



ZERO

1

KNOWLEDGE

A Workshop on Zero-Knowledge Trust

EDCON 2024 x ZKT NETWORK

WELCOME TO

ZKTNETWORK!



ZKT Network DevRel

@MadaoChris on TG / X

QRCode



bit.ly/zkt-sdk-workshop-edcon

About ZKT Network

- We're a all-chain Trust Machine
- We provide Trust Score Oracle for developers & protocols.
- We are working to create a better privacy and compliance blockchain based on Trust.

Introducing ZKT SDK 🎉

- ZKT SDK is the oracle calling SDK we provide to various public chain developers.
- This Trust Score represents the aggregated result of an address's on-chain and off-chain behaviors.
- This SDK supports multiple chains and can be used on multiple L1/L2.

What can I do with ZKT SDK?

- Get scores through the Trust Score oracle and provide different types of services to users of the DeFi protocol
- Identify risky users by screening out low scores and protect users' assets.
- Create your own Trust Score Persona and get your own Trust Score. Users with high Trust Score will be rewarded by the network.
- ...and a lot more on the way!

LFG!!! How we can start?

```
forge install ZKTLabs/zktnetwork
```

Easy! What is next?

- Documentation - <https://docs.zkt.network/zkt/product/zkt-sdk>
- Discord - <https://discord.gg/eagyPuaGj5>
- Github Discussions - <https://github.com/ZKTLabs/zktnetwork/discussions>
- Github repo - <https://github.com/ZKTLabs/zktnetwork>
- Examples - <https://github.com/ZKTLabs/zkt-examples/tree/main>

Example 1: Basic usage

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import {ComplianceAggregatorV2} from "@zktnetwork/v0.2/abstract/ComplianceAggregatorV2.sol";

contract Counter is ComplianceAggregatorV2 {
    uint256 public count;

    constructor(address _versionedMerkleTreeStub) ComplianceAggregatorV2(_versionedMerkleTreeStub)
    {
        count = 0;
    }

    /// @notice Increment the counter, only if the caller is verified
    function increment(bytes32[] memory proof, bytes memory encodedData) external {
        require(stub.verify(proof, encodedData), "Counter: Invalid proof");

        require(msg.sender == stub.getAccount(encodedData, true), "Counter: Invalid account");
        require(stub.getScore(encodedData, true) > 60, "Counter: Invalid score");

        count += 1;
    }

    /// @notice Increment the counter, only if the caller is verified
    function decrement(bytes32[] memory proof, bytes memory encodedData) external {
        require(stub.verify(proof, encodedData), "Counter: Invalid proof");

        require(msg.sender == stub.getAccount(encodedData, true), "Counter: Invalid account");
        require(stub.getScore(encodedData, true) > 90, "Counter: Invalid score");
        count -= 1;
    }
}
```

Example 2: Integration for Uniswap hook

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.4;

import {ComplianceAggregatorV2} from "@zktnetwork/v0.2/abstract/ComplianceAggregatorV2.sol";
import {BaseHook} from "v4-periphery/BaseHook.sol";
import "v4-core/interfaces/IPoolManager.sol";
import "v4-core/types/PoolKey.sol";
import "v4-core/libraries/Hooks.sol";

contract ZKTUniswapV4ComplianceHook is BaseHook {
    uint256 public beforeSwapCounter;
    uint256 public validScore;
    bool public bypass;

    constructor(
        address _versionedMerkleTree,
        IPoolManager _poolManager,
        uint256 _validScore,
        bool _bypass
    ) BaseHook(_poolManager) ComplianceAggregatorV2(_versionedMerkleTree) {
        beforeSwapCounter = 0;
        validScore = _validScore;
        bypass = _bypass;
    }

    function getHookPermissions() public view returns (Hooks.Permissions) {
        return Hooks.Permissions({
            beforeInitialize: false,
            afterInitialize: false,
            beforeAddLiquidity: false,
            afterAddLiquidity: false,
            beforeRemoveLiquidity: false,
            afterRemoveLiquidity: false,
            beforeSwap: true,
            afterSwap: false,
            beforeDonate: false,
            afterDonate: false
        });
    }

    function beforeSwap(address, PoolKey calldata, IPoolManager.SwapParams calldata, bytes calldata data)
        external
        virtual
        override
        returns (bytes4)
    {
        (
            bytes32[] memory proof,
            bytes memory encodedData
        ) = abi.decode(data, (bytes32[], bytes));
        require(stub.verify(proof, encodedData), "ZKTUniswapV4ComplianceHook: Invalid proof");

        if (!bypass) {
            require(tx.origin == stub.getAccount(encodedData, true), "ZKTUniswapV4ComplianceHook: Invalid account");
        }
        require(stub.getScore(encodedData, true) > validScore, "ZKTUniswapV4ComplianceHook: Invalid score");
        beforeSwapCounter += 1;
        return BaseHook.beforeSwap.selector;
    }
}
```

```
// .....
```

```
function beforeSwap(address, PoolKey calldata, IPoolManager.SwapParams calldata, bytes calldata data)
    external
    virtual
    override
    returns (bytes4)
{
    (
        bytes32[] memory proof,
        bytes memory encodedData
    ) = abi.decode(data, (bytes32[], bytes));
    require(stub.verify(proof, encodedData), "ZKTUniswapV4ComplianceHook: Invalid proof");

    if (!bypass) {
        require(tx.origin == stub.getAccount(encodedData, true), "ZKTUniswapV4ComplianceHook: Invalid account");
    }
    require(stub.getScore(encodedData, true) > validScore, "ZKTUniswapV4ComplianceHook: Invalid score");
    beforeSwapCounter += 1;
    return BaseHook.beforeSwap.selector;
}
```



THANK YOU

Get in touch ykw@zkt.network