. We're not going to bet massive markets on an under-researched blockchain that proves to be insecure or not scalable in the long run. Cardano also provides a great functional programming toolset that improves software correctness and efficient transactions that are inexpensive and fast to execute.

3.1 Vesting

3.1.1 Native Tokens

Name	Description	Minting Policy
INDY	INDY Token has many utilities in our protocol. For this version, user can use INDY Token to open a proposal and earn the voting powers.	Minted before launch with total supply 35 million.

3.1.2 OnChain



Parameters

- `indyToken :: Value.AssetClass`. AssetClass of INDY Token.

Outputs:

Type	Description	Datum	Values
TeamVesting	Each output corresponding to one reward period.	• *validTime*: the time after that team members can withdraw reward at this output. • *members*: map of team members' public keys and their reward.	INDY (based on each output)

Validation Rule:
Consume TeamVesting (*validTime*, *members*) output with WithdrawINDY(*pkh*, *amount*) redeemer:

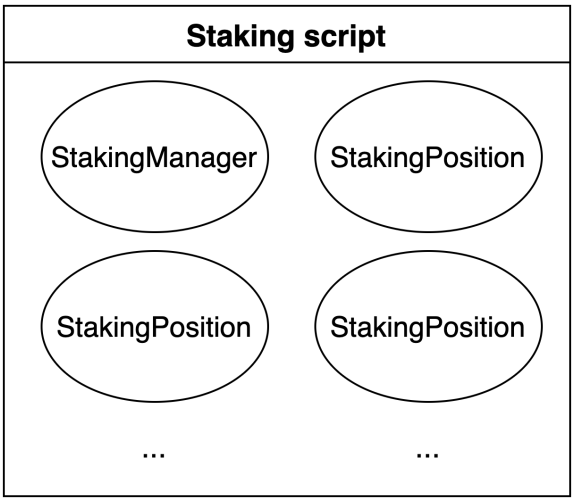
1. The transaction must be signed by *pkh*.
2. The time range of transaction must be after *validTime*.
3. The *amount* is less than or equal to user's reward.
4. The transaction produces an TeamVesting (*validTime*', *members*') output with the following condition:
   (a) *validTime*' = *validTime*
   (b) *member*'[*pkh*] = *member*[*pkh*] - *amount*
   (c) Amount INDYs locked at the output must be updated properly.

3.2 Staking

3.2.1 Native Tokens

Name	Description	Minting Policy
StakingManagerNFT	- The NFT identifies the authentic StakingManager output. - The NFT must be stored at StakingManager output. - Validator scripts ensure that this NFT always stays at the StakingManager output.	The protocol mints exactly 1 token, before launch.
StakingToken	Identify the authentic StakingPosition output.	The transaction must consume StakingManagerNFT.

3.2.2 OnChain



Parameters

- `stakingManagerNFT :: Value.AssetClass`. NFT of StakingManager.
- `stakingToken :: Value.AssetClass`. Token for identifying authentic Staking Position output.
- `proposalToken :: Value.AssetClass`. Token for identifying authentic Proposal output.
- `indyToken :: Value.AssetClass`. INDY token.

Outputs:

Type	Description	Datum	Values
StakingManager	- Only one output of this type is stored in the script. - To create a StakingPosition output, the user must consume this output in the transaction.	*totalStake*: The total amount of staked INDY *stakedPkh*: list of public key hash staked some INDY	StakingManagerNFT

Validation Rule:

1. Consume the StakingManager (*totalStake*, *stakedPkh*) output with CreateStakingPosition *owner* redeemer:
   (a) The transaction must be signed by *owner*.
   (b) $owner \notin stakedPkh$
   (c) Tx mint/burn amount = {1 StakingToken}
   (d) The transaction must produce one StakingPosition (*owner*, *lockedAmount*) output to the StakingScript. Let *#indy* be the amount of INDY stored in this output, we have the following conditions:
      i. $lockedAmount = \{\}$
      ii. Value = {*#indy* INDY, 1 StakingToken}
   (e) The transaction must produce one StakingManager (*totalStake*', *stakePkh*') output to the StakingScript:
      i. $totalStake' = totalStake + \#indy$
      ii. $stakePkh' = stakePkh \cup \{owner\}$
      iii. Value = {1 StakingManagerNFT}
2. Consume StakingManager output with UpdateTotalStake redeemer:
   (a) The transaction must consume one StakingPosition output.
3. Consume StakingManager output with View redeemer:
   (a) The transaction must produce the exact same StakingManager output to StakingScript.
   (b) Cannot mint any StakingToken.
4. Consume StakingPosition (*owner*, *lockedAmount*) output with AdjustStakedAmount *amt* redeemer:
   (a) The transaction must consume exactly one StakingPosition output.

(b) The transaction must be signed by *owner*.

(c) Tx mint/burn amount = {}

(d) Let *indy* be the amount of INDY tokens stored at this output. Let *lockedIndy* be the amount of INDY tokens locked because of voting. We have: *indy* - *amt* $\geq$ *lockedIndy*

(e) The transaction must consume StakingManager (*totalStake*, *stakedPkh*) output and produce StakingManager (*totalStake*', *stakedPkh*') output to the StakingScript:

i. $totalStake' = totalStake + amt$

ii. $stakedPkh' = stakedPkh$

iii. Value = {1 StakingManagerNFT}

(f) Then transaction must produce StakingPosition (*owner*', *lockedAmount*') output to the StakingScript:

i. $owner' = owner$

ii. $lockedAmount' = lockedAmount$

iii. Value = {(*indy* + *amt*) INDY, 1 StakingToken}

5. Consume exactly 1 StakingPosition (owner, lockedAmount) output with Unstake redeemer:

(a) The transaction must be signed by owner.

(b) $lockedAmount = \{\}$

(c) Tx mint/burn amount = {-1 StakingToken}

(d) The transaction must consume StakingManager (*totalStake*, *stakedPkh*) output and return the correct one (*totalStake*', *stakedPkh*') to the StakingScript:

i. $totalStake' = totalStake - indy$, where *indy* is the amount of INDY tokens stored at the StakingPosition output.

ii. $stakePkh' = stakePkh \setminus owner$

iii. Value = {1 StakingManagerNFT}

6. Consume exactly 1 StakingPosition (*owner*, *lockedAmount*) output with Lock redeemer:

(a) The transaction must not consume StakingManager.

(b) The transaction must be signed by *owner*.

(c) The transaction must consume a Proposal Output in its Voting Period.

7. Consume exactly 1 StakingPosition (*owner*, *lockedAmount*) output with Unlock redeemer:

(a) The transaction must not consume StakingManager.

(b) The transaction must be signed by *owner*.

(c) Must produce 1 StakingPosition (*owner*', *lockedAmount*') output to the StakingScript:

i. $owner' = owner$

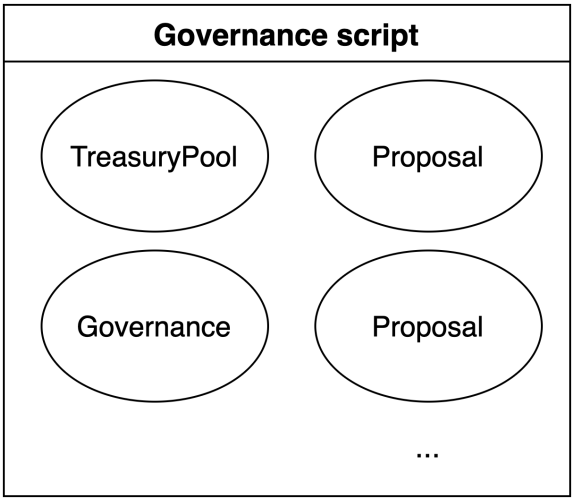
ii. $lockedAmount' = lockedAmount \backslash x \in lockedAmount \mid x.endTime < txValidRange$

3.3 Governance

3.3.1 Native Tokens

Name	Description	Minting Policy
GovNFT	- Identify authentic Governance output. - Governance Script ensures that this NFT always stays at the Governance output.	The protocol mints exactly 1 token, before launch.
TreasuryPoolNFT	- Identify the authentic TreasuryPool output. - Governance Script ensures that this NFT always stays at the TreasuryPool output.	The protocol mints exactly 1 token, before launch.
ProposalToken	- Identify the authentic proposal. - Validator scripts ensure that this token always stays at Proposal output.	The transaction must consume GovNFT.

3.3.2 OnChain



Parameters

- `governanceNFT :: Value.AssetClass`. NFT of Governance.
- `treasuryPoolNFT :: Value.AssetClass`. NFT of Treasury Pool.
- `proposalAuthToken :: Value.AssetClass`. Token identifying authentic proposals.
- `iAssetToken :: Value.AssetClass`. Token identifying authentic iAsset outputs.
- `stabilityPoolToken :: Value.AssetClass`. Token identifying authentic Stability Pool outputs.
- `lpOutputToken :: Value.AssetClass`. Token for identifying authentic LPToken outputs.
- `indyAsset :: Value.AssetClass`. INDY Token.
- `stakingParams :: StakingParams`. The parameters to interact with Staking script.
- `cdpValHash :: Ledger.ValidatorHash`. Hash of the CDP script. We need to place iAsset output there.
- `sPoolValHash :: Ledger.ValidatorHash`. Hash of the Staking script. We need to place the staking manager and staking position outputs there.
- `liquidityValHash :: Ledger.ValidatorHash`. Hash of Liquidity Pool script. We need to place the LPToken output there.

Outputs:

Type	Description	Datum	Values
Governance	- Only one output of this type is stored in the script. - To create a Proposal output, the user must consume this output in the transaction. - To store the protocol parameters.	*currentProposal*: The number of opened proposals. *iAssetList*: iAsset whitelist. *propotolParams*: The params of the protocol.	GovNFT: 1
Proposal	Each proposal output representing a proposal.	*psId*: Index of the proposal. *psOwner*: The public key hash that creates the proposal. *psContent*: The content of the proposal. More details are in *Table* 1 . *psStatus*: The status of a proposal. More details are in *Table* 1 . *psEndTime*: The slot that proposal's voting period ends, calculated from votingPeriod when creating. *psDescription*: Explain why this proposal is necessary.	ProposalToken: 1 INDY: Proposal Deposit
Treasury Pool	Store Proposal Deposit INDY tokens that are not returned to the proposals' creators.		TreasuryPoolNFT INDY: Store Proposal Deposit INDY.

Table 1: Proposal Detail

Proposal's features	Types
psContent	• ProposeAsset: Whitelist a new iAsset – *iaName*: Name of the iAsset – *iaMinRatio*: Minimum Collateral Ratio • ProposeLPToken: Whitelist a new LP-Token – *lpAsset*: Asset class of the LPToken – *lpWeight*: Reflect with iAsset, 300 if token is INDY, 100 if iAsset minted for the first year, 30 otherwise. • ModifyProtocolParam: *protocolParams*
psStatus	• InProgress – *nYes*: yes votes – *nNo*: no votes – *nAbstain*: abstain votes • Passed – *passedTime*: The slot that the proposal passed

Validation Rule:

1. Consume Governance (*currentProposal*, *iAssetList*, *protocolParams*, *lpTokenList*) output with CreateProposal *sTime* redeemer:
   (a) The transaction must produce a Governance output to the Governance script with:
      i. Datum = Governance (*currentProposal + 1*, *iAssetList*, *protocolParams*, *lpTokenList*).
      ii. Value = {1 GovNFT}
   (b) The transaction must produce a new Proposal (*id, owner, content, status, psEndTime, description*) output to the Governance script:
      i. The transaction must be signed by *owner*.
      ii. $id = currentProposal + 1$
      iii. status = InProgress 0 0 0.
      iv. $psEndTime = sTime + votingPeriod$
      v. If content is ProposeAsset , the proposed $iAsset$ must not exist.
      vi. If content is ProposeLPToken, the proposed LPToken must not exist and $lpWeight$ must correct.
      vii. Value = {Proposal Deposit INDY, 1 ProposalToken}
   (c) The time range of transaction must be included in *(sTime - biasTime, sTime + biasTime)*.
   (d) Tx mint/burn amount = {1 ProposalToken}
2. Consume Proposal *(id, owner, content, status, endSlot, description)* output with Vote *option amount* redeemer:
   (a) The proposal must actually be in its Voting Period:
      i. The *status* must be InProgress.
      ii. $txValidRange \leq endSlot$
   (b) User cannot lock negative amount to vote.
   (c) The total locked amount must be less than or equal to the staked amount.
   (d) The transaction must produce a Proposal(*id', owner', content', status', endSlot', description'*) output to the GovernanceScript:
      i. *id', owner', content', endSlot', description'* must remain the same.
      ii. The *status'* must be updated properly.
      iii. The value of the Proposal Output must stay the same.
   (e) The transaction must consume exactly one StakingPosition (*owner, lockedAmount*) output:

i. $id \notin keys(lockedAmount)$ (owner didn't vote for this proposal before)

(f) The transaction must produce exactly one StakingPosition (*owner', lockedAmount'*) output to the StakingScript:

i. Value must stay the same

ii. $owners' = owner$

iii. $lockedAmount' = lockedAmount \cup (id, amount)$

3. Consume Proposal (*id, owner, content, status, endSlot, description*) output with EndProposal *pTime* redeemer:

(a) The transaction must consume the StakingManager (*totalStake, ...*) output and return the exact same one.

(b) The transaction must consume Governance (*protocolParameters, ...*) output and return the exact same one.

(c) The time range of transaction must be included in (*pTime - biasTime, pTime + biasTime*).

(d) The proposal must not be in its Voting Period and still InProgress:

i. The *status* must be InProgress.

ii. $txValidRange > endSlot$

(e) Let $participantRatio = \frac{nYes+nNo+nAbstain}{total\_stake}$

i. $participantRatio \geq quorum$: All the INDY locked in this proposal must be sent to owner.

ii. $participantRatio < quorum$: All the INDY locked in this proposal must be sent to the TreasuryPool.

(f) Let $yesRatio = \frac{nYes}{nYes+nNo+nAbstain}$

i. $yesRatio \geq threshold$ & $participantRatio \geq quorum$:

A. The transaction must produce exactly one Proposal output (*id', owner', content', status', endSlot', description'*) to the GovScript:

B. $id = id', owner = owner', content = content', endSlot = endSlot', description = description'$.

C. *status* = Passed *pTime*.

ii. Otherwise:

A. Tx mint/burn amount = {-1 ProposalToken}

4. Consume Proposal (*id, owner, content, status, psEndTime, description*) output with Execute redeemer:

(a) The transaction must consume Governance (*protocolParameters, ...*) output.

(b) We can indeed execute this proposal:

i. The *status* must be Passed.
ii. $txValidRange \geq passedTime + effectivedelay$
iii. $txValidRange < psEndTime + expirationperiod$

(c) We execute the proposal properly, produce the correct Governance output, and mint the correct amounts of all relevant Cardano native tokens:

i. PropsalAsset *iAsset*:
A. The transaction must produce exactly one Governance (*currentProposal', iAssetList', protocolParams'*) output to the GovernanceScript:
- $currentProposal' = currentProposal, protocolParams' = protocolParams$
- $iAssetList' = iAssetList \cup iAsset$
- Value = {1 GovNFT}

B. A StabilityPool (*iAsset*, {}, {}) output must be produced to the StabilityPoolScript.
C. An iAsset (*iAsset*) output must be produced to the CDPScript.
D. Token minted/burned = {1 ProposalToken, 1 StabilityPoolToken, 1 iAssetToken}

ii. ProposeLPToken *lpToken*:
A. The transaction must change the Governance output properly (add *lpToken* to whitelist)
B. A LPToken output must be produced to the Liquidity script.
C. Token minted/burned = {1 ProposalToken, 1 LPAuthToken}

iii. ChangeProtocolParameter *protocolParams*:
A. The transaction must produce exactly one Governance (*currentProposal', iAssetList', protocolParams'*) output to the GovernanceScript:
- $currentProposal' = currentProposal, iAssetList' = iAssetList$
- $protocolParams' = protocolParams$
- Value = {1 GovNFT}

B. Token minted/burned = {-1 ProposalToken}

5. Consume Governance output with ConsumeWithProposal redeemer:

(a) The transaction must consume exactly one Proposal output (*id, owner, content, status, psEndTime, description*):

i. We can't vote for this proposal:
A. status != InProgress
B. or $txValidRange > psEndTime$

6. Consume TreasuryPool output with AddFee redeemer:

(a) Can not withdraw from pool.

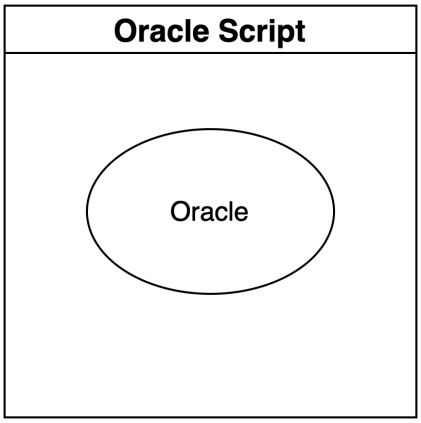
NOTE: When we consume the Proposal output with EndProposal or Execute redeemer, we also validate the outcome of the Governance output. However, we don't do that when consuming with Vote redeemer. a-i is a way to work around it. The design will be much cleaner, more intuitive, and more coherent when Plutus allows us to view the Redeemers all inputs in the transaction.

3.4 Oracle

3.4.1 Native Tokens

Name	Description	Minting Policy
OracleNFT	- Identify the authentic Oracle output. - Must be stored at Oracle output. - Validator scripts ensure that this NFT always stays at the Oracle output.	The protocol mints exactly 1 token, before launch.

3.4.2 OnChain



Parameters

- `oracleNFT :: Value.AssetClass`. NFT of Oracle.
- `owner :: Ledger.PubKeyHash`. The owner of Oracle has permission to update prices.
- `oracleFee :: OnChainDecimal`. The fee must be paid to read Oracle utxo.

Outputs:

Type	Description	Datum	Values
Oracle	- Only the sole owner has permission to update the price. - Users must consume this output to get the price of the asset. - The user must send a small read fee to Oracle's owner to consume this output.	Map of iAssets and their prices.	OracleNFT.

Validation Rule:

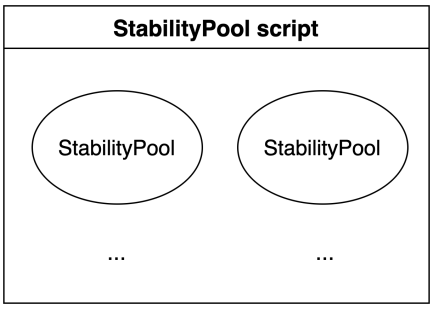
1. Consume Oracle output with FeedPrice redeemer:
   (a) The transaction must return Oracle output with its NFT to the Oracle Script.
   (b) The transaction must be signed by the Oracle owner.
   (c) The new Oracle datum must reflect the FeedPrice update properly.
   (d) The new price must be positive.
2. Consume Oracle output with Use redeemer:
   (a) The transaction must return Oracle output with its NFT to the Oracle Script.
   (b) The Oracle datum must stay the same.
   (c) The transaction must send a small read fee to Oracle's owner.

3.5 Stability Pool

3.5.1 Native Tokens

Name	Description	Minting Policy
StabilityPoolToken	Identify the authentic StabilityPool output.	The transaction must spend GovNFT.

3.5.2 OnChain



Parameters

- `stabilityPoolToken :: Value.AssetClass`. StabilityPoolToken.
- `assetSymbol :: Value.CurrencySymbol`. iAsset currency symbol.
- `cdpToken :: Value.AssetClass`. Token for identifying authentic CDP output.

Outputs:

Type	Description	Datum	Values
StabilityPool	Each StabilityPool output holds different flavors of iAsset.	*iAsset*: Type of iAsset. *Stability Providers*: Map of stability providers and their deposited amount. *Rewards*: Map of stability providers and their reward that is the amount of ADA obtained in the pool.	StabilityPoolToken: 1. iAsset: Funded by Stability Providers. ADA: Collateral transferred to StabilityPool from liquidated CDP.

Validation Rule:

1. Consume the StabilityPool (*iAsset, stabilityProviders, rewards*) output with LiquidateCDP redeemer:
   (a) The transaction must consume exactly one StabilityPool output.
   (b) The transaction must consume exactly one CDP (*id, owner, iAsset', mintedAmount*) output:
      i. $iAsset = iAsset'$
      ii. StabilityPool has enough funds to cover the subtraction of the debt and *mintedAmount*.

(c) The transaction must produce exactly one StabilityPool (*iAsset', stabilityProviders', rewards'*) to the StabilityPoolScript:

i. $iAsset = iAsset'$

ii. The *reward*, *stabilityProviders'* must be updated properly.

iii. Value = {*totalCollateralReward* ADA, *poolFund* iAsset, 1 StabilityPoolToken}

(d) Tx Mint/burn amount = {*mintedAmount* iAsset, (-1) CDPToken}

2. Consume the StabilityPool (*iAsset, stability providers, rewards*) output with AdjustSP redeemer:

(a) The transaction must consume exactly one StabilityPool output with StabilityPoolToken.

(b) Users cannot have a negative deposited amount after updating.

(c) Must produce one StabilityPool (*iAsset', stabilityProviders', rewards'*) output:

i. $iAsset' = iAsset$

ii. $stabilityProviders' = stabilityProviders$ with updates

iii. $rewards' = rewards$

iv. Value = {*oldTotalCollateralReward* ADA, *newPoolFund* iAsset, 1 StabilityPoolToken}

(d) Tx Mint/burn amount = {}

3. Consume the StabilityPool (iAsset, stability providers, rewards) output with WithdrawSPReward redeemer:

(a) The transaction must consume exactly one StabilityPool output and return one output with StabilityPoolToken.

(b) The transaction must withdraw a positive amount.

(c) Users can not have a negative reward amount after updating.

(d) Must produce one StabilityPool (*iAsset', stability providers', rewards'*) output:

i. $iAsset' = iAsset$

ii. $stabilityproviders' = stabilityproviders$ with updates

iii. $rewards' = rewards$

iv. Value = {*newTotalCollateralReward* ADA, *oldPoolFund* iAsset, 1 StabilityPoolToken}

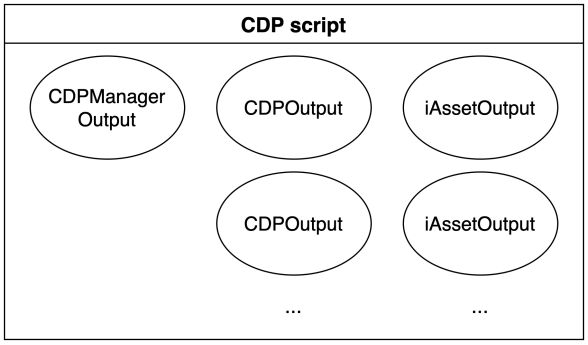
(e) Tx Mint/burn amount = {}

3.6 Collateralized Debt Position (CDP)

3.6.1 Native Tokens

Name	Description	Minting Policy
CDPManagerNFT	- The NFT identifies the authentic CDPManager output. - Must be stored at CDPManager output. - Validator scripts ensure that this NFT always stays at the CDPManager output.	The protocol mints exactly 1 token, before launch.
CDPToken	Identify the authentic CDP output.	The transaction must spend CDPManagerNFT or consume a CDPToken.
iAssetToken	- Identify the authentic iAsset output. - Validator scripts ensure that this token always stays at iAsset output.	The transaction must consume GovNFT.
iAssets (iTSLA, iXAU,...)	Synthetic version of TSLA, XAU,...	The transaction must either consume or spend a CDPToken.

3.6.2 OnChain



Parameters

- `cdpManagerNFT :: Value.AssetClass`. NFT of CDPManager.
- `cdpAuthToken :: Value.AssetClass`. Token for identifying authentic CDP output.

- `cdpAssetSymbol :: Value.CurrencySymbol`. Currency Symbol of all iAssets.
- `iAssetToken :: Value.AssetClass`. Token identifying authentic iAsset.
- `oracleParam :: OracleParams`. The parameters to interact with Oracle Script.
- `stabilityPoolToken :: Value.AssetClass`. Token identifying authentic Stability Pool.

Outputs:

Type	Description	Datum	Values
CDPManager	- Only one output of this type is stored in the script. - To create a CDP output, the user must consume this output in the transaction.	*currentCounter*: The number of opened CDPs.	CDPManagerNFT: 1.
CDP	- Each CDP output representing an individual position.	*id*: Index of CDP. *owner*: The public key hash that owns this CDP. *iAsset*: Type of iAsset minted of this CDP. *mintedAmount*: Amount of iAsset minted from this position. *minRatio*: The minimum collateral ratio of iAsset.	CDPToken: 1. ADA: collateral locked in this position.
iAsset	Each iAsset output representing an iAsset.	*iaName*: the name of iAsset. *iaMinRatio*: The minimum collateral ratio of iAsset.	iAssetToken: 1

Validation Rule:

1. Consume the CDPManager (*currentCounter*) output with CreateCDP redeemer:

(a) The transaction must produce CDPManager (*currentCounter*') to the CDPScript:
   i. $currentCounter' = currentCounter + 1$
   ii. Value = {1 CDPMangerNFT}

(b) The transaction must produce 1 CDP (*id, owner, iAsset, mintedAmount*) output:
   i. The transaction must be signed by owner.
   ii. $id = currentCounter + 1$
   iii. $mintedAmount \geq 0$
   iv. Collateral Ratio is above the Minimal Collateral Ratio.

(c) Tx mint/burn = {1 CDPToken, *mintedAmount* (*cdpAssetSymbol*, *iAsset*)}

(d) The transaction must consume the Oracle output to get the prices.

(e) The transaction must consume the iAsset output with $iaName = iAsset$.

2. Consume CDP (*id, owner, iAsset, mintedAmount*) output with AdjustCDP redeemer:

(a) The transaction must be signed by *owner*.

(b) Must produce exactly one CDP output (*id', owner', iAsset', mintedAmount'*) to the CDPScript:
   i. $id = id', owner = owner', iAsset = iAsset'$
   ii. $mintedAmount' \geq 0$
   iii. Collateral Ratio is above the Minimal Collateral Ratio.

(c) Tx mint/burn = {(*mintedAmount' - mintedAmount*) (*cdpAssetSymbol*, *iAsset*)}

(d) The transaction must consume the Oracle output to get the prices.

3. Consume CDP (*id, owner, iAsset, mintedAmount*) output with CloseCDP redeemer:

(a) The transaction must consume exactly one CDP output.

(b) The transaction must be signed by *owner*.

(c) Tx mint/burn = { -1 CDPToken, -*mintedAmount* (*cdpAssetSymbol*, *iAsset*)}

4. Consume the CDP (*id, owner, iAsset, mintedAmount*) output with LiquidateCDP redeemer:

(a) The transaction must consume exactly one CDP output.

(b) The transaction must consume exactly one StabilityPool output.

(c) Tx mint/burn = {-1 CDPToken}

(d) The collateral ratio of CDP must be under Minimal Collateral Ratio.

(e) The transaction must consume the Oracle output to get the prices.

5. Consume *iAsset* output with View redeeemer:

   (a) The transaction must return iAsset output with its iAssetToken.

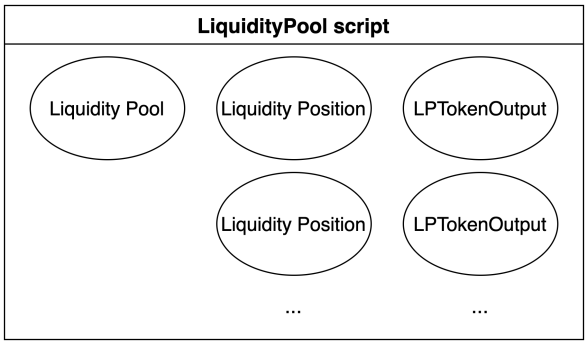
   (b) The iAsset datum must stay the same.

3.7 Liquidity Pool

3.7.1 Native Tokens

Name	Description	Minting Policy
LiquidityPoolNFT	Identify the authentic Liquidity-Pool output.	The protocol mints exactly 1 token, before launch.
LiquidityPosition-Token	Identify the authentic Liquidity-Position output.	The transaction must consume LiquidityPoolNFT.
LPTokens	User must add liquidity to iAsset-StableCoin/ INDY-StableCoin pool to receive.	Produced by DEX
LPAuthToken	Identify the authentic LPToken output	Then transaction must consume GovNFT.

3.7.2 OnChain



Parameters

- `liquidityPoolNFT :: Value.AssetClass`. NFT for identifying authentic Liquidity Pool output.

- `liquidityPositionToken :: Value.AssetClass`. Token for identifying authentic Liquidity Position output.
- `lpAuthToken :: Value.AssetClass`. Token for identifying authentic LPToken output.
- `indyToken :: Value.AssetClass`. INDY token.
- `vestingPeriods :: [(Ledger.POSIXTime, Ledger.POSIXTime), OnChainDecimal)]`. Map of the vesting periods (start, end) to the amount of INDY tokens released in that period. INDY tokens will released uniformly throughout the period.

Outputs:

Type	Description	Datum	Values
LiquidityPool	Only one output locked all INDY reward for liquidity pool	*totalLPTokenStake*: weighted sum of all staked LP tokens. *avgINDYPoolSnapshot*: more explanation below. *lastUpdateSnapshot*: the latest time pool was updated. *listStakers*: list of public key hash staked LPTokens.	INDY: INDY rewards for providing iAsset and INDY liquidity for DEX. LiquidityPoolNFT
LiquidityPosition	- Each output corresponding to a staking position of a user. - This output is created when the user stake LPToken for the first time.	*lpOwner*: owner of this Liquidity Position. *avgINDYPositionSnapshot*: avgINDYPoolSnapshot at the last time the owner interacts with his/her position. *totalPositionStaked*: weighted sum of all LP Tokens staked by lpOwner.	LiquidityPositionToken: 1. LPTokens: staked amount in this output.
LPToken	Each LPToken output representing an LPToken	*lpTokenAsset*: AssetClass of LPToken *weight*: LPToken's weight.	lpTokenOutput: 1

Validation Rule:

1. Consume LiquidityPool (*totalLPTokenStake, avgINDYPoolSnapshot, lastUpdateSnapshot, listStakers*) output with CreateLP (*owner, amount, currentTime*) redeemer

(a) The transaction must be signed by *owner*.

(b) $owner \notin listStakers$

(c) $currentTime > lastUpdateSnapshot$

(d) (*currentTime - biasTime, currentTime + biasTime*) contains the time range of transaction.

(e) Tx mint/burn amount = {1 LiquidityPositionToken}

(f) The transaction must consume only one LPToken (*lpTokenAsset, weight*) output and return the same one.

(g) Then transaction must produce one LiquidityPool (*totalLPTokenStake', avgINDYPoolSnapshot', lastUpdateSnapshot', listStakers*) output.

i. Datum

- $totalLPTokenStake' = totalLPTokenStake + amount * weight$
- If totalLPTokenStake = 0:

$$avgINDYPoolSnapshot' = avgINDYPoolSnapshot$$

- Else: $avgINDYPoolSnapshot' =$

$$avgINDYPoolSnapshot + \frac{Reward_{lastUpdateSnapshot \to currentTime}}{totalLPTokenStake}$$

- $lastUpdateSnapshot' = currentTime$
- $listStakers' = listStakers \cup owner$

ii. Value = LiquidityPoolNFT and:

- If $totalLPTokenStake = 0$:

$$oldINDYAmount - Reward_{lastUpdateSnapshot \to currentTime} INDY.$$

- Else : *oldINDYAmount* INDY

(h) The transaction must produce one LiquidityPosition (*owner, avgINDYPositionSnapshot, totalPositionStaked*) output.

i. $avgINDYPositionSnapshot = avgINDYPoolSnapshot'$

ii. $totalPositionStaked = amount * weight$

iii. Value = {1 LiquidityPositionToken, *amount* lpTokenAsset}

2. Consume LiquidityPool (*totalLPTokenStake, avgINDYPoolSnapShot, lastUpdateSnapshot, listStakers*) output with AdjustLP (*currentTime, amount*) redeemer:

(a) The transaction must consume one LiquidityPosition(_, _, *totalPositionStaked*) with LiquidityPositionToken.

(b) *amount* must be correct.

(c) The transaction must consume the LPToken (*lpTokenAsset, weight*) output and return the same one.

(d) (*currentTime - biasTime, currentTime + biasTime*) contains txValidRange.

(e) The transaction must return the correct Liquidity Pool (*totalLPTokenStake', avgINDYPoolSnapShot', lastUpdateSnapshot', vestingPeriods', listStakers'*) output:

i. Datum

- $totalLPTokenStake' = totalLPTokenStake + amount * weight$
- If totalLPTokenStake = 0 : $avgINDYPoolSnapshot' = avgINDYPoolSnapshot$
- Else :$avgINDYPoolSnapshot' =$

$$avgINDYPoolSnapshot + \frac{Reward_{lastUpdateSnapshot \to currentTime}}{totalLPTokenStake}$$

- $lastUpdateSnapshot' = currentTime$
- $listStakers = listStakers'$

ii. Value

- 1 LiquidityPoolNFT
- old amount of INDY - $(avgINDYPoolSnapShot - avgINDYPositionSnapshot) * totalPositionStaked$

3. Consume LiquidityPosition (*lpOwner, avgINDYPositionSnapshot, totalPositionStaked*) output with AdjustLP(*currentTime, amount*) redeemer:

(a) The transaction must consume one LiquidityPool output with LiquidityPoolNFT.

(b) The transaction must consume one LPToken (*lpTokenAsset, weight*) output and return the same one.

(c) The transaction must produce a LiquidityPool(_, *avgINDYPoolSnapShot'*, _) output.

(d) The transaction cannot mint/burn any tokens.

(e) The transaction must be signed by *lpOwner*.

(f) The transaction must return the correct LiquidityPosition (*lpOwner', avgINDYPositionSnapshot', totalPositionStaked'*) output:

i. Datum:

- $avgINDYPositionSnapshot' = avgINDYPoolSnapShot'$.
- $lpOwner' = lpOwner$
- $totalPositionStaked' = totalPositionStaked + amount * weight$

ii. Value:

- 1 LiquidityPositionToken
- The sum of old staked amount and *amount lpTokenAsset* .

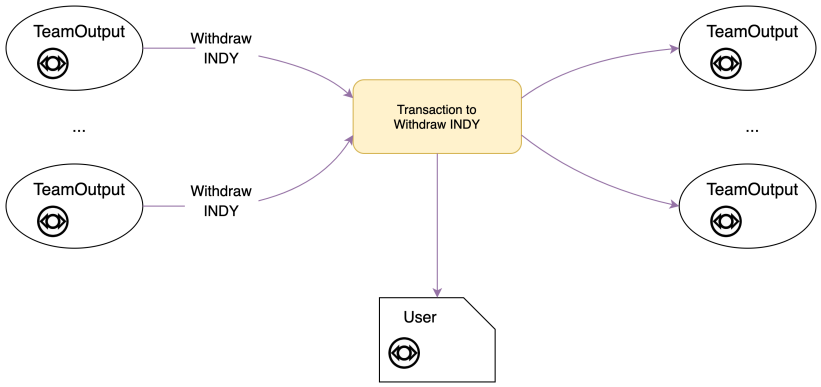
4. Consume LPToken (*lpToken, weight*) output with View redeemer:
   (a) The transaction must return the same LPToken output with LPAuthToken.

4 OffChain Endpoints

4.1 Vesting

4.1.1 Withdraw

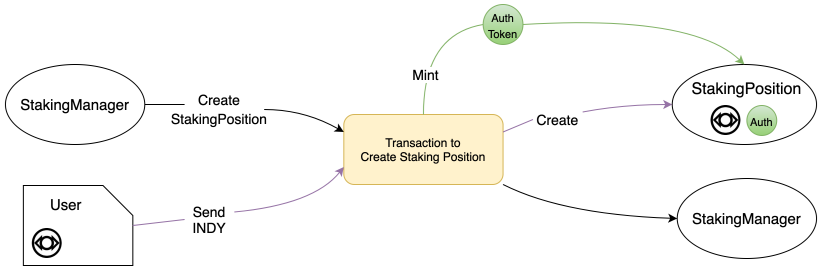
Team member can withdraw their available reward through Withdraw transaction.



4.2 Staking

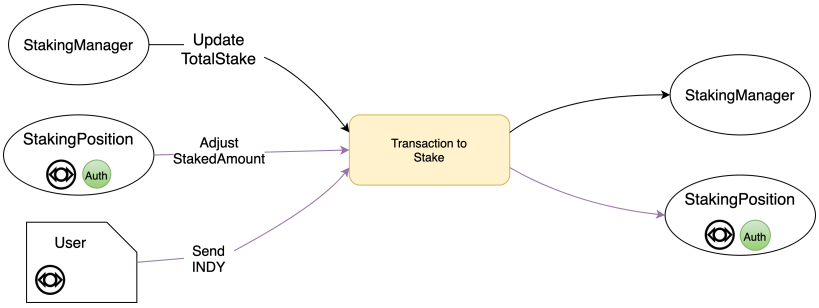
4.2.1 Create Stake Position

CreateStakePosition transaction allows users to create a stake position and stake their INDY.



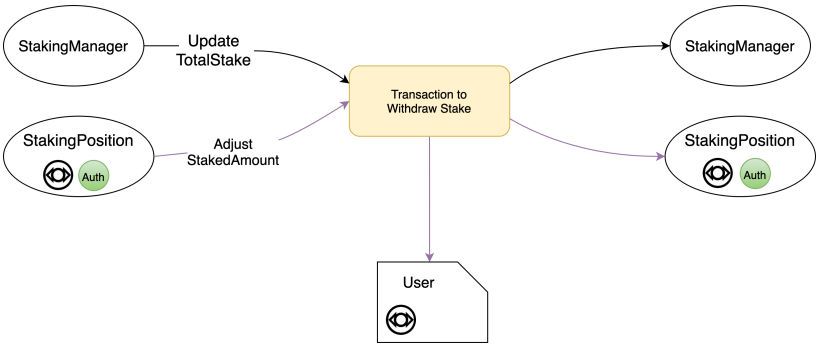
4.2.2 Stake

Stake transaction allows users to deposit INDY to their Staking position.



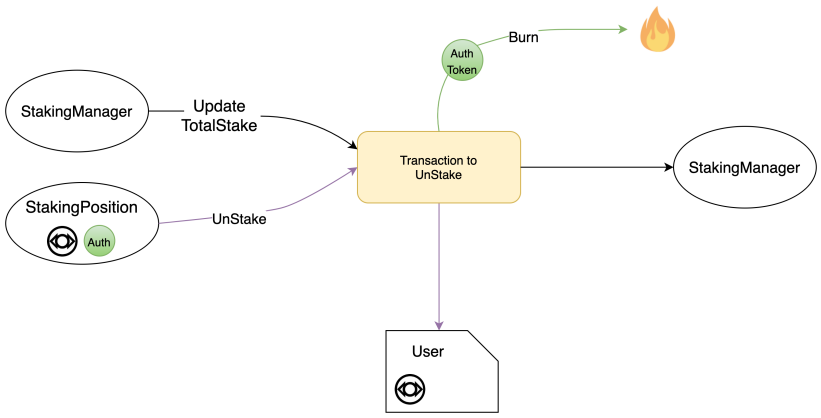
4.2.3 Withdraw from Stake Position

WithdrawStake transaction allows users to withdraw INDY from their Staking position.



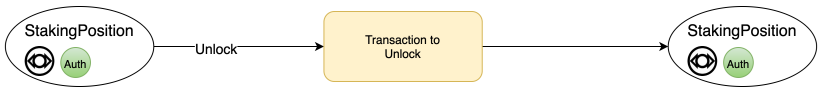
4.2.4 Unstake

Unstake transaction allows users to close their StakingPosition and withdraw all staked INDY.



4.2.5 Unlock

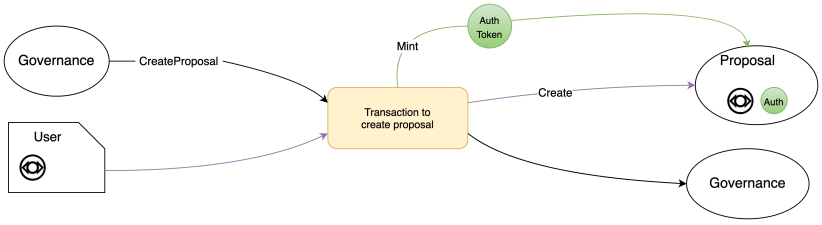
Unlock transaction allows users to unlock all INDY that was locked in the expired proposals



4.3 Governance

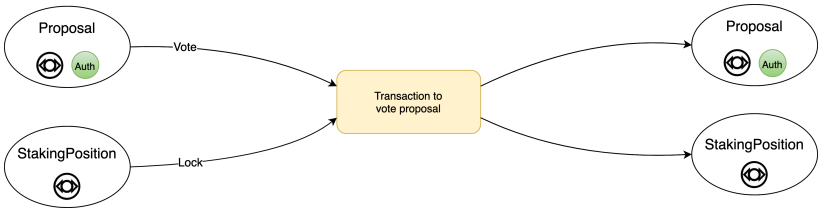
4.3.1 Create Proposal

Create Proposal transaction allows the user to create a Proposal and lock the amount of INDY as a fee.



4.3.2 Vote

Vote transaction allows the user to vote for a proposal with a vote option and vote fee.



4.3.3 End Proposal

End Proposal flow allows the user to end the proposal and pay the creation fee to the owner or Treasury Pool.

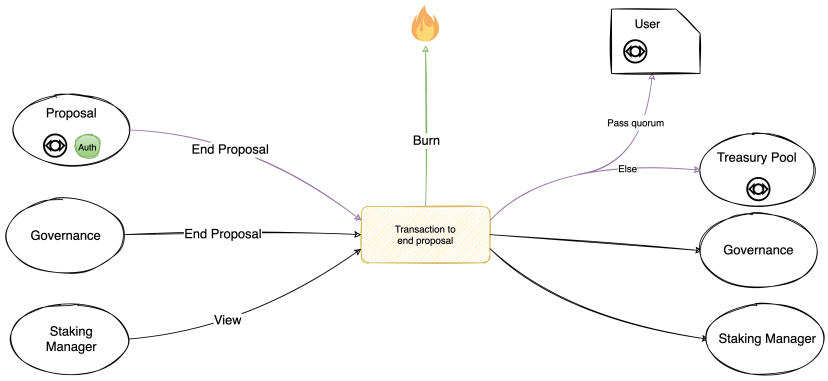


Figure 1: End a failed proposal

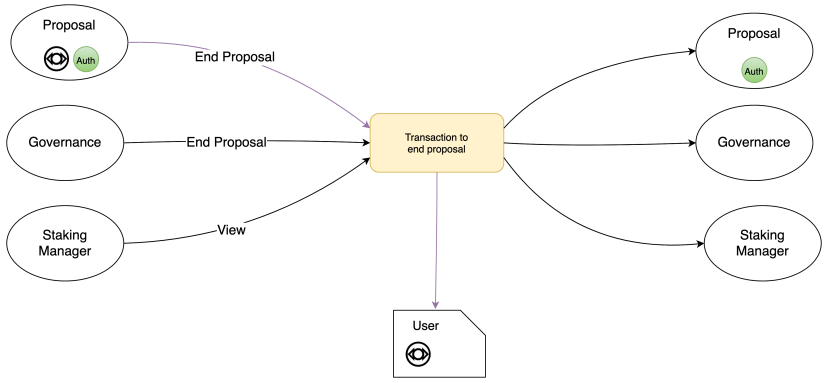


Figure 2: End a success proposal

4.3.4 Execute

Execute Asset allows the user to execute a passed proposal. There are 2 types of proposals:

- `UpdateProtocolParams` : To update protocol params in Gov output, output has only Gov.
- `ProposeAsset` : To execute a new iAsset, the output includes Gov, iAsset, and StabilityPool.

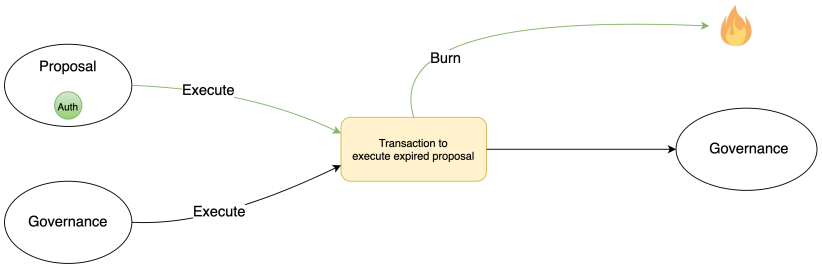


Figure 3: Execute an expired proposal

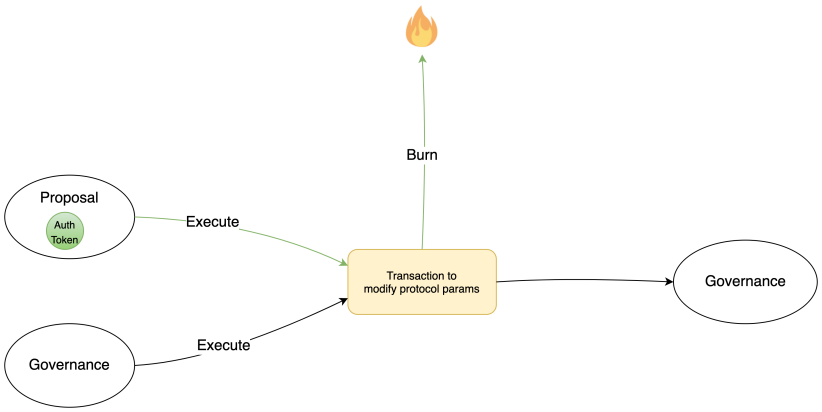


Figure 4: Execute a proposal to modify protocol params

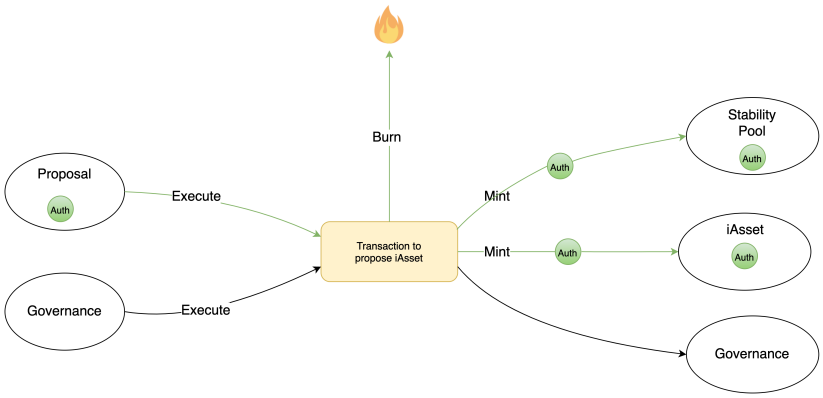


Figure 5: Execute a proposal to propose an asset

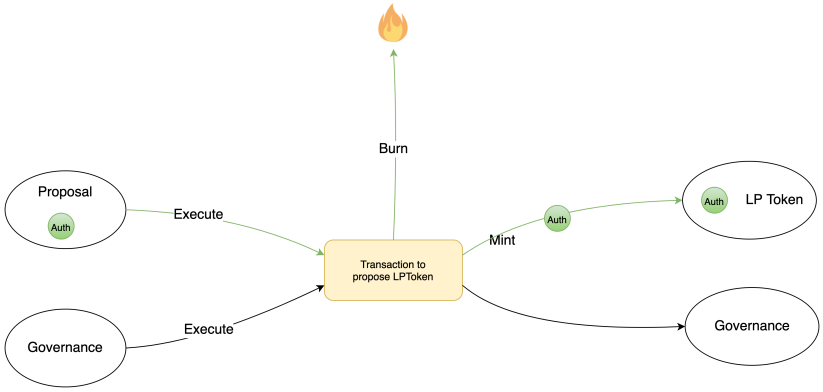


Figure 6: Execute a proposal to propose an LPToken

4.4 Oracle

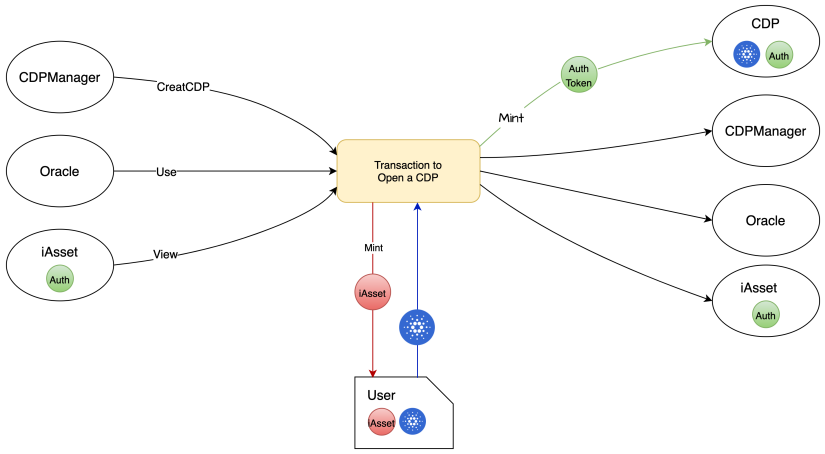
4.4.1 Feed Price

FeedPrice transaction flow allow oracles' owner to update the price of an iAsset

4.5 CDP

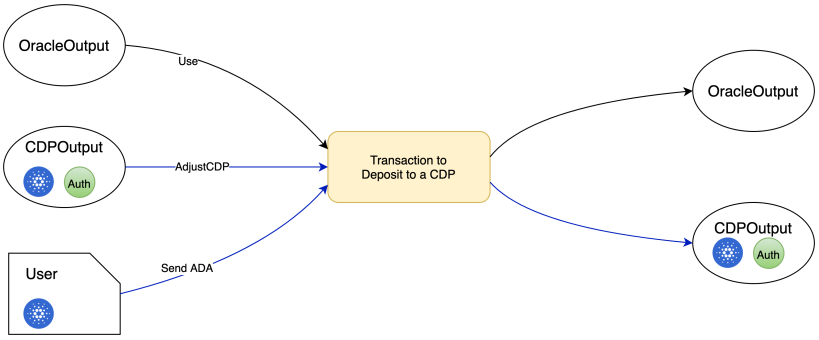
4.5.1 Open CDP

Open position allows users to create a CDP with Ada as a collateral asset and mint iAsset (iTSLA, iXAU,...).

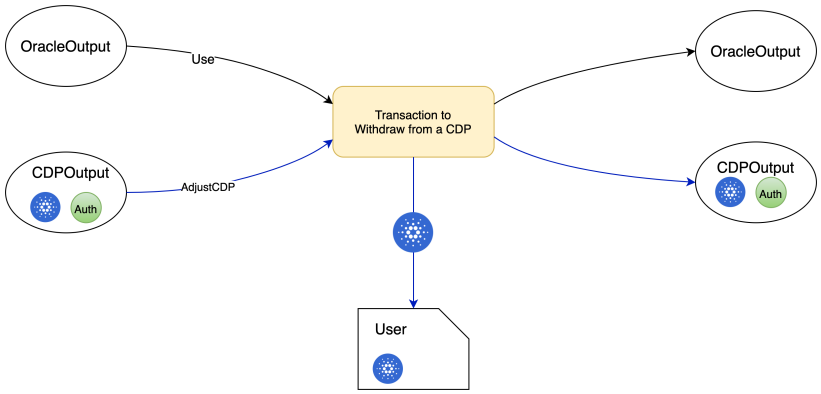


4.5.2 Update CDP (deposit, withdraw, mint, burn, close, liquidate)

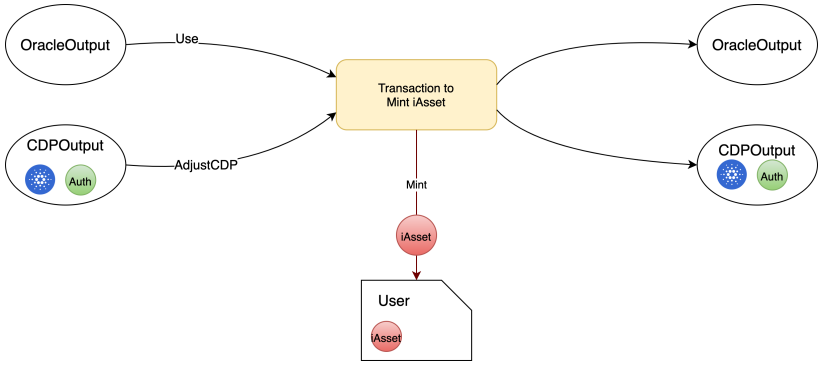
Deposit transaction allows users to deposit ADA to their CDP.



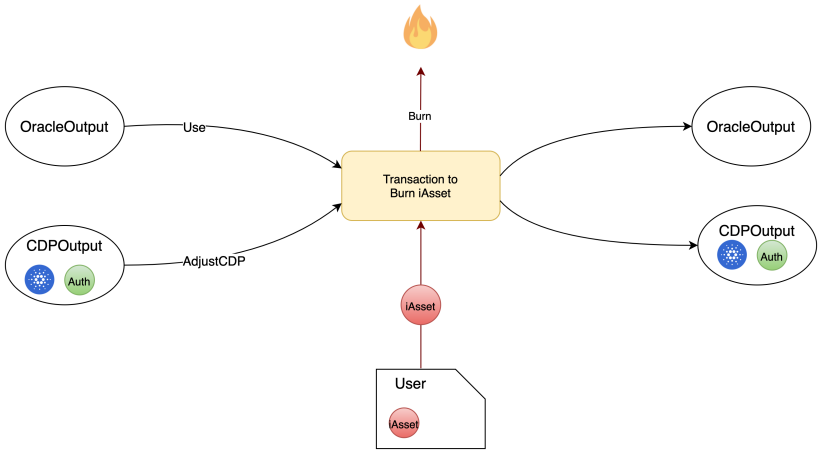
Withdraw transaction allow users to withdraw ADA from their CDPs



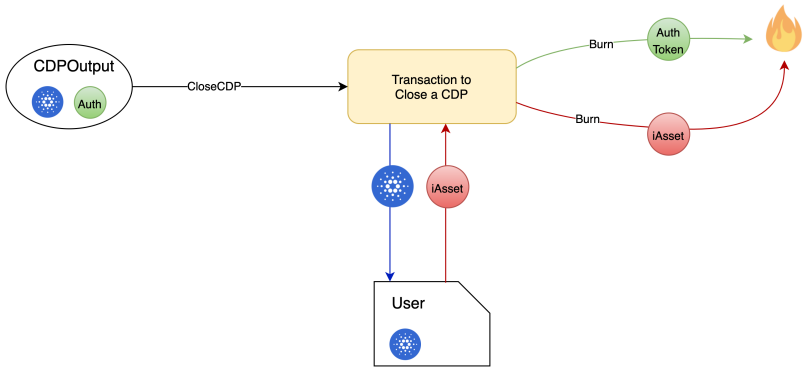
Mint transaction allows users to mint new iAsset.



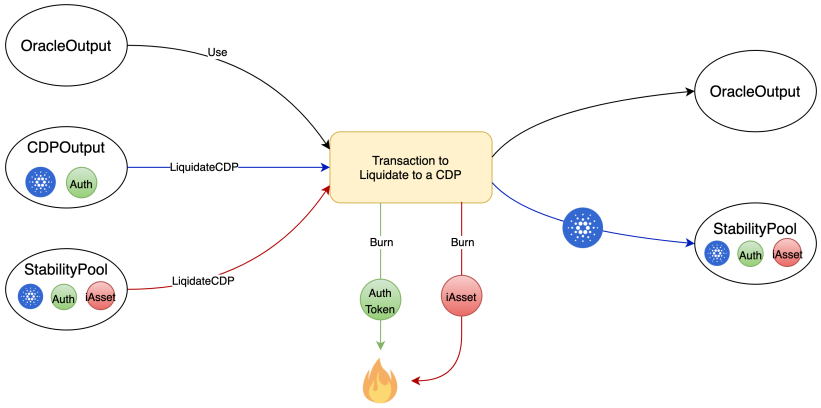
Burn transaction allows users to burn their iAsset (iTSLA, iGOLD,...).



Close transaction allows users to close their CDP then retrieve their ADA collateral.



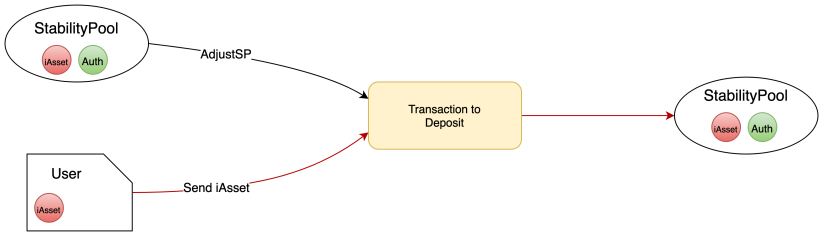
Liquidate transaction allows users to liquidate a CDP: burn iAsset in the stability pool and pay ADA reward from the collateral of CDP to stability pool providers.



4.6 Stability Pool

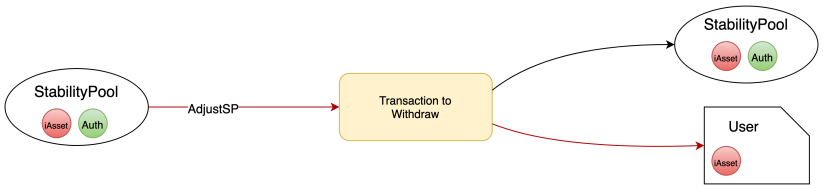
4.6.1 Deposit

Deposit transaction allows users to deposit their iAsset to a Stability Pool.



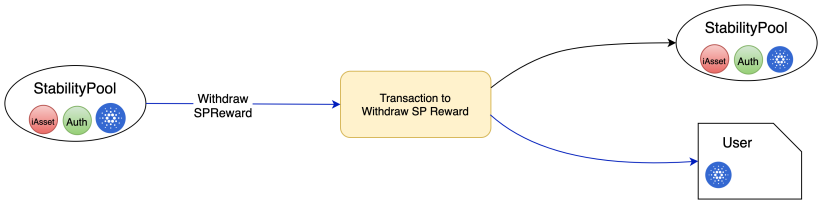
4.6.2 Withdraw

Withdraw transaction flow allows users to withdraw their deposited iAsset from Stability Pool.



4.6.3 Withdraw Reward

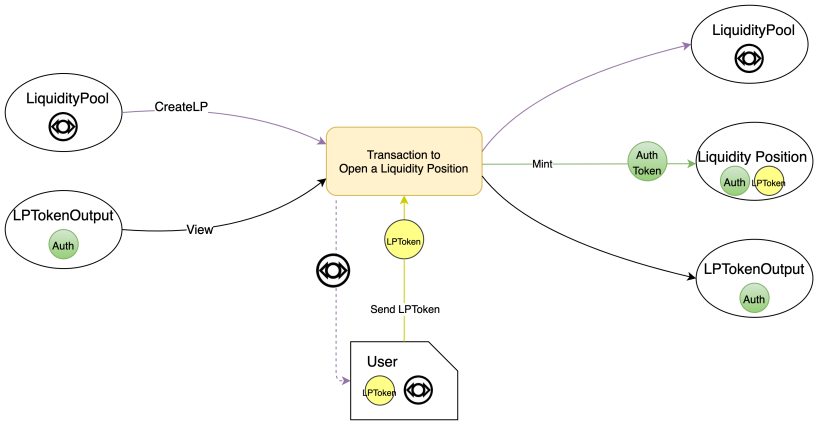
WithdrawReward transaction allows users to withdraw their ADA reward from Stability Pool.



4.7 Liquidity Pool

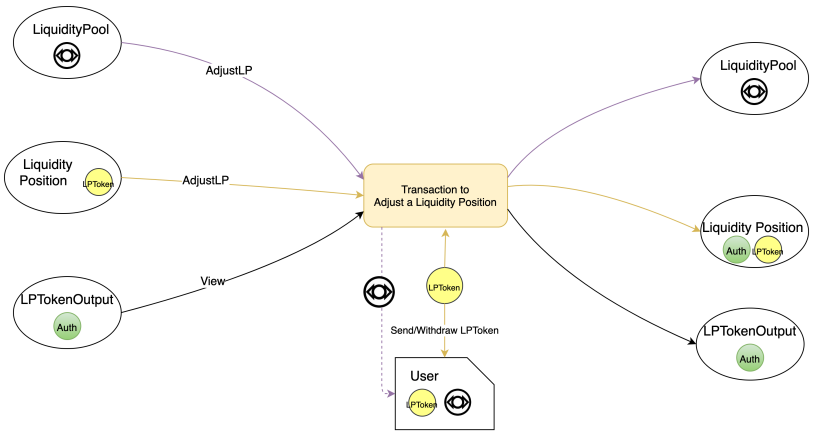
4.7.1 Open a Liquidity Position

The transaction allows the user to create a liquidity position in the liquidity pool and maybe receive INDY reward from the pool if he/she is the first provider.

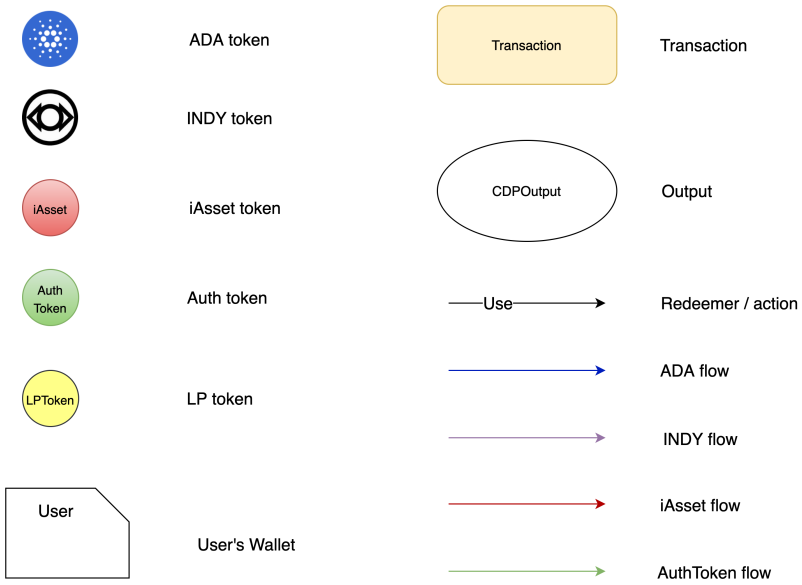


4.7.2 Adjust a Liquidity Position

The transaction allow the users to update their liquidity positions and maybe receive INDY reward from the pool.



5 Glossary



6 Experiments

Smart contract implementation and tests can be found here: `https://github.com/IndigoProtocol/smart-contracts`.

7 Related Work

This work is highly influenced by Mirror Finance (`https://mirror.finance`) & Liquity (`https://www.liquity.org/`)

8 Conclusions

The activity of trading cryptocurrencies and other real-world assets is steadily spanning beyond the 'professional trader' to the everyday person, globally. The creation of centralized applications like Robinhood & eToro, along with social media figureheads, have driven this exponential rise in interest and investment. Indigo Protocol will satisfy this rapidly growing demand with decentralized synthetic asset exposure for the common person. Indigo protocol will continue to grow significantly in the months and years ahead. Exploring and creating industry partnerships will allow new ideas and concepts to be deployed as blockchain technology and DeFi inevitably grows and evolves as a whole, while enhancements to the protocol will be driven by the user community and governance

proposal process. This will ensure that the user community maintains the protocol as the heirs to the platform in true DAO fashion.

References

[1] M. Chakravarty, James Chapman, K. Mackenzie, Orestis Melkonian, M. P. Jones, and P. Wadler. The extended utxo model. In *Financial Cryptography Workshops*, 2020.