Tinyman Liquid Staking

Repo: https://github.com/tinymanorg/tinyman-consensus-staking/tree/main/

Overview

Tinyman Liquid Staking will give Algo holders an opportunity to easily put their Algo to work securing the network and earning rewards. The liquid staking token will further allow them to continue to use their Algo in DeFi protocols like Tinyman AMM while earning swap fees, farming rewards and Algorand staking incentives rewards.
For users who do not want to participate in DeFi they may still stake with Tinyman and additionally earn TINY rewards if they wish through re-staking.

This document describes the two contracts & protocols for Liquid Staking (the tAlgo token) and Re-Staking.

tAlgo - Tinyman Liquid Staking

The tAlgo liquid token is issued by the tAlgo application. The application issues tAlgo in exchange for Algo. The Algo added to this application is collectively owned by the holders of tAlgo and can be redeemed at any time. The application account(s) is brought online to participate in consensus and potentially earns rewards. As rewards are earned the value of tAlgo increases relative to Algo. This means that users will receive more Algo when they “burn” their tAlgo compared to the Algo they initially deposited when “minting” tAlgo, provided the associated node has proposed blocks in the meantime.

Algorand Consensus does not use slashing so there is no risk of tAlgo’s value decreasing relative to Algo.

Formulae

```
RATE = algo_balance / minted_talgo
talgo_amount = algo_amount / RATE
algo_amount = talgo_amount * RATE
```

App Details

- This is a singleton app.

- This app is immutable.
- This app creates and issues the tAlgo ASA.
- This app manages up to 5 accounts that can be brought online to participate in consensus.
- The accounts potentially earn Algo through incentivised consensus when online.
- The app issues tAlgo in exchange for Algo.
- The Algo in the accounts is collectively owned by the holders of tAlgo.
- The accounts can be brought online for a node by the node_manager account set for each account
- The stake_manager account can move Algo between the accounts to redistribute the stake as necessary.
- The rewards from all accounts are pooled.
- The node_managers receive no reward or payment from this app.

tAlgo ASA

UnitName	TALGO
Name	tALGO
Total	10_000_000_000_000_000
Decimals	6
URL	https://tinyman.org
Creator	{tALGO Application Address}
ReserveAddress	{tALGO Application Address}
ManagerAddress	None
FreezeAddress	None
ClawbackAddress	None
DefaultFrozen	False

Staking Accounts

The app uses up to 5 accounts in total to hold staked Algo. The number 5 was chosen because a reference to each additional account is needed in each transaction to calculate the full balance and the received rewards. The Algorand protocol allows up to 4 account references with an app call and includes an extra implicit reference to the application account. Extra app call transactions could be included to increase the number of available account references but 5 accounts appears to be sufficient (see calculations below regarding max stake).

The Algorand protocol limit is currently 70M Algo per online account to remain eligible for proposer rewards. If any of the accounts does breach 70M Algo then the Algo in that account will stop earning rewards.

The tAlgo app stops minting when the balance of account 0 reaches 65M*. The extra 1M is to allow ample space for rewards to accumulate. It is expected that the stake manager will redistribute stake to the additional accounts before account 0 reaches the limit, until the limit is reached in all accounts.

A maximum of 345M* Algo (65M*5) can be staked with tAlgo.

*The 65M amount is a default value and can be updated by the manager.

Going Online/Offline

For an account to participate in consensus it needs to issue a KeyReg transaction with the participation keys for a node. It is possible to issue this transaction using inner transactions from an application. We are concerned however that future Algorand protocol upgrades will change the required fields of the KeyReg transaction*. If the app was using an inner transaction then it would not be able to successfully register online unless it was upgradable. We want to avoid upgradability due to the serious risks, trust issues and legal implications of an upgradable app. We have therefore decided to go use an alternative approach involving temporary rekeys for the KeyReg transaction.

When the node manager wishes to issue a KeyReg transaction they must sign it with their own key and include it in a group immediately after an app call with the change_online_status method. This method issues an inner transaction to rekey the relevant account to the node manager if and only if the next transaction is a KeyReg and the transaction rekeys back to the application account. Safety is ensured by only rekeying to the node manager account for the duration of one transaction and asserting its type.

The specifics of the other fields of the transaction are purposely not checked to allow for future flexibility. It is assumed that any upgrades to the Algorand protocol would not introduce ‘dangerous’ functionality in the KeyReg transaction. In any case the KeyReg can only be issued by the designated node manager, a semi-trusted role.

* Note this is not a purely theoretical scenario. Such a change was introduced to the KeyReg transaction with the introduction of StateProofs and state proof keys in 2022.

Node Running

It is intended that the Tinyman Organisation will contract professional service providers to run nodes for the protocol. This will initially be organized by the current Tinyman development team but later it is intended there will be a request for proposals and a vote in Tinyman Governance to allocate funds for the node running operations for a fixed time. Governance may vote to delegate managerial responsibility to a team or individual to ensure quality of service and reassign node running duties as necessary. A stake manager will also be appointed to manage the stake distribution between nodes. Over time it may be possible to transition a number of these responsibilities to permissionless contracts.

Node running quality control and compensation are out of the scope of this protocol.

User Flow

1. Opt-In to tAlgo
2. Mint tAlgo
    a. Transfer Algo to App Account
    b. Call mint(algo_amount)
    c. Receive tAlgo
3. Burn tAlgo
    a. Transfer tAlgo to App Account
    b. Call burn(talgo_amount)
    c. Receive Algo

Manager Flow

1. Deploy app, calling create_application
2. Transfer 2.0 Algo to app account
3. Call init()
4. Call set_node_manager for 1 or more node accounts
5. Call change_online_status with a keyreg in a group to bring an account online
6. Call move_stake to distribute stake between nodes as necessary

Permissions & Roles

The system requires a number of permissioned functions to operate. These permissions are role based to limit their scope.

It is an explicit design goal that a compromised manager account cannot affect access to the users' principal stake or accrued rewards in any way.

Note: None of these roles has permission to upgrade the application, remove Algo from the application accounts or rekey the accounts other than for KeyReg.

Manager

Global key: “manager”
Permissions:

- propose_manager
- set_node_manager
- set_stake_manager
- set_fee_collector
- set_protocol_fee
- set_max_account_balance

Proposed Manager

Global key: “proposed_manager”
Permissions:

- accept_manager

Node Manager

Global key: “node_manager_{i}”
Permissions:

- change_online_status
- Sign KeyReg transaction (only while grouped with change_online_status)

Stake Manager

Global key: “stake_manager”
Permissions:

- move_stake

Global State

Name	**Data Type**	**State Type**	**Comments**
talgo_asset_id	uint64	Constant	Set at init
account_0	Address (bytes[32])	Constant	Set at init
account_1	Address (bytes[32])	Constant	Set at init
account_2	Address (bytes[32])	Constant	Set at init

account_3	Address (bytes[32])	Constant	Set at init
account_4	Address (bytes[32])	Constant	Set at init
node_manager_0	Address (bytes[32])	Configurable	
node_manager_1	Address (bytes[32])	Configurable	
node_manager_2	Address (bytes[32])	Configurable	
node_manager_3	Address (bytes[32])	Configurable	
node_manager_4	Address (bytes[32])	Configurable	
manager	Address (bytes[32])	Configurable	
proposed_manager	Address (bytes[32])	Configurable	
stake_manager	Address (bytes[32])	Configurable	
fee_collector	Address (bytes[32])	Configurable	
protocol_fee	uint64	Configurable	0-100
max_account_balance	uint64	Configurable	MicroAlgo
initial_balance	uint64	Internal State	MicroAlgo
minted_talgo	uint64	Internal State	Micro Units
algo_balance	uint64	Internal State	MicroAlgo
rate	uint64	Internal State	Rate * 1,000,000,000,000
total_rewards	uint64	Internal State	MicroAlgo All time rewards
protocol_talgo	uint64	Internal State	tAlgo claimable to fee_collector

Methods

Init

This method initializes the application, creates tALGO asset, creates “auxiliary” accounts that are managed through the application.

1. App Call:
   Sender: Any Address
   Index: tAlgo App ID
   OnComplete: NoOp
   App Args: [“init”]
   Inner Transactions:
   1. Asset Config:
      Sender: tALGO Application Address
      Receiver: tALGO Staking Application Address
      AssetUnitName: "TALGO"
      AssetName: "tALGO"
      AssetTotal: 10_000_000_000_000_000
      AssetDecimals: 6
      AssetURL: "https://tinyman.org"
      AssetReserve: tALGO Application Address

←—-- Repeat for each auxiliary account (4x) —-------------------------->

   2. App Call:
      OnComplete: DeleteApplication
      ApprovalProgram: {AUX_PROGRAM}
      ClearState: "\x0A\x81\x01"
      Inner Transactions:
      1. Pay:
         Sender: {AUX_PROGRAM Created Account Address}
         Receiver: tALGO Application Address
         RekeyTo: tALGO Application Address
         Amount: 0
   3. Pay:
      Sender: tALGO Application Address
      Receiver: {AUX_PROGRAM Created Account Address}
      Amount: 100_000

←—--->

Side Effects: Initialize the global state variables, rate being 1 initially. Store every created auxiliary account address as “account_{i}”, while 1 <= i <= 4, into global state variables.

Mint

This method is used to stake ALGO into the stakepool and get tALGO in return.

1. Payment:
   Sender: User Address

Receiver: tALGO Application Address
Amount: <algo_amount>

2. App Call:
   Sender: User Address
   Index: tAlgo App ID
   OnComplete: NoOp
   App Args: [“mint”, <algo_amount: int>]
   Inner Transactions:
   1. Asset Transfer:
      Sender: tALGO Application Address
      Receiver: User Address
      Index: tALGO Asset Id
      Amount: {(algo_amount * RATE_SCALER) / rate}

Side Effects: `update_rewards_and_rate` internal function is called, rate and other balance related global states are updated.

Burn

This method is used to burn tALGO and receive ALGO in return.

1. Asset Transfer:
   Sender: User Address
   Receiver: tALGO Application Address
   Index: tALGO Asset Id
   Amount: <talgo_amount>

2. App Call:
   Sender: User Address
   Index: tAlgo App ID
   OnComplete: NoOp
   App Args: [“burn”, <talgo_amount: int>]
   Inner Transactions:
   1. Pay:
      Sender: tALGO Application Address
      Receiver: User Address
      Amount: {(talgo_amount * rate) / RATE_SCALER}

Side Effects: `update_rewards_and_rate` internal function is called, rate and other balance related global states are updated.

Sync

This method is a wrapper around the `update_rewards_and_rate` internal function.

1. App Call:
   Sender: User Address
   Index: tAlgo App ID
   OnComplete: NoOp
   App Args: [“sync”]

Side Effects: Rate and other balance related global states are updated.

Claim Protocol Rewards

This method is used to claim the tALGO that is accumulated through fees.

1. App Call:
   Sender: Any Address
   Index: tAlgo App ID
   OnComplete: NoOp
   App Args: [“claim_protocol_rewards”]
   Inner Transactions:
   1. Asset Transfer:
      Sender: tALGO Application Address
      Receiver: {Fee Collector Address}
      Index: tALGO Asset Id
      Amount: {protocol_talgo}

Side Effects: Accumulated tALGO is sent to the fee collector and `protocol_talgo` global state variable is set to 0.

Propose Manager

First step of the 2 step manager changing mechanism. This system is introduced for ensuring that the new manager is accessible and parameter mistakes are prevented.

Transaction Group:
1. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App Id
   OnComplete: NoOp

App Args: [“propose_manager”, <new_manager_address: bytes[32]>]

Side Effects: “proposed_manager” global state is updated with the <new_manager_address> parameter.

Accept Manager

Previously proposed manager account can accept the manager position by calling this method. This step ensures that the proposed account is correct and accessible.

Transaction Group:

1. App Call:
   Sender: Proposed Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“accept_manager”]

Side Effects: “manager” global state is updated with the “proposed_manager” value. “proposed_manager” global state is set to null.

Set Node Manager

This manager method is used to assign an account for the node that is described by its index.

Transaction Group:

1. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“set_node_manager”, <node_index: int>, <new_node_manager: bytes[32]>]

Side Effects: “node_manager_{node_index}” global state variable is updated with the given `new_node_manager` parameter.

Set Stake Manager

This manager method is used to assign the stake_manager, the account that has permission to call move_stake.

Transaction Group:

1. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“set_stake_manager”, <new_stake_manager: bytes[32]>]

Side Effects: “stake_manager” global state variable is updated.

Set Fee Collector

This manager method is used to assign the fee collector; the account that the protocol fees will be sent to when the `claim_protocol_rewards` method is called.

Transaction Group:

2. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“set_fee_collector”, <new_fee_collector: bytes[32]>]

Side Effects: “fee_collector” global state variable is updated.

Set Protocol Fee

This manager method is used to set the protocol fee percentage.
E.g. for 10% the fee_amount should be 10

Transaction Group:

1. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“set_protocol_fee”, <fee_amount: int>]

Side Effects: “protocol_fee” global state variable is updated.

Set Max Account Balance

This manager method is setting the maximum amount of ALGO the application and auxiliary accounts can hold each.
Max_amount must be microAlgo.

Transaction Group:

1. App Call:
   Sender: tALGO Application Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“set_max_account_balance”, <max_amount: int>]

Side Effects: “max_account_balance” global state variable is updated.

Change Online Status

This method allows the node manager to make a keyreg transaction on behalf of the node’s related account, “account_{node_index}”. It is ensured that the transaction after the application call is a keyreg transaction and rekeys the account back to the application account.

Note: If the KeyReg transaction fee is > 0 the same amount of Algo must be paid to the node account address (global “account_{node_index}”) in the transaction preceding the App Call in the group. According to the current Algorand consensus parameters the fee must be set to 2.0 Algo the first time an account goes online to become eligible for rewards. The same fee must be applied when going back online after being evicted.

Transaction Group:

1. App Call:
   Sender: Node Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“change_online_status”, <node_index: int>]
   Inner Transactions:
    1. Pay:
       Sender: “account_{node_index}”
       Receiver: Node Manager Address
       RekeyTo: Node Manager Address
2. Keyreg:

Sender: “account_{node_index}”
RekeyTo: tALGO Application Address
{Additional Keyreg fields for going offline/online}

Side Effects: This method affects the account’s participation.

Move Stake

This method can be called only by the stake manager. It is used to move the staked ALGO between application and auxiliary accounts.

Amount must be specified in microAlgo.

Transaction Group:

1. App Call:
   Sender: Stake Manager Address
   Index: tALGO App ID
   OnComplete: NoOp
   App Args: [“move_stake”, <from_index: int>, <to_index: int>, <amount: int>]
   Inner Transactions:
   1. Pay:
      Sender: “account_{from_index}”
      Receiver: “account_{to_index}”
      Amount: <amount>

Side Effects: Account balances are affected.

tAlgo Staking (aka ReStaking)

The tALGO Staking application allows tAlgo holders to stake their tAlgo to earn TINY rewards. The user stakes their tAlgo by transferring their tAlgo to the contract in the appropriate transaction group. The user receives an equal amount of stAlgo in return. This is a non-liquid (frozen) asset. When the user unstakes the stAlgo is clawbacked and an equal amount of tAlgo is returned. The TINY rewards accumulate every second and can be claimed at any time.

The sole purpose of stAlgo is to give users an ASA that represents their locked value so their wallets and portfolio trackers can give accurate balance information without requiring a special integration with the restaking app. This is a design requirement because the target audience for tAlgo and tAlgo ReStaking includes users who would not normally participate in DeFi and would expect to see their full portfolio balance in their wallets.

TINY rewards are allocated for fixed periods set by the manager. The amount and duration of the period determines the total rate of rewards per second. The rate of reward per unit of staked tAlgo is determined by the rate of rewards per second and the total staked amount.
The process for funding the rewards and determining the amount of rewards is outside the scope of this protocol.

Formulae

We heavily rely on these calculations:
[0] https://github.com/stakewithus/notes/blob/main/excalidraw/staking-rewards.png

Using a reward rate per time, rewards are accumulated to distribute on per unit staked later on.

```
time_delta = Global.LatestTimestamp - “last_update_timestamp”

reward_rate_per_unit_per_time = (reward_rate_per_time * RPU_SCALER) /
total_staked_amount

accumulated_rewards_per_unit = accumulated_rewards_per_unit +
(reward_rate_per_unit_per_time * time_delta)
```

This value is used to calculate an user’s earned reward amount for the staked amount and the timedelta:

```
rewards_per_unit_delta = accumulated_rewards_per_unit -
user_state.accumulated_rewards_per_unit_at_last_update
```

```
rewards_delta = (user_state.staked_amount * rewards_per_unit_delta) /
RPU_SCALER
```

Global State

Reward related global states:

- "current_reward_rate_per_time": Calculated by the "total_reward_amount / timedelta". This variable is denoting the number of TINY amounts that will be given per seconds.

- "current_reward_rate_per_time_end_timestamp": Expiration timestamp for the "current_reward_rate_per_time" variable. It denotes the end of the reward period that is set by the manager.

- "accumulated_rewards_per_unit": Between the two app usage, user's accumulated rewards are calculated on a per staked unit basis. This variable denotes the total accumulated amount that will be used to calculate the user's share.

Box Storage

tALGO staking contract uses given struct to hold user information:

```
# Name: <user_address>
# Size: 24 bytes.
struct UserState:
   staked_amount: int
   accumulated_rewards_per_unit_at_last_update: int
   accumulated_rewards: int
   timestamp: int
end
```

The box data is updated in each user action with the `update_user_state` internal function.

stAlgo ASA

UnitName	STALGO
Name	Staked tALGO

Total	10_000_000_000_000_000
Decimals	6
URL	https://tinyman.org
Creator	{tAlgo Staking Application Address}
ReserveAddress	{tAlgo Staking Application Address}
ManagerAddress	None
FreezeAddress	{tAlgo Staking Application Address}
ClawbackAddress	{tAlgo Staking Application Address}
DefaultFrozen	True

User Flow:

1. Opt-in to stALGO
2. Increase Stake
    a. Lock TINY to obtain required TINY power.
    b. Transfer the box cost (Algo) for the first increase stake.
    c. Transfer tAlgo to stake.
    d. Call increase stake.
    e. Receive stALGO.
3. Decrease Stake
    a. Call decrease stake.
    b. stALGO sent back using clawback.
    c. Receive tALGO.
4. Claim Rewards
    a. Call claim rewards.
    b. Receive TINY

Permissions & Roles

The system requires a permissioned function to operate. A single manager role is used for the two actions to configure parameters.

Note: No role has permission to upgrade the application, remove any assets from the application account, rekey the account, or clawback any asset.

Manager

Global key: “manager”
Permissions:

- propose_manager
- set_tiny_power_threshold
- set_reward_rate

Proposed Manager

Global key: “proposed_manager”
Permissions:

- accept_manager

Methods

Init

This is a one time transaction for opting into TINY and tALGO assets, creating the stALGO asset.

Transaction Group:

1. App Call:
   Sender: Talgo Staking Manager Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“init”]
   Inner Transactions:
    1. Asset Transfer:
       Sender: Talgo Staking Application Address
       Receiver: Talgo Staking Application Address
       Amount: 0
       Asset Index: TINY Asset Id

    2. Asset Transfer:
       Sender: Talgo Staking Application Address
       Receiver: Talgo Staking Application Address
       Amount: 0
       Asset Index: tALGO Asset Id

    3. Asset Config:
       Sender: Talgo Staking Application Address
       Receiver: Talgo Staking Application Address

AssetUnitName: "STALGO"
AssetName: "Staked tALGO"
AssetTotal: 10_000_000_000_000_000
AssetDecimals: 6
AssetURL: "https://tinyman.org"
AssetReserve: Talgo Staking Application Address
AssetClawback: Talgo Staking Application Address
AssetFreeze: Talgo Staking Application Address
AssetDefaultFrozen: 1

Propose Manager

First step of the 2 step manager changing mechanism. This system is introduced for ensuring that the new manager is accessible and parameter mistakes are prevented.

Transaction Group:

1. App Call:
   Sender: Talgo Staking Manager Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“propose_manager”, <new_manager_address: bytes[32]>]

Side Effects: “proposed_manager” global state is updated with the <new_manager_address> parameter.

Accept Manager

Previously proposed manager account can accept the manager position by calling this method. This step ensures that the proposed account is correct and accessible.

Transaction Group:

1. App Call:
   Sender: Proposed Manager Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“accept_manager”]

Side Effects: “manager” global state is updated with the “proposed_manager” value. “proposed_manager” global state is set to null.

Set TINY Power Threshold

This method is used to set the TINY power requirement for the user's first `increase_stake` call and `claim_rewards` calls.

Transaction Group:

1. App Call:
   Sender: Talgo Staking Manager Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“set_tiny_power_threshold”, <threshold: int>]

Side Effects: “tiny_power_threshold” global state variable is updated with the <threshold> parameter.

Set Reward Rate

This method is used to set the TINY amount that will be given per seconds (`reward_rate_per_time`) until `end_timestamp` is reached. The new rate is valid after the transaction until the `end_timestamp`.

`total_reward_amount` is the amount of TINY (microunits) to be distributed until `end_timestamp`.

Transaction Group:

1. App Call:
   Sender: Talgo Staking Manager Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“set_reward_rate”, <total_reward_amount: int>, <end_timestamp: int>]

Prerequisites: Application’s TINY balance should be enough to distribute the unclaimed reward amount and the `total_reward_amount`.

Side Effects: State is folded till the transaction’s timestamp and state is updated.
“current_reward_rate_per_time” is calculated from the parameters and updated.
“current_reward_rate_per_time_end_timestamp” is updated with `end_timestamp`.
“total_reward_amount_sum” is adjusted accordingly.

Apply Rate Change

This method is used to fold up the state and set the `reward_rate_per_time` to 0, signifying the end of the current reward period. The app is blocked for the user methods until this method is called. Side effects are only applied if the “current_reward_rate_per_time_end_timestamp” is exceeded.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“apply_rate_change”]

Side Effects: “current_reward_rate_per_time” is set to 0.
“current_reward_rate_per_time_end_timestamp” is set to MAX_UINT64, practically it is ensured that the rate is 0 until a `set_reward_rate` call.

Update State

This method is a wrapper public method for `update_state_internal` function. It is used to do the calculations described in the global state section, updating a subgroup of the global state variables that are described as the “state” in the contract. This method will fail if “current_reward_rate_per_time” is invalid for the transaction’s `Global.LatestTimestamp`. This situation requires an `apply_rate_change` call as it is described in the documentation.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“update_state”]

Side Effects: “accumulated_rewards_per_unit” and “last_update_timestamp” are updated.

Increase Stake

This method is used by the user for increasing the staked amount of tALGO.

Transaction Group:

1. Asset Transfer:
   Sender: User Address
   Receiver: Talgo Staking Application Address
   Amount: <amount>
   Asset Index: tALGO Asset Id

2. App Call:
   Sender: User Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: ["increase_stake", <amount: int>]
   Foreign Apps: [Tinyman Governance Vault App Id]
   Foreign Assets: [stALGO Asset Id]
   Boxes: [
   (0, <user_address_decoded>),
   (Tinyman Governance Vault App Id, <user_address_decoded>)
   ]
   Inner Transactions:
   1. App Call:
      Sender: Talgo Staking Application Address
      Index: Tinyman Governance Vault App Id
      OnComplete: NoOp
      App Args: ["get_tiny_power_of", <address: bytes[32]>]

   2. Asset Transfer:
      Sender: Talgo Staking Application Address
      AssetSender: Talgo Staking Application Address
      Receiver: User Address
      Amount: <amount>
      Asset Index: stALGO Asset Id

Prerequisites: User should be opted into stALGO asset. Users must hold an equal or higher amount of TINY power that is represented as the "tiny_power_threshold" global state variable for the first call.

Side Effects: `update_state_internal` function is called and state is updated.
The UserState box that is represented with the user's address is created and or updated. An equal amount of stALGO is sent to the user using clawback.

Decrease Stake

This method is used by the user for decreasing the staked amount of tALGO.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“decrease_stake”, <amount: int>]
   Foreign Assets: [tALGO Asset Id, stALGO Asset Id]
   Boxes: [
       (0, <user_address_decoded>),
   ]
   Inner Transactions:
    1. Asset Transfer:
       Sender: Talgo Staking Application Address
       AssetSender: User Address
       Receiver: Talgo Staking Application Address
       Amount: <amount>
       Asset Index: stALGO Asset Id

    2. Asset Transfer:
       Sender: Talgo Staking Application Address
       Receiver: User Address
       Amount: <amount>
       Asset Index: tALGO Asset Id

Side Effects: `update_state_internal` function is called and state is updated.
The UserState box that is represented with the user's address is updated.
tALGO is sent back to the user and an equal amount of stALGO is clawbacked from user to application account.

Claim Rewards

This method is used by the user to claim accumulated rewards from the application.

Transaction Group:

1. App Call:
   Sender: User Address
   Index: Talgo Staking App ID
   OnComplete: NoOp
   App Args: [“claim_rewards”]

Foreign Apps: [Tinyman Governance Vault App Id]
Foreign Assets: [TINY Asset Id]
Boxes: [
(0, <user_address_decoded>),
(Tinyman Governance Vault App Id, <user_address_decoded>)
]
Inner Transactions:

1. App Call:
   Sender: Talgo Staking Application Address
   Index: Tinyman Governance Vault App Id
   OnComplete: NoOp
   App Args: [“get_tiny_power_of”, <address: bytes[32]>]

2. Asset Transfer:
   Sender: Talgo Staking Application Address
   Receiver: User Address
   Amount: <UserState.accumulated_rewards>
   Asset Index: TINY Asset Id

Side Effects: UserState.accumulated_rewards is set to 0.