LYNKPAY.ME

ERC20 Payment processing for LynkPay

Abstract

This paper outlines the technical implementation of **LynkPay**, a crypto payment processor designed to facilitate ERC20 token payments, primarily for USDT. The solution offers a **non-custodial by design** system for securely collecting payments in cryptocurrency. However, LynkPay is structured to collect and hold tokens temporarily in a wallet controlled by the platform in order to facilitate conversion into fiat currency for the merchants. This approach eliminates custodial risks while supporting the final transfer of funds as cash or fiat currency through exchange offices. This document provides an in-depth technical overview, explaining the system architecture, smart contract flow, and operational details.

Introduction

Motivation

Traditional payment processors often introduce custodial risks, requiring merchants to trust centralized entities to handle their funds. With cryptocurrency payments, especially in ERC20 tokens, there is an opportunity to offer non-custodial solutions that allow direct token transfers. However, for practical reasons, especially with cryptocurrency-to-fiat conversions, it is often necessary to centralize the receipt of tokens before converting to fiat. LynkPay leverages a hybrid approach to balance non-custodial security with the need for fiat payouts to merchants.

Objective

LynkPay's goal is to provide a seamless payment processing system that allows merchants to receive payments in ERC20 tokens (such as USDT), but ensures that the funds are ultimately transferred to the merchants in fiat currency (such as USD, EUR). LynkPay achieves this by using a **non-custodial** payment processor at the blockchain level, while centralizing token receipt temporarily within LynkPay's wallet to facilitate conversion to fiat through partnered exchange offices.

Use case

User flow

1. **Merchant Sets Up a Payment Address**: The merchant generates a payment address using the LynkPay forwarder factory.
2. **Customer Deposits Funds**: The customer transfers ERC20 tokens (such as USDT) to the computed address generated by the merchant's forwarder.
3. **Tokens Sent to LynkPay's Wallet**: After deployment of the forwarder contract, the ERC20 tokens are forwarded to a wallet controlled by LynkPay.
4. **Token Conversion and Payout**: LynkPay works with exchange offices to convert the received tokens into fiat currency and transfer the equivalent amount to the merchant's bank account.

Benefits

- **Security**: The system is **non-custodial by design**, with the forwarder contract ensuring that tokens are only held temporarily in LynkPay's wallet for conversion.
- **Fiat Integration**: By working with exchange offices, LynkPay converts the cryptocurrency into fiat for merchants, who can then use the funds in their local currency.
- **Reduced Custodial Risk**: While the funds are held in LynkPay's wallet, they are only temporarily controlled, and full transparency is maintained through smart contract interactions.
- **Automated Fund Flushing**: The forwarder contract ensures automatic fund forwarding once tokens are received, removing manual intervention.

Technical Design

Contract Components

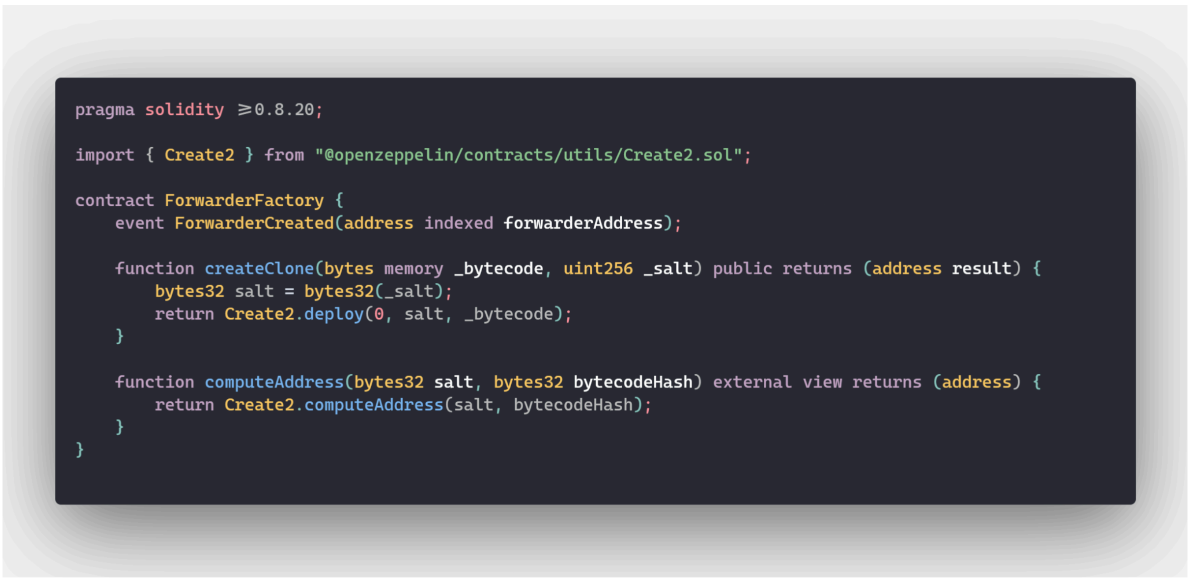
1. **Factory Contract (`ForwarderFactory`)**: Deploys individual forwarder contracts using the `CREATE2` opcode, ensuring predictable contract addresses and allowing merchants to securely receive payments.
2. **Forwarder Contract (`Forwarder`)**: A minimal contract that temporarily holds funds and automatically forwards them to a wallet controlled by LynkPay, where the tokens can be converted to fiat.

Detailed Code Walkthrough

Factory Contract

The `ForwarderFactory` contract allows merchants to generate forwarder addresses that users can send tokens to. These addresses are deterministic, meaning they can be precomputed using the salt and bytecode hash. This enables merchants to share their payment address before any transaction takes place.

- **`createClone(bytes _bytecode, uint256 _salt)`**: Deploys a forwarder contract using the provided bytecode and salt, returning the contract's address.
- **`computeAddress(bytes32 salt, bytes32 bytecodeHash)`**: Computes the address where the contract will be deployed, allowing merchants to share the address with customers before actual deployment.



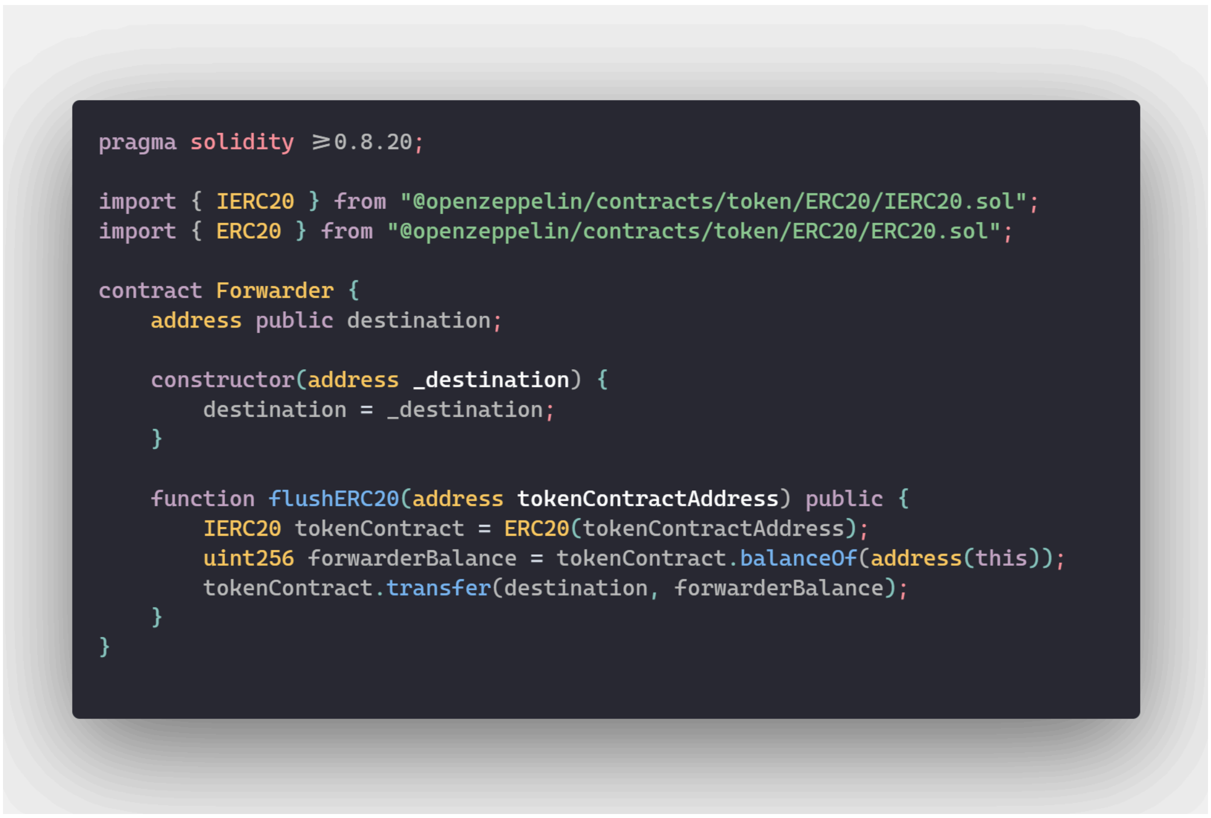
```solidity
pragma solidity >=0.8.20;

import { Create2 } from "@openzeppelin/contracts/utils/Create2.sol";

contract ForwarderFactory {
    event ForwarderCreated(address indexed forwarderAddress);

    function createClone(bytes memory _bytecode, uint256 _salt) public returns (address result) {
        bytes32 salt = bytes32(_salt);
        return Create2.deploy(0, salt, _bytecode);
    }

    function computeAddress(bytes32 salt, bytes32 bytecodeHash) external view returns (address) {
        return Create2.computeAddress(salt, bytecodeHash);
    }
}
```

LYNKPAY.ME

Forwarder Contract

The `Forwarder` contract holds funds temporarily and forwards them to LynkPay's wallet. The contract contains a `flushERC20` function to transfer tokens to LynkPay, enabling the exchange of cryptocurrencies for fiat.

- **`flushERC20(address tokenContractAddress)`**: Transfers the entire balance of ERC20 tokens held in the forwarder contract to LynkPay's wallet.
- **Constructor**: The constructor initializes the destination address, which in this case, is LynkPay's wallet where tokens are forwarded.

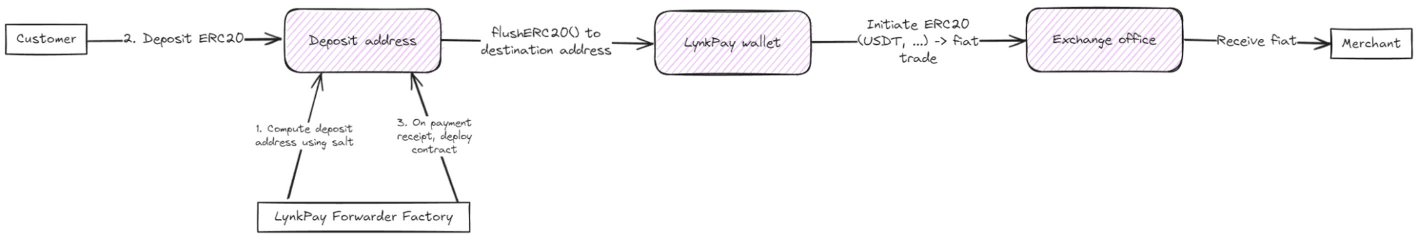


```
pragma solidity ≥0.8.20;

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract Forwarder {
    address public destination;

    constructor(address _destination) {
        destination = _destination;
    }

    function flushERC20(address tokenContractAddress) public {
        IERC20 tokenContract = ERC20(tokenContractAddress);
        uint256 forwarderBalance = tokenContract.balanceOf(address(this));
        tokenContract.transfer(destination, forwarderBalance);
    }
}
```

Step-by-Step Walkthrough

Payment Flow

1. **Customer Transfers Funds**:
   a. The customer transfers ERC20 tokens (such as USDT) to the computed forwarder address.
2. **Deployment and Forwarding**:
   a. After the forwarder contract is deployed, the tokens are automatically forwarded to LynkPay's wallet using the `flushERC20` function.
3. **Token Conversion to Fiat**:
   a. LynkPay works with exchange offices to convert the received tokens into fiat currency.
4. **Merchant Receives Fiat Payment**:
   a. LynkPay transfers the fiat equivalent to the merchant, completing the payment process.


Customer
2. Deposit ERC20
Deposit address
flushERC20() to destination address
LynkPay wallet
Initiate ERC20 (USDT, ...) -> fiat trade
Exchange office
Receive fiat
Merchant
1. Compute deposit address using salt
3. On payment receipt, deploy contract
LynkPay Forwarder Factory

Conclusion

LynkPay offers a **non-custodial by design** solution for ERC20 token payments, leveraging the power of Ethereum smart contracts to create secure payment addresses for merchants. Although the system is designed to be non-custodial, tokens are temporarily routed to a wallet controlled by LynkPay to enable cryptocurrency-to-fiat conversion. This approach strikes a balance between decentralization and real-world functionality, ensuring merchants can receive their payments in fiat currency. The system automates token forwarding and simplifies the payment process for merchants, offering a scalable solution for modern payment systems.