DelegatedClaimCampaigns - Smart Contract Technical Docs & Architecture

Smart Contract: DelegatedClaimCampaigns.sol

References of previous contracts:
https://github.com/hedgey-finance/Locked_VestingTokenPlans/tree/master/technical%20documentation

Background: Hedgey Finance builds onchain tools for automated vesting and token lockups and distributions. This contract is a tool used to distribute tokens to hundreds of thousands of wallets, using a merkle tree based claim solution. The contract uploads a root of a merkle tree, where the proofs and amounts are generally held in an offchain storage like amazon S3 or IPFS for users to access in order to claim their tokens. This contract has the additional functionality that it can require claimers to delegate their tokens at the same time they are claiming them, if the token is an ERC20Votes standard and the creator mandates such delegation. This is an important feature for DAOs launching where they need a set minimum amount of tokens to be part of governance for the DAO to function and meet onchain quorums for proposals.

Purpose:
The DelegatedClaimCampaigns contract is a tool to be used by DAO projects when launching their token, to efficiently distribute tokens to a large number of members and bake in vesting and unlock schedules. The tool can enforce recipients to delegate tokens, and it can also use EIP712 signatures for gasless transactions to make claiming super efficient for users if projects want to sponsor the claims and pay for gas.

Functional Overview

Roles

Role Name	Description	Jobs / Functions
Campaign Creator / Admin	The creator of the claim campaigns. The admin creates the merkle tree and uploads the root, along with decisions of the lockup or vesting schedule and if the tokens are required to be delegated at claim transaction.	- Creating the claim campaigns - Cancelling campaigns to end them
Claimer	Individual Wallets that claim tokens, or claim locked or vesting tokens, with additional delegation requirements	- Claim tokens - Claim and delegate tokens
Paymaster / Claimer OBO	A paymaster that is claiming tokens on behalf of claimaints	Claiming tokens on behalf Claim and delegate tokens

Core Functions

Function Name	Description	Role / User
Create unlocked campaign	Function for the creator to setup a claim campaign where the tokens will be unlocked, ie liquid when claimed	Campaign Creator / Admin
Create locked campaign	Function for the creator to setup a claim campaign where tokens will be locked inside a vesting or lockup plan when claimed, and so users receive the lockup plan instead of liquid tokens	Campaign Creator / Admin
claim	Function for a recipient to claim their tokens, regardless of lock or unlocked. Tokens must not be required to be delegated when being claimed	Claimer
claimMultiple	Function for claimer to claim tokens from multiple claim campaigns, must not require delegation	Claimer
claimWithSig	Function for a paymaster to claim tokens on behalf of a claimer, tokens delivered to claimer, must not require delegation	Paymaster / Claimer OBO
claimMultipleWithSig	Function for a paymaster to claim tokens from multiple campaigns on behalf of a claimer, tokens delivered to claimer and must not require delegation	Paymaster / Claimer OBO
claimAndDelegate	Function for claimers to claim tokens where the campaign requires delegation	Claimer
claimAndDelegateWithSig	Function for a paymaster to claim on behalf of a claimer, and also delegate on their behalf	Paymaster / Claimer OBO

Smart Contract Technical Definitions & Design

Contract Inheritance

Open Zeppelin Imports:

@openzeppelin/contracts/utils/ReentrancyGuard.sol
@openzeppelin/contracts/utils/cryptography/MerkleProof.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/token/ERC721/utils/ERC721Holder.sol
@openzeppelin/contracts/utils/cryptography/EIP712.sol
@openzeppelin/contracts/utils/cryptography/ECDSA.sol
@openzeppelin/contracts/utils/Nonces.sol

Hedgey Contract Imports:

TransferHelper.sol - Library to assist with transferring tokens to and from beneficiaries and contracts

Interfaces:
IVestingPlans.sol - Interface for creating a vesting plan when claiming tokens to be vested
ILockupPlans.sol - Interface for creating a lockup plan when claiming tokens that are locked
IDelegatePlan.sol - Interface for delegating the tokens in a vesting or lockup plan
IERC20Votes.sol - Interface for delegating base ERC20Votes tokens with delegateBySig

Contract Objects, Variables & Functions

Enums

```
enum TokenLockup {
   Unlocked,
   Locked,
   Vesting
 }
```

Unlocked means that tokens claimed are liquid and not locked at all
Locked means that the tokens claimed will be locked inside a TokenLockups plan
Vesting means the tokens claimed will be locked inside a TokenVesting plan

Structs

```
struct ClaimLockup {
   address tokenLocker;
```

```
    uint256 start;
    uint256 cliff;
    uint256 period;
    uint256 periods;
  }
```

The ClaimLockup struct is required for creating lockedclaimcampaigns and defines the lockup plan. This struct is used for when tokens are claimed it will lookup this struct and is used to create the lockup plan.
tokenLocker is the address of the TokenLockup or TokenVesting plans contract that will lock the tokens
start is the start date when the unlock / vesting begins
cliff is the single cliff date for unlocking and vesting plans, when all tokens prior to the cliff remained locked and unvested
period is the amount of seconds in each discrete period.
periods is the total number of periods that the tokens will be locked or vested for

```
struct Campaign {
    address manager;
    address token;
    uint256 amount;
    uint256 start;
    uint256 end;
    TokenLockup tokenLockup;
    bytes32 root;
    bool delegating;
  }
```

This struct is used generally to define the Campaign of any claim campaign.
manager is the address of the campaign manager who is in charge of cancelling the campaign - AND if the campaign is setup for vesting, this address will be used as the vestingAdmin wallet for all of the vesting plans created. The manager is typically the msg.sender wallet, but can be defined as something else in case.
token is the address of the token to be claimed by the wallets, which is pulled into the contract during the campaign
amount is the total amount of tokens left in the Campaign. this starts out as the entire amount in the campaign, and gets reduced each time a claim is made
start is the start time of the campaign when ppl can begin claiming their tokens
end is a unix time that can be used as a safety mechanism to put a hard end date for a campaign, this can also be far far in the future to effectively be forever claims
tokenLockup is the enum (uint8) that describes how and if the tokens will be locked or vesting when they are claimed. If set to unlocked, claimants will just get the tokens, but if they are Locked / vesting, they will receive the NFT Tokenlockup plan or vesting plan
root is the root of the merkle tree used for the claims.
delegating is a boolean defining whether the claims need to be delegated when claimed or not

```
struct SignatureParams {
   uint256 nonce;
   uint256 expiry;
   uint8 v;
   bytes32 r;
   bytes32 s;
 }
```
This struct is used for the claim by signature functions.

Global Variables

mapping(bytes16 => Campaign) public campaigns: Mapping of the UUID to a specific claim campaign

mapping(bytes16 => ClaimLockup) public claimLockups: Mapping of the UUID of locked claim campaigns to the claim lockup struct

mapping(bytes16 => bool) public usedIds: Mapping of UUIDs that have been used - prevents ability of duplication

mapping(bytes16 => mapping(address => bool)) public claimed: Mapping of UUID to address to boolean to store if a wallet has claimed tokens from a specific campaign. Turned to true after claiming, and is checked that is false before a claim can happen.

mapping(bytes16 => address) private _vestingAdmins: Mapping of the UUID to vesting admins which may be different from the campaign manager - only used for vesting version of campaigns

State Changing Functions

function createUnlockedCampaign(bytes16 id, Campaign memory campaign, uint256 totalClaimers) external
Function to create an unlocked claim campaign. This function will physically pull tokens from the msg.sender into the contract, and creat the claim campaign and store its data in storage, mapped to the id. It will check that the id has not been used before. If delegation is turned on to true, it will check that the token is an ERC20Votes standard so that claimers will not have a reverted error when claiming and delegating. totalClaimers is specifically for event tracking.

function createLockedCampaign(bytes16 id, Campaign memory campaign, ClaimLockup memory claimLockup, address vestingAdmin, uint256 totalClaimers) external
Function to create a locked claim campaign, where tokens are send to a lockup or vesting contract, and the recipient receives the vesting lockup NFT instead of liquid tokens. This function checks that the id is not already in use or has been used. Total claimers is specifically for event tracking as it is not possible to ascertain this data strictly by the merkle tree root.

function cancelCampaign(bytes16 campaignId) external
Function to cancel an existing campaign by the campaign manager. This function can be called at anytime, even if the campaign has not ended. It will return any unclaimed tokens back to the manager and delete the campaign and associated lockup details from storage.

function claim(bytes16 campaignId, bytes32[] calldata proof, uint256 claimAmount) external
Primary function for wallets to claim directly. They need the campaign ID, proof and amount they are allocated. This is made simplest using a Hedgey UI dApp, as otherwise it can be difficult to know this information and perform the claim. The claim can be for either locked or unlocked tokens, but does not delegate anything. It will check if the campaign is a locked, vesting or unlocked version, and then internally process the claim according to the enum type. If the tokens are unlocked the claimer will receive the tokens directly from the contract. If the tokens are locked or vesting, the function will transfer tokens to the vesting or lockup contract, create a new vesting / lockup plan, where the claimer is the recipient and beneficiary of the vesting / lockup plan.

function claimMultiple(bytes16[] calldata campaignIds, bytes32[][] calldata proofs, uint256[] calldata claimAmounts) external
This function is used by a wallet to claim tokens from multiple campaigns at the same time. This is useful for some DAOs that intend to transfer some tokens that are immediately and initially unlocked at TGE and then a second batch of tokens that will unlock over a set period of time. This gives the claimer ability to claim both in a single transaction. It handles the internal logic of claiming locked vs unlock tokens based on the campaign ids.

function claimWithSig(bytes16 campaignId, bytes32[] calldata proof, address claimer, uint256 claimAmount, SignatureParams memory claimSignature) external
This function allows a paymaster or wallet to claim tokens on behalf of another wallet, using the EIP712 signature pattern. The function checks that the signature matches the claimer signature, and then processes the transaction based on the claim campaign details delivering tokens to the claimer instead of the msg.sender.

function claimMultipleWithSig(bytes16[] calldata campaignIds, bytes32[][] calldata proofs, address claimer, uint256[] calldata claimAmounts, SignatureParams memory claimSignature) external
This function allows a paymaster or account to claim multiple claims on behalf of another wallet. In this case the EIP712 signature check only checks that the first array of signatures matches the claimer wallet and then processes them all, as the claimer wallet is the same for all of the claims, it cannot be multiple different claimer addresses.

function claimAndDelegate(bytes16 campaignId, bytes32[] memory proof, uint256 claimAmount, address delegatee, SignatureParams memory delegationSignature) external
Function to claim and delegate tokens in a single transaction. This function must be used when the claim campaign delegation is set to true. This will claim the liquid tokens or locked tokens and within the same transaction delegate the tokens to the delegatee. It will check that the delegation has actually been processed correctly. If the tokens claimed are unlocked, it will use the delegationSignature to call the

delegateBySig function on ERC20Votes to perform the delegation. If the tokens are locked / vesting, this signature params can be ignored as the delegation is done by the lockup / vesting contract natively without a delegation sig requirement.

function claimAndDelegateWithSig(bytes16 campaignId, bytes32[] memory proof, address claimer, uint256 claimAmount, SignatureParams memory claimSignature, address delegatee, SignatureParams memory delegationSignature) external
This function allows a paymaster or another account to claim and delegate tokens on behalf of another user, where they have both the delegation signature and claim signature using the EIP712 standard. This follows a similar pattern as the claimAndDelegate function, having performed the check that the claimer has given authorization via signature to the msg.sender to claim on their behalf.

View Functions

function verify(bytes32 root, bytes32[] memory proof, address claimer, uint256 amount) public pure returns (bool)
This function uses the open zeppelin merkle proof standard contract to check if a root, proof, address and amount are in a merkle tree. It takes all of these properties to verify and validate, and returns true. This is a core function that is used by all claim functions to prove that the claimer can in fact claim the tokens.