Chainlink Rewards Contracts Technical Design

Motivation

Chainlink Rewards (CLR) is a community engagement and rewards program designed to incentivize active participation in the Chainlink Network. The contracts in this document are designed to allow projects in the Chainlink Build program (CLR projects) to deploy an on-chain claim mechanism of their tokens for Chainlink ecosystem participants.

Design

NOTE: This design includes additional optional functionality that may or may not be used at launch. Documentation and code also contains parameter config values. These config values are for illustration purposes only to explain how the code executes through examples. Offchain mechanism design may be subject to change. The scope of this audit review is focused on the onchain smart contract logic.

Glossary

- **CLR**: A shorthand for Chainlink Rewards
- **Build**: A program that accelerates the growth of early-stage and established projects in the Chainlink ecosystem by providing enhanced access to Chainlink services and technical support. Chainlink Rewards enables Chainlink Build projects to make their native tokens claimable by Chainlink ecosystem participants.
- **User**: The participants of the CLR program. This includes Chainlink ecosystem participants.
- **Contract Manager**: The user with the role to modify certain CLR parameters and allowlist projects.
- **CLR Project**: The dApps that participate in the CLR program. Also referred to as projects.
- **CLR Project Admin**: The admin of the allowlisted CLR Project who will have permission to deploy their own claim contracts and fund tokens into them.
- **CLR Project Token**: The tokens created and launched by the CLR Projects. Projects make a portion of their tokens available to the CLR program, possibly across multiple seasons, and users who allocate their credits to a project will be able to claim the project's tokens.
- **Season**: A period during which a number of selected CLR project tokens are made available to ecosystem participants. A project's tokens can be split across multiple seasons.
- **Credits**: Offchain calculation based on allocation of ecosystem participants.
- **Allocation Period**: The period during which ecosystem participants can allocate credits to the CLR projects.
- **Unlock Period**: The period during which ecosystem participants can start the claim process for their CLR project tokens. The total amount claimable is based on the allocations made during the allocation period, and tokens become claimable during the unlock period.

High-level Architecture

A season goes through different states in the following order:

1. Allocation Period
2. Unlock Period
3. Post-season
4. Refunding

In Allocation Period,

- Contract Manager deploys the BUILDFactory contract (if it hasn't been deployed or needs an upgrade) and sets the season's global unlock start time.
- Contract Manager allowlists CLR Projects.
- CLR Project Admin deploys the BUILDClaim contract (if it hasn't been deployed or needs an upgrade) through the BUILDFactory.
- CLR Project Admin deposits the Project tokens into their BUILDClaim contract. They can deposit tokens in advance and use only part of the deposits for each season.
- Contract Manager can cancel the season before the unlock period starts.
- Users can allocate credits to one or more projects through an off-chain mechanism.
- At any given time, there will only be one active allocation period within the CLR program.
- Once the allocation period ends, Contract Manager calculates for each project how many tokens each user can claim, according to credit allocations. They will create a merkle tree with the user address and claimable amount tuples as the leaves.
- Contract Manager sets project-specific configs, such as the total token amount to be unlocked in the season, unlock delay, unlock duration, percentage of the instantly unlocked tokens, and the merkle root for allowlisting users and their claimable amounts.
    - It is possible that a project doesn't participate in all seasons continuously. In this case, the project will remain allowlisted but the season config for the project will not be set. The project will not be available for credit allocation in that season.

In Unlock Period,

- Each CLR Project unlocks its tokens to the users who have allocated credits to it.
- Depending on the project's unlock delay configuration, users may need to wait until they can start claiming the unlocked tokens.
- Each project may have a different unlock delay and duration.
- Since different projects may have different unlock durations, there may be multiple seasons in the unlock period at the same time.

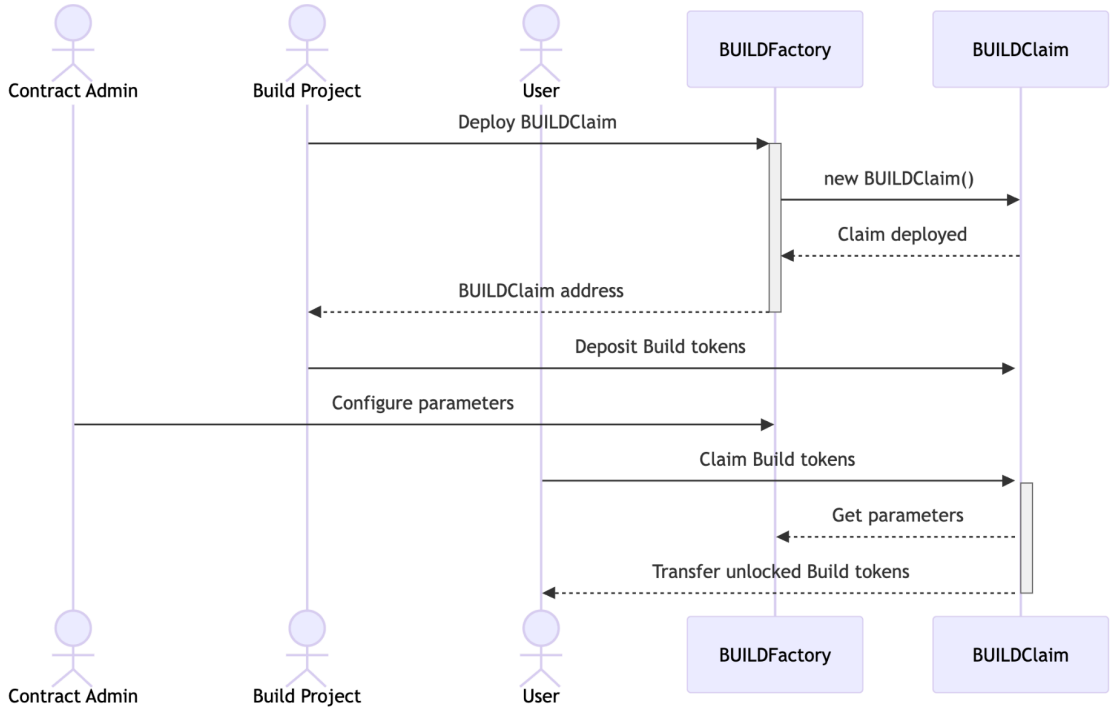
In Post-Season,

- The token unlock period ends for the specific CLR Project.
- Users can keep claiming the project tokens.
- Contract Manager can start the refunds for a CLR Project.

In Refunding State, (Optional)

- Users who haven't claimed any rewards during the season for a project won't be able to claim the remaining unlocked project tokens.
- Users who have claimed any rewards during the season for a project can continue to claim the remaining unlocked project tokens.
- The transition to a refunding state is irreversible.

Below is a sequence diagram of the general flow of a season:



Unlock Mechanism

Tokens can be rewarded to the users in 3 different manners:

- Instant-claim (Base tokens)
  - Users can claim the instant-claim tokens as soon as the unlock delay is over. Each user's amount is proportional to their share of the allocations to the project, $instant\ claim\ token\ amount\ *\ \frac{user\ allocated\ amount}{total\ allocated\ amount}$.
- Unlock (Part of bonus tokens)
  - After the unlock delay (which can also be 0), the token can be linearly unlocked over a project-specific unlock period.

    - Users get their share of the unlock tokens proportional to their allocated credits, i.e., $bonus\ vested\ token\ amount\ *\ \frac{user\ allocated\ amount}{total\ allocated\ amount}$.
- Unlock early vest (Part of bonus tokens)
    - Users can one time only “early vest” and receive a portion of their unlocked tokens. They receive an amount that is proportional to the time remaining in a linear increasing band, ie

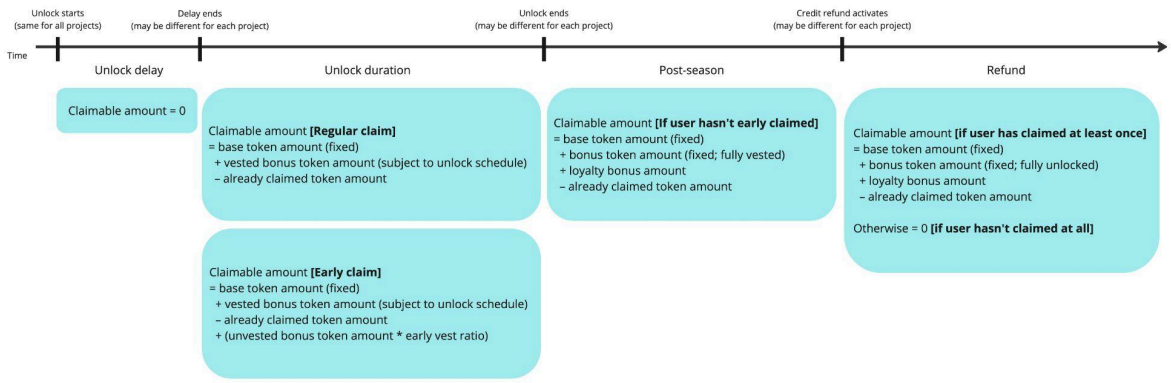
$\mathrm{minRatio}$: the **minimum** Early Vest Ratio at $t = 0$.

$\mathrm{maxRatio}$: the **maximum** Early Vest Ratio at $t = T$.

$$R(t) = \mathrm{minRatio} + \left(\mathrm{maxRatio} - \mathrm{minRatio}\right) \times \frac{t}{T}.$$

$$bonus\ unvested\ token\ amount\ *\ R(t)\ *\ \frac{user\ allocated\ amount}{total\ allocated\ amount}$$

The diagram below illustrates all potential different stages and claimable amounts during those stages.



Seasons support linear unlock schedules with the following formula:

$$Claimable\ tokens\ =\ bonus\ tokens\ *\ \frac{block.timestamp - (\ time_{unlockStartsAt} + unlock\ delay)}{time_{unlockEndsAt} - (time_{unlockStartsAt} + unlock\ delay)} + base\ tokens$$

(Note: the formula is simplified and does not check if the unlock ended, loyalty tokens after unlock ended, etc.)

As the first part of the formula shows, the unlock schedule applies to the bonus tokens.
The base tokens are “unlocked” immediately once the unlock delay is passed.

The unlock start timestamp is the same for all Projects (BUILDClaims) for a season, whereas the unlock end timestamp, which equals unlock start time + unlock delay + unlock duration, and the total tokens may vary for different Projects.

Each user's total token amount per project (= base tokens + bonus tokens) is calculated offchain and put into a merkle tree. The roots of the merkle trees are set in the BUILDFactory, and when users claim from one or more projects, they get their merkle proofs and the total token amounts from the Backend and use them as arguments to the claim functions. The claim functions verify the proofs, calculate the claimable tokens, and transfer the amounts to the users.

Early Vesting and Loyalty Token Redistribution (Optional)

This is an optional functionality that allows users to claim their tokens without waiting for the unlock duration, but only a portion of their unvested tokens can be claimed as part of early vest. The claimable portion, a.k.a. the early vest ratio, is a linear function of time that has an inverse relationship with the time elapsed during the unlock period, with the maximum and minimum ratios defined as project config per season. When claiming with the isEarlyClaim flag on, users receive a portion of their unvested tokens, calculated as $unvested\ bonus\ tokens\ *\ early\ vest\ ratio$, as well as their vested tokens (as defined in the Unlock mechanism section above). Once early claimed, the user's unclaimed tokens do not unlock, and the user cannot execute any more early or regular claims for the project's season.

The unclaimed tokens go into a loyalty pool within the same BUILDClaim contract and are tracked per season. Once the unlock duration ends, users who haven't early vested get rewarded with the loyalty tokens in proportion to their credits, in addition to their maxTokenAmount. The loyalty tokens are calculated as follows:

$$User's\ loyalty\ tokens\ =\ Total\ loyalty\ tokens\ *\ User\ credits/\ Total\ credits\ of\ those\ who\ didn't\ early\ vest$$

Refunding (Optional)

Refunding is an optional phase that can occur after the post season state has ended and the project admin starts the refund phase for a season. In this state, users can no longer claim the tokens and their share of project tokens will be released back so that the admin can either withdraw or allocate them to future seasons. If there are any loyalty tokens that the refund-eligible users waive, those will also be released back to the admin. If a user has claimed at least once throughout the season, the user is not eligible for a refund and instead can continue claiming any remaining unlocked and/or loyalty tokens.

Batch claims (Optional)

Batching claims across seasons is an optional functionality and can be done by using an array of ClaimParams as an argument to the claim function. Batching across projects means the user needs to use an external forwarder contract, such as Multicall3 if it's an EOA or multiSend if it's a Gnosis Safe multisig wallet. When the multicall3 is used, the `msg.sender` is the multicall3 contract, rather than the user's EOA. We do not allow early claims in this case, in order to reduce the risk of a user being tricked by a malicious actor into claiming less than the full amount and increasing the loyalty token pool. In addition, we'll use different proofs for regular and early claims so that even if a user regular claims on-chain and the proofs become publicly available, the early claim proofs are not and can't be used by such attackers.

Delegation

Users can delegate their rights to claim to another wallet and use the delegated-to wallet to claim. We plan to use the DelegateRegistry contract built by delegate.xyz to verify that the `msg.sender` is a valid delegate of the claimant, `user`, only when the ClaimParams include an early claim (isEarlyClaim = true). As we do not check on-chain the identity of the `msg.sender` when it's a regular claim, we do not check the delegation status either. Only the ToS-signed valid users or their delegates will be able to access the valid merkle proofs on the Frontend.

System Invariants

Access Controls

- After funding a Claim contract, CLR projects should not be able to halt, exit, or do anything harmful to the system later without the admin's permission.
- Only Contract Admin can update configs, allowlist CLR Project Admins (Project Admins), start a refund, and schedule or cancel withdrawals.
- Only Project Admins can execute a scheduled withdrawal.

Project Management

- Only allowlisted actors can deploy Claim contracts.

- Claim contracts can only be deployed and configured properly through the Factory contract.

Claiming and Vesting

- Users can only claim unlocked tokens or if enabled, early claim a portion of the unlocked tokens.
- Tokens vest according to the schedule (unlock duration, delay, etc.)
- CLR projects can withdraw only the allowed/available amount.
- Token amount validation calculation does not under/overflow.

General

- Contracts can’t be bricked.

Operations

Allowlisting CLR Projects

The Contract Manager can add or remove CLR Projects to the allowlist by calling `addProjects` and `removeProjects` functions on the BUILDFactory.

Deploying the claim contracts

The CLR Projects can deploy BUILDClaim contracts through the `deployClaim` function of the BUILDFactory.

Depositing CLR project tokens

The CLR Projects can deposit the project tokens to their BUILDClaim contracts by calling the `deposit` function on their claim contracts.

Prerequisites for this operation are:

- CLR Project must be allowlisted
- CLR Project Admin must have deployed the BUILDClaim contract through the BUILDFactory

- CLR Project Admin must have enough project tokens and have increased their token approval for the claim contract enough so the contract can transfer from the admin to itself

Configuring seasons

The Contract Manager can configure a season's parameters by calling the `setSeasonUnlockStartTime` and `setProjectSeasonConfig` functions on the BUILDFactory.

The order of operations will be:

1. Contract Manager calls `setSeasonUnlockStartTime` on the BUILDFactory, as the season must be created first to configure project-specific season parameters)
2. Contract Manager calls `setProjectSeasonConfig` on the BUILDFactory for each project participating in the season

Starting refunds

The Contract Manager can start the refund process for a specific project's season by calling the `startRefund` function on the BUILDFactory.

Withdrawing CLR project tokens

The CLR Projects can withdraw the deposited tokens not assigned to a season by calling the `withdraw` function on their BUILDClaim contract.

Prerequisites for this operation are:

- The withdrawal needs to be approved and scheduled by the Contract Manager.
- There must be non-zero leftover tokens that haven't been assigned to a season or have been released because of the refund process.

The order of operations will be:

1. Contract Manager calls `scheduleWithdraw` on the BUILDFactory contract, specifying the project's token address, the withdrawal recipient, and the amount.
2. CLR Project Admin calls `withdraw` on the BUILDClaim contract.

Upgrading contracts

We expect two general scenarios where contract upgrades are needed. One is when a new feature is being introduced, and another is when an error in the contracts is discovered. In either case, the Contract Manager and Projects will need to upgrade all the contracts, including the

BUILDFactory and the BUILDClaim. An exception would be when a Project's admin wallet has been compromised or they have become malicious, in which case we will treat the claim contracts of those affected projects individually.

Regular upgrades for all (new features)

The upgrade only affects future seasons. Projects will be allowed to transfer tokens that haven't been allocated to any of the existing seasons and deposit them into new BUILDClaim contracts. It is also possible for a Project to not deploy a new version and keep using its old BUILDClaim contract.

The order of operations will be:

1. Contract Manager deploys a new BUILDFactory and configures it by calling `setSeasonUnlockStartTime` and `addProjects`
2. Each Project deploys a new BUILDClaim through the `deployClaim` on the new BUILDFactory
3. Contract Manager sets the new BUILDClaim contract as the withdrawal recipient for each old BUILDClaim by calling `scheduleWithdraw` on the old BUILDFactory
4. Each Project transfers tokens to their new BUILDClaim with the `withdraw` call.
    a. Since the claim contracts are not paused, Projects will only be able to withdraw the tokens that have not yet been assigned to a season. If there are refunds from previous seasons, those are available for withdrawal as well.
5. Contract Manager closes the old BUILDFactory by calling `close` on the contract.

Emergency upgrades for all

The contracts must be paused immediately and there may be one or more seasons that are in their unlock period. In this case, the Projects will be allowed to migrate the unallocated tokens as well as allocated but locked tokens to the new contract.

The order of operations will be:

1. Contract Manager pauses the BUILDFactory and all BUILDClaim contracts by calling `emergencyPause` on the BUILDFactory
2. Contract Manager deploys a new BUILDFactory and configures it by calling `setSeasonUnlockStartTime` and `addProjects`
3. Each Project deploys a new BUILDClaim by calling `deployClaim` on the BUILDFactory
4. If it's been determined that the error is not in the BUILDClaim contracts and the claim logic isn't at risk, Contract Manager will close the BUILDFactory through the `close` function and unpause the BUILDClaim contracts by calling `emergencyUnpause`, so that users can claim their unlocked tokens.

    a. Otherwise, the BUILDClaim contracts will remain paused.
5. Contract Manager sets the new BUILDClaim contract as the withdrawal recipient on each old BUILDClaim by calling `scheduleWithdraw` on the old BUILDFactory
6. Each Project transfers tokens to their new BUILDClaim with the `withdraw` call.
    a. If the old claim contracts are still emergency paused, it allows withdrawing all remaining tokens in the contracts, but if they are unpaused, it only allows up to the unallocated and refunded amounts.

Emergency upgrade/pause for individual projects

Contract Manager will pause the claim contract for the specific CLR project by calling `pauseClaimContract` on the BUILDFactory and perform any additional operations as needed, which will depend on the situation.

In general, the order of operations will be:

1. Contract Manager pauses the BUILDClaim contract(s) for the project(s) by calling `pauseClaimContract` on the BUILDFactory
2. If it's been verified that the project can be re-added to the system, Contract Manager performs the following steps
    a. If the project needs to deploy a new token or change their admin wallet, call `addProjects` on the BUILDFactory with the new values
    b. If the issue isn't in their tokens or admin wallets, call `unpauseClaimContract` on the BUILDFactory

When the BUILDFactory is paused,

- Takes precedence over the individual claim contract pause/unpause, i.e., unpausing individual claim contracts is not possible
- Contract Manager cannot start the refund process
- Projects cannot deploy claim contracts
- Projects cannot deposit more tokens to the claim contracts
- Users cannot claim unlocked tokens

When a BUILDClaim is paused individually,

- Project cannot deposit more tokens to the specific claim contract
- Users cannot claim unlocked tokens from the particular claim contract
- Users can still claim unlocked tokens from other claim contracts that aren't paused

Once the BUILDFactory is closed,

- It cannot be reopened
- Contract Manager cannot add or remove projects

- Contract Manager cannot set the season configs (unlock config maximums, season unlock start times)
- Contract Manager cannot set the project-specific season configs
- Projects cannot deploy claim contracts
- Projects cannot deposit more tokens to the claim contracts
- Users can still claim unlocked tokens from claim contracts
- If there's an ongoing season when the contract is closed, it will continue with the unlock schedule