



# Introduction

## **Wallet and Bank**

Truly decentralized finance requires decentralized wallets and banks.

## **KeeperC**

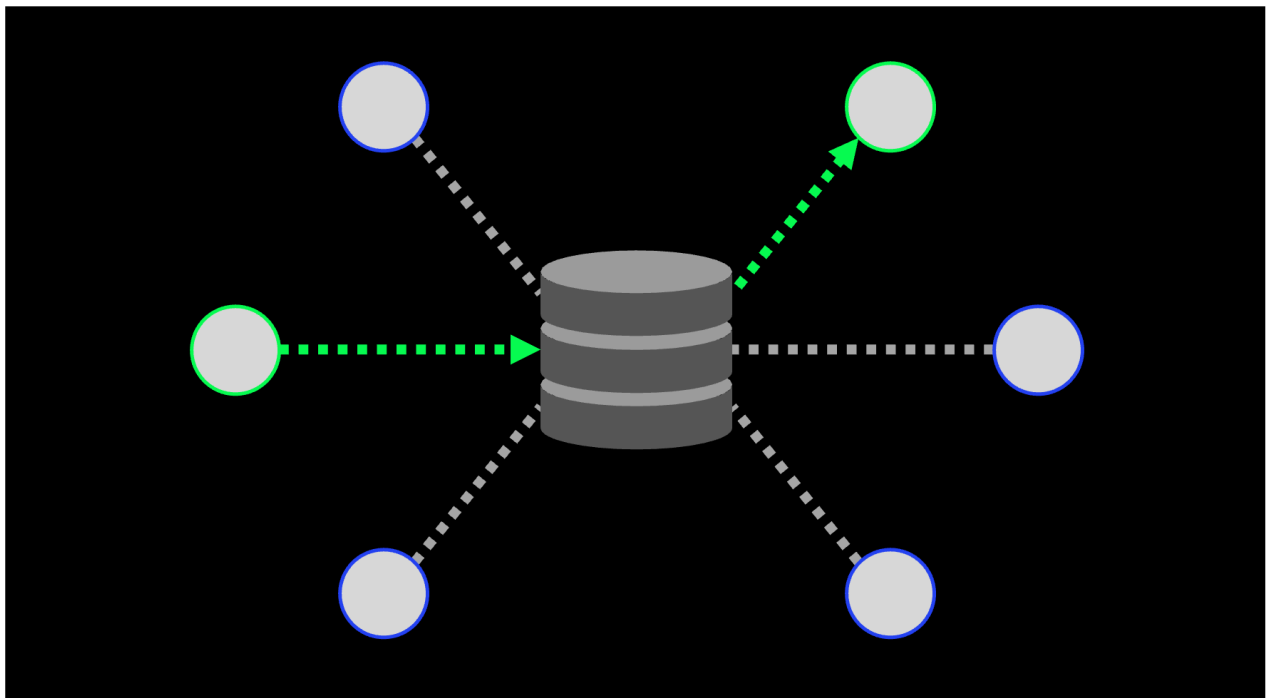
To provide security and convenience without compromising decentralization, we propose KeeperC, a contract wallet based on smart contracts, Chainlink Automation, and O...

# Wallet and Bank

Truly decentralized finance requires decentralized wallets and banks. While we're already familiar with decentralized, non-custodial wallets such as Metamask, what about **decentralized banks**?

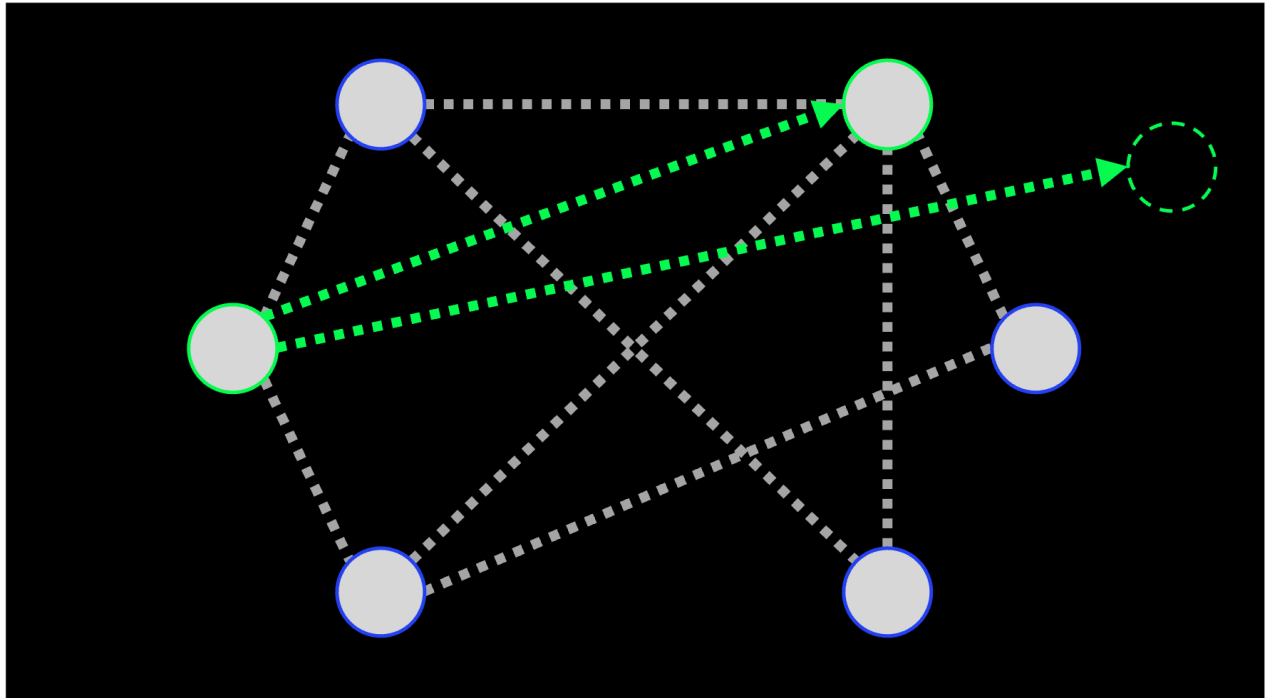
**FIG 1. CENTRALIZED**

- ✕



**FIG 2. DECENTRALIZED**

- X



Unfortunately, the blockchain should not have a centralized player.

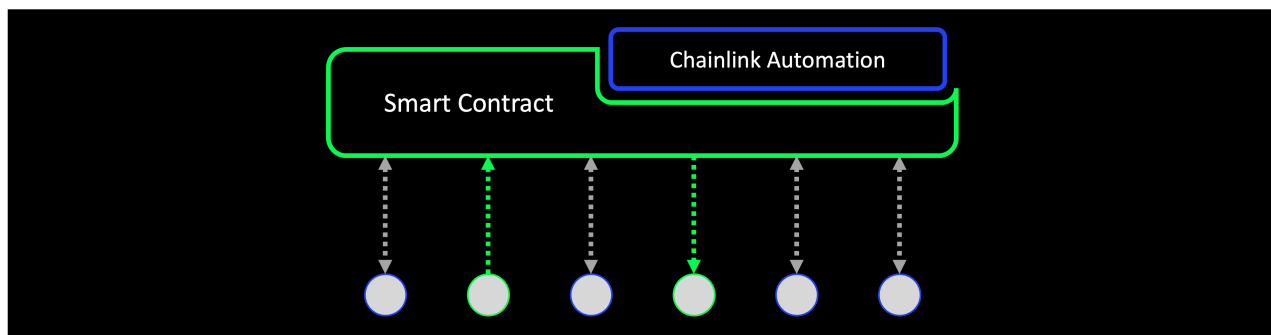
Therefore, blockchain users cannot enjoy the convenience of centralized institutions, such as scheduled transfer and mis-transferred asset recovery.

# KeeperC

To provide security and convenience without compromising decentralization, we propose KeeperC, a contract wallet based on smart contracts, Chainlink Automation, and OpenAI API.

FIG 3. KEEPERC20

- ✕




Currently, KeeperC offers the following powerful functions:

- **Scheduled Transfer**: You can schedule future transfers of your tokens, making it easier to manage your transactions.
- **Recoverable Transfer**: If you accidentally send your tokens to the wrong address, KeeperC can help you recover them.
- **Expirable Approve**: You can set a time limit for token approvals, increasing the safety of using dApps.

KeeperC includes a **Fraud Detection System** that monitors your transactions and alerts you if any suspicious activity is detected, protecting you against potential hacks and scams.

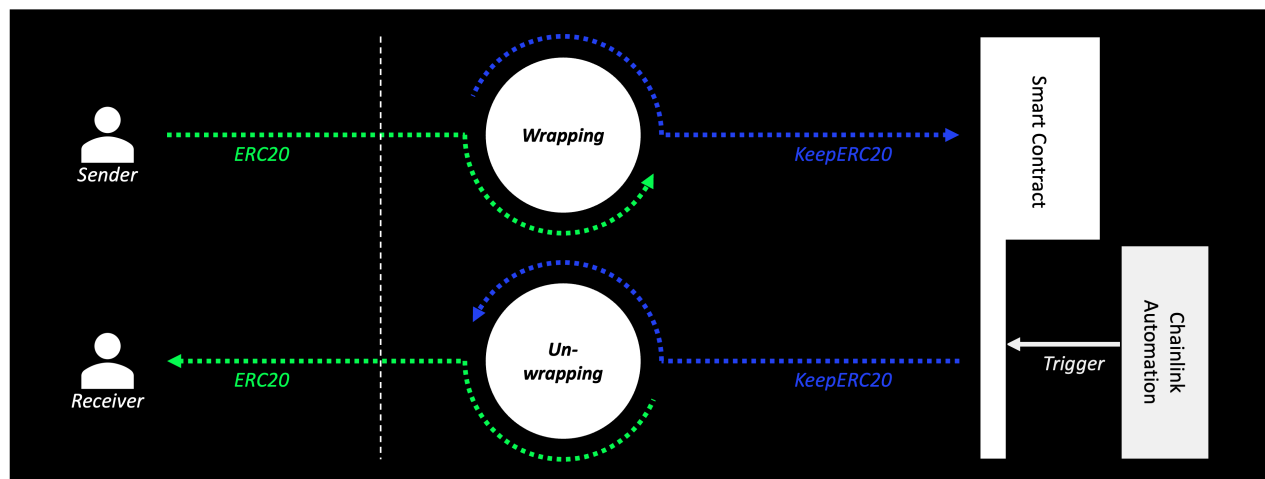
KeeperC comes with a **chatbot** that uses AI to provide you with natural language support and make it easier to manage your tokens. You can also use KeeperC's

chatbot via the popular messaging app, **Telegram**.

We will continuously add more convenient and secure functions to  KeeperC to ensure that your tokens remain safe and secure.

# Overview

FIG 4. OVERVIEW OF KEEPERC20



- When a user sends ERC20 tokens to KeeperC, they are wrapped into KeeperC tokens, allowing the user to take advantage of various features such as scheduled transfers, lost money recovery, expirable approval, and fraud detection through smart contracts and Chainlink Automation.
- When the recipient receives the tokens, they are automatically unwrapped back into the original ERC20 tokens, making it easy to use the tokens with other dApps.

However, in some cases, simple token wrapping may not be enough to perform certain tasks, such as ownership-related function calls.

To address these cases, KeeperC creates a safe contract wallet internally and provides these functions seamlessly to ensure the security and convenience of users' tokens.

With KeeperC, you can enjoy a secure and convenient way to manage your tokens while participating in the decentralized blockchain ecosystem.

# Functions

## Scheduled Transfer

The scheduled transfer is a function that automatically transfers ERC20 tokens after a few blocks. A series of bytes can be transferred together, so the contract call is also p...

## Recoverable Transfer

The recoverable transfer can be treated as a kind of insurance because users can use it to prepare for mispayment.

## Expirable Approve

ERC20's Approve is often the target of attack. The expirable-approve reduces the possibility of the hack by automatically canceling approval.

# Scheduled Transfer

The scheduled transfer is a function that automatically transfers ERC20 tokens after a few blocks. A series of bytes can be transferred together, so the contract call is also possible. It is monitored and managed through the Upkeep of Chainlink Automation.

A predefined fee is collected as an ERC20 token when requesting a scheduled transfer.



# Recoverable Transfer

The recoverable transfer can be treated as a kind of insurance because users can use it to prepare for mispayment. When the tokens were sent to an address where the private key didn't exist or sent to the wrong contract, there was no way to recover them before.

However, if you use KeeperC's recoverable transfer, you can get them back completely. Except for some fees.

The asset transfer through this function is finalized only when the receiver publishes unwrap transaction. If tokens are sent to the wrong address so no one can access them, the tokens will automatically return to the sender through Chainlink Automation after the expiration.

# Expirable Approve

ERC20's *Approve* is often the target of attack. The expirable-approve reduces the possibility of the hack by automatically canceling approval.

Since the *increase* and *decrease* in ERC20's *Allowance* is a function that only the token owner can call, a contract wallet is created internally to control it with the KeeperC contract and Chainlink automation.



# Fraud Detection

KeeperC comes with a fraud detection system that monitors your transactions and alerts you if any suspicious activity is detected. This feature helps protect you from potential hacks, scams, and phishing attacks.

Current fraud detection in KeeperC is denylist-based phishing address detection. This feature compares the address you are sending tokens to against a denylist of known phishing addresses. If a match is found, KeeperC will alert you and prevent the transaction from going through, protecting your tokens from being sent to a fraudulent address.

To ensure the highest level of security, KeeperC's denylist is constantly updated to include the latest phishing addresses. Additionally, if you encounter a new phishing address that is not on the denylist, you can report it to [KeeperC's support team](#), and they will investigate and add it to the denylist if necessary.

# Chatbot

## Integrated Chatbot

KeeperC's chatbot is integrated into both the web app and the extension to provide users with convenient natural language support.

## Telegram Bot

In addition to the chatbot integrated into the web app and extension, KeeperC also provides a Telegram bot (@keeperc\_bot) that allows you to use the chatbot via the popul...

# Integrated Chatbot

KeeperC's chatbot is integrated into both the web app and the extension to provide users with convenient natural language support. If you need help with anything related to KeeperC, simply type your query into the chatbot, and it will suggest features and functions on KeeperC that you can use.

For example, if you're worried about losing your money by accidentally sending it to the wrong address, you can ask the chatbot for help. The chatbot will recommend the "Recoverable Transfer" function of KeeperC, which allows you to recover your tokens if you accidentally send them to the wrong address. The chatbot will also provide a brief explanation of how the function works and why it's a good option for you.

# Telegram Bot

In addition to the chatbot integrated into the web app and extension, KeeperC also provides a Telegram bot (@keeperc\_bot) that allows you to use the chatbot via the popular messaging app. You can join KeeperC's Telegram group (<https://t.me/+8l6TVyQhoSI4YWJl>) to access the KeeperC chatbot and connect with other users to discuss the latest updates and features of the wallet.

The bot works in the same way as the chatbot in the web app and extension, providing natural language support and suggesting features and functions on KeeperC that you can use.

# Live Demo

## Web App

KeeperC is a contract wallet that allows users to participate in the system with any ERC20 token. However, for test purposes, KeeperC currently only supports the TERC20 to...

## Chrome Extension

KeeperC is a contract wallet that allows users to participate in the system with any ERC20 token. However, for test purposes, KeeperC currently only supports the TERC20 to...

## Information

The KeeperC is a work-in-progress and is provided as-is, without any warranties or guarantees of its effectiveness. Its use is entirely at your own risk. Use the web app and e...

# Web App

## ⚠ CAUTION

KeeperC is a contract wallet that allows users to participate in the system with any ERC20 token. However, for test purposes, KeeperC currently only supports the `TERC20` token.

*Mumbai testnet is used for a concrete example.*

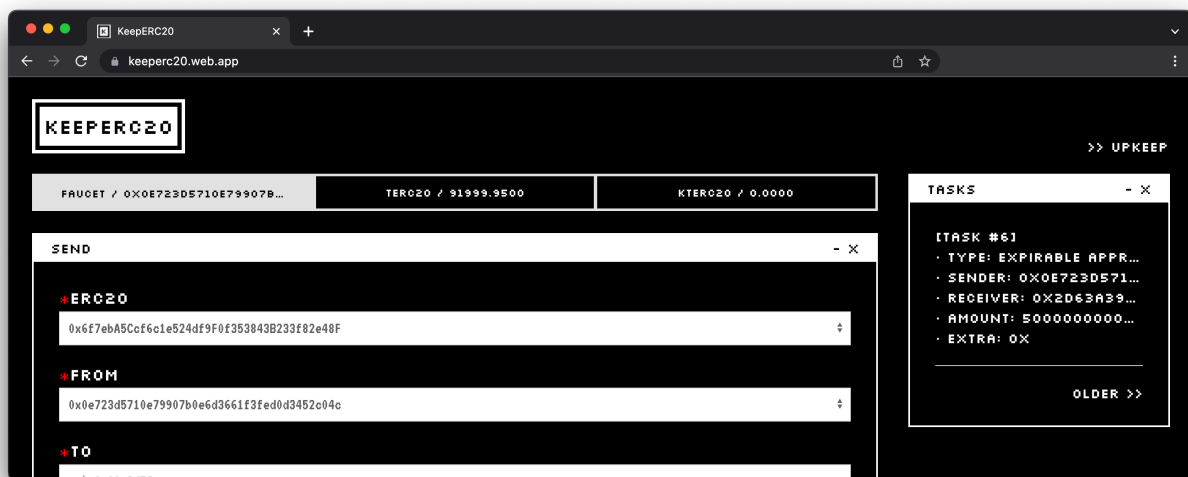
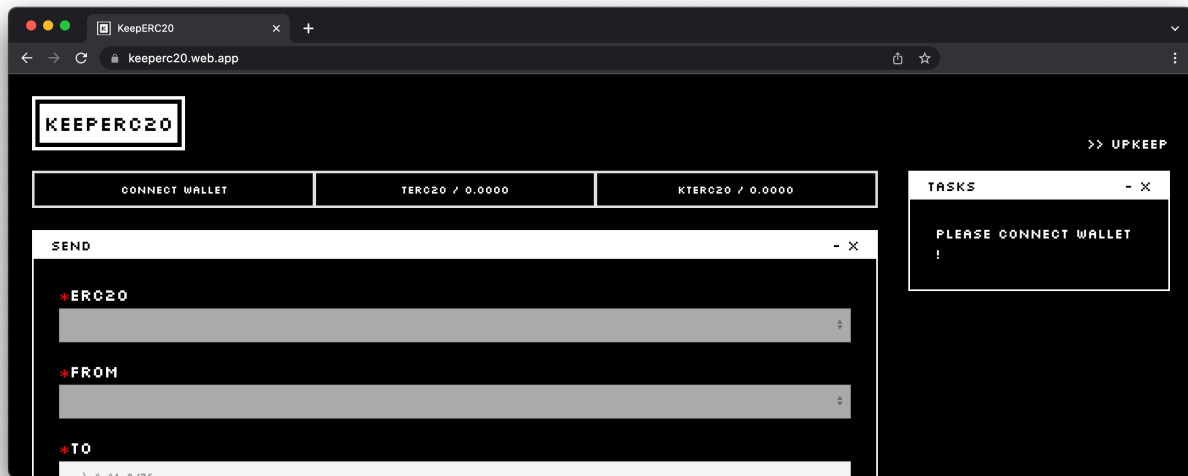
- 🔗 [WebApp](#)
- ✉ Feedback: [lukepark327@gmail.com](mailto:lukepark327@gmail.com)

---

## Faucet

The faucet for Test ERC20 (TERC20) tokens is available! First, click the "Connect Wallet" button. Then, click the "Faucet" button appearing at the same location.





# Features

You can check your transaction history in the TASKS window.

[Live Demo](#)[Chrome Extension](#)

# Chrome Extension

## CAUTION

KeeperC is a contract wallet that allows users to participate in the system with any ERC20 token. However, for test purposes, KeeperC currently only supports the `TERC20` token.

*Mumbai testnet is used for a concrete example.*

-  [Download \(Code\)](#)
-  Feedback: [lukepark327@gmail.com](mailto:lukepark327@gmail.com)

---

## Faucet

The faucet for Test ERC20 (TERC20) tokens is available! First, please login. Then, click the "TERC20 / <YOUR\_BALANCE>" button to receive 100 TERC20 tokens.


## Features

The KeeperC Chrome Extension provides a user-friendly interface to interact with the KeeperC protocol. It has its own wallet functionalities, so you don't need other wallets!

Here's how to use it:

1. Download the KeeperC Chrome Extension from the [GitHub repository](#). Please

note that it's not available on the Chrome Web Store yet.

2. Run `npm run build` to update the `dist` folder, then navigate to <chrome://extensions/> and click on "Load unpacked" to import the updated `dist` folder. You'll need to enable **Developer mode** first.
3. Get test tokens by clicking the "FAUCET / 0x1234..." button.
4. Enjoy  KeeperC!

[Live Demo](#)[Information](#)

# Information

## DANGER

The KeeperC is a work-in-progress and is provided as-is, without any warranties or guarantees of its effectiveness. Its use is entirely at your own risk. Use the web app and extension with caution and thoroughly review its code before use to avoid potential loss of funds.

## Contracts

### INFO

You'll need to give allowance to KeeperC contract on TERC20 and KTERC20 contracts for the first time usage.

### ERC20 Token

- address: "0x6f7ebA5Ccf6c1e524df9F0f353843B233f82e48F"
- name: "Test ERC20"
- symbol: "TERC20"

### KeepERC20 Token (about TERC20)

- address: "0x09955185759C8A5d1668AE9C843185a063b39516"
- name: "Keep Test ERC20"
- symbol: "KTERC20"

KeepERC20 Factory

- address: "0x516e99AccB8Ebd6FC04C5FE4C516b8fF172a37e7"

## Chainlink Automation

- [KeepERC20-TERC20 Upkeep](#)
- [Register New Upkeep](#)

# Register Upkeeps

Mumbai testnet is used for a concrete example.

## Mumbai Testnet MATIC and LINK

You need sufficient MATICs and LINKs for registering Upkeep.

### Mumbai Faucet

- [Polygon](#)
- [Alchemy](#)

### LINK Faucet

- [Chainlink](#)

## Register Upkeep

Goto [Chainlink Automation](#) page.

Chainlink | Automation

Polygon Mumbai0x5e14...2c80

NEW Check your eligibility for early access to Staking v0.1. [View now.](#)

### Chainlink Automation

Automate your smart contract with Chainlink's hyper-reliable Automation network.

[Register new Upkeep](#)[Go to the docs](#)[How it works](#)

Registry address  
0x02777053d6764996e594c3E88AF1D58D5363a2e6

#### My upkeeps

ActiveCancelled

No upkeeps found

We can't find any upkeeps for your wallet address. Register a new Upkeep to get started.

#### Recent Upkeeps

Name	Status	Address	Balance LINK
cryptoBet-Nov	Active	0x54079e52623a9c4d7fc73eb8c78ae7a48561769e	15
DNA	Active	0xf5279d0d0286b13121e0542cb9322d3ab9746254	4.97

Register new upkeep through:

- Select `Custom logic`
- Input address of KeepERC20 contact to perform Upkeep on

**NEW** Check your eligibility for early access to Staking v0.1. [View now.](#)

[Home](#) /

## Register new Upkeep

Automate your smart contract with Chainlink's hyper-reliable Automation network.

### Trigger

Select the trigger mechanism for automation

☐ Time-based

☐ Custom logic

### Automation triggers

The trigger specifies what Automation Nodes should look at to determine if your Upkeep should be performed.

Time-based uses a time schedule (CRON) to execute your smart contract function according to the schedule.

Custom logic uses an Automation-compatible contract that you deployed to determine when to perform your Upkeep.

[Learn more](#) about an Automation-compatible contracts.

Need help or have questions? [Talk to an expert](#) or visit the [Automation webpage](#) to learn more

Stay updated on the latest from Chainlink

Enter your email address

[Sign up](#)

[Home](#) /

## Register new Upkeep

Welcome to Chainlink Automation - fully automate your contract in two simple steps.

### Trigger

Custom logic

## Target contract address

0x09955185759C8A5d1668AE9C843185a063b39516

Address of your Automation compatible contract to perform Upkeep on.

⚠ Unable to verify if this is an Automation compatible contract.

Next

## Upkeep address

When using custom logic, you need to provide the address of your Automation-compatible contract. [Learn more](#) about creating an Automation-compatible contract. Please follow our [best practices](#) when creating your contract. Your deployed contract does not need to be "verified" to use it with Chainlink Automation.

Need help or have questions? [Talk to an expert](#) or visit the [Automation webpage](#) to learn more

Stay updated on the latest from Chainlink

Enter your email address

[Sign up](#)

Set `check data` for pagination (lowerBound, upperBound) For example,

[illegible]

You can submit multiple Upkeeps for one KeepERC20 contact with different pagination.

## CAUTION

Registering the same contract's Upkeeps with the same `check_data` is possible but useless and inefficient.

### Upkeep details

Upkeep name

KeepERC20-TERC20

Provide a name for your Upkeep to easily manage it in the Automation UI.

Admin Address

0x5e144ee53c6f3305362bda2bc019081405bc2c80

The address for the administrator of this Upkeep.

Gas limit

500000

Amount of gas to provide the target contract when performing Upkeep. This will impact minimum balance requirements, and should be approximately the maximum amount of gas the transaction might use.

Starting balance (LINK)

10

Deposit LINK to your Upkeep. Select an amount that will satisfy multiple performances to start, then fund the Upkeep directly once it's operational.

Check data (Hexadecimal) *Optional*

0x0000000000000000000000000000000000000000000000000000000000000000

Pass static data into your checkUpkeep function. This will be converted to bytes. See [docs](#) for details.

**Need LINK for testing?** Visit the [Chainlink Polygon Mumbai Faucet](#) to receive testnet LINK.

### Project information

This information will not be displayed publicly.

Your email address *Optional*

Finally, register Upkeep.

impact minimum balance requirements, and should be approximately the maximum amount of gas the transaction might use.

Starting balance (LINK)

10

Deposit LINK to your Upkeep. Select an amount that will satisfy multiple performances to start, then fund the Upkeep directly once it's operational.

Check data (Hexadecimal) *Optional*

0x0000000000000000000000000000000000000000000000000000000000000000

Pass static data into your checkUpkeep function. This will be converted to bytes. See [docs](#) for details.

**Need LINK for testing?** Visit the [Chainlink Polygon Mumbai Faucet](#) to receive testnet LINK.

### Project information

This information will not be displayed publicly.

Your email address *Optional*

lukepark327@gmail.com

We will only use this email address for Upkeep.

Project name *Optional*

KeepERC20-TERC20

Register Upkeep

Submit registration request

Receive confirmation

## Upkeep registration request submitted successfully

You can view your Upkeep registration via button below. If it's pending approval, you will receive an email once it's approved.

View your transaction here:

0x7a9b16a494f4f1eb7d9f03bfa5e3ccc29db3377814a0837a43da0cacfb419317

View Upkeep

Need help or have questions? Talk to an expert or visit the [Automation webpage](#) to learn more



This is the sample **Upkeep service** named **KeepERC20-TERC20**.

You can see the Upkeep's transaction history:

Status

Active

Date ran

November 18, 2022 at 06:47 UTC

Balance

9.995496353893579 LINK

Minimum Balance ⓘ

0.26250595365487783 LINK

---

Registry address

0x0277...a2e6 ⓘ

Total spent

0.00450364610642084 LINK

## Details ^

Registration

Owner address

0x5e14...2C80 ⓘ

Date

November 18, 2022 at 05:10 UTC

Transaction Hash

0x7a9b...9317 ⓘ

Upkeep

ID

934015...1283 ⓘ

Address

0x0995...9516 ⓘ

Gas limit

500,000

Trigger

Type

Custom

Check data (Base16)

0x0000000000000000000000000000000000000000000000000000000000000000  
0000000000000000000000000000000000000000000000000000000000000000  
0000000000000000000000000000000000000000000000000000000000000000

## History

Date UTC	Sender	Transaction hash	Activity type	Amount LINK	Balance LINK
November 18, 2022 at 06:47	0xb71...69a9 ⓘ	0xa7f...6cae ⓘ	Perform Upkeep	-0.004249181120339975	10
November 18, 2022 at 05:58	0x97F...0dA7 ⓘ	0x38ff...09ad ⓘ	Perform Upkeep	-0.000254464986080865	10
November 18, 2022 at 05:10	0xdb8e...5a1d ⓘ	0x7a9b...9317 ⓘ	Fund Upkeep	10	10

Prev

Showing 1 to 3 of 3 entries

New



For Developers

# For Developers



## Intefaces

Please refer to GitHub.



## Deploy

See KeepERC20-wrapper for more details.

# Interfaces

Please refer to [GitHub](#).

---

## IKeepERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.17;

interface IKeepERC20 {
    //===== Params =====//

    enum TaskType {
        Schedule,
        Recovery,
        Expire
    }

    struct Task {
        TaskType taskType;
        address sender;
        address receiver;
        uint256 amount;
        bytes extraField;
        bool active;
    }

    //===== Events =====//
```

# IKeepERC20Factory.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.17;

interface IKeepERC20Factory {
    //===== Events =====//

    event SetScheduleFeeRatio(uint256 prev, uint256 curr);
    event SetRecoveryFeeRatio(uint256 prev, uint256 curr);
    event SetFeeTo(address prev, address curr);

    event NewKeepERC20(
        address originalToken,
        uint256 scheduleFeeRatio,
        uint256 recoverFeeRatio,
        address feeTo
    );

    //===== Initialize =====//

    function setScheduleFeeRatio(uint256 newScheduleFeeRatio_)
    external;

    function setRecoveryFeeRatio(uint256 newRecoveryFeeRatio_)
    external;

    function setFeeTo(address newFeeTo_) external;

    //===== View Functions =====//

    function keepOf(address token) external view returns
    (address);
```

## IKeepERC20Upkeep.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.17;

interface IKeepERC20Factory {
    //===== Automation =====//

    function pagination(uint256 lowerBound, uint256 upperBound)
        external
        pure
        returns (bytes memory);

    function checkUpkeep(bytes calldata checkData)
        external
        view
        returns (bool upkeepNeeded, bytes memory performData);

    function performUpkeep(bytes calldata performData) external;
}
```

## IWallet.sol

```
// SPDX-License-Identifier: MIT

pragma solidity 0.8.17;

import "@openzeppelin/contracts/access/Ownable.sol";

interface IWallet {
```















# Deploy

See [KeepERC20-wrapper](#) for more details.

---

## Requirements

```
$ npm install
```

## Set `.env`

and/or `.env.test` for test environment.

Now we can use pre-defined values as environment variable, with a prefix `dotenv` `-e <ENV> --`. For example, `dotenv -e .env.test --`.

## Run Node

```
$ dotenv -e .env.test -- npx hardhat node --network hardhat
```

# Deploy

```
$ dotenv -e .env -- npx hardhat run scripts/deploy.js --network localhost
```

*Mumbai testnet is used for a concrete example.*

```
$ dotenv -e .env.test -- npx hardhat run scripts/deploy.js --network mumbai
```

Compiled 2 Solidity files successfully

<Set>

Owner: 0x1ccE14942bD77f5c8EdFe408f7116595E18ccaF4  
(0.19886533549475788 ETH)

User1: 0x0E723d5710E79907b0E6D3661F3fed0D3452C04c  
(0.8923594874453868 ETH)

User2: 0xdBf13a0374E70f01DB7d1a570Be84e067B9E1Be1 (0 ETH)

Fee: 0x21De12f081958D5590AB70C172703345286bcDc9 (0 ETH)

<Deploy>

Deploy Token:

0x6f7ebA5Ccf6c1e524df9F0f353843B233f82e48F

Deploy Factory:

0x516e99AccB8Ebd6FC04C5FE4C516b8fF172a37e7

Deploy KeepToken:

0x09955185759C8A5d1668AE9C843185a063b39516



Usecase

# Usecase

There can be many use-cases of KeeperC.

For example, it is possible to invest in **Dollar Cost Averaging** that purchases a certain amount of cryptocurrency periodically through DEX.

KeeperC allows users to use blockchain more conveniently and safely. We expect that KeeperC will open a new horizon for using blockchain and DApp.