

# NoirCash

Transacciones anónimas con Noir y zero knowledge.  
Protege tu privacidad mientras garantizas que tus  
fondos no provienen de actividades maliciosas.





# El equipo



**Nico**  
Dev



**Victor**  
Dev



**Lila**  
Dev



**Eugenio**  
Dev

# Problema

Las soluciones actuales mezclan ether legitimos con fondos ilegítimos.



## 01

Las soluciones actuales dan anonimidad, pero al mismo tiempo no son 100% éticas

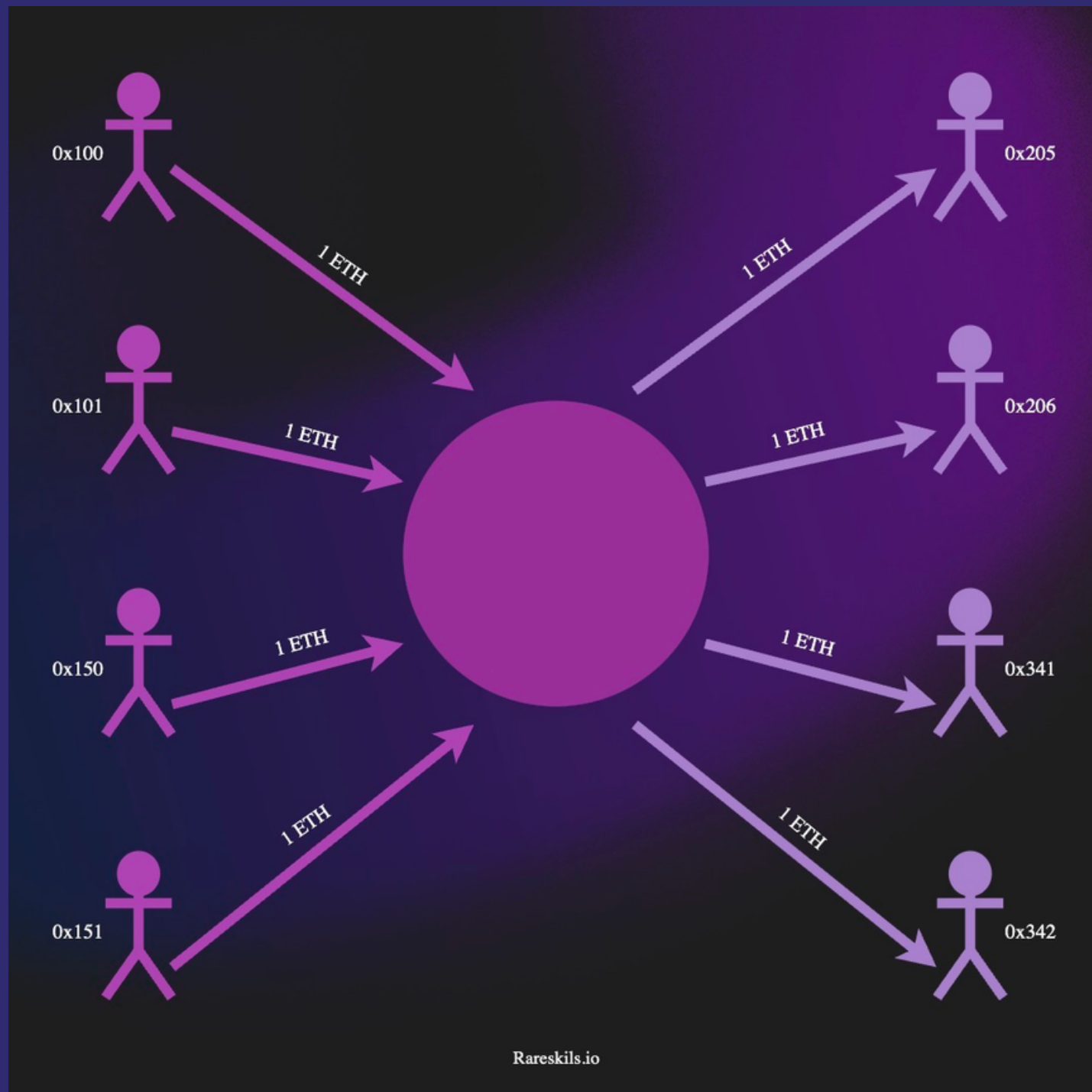
## 02

Cuando depositamos ether para obtener privacidad estamos mezclando nuestro ether con otros ethers que pueden provenir de actividades consideradas ilegales.

## 03

Al retirar nuestro ether no hay forma de saber que nuestro ether proviene de actividades lícitas.

# Funcionamiento actual

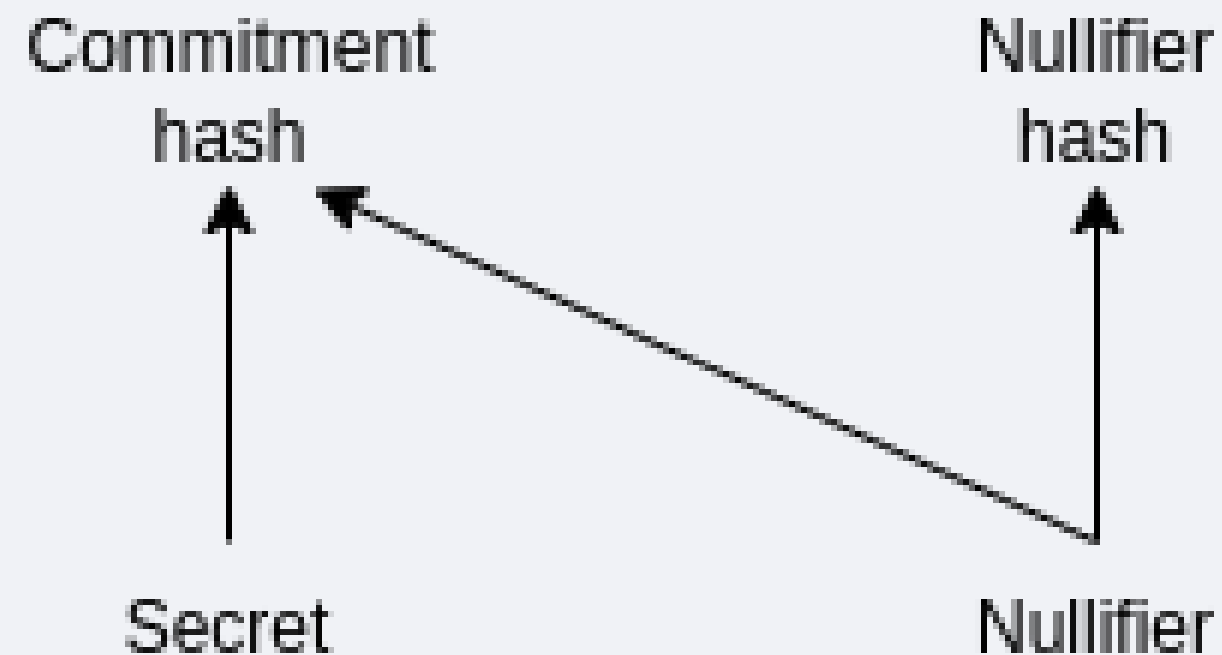


**Al depositar obtenemos un commitment**

**Con ese commitment podemos crear una zk proof para retirar y nullificar el commitment, sin decir cual es el commitment**

**El problema es que hay dinero de malos actores y no tenemos forma al retirar de que no somos malos actores.**

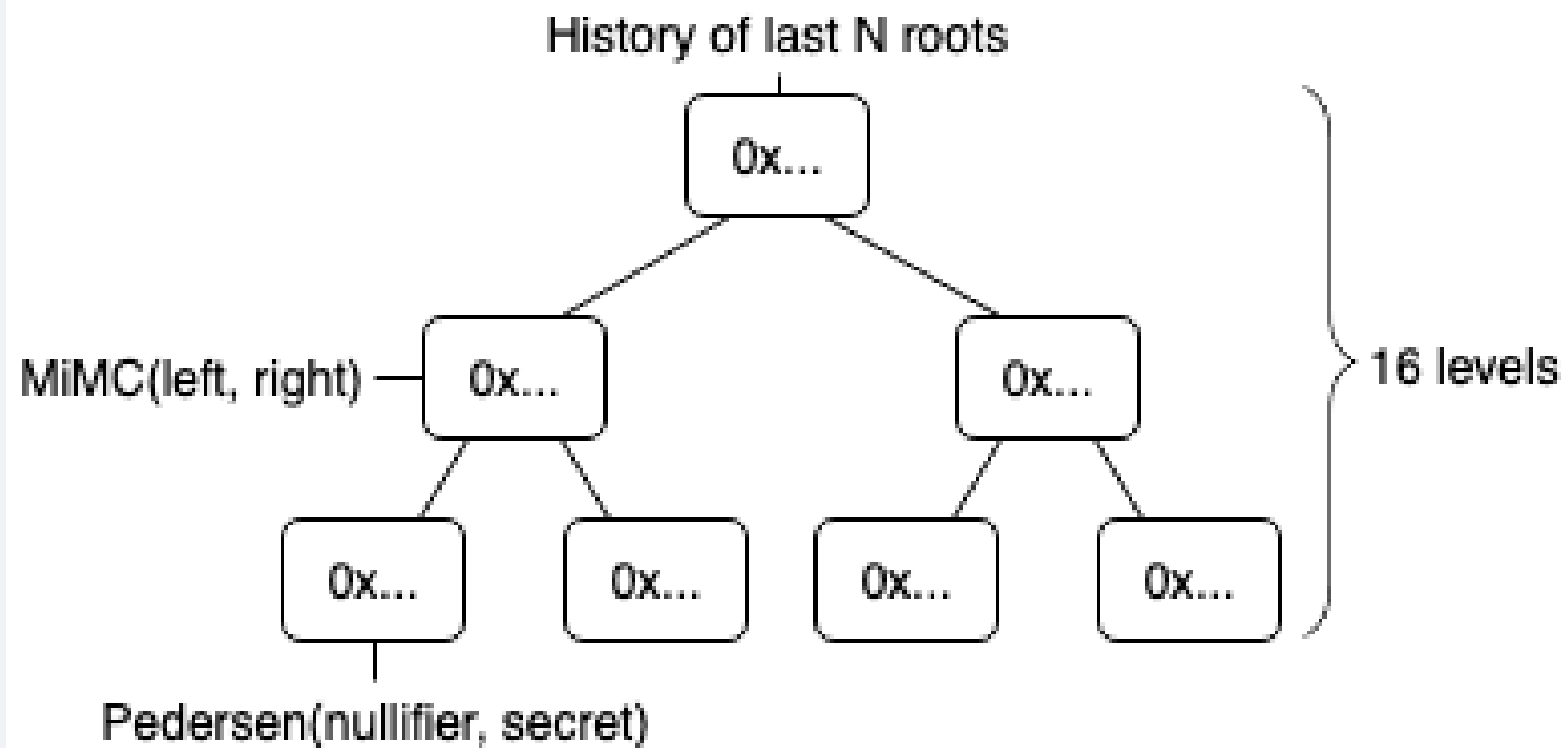
# Funcionamiento actual



**Con un nullifier y un secreto se genera el commitment.**

**Si cambia el nullifier cambia el commitment.**

# Funcionamiento actual



## On deposit:

- Generate random secret and nullifier
- Compute their hash (commitment)
- Check that user sent correct amount ETH
- Insert user commitment into the tree

## On withdrawal:

- User proves that he knows merkle path to certain leaf and preimage to this leaf
- User reveals only nullifier part of his commitment, it is used to track spent notes
- User supplies address to withdraw to and optional fee to address that submits withdraw transaction
- Check SNARK proof
- Check that nullifier is not spent
- Save nullifier
- Release funds

# 01

Noir cash cuenta con un sistema de blacklist, manejado por una DAO

# 02

Al momento de retirar ademas de proveer un nullifier hay que enviar una zkproof para demostrar que no esta en el blacklist

# 03

Si el usuario no puede demostrar que no esta en la blacklist, la address de salida sera agregada a la blacklist



## Solución

Escribe de una a tres formas en que tu empresa propone resolverlos.



# Diseño

Vista del como interactuar para depositar

## NoirCash


Fund

Withdraw

Token amount

Max

0

 ETH

Recipient address

0xd8dA6BF2696...D37aA96045

Deposit





# Diseño

Vista del como interactuar para depositar

## NoirCash

Fund

Withdraw

Recipient address


Place recipient address here

Commitement

Place commitment proof

Not blacklisted proove

Proove you are not blacklisted

Withdrawal method	Relayer 
Relayer fee	0.05 %
L1 network fee	0.017675 ETH
Total fee	-
To receive	-
<b>Total</b>	-

Withdraw

# Qué sigue

Si bien resuelve los problemas planteados, creemos que hay algunas cuestiones mas que resolver.

Fees? como funciona la DAO? como se maneja la blacklist? si el dinero esta blacklistado tiene que tener un fee mayor? como se distribuyen los tokens de la DAO para votacion?



Title

```
1 // // https://tornado.cash
2 /*
3  * d888888P                                dP                                a888888b.                                dP
4  *   88                                88                                d8'  `88                                88
5  *   88      .d8888b. 88d888b. 88d888b. .d8888b. .d888b88 .d8888b. 88      .d8888b. .d8888b. 88d888b.
6  *   88   88' `88 88' `88 88' `88 88' `88 88' `88 88   88' `88 Y8ooooo. 88' `88
7  *   88   88. .88 88      88   88 88. .88 88. .88 88. .88 dP Y8. .88 88. .88      88 88   88
8  *   dP   `88888P' dP      dP   dP `88888P8 `88888P8 `88888P' 88   Y88888P' `88888P8 `88888P' dP   dP
9  *   ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
10 */
11
12 // SPDX-License-Identifier: MIT
13 pragma solidity ^0.7.0;
14
15 import "./MerkleTreeWithHistory.sol";
16 import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";
17
18 interface IVerifier {
19     function verifyProof(bytes memory _proof, uint256[6] memory _input) external returns (bool);
20 }
21
22 abstract contract Tornado is MerkleTreeWithHistory, ReentrancyGuard {
23     IVerifier public immutable verifier;
24     uint256 public denomination;
25
26     mapping(bytes32 => bool) public nullifierHashes;
27     // we store all commitments just to prevent accidental deposits with the same commitment
28     mapping(bytes32 => bool) public commitments;
29
30     event Deposit(bytes32 indexed commitment, uint32 leafIndex, uint256 timestamp);
31     event Withdrawal(address to, bytes32 nullifierHash, address indexed relay, uint256 fee);
32
33     /**
34      * @dev The constructor
35      * @param _verifier the address of SNARK verifier for this contract
36      * @param _hasher the address of MiMC hash contract
37      * @param _denomination transfer amount for each deposit
38      * @param _merkleTreeHeight the height of deposits' Merkle Tree
39      */
40     constructor(
41         IVerifier _verifier,
42         IHasher _hasher,
43         uint256 _denomination,
44         uint32 _merkleTreeHeight
45     ) MerkleTreeWithHistory(_merkleTreeHeight, _hasher) {
46         require(_denomination > 0, "denomination should be greater than 0");
47         verifier = _verifier;
48         denomination = _denomination;
49     }
50
51     /**
52      * @dev Deposit funds into the contract. The caller must send (for ETH) or approve (for ERC20) value equal
53      to or `denomination` of this instance.
54      * @param _commitment the note commitment, which is PedersenHash(nullifier + secret)
55      */
56     function deposit(bytes32 _commitment) external payable nonReentrant {
57         require(!commitments[_commitment], "The commitment has been submitted");
58
59         uint32 insertedIndex = _insert(_commitment);
60         commitments[_commitment] = true;
61         _processDeposit();
62
63         emit Deposit(_commitment, insertedIndex, block.timestamp);
64     }
65
66     /** @dev this function is defined in a child contract */
67     function _processDeposit() internal virtual;
68
69     /**
70      * @dev Withdraw a deposit from the contract. `proof` is a zkSNARK proof data, and input is an array of
71      circuit public inputs
72      */
73     function withdraw(bytes32 indexed commitment, uint32 indexed leafIndex, address indexed relay, uint256 fee,
74         bytes memory proof) external {
75         require(commitments[commitment], "Commitment not found");
76         require(verifier.verifyProof(proof, [commitment, leafIndex, relay, fee, denomination, 0]), "Invalid proof");
77         _processWithdrawal(commitment, leafIndex, relay, fee);
78     }
79
80     function _processWithdrawal(bytes32 commitment, uint32 leafIndex, address relay, uint256 fee) internal {
81         bytes32 nullifierHash = _hasher.hash(commitment, leafIndex);
82         nullifierHashes[nullifierHash] = true;
83         emit Withdrawal(relay, nullifierHash, relay, fee);
84     }
85 }
```

Gracias