Universal Rewards Distributor (URD)

The Universal Rewards Distributor (URD), is a smart contract allowing the distribution of multiple ERC20 tokens from a single offchain computed Merkle tree.

Full repository here .

Each URD contract has an owner and a group of updaters (chosen by the owner). Values submitted by updaters are timelocked and can be revoked by the owner or overriden by another updater. However, this timelock can be set to 0 if the URD owner does not need it.

Use Case Example

Assume the Owner is a DAO with a periodic rewards mechanism. Each month, a Gelato bot runs a script to create a Merkle tree that distributes TokenA and TokenB.

During the setup, the DAO configures the timelock based on the risk of updater corruption, let's say 3 days, and adds a Gelato bot as an updater. If the DAO already has a distribution at the time of the URD deployment, it can define an initial root, or use an empty root if not.

Each month, the Gelato bot proposes a new root. For 3 days, the DAO has the opportunity to run checks on this root. After these 3 days, if the DAO did not revoke the root, anyone can accept this value.

The DAO must transfer the correct amount of tokens to the URD to allow all claimants to claim their rewards. If the DAO does not provide enough funds, the erc20 transfer will fail.

Attaching an IPFS Hash

- Using a Merkle tree delegates all root computation offchain, leaving no information about the tree onchain (except for the root). This makes it challenging for an integrator (or a claimer) to understand the Merkle tree or to know which proof to use. To address this, each root can be linked to an IPFS hash, which can link to any data. We recommend that all users follow the same format for the IPFS hash to facilitate integration. The suggested format is as follows:

{ "id": "A string id of the Merkle tree, can be random (you can use the root)", "metadata": { "info": "a key value mapping allowing you to add information" }, "root": "The merkle root of the tree", "tree": [{ "account": "The address of the claimer", "reward": "The address of the reward token", "claimable": "The claimable amount as a big number string", "proof": ["0x1...", "0x2...", "...", "0xN..."] }] } * We recommend not including spaces or new lines when uploading to IPFS, to ensure a consistent method of uploading your JSON file. * We also recommend sorting the tree by the account address and the reward token address, to ensure a consistent order.

Owner Specifications

- The URD is an owner-managed contract, meaning the owner has full control over its distribution. Specifically, the owner can bypass all timelocked functions, modify the timelock, add or remove updaters, and revoke the pending value at their discretion.
- If the owner is set to the zero address, the contract becomes ownerless. In this scenario, only updaters can submit a root. Furthermore, thetimelock
- value becomes unchangeable, and pending values can still be overridden by updaters.
- If there are neither owner nor updaters, the URD becomes trustless. This is useful for creating a one-time distribution that must remain unchanged. This can be accomplished at the contract's creation by providing only a Merkle root and an optional IPFS hash. After this, the contract's sole functionality is to claim rewards.
- It is possible to create a URD with no root, owner, or updaters. While this might seem pointless, it is the URD creator's responsibility to configure the URD correctly.

Updaters Specifications

- Multiple updaters can be defined for a single URD. The owner can add or remove updaters at any time, instantly, without a timelock.
- All updaters share a single pending value. This means they can override the pending value (if any) at any time.
- If a pending value is not yet used as the main root, any submissions by updaters will override the pending value.

Having multiple updaters can lead to situations where the pending values are subject to multiple concurrent propositions. The owner must manage this scenario to maintain the URD's functionality. We recommend not having different reward distribution strategies in a single URD. All updaters should agree on a distribution mechanism and a root. The use case for having multiple updaters is to enhance resilience and security.

Considerations for Merkle Tree

- The claimable amount for a given user must always exceed the amount provided in the previous Merkle tree. If a claimer has claimed an amount higher than theclaimable
- amount in the Merkle tree (if claimed from a previous root), the transaction will revert with an underflow error.
- We recommend merging all the (reward, user) pairs into a single leaf. If you wish to have two different leaves for one (reward, user) pair, the user will be able to claim the larger amount from the two leaves, not the sum of the two.
- Merkle trees can be generated withOpenzeppelin library
- .
- The leaves of the merkle tree have to be hashed twice. It is natively supported by the Openzeppelin library.
- The URD supports empty root. This means that, at any time, updaters or owner can submit a 0 hash root. It can be used to deprecate the URD.

Claim Rewards

- The arguments for the claim function must be provided off-chain. If an IPFS hash is given, the IPFS content likely includes the parameters.
- Anybody can make a claim on behalf of someone else. This means you can claim for multiple users in one transaction by using a multicall.
- An update to the root can increase the claimable amount and modify the proof, even if the claimable amount remains unchanged.
- A call to the claim function will claim only one token. If you wish to claim multiple tokens, use a multicall.

The factory

You can create a URD using a factory. This factory simplifies the indexing of the URD offchain and validates any URDs created through it. While its use is optional, it offers a convenient alternative to directly deploying a URD by submitting the bytecode.

Skim non claimed rewards

A rewards program can have a deadline for users to claim tokens. After this deadline, the owner can skim the rewards that were not claimed. To do so, the owner has to create a Merkle tree that is sending the rewards to the owner. Then, the owner can submit this Merkle tree to the URD. The URD will then allow the owner to claim the rewards that were not claimed by the users.

Limitations

The pending root is not a queue

The pending root does not have a queue mechanism. As a result, when a root updater pushes a root to the pending root in a distribution with a timelock, it effectively erases the previous root. This implies that a compromised root updater can continuously suggest a root, resetting the timelock of the pending root.

Furthermore, if the pending root is ready to be accepted but a root updater suggests a new root simultaneously, the pending root is erased and the timelock restarts. This situation can lead to endless loops of the pending root, especially if an automation mechanism suggests a root at intervals shorter than the timelock.

This behavior is acknowledged and should be considered when designing a strategy using the URD. Ensure the epoch interval for updating the root is longer than the timelock, and define these parameters accordingly.

An alternative solution is to build a queue mechanism on top of the URD with a 0 timelock distribution. Here, the root updater could be a queue designed as theDelay Modifier of Zodiac .

Getting Started

Installation

yarn cp .env.example .env

Development

Running tests requires forge fromFoundry .

yarn test

Deployment

- Add the desired network key and its corresponding RPC url tofoundry.toml
- yarn deploy {network} --broadcast --sender {sender}
- followed with appropriate private key management parameters

[!NOTE] If the provided network's RPC url uses a variable environment (such asALCHEMY_KEY), it should be defined in your.env For example:

yarn deploy goerli --broadcast --ledger --sender 0x7Ef4174aFdF4514F556439fa2822212278151Db6 [!NOTE] Broadcast run logs are to be committed to this repository for future reference.

Etherscan verification

yarn verify --watch --chain-id {chainid} --etherscan-api-key {key} {address} src/UrdFactory.sol:UrdFactory

Audits

All audits are stored in theaudits ' folder.

License

The URD is licensed underGPL-2.0-or-later , seeLICENSE .

Previous Addresses Next Track Rewards Emission