ZERO KNOWLEDGE PROOFS

Semaphore - overview

Semaphore is a zero-knowledge protocol that allows you to cast a signal (for example, a vote or endorsement) as a provable group member without revealing your identity. Additionally, it provides a simple mechanism to prevent double-signaling.

Use cases include **private voting, whistleblowing, anonymous DAOs and mixers**.

You can find more information on these links:

Link 1 >
Link 2 >

Semaphore - features

With Semaphore, you can allow your users to do the following:

1. **Create a Semaphore identity.**
2. **Add their Semaphore identity to a group (i.e. *Merkle tree*).**
3. **Send a verifiable, anonymous signal (e.g a vote or endorsement).**

When a user broadcasts a signal (for example: a vote), Semaphore zero-knowledge proofs can ensure that the user has joined the group and hasn't already cast a signal with their nullifier.

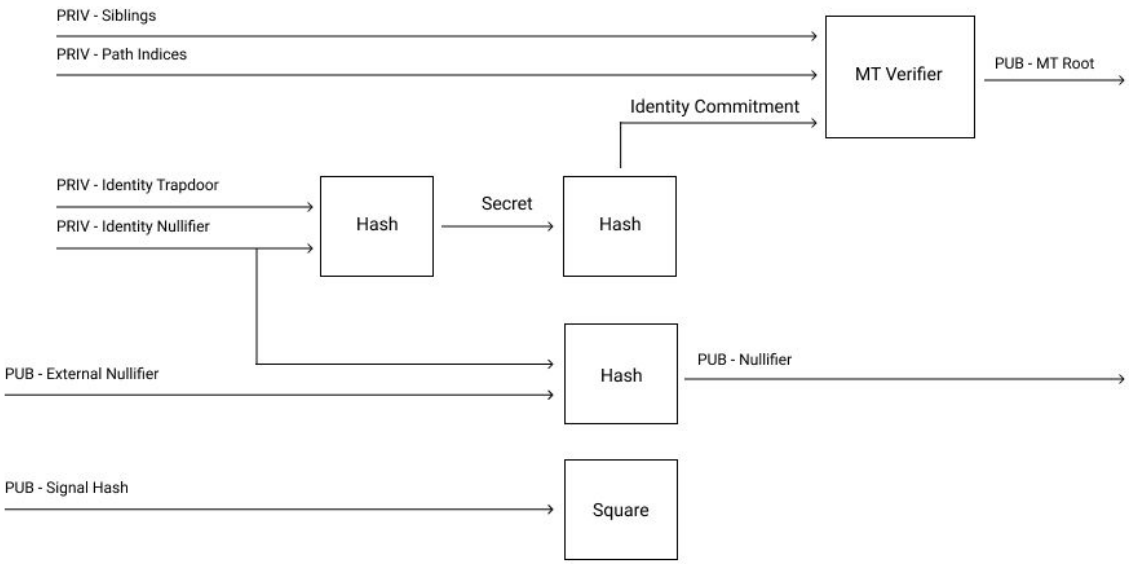
* *You can find more information on this link:*

Link 1 >

Semaphore - circuits

The Semaphore circuit is the heart of the protocol and consists of three parts:

- **Proof of membership**
- **Nullifier hash**
- **Signal**



There's More 👉

Semaphore - membership proof

The circuit hashes the hash of the identity nullifier with the identity trapdoor to generate an **identity commitment**. Then, **it verifies the proof of membership against the Merkle root and the identity commitment.**

Private inputs:

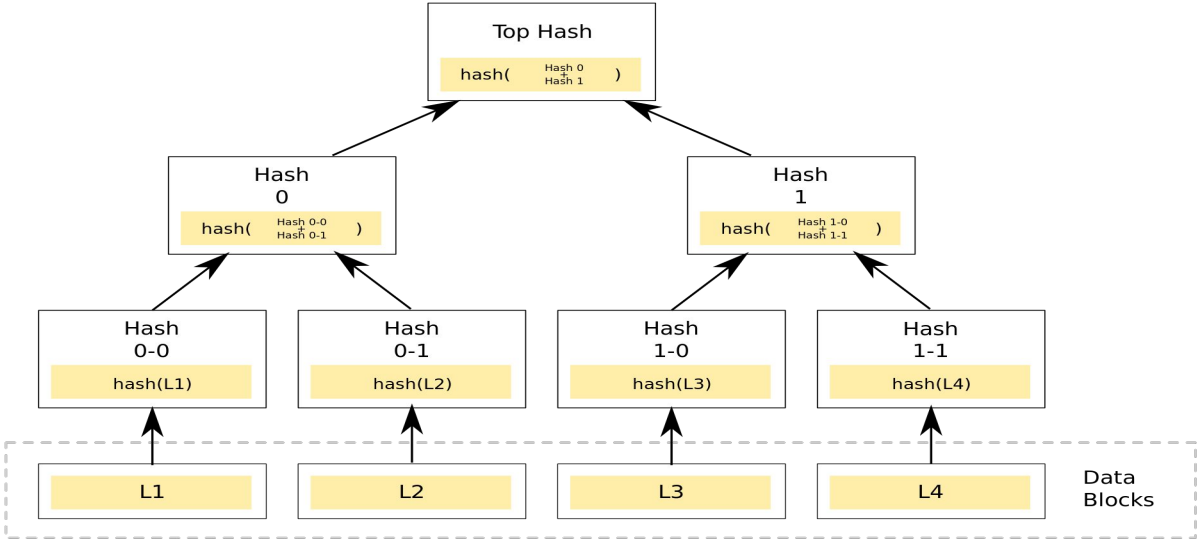
- **treeSiblings[nLevels]**: the values along the Merkle path to the user's identity commitment,
- **treePathIndices[nLevels]**: the direction (0/1) per tree level corresponding to the Merkle path to the user's identity commitment,
- **identityNullifier**: the 32-byte identity secret used as nullifier,
- **identityTrapdoor:** the 32-byte identity secret used as trapdoor.

Public outputs:

- **root:** The Merkle root of the tree.

There’s More 👉

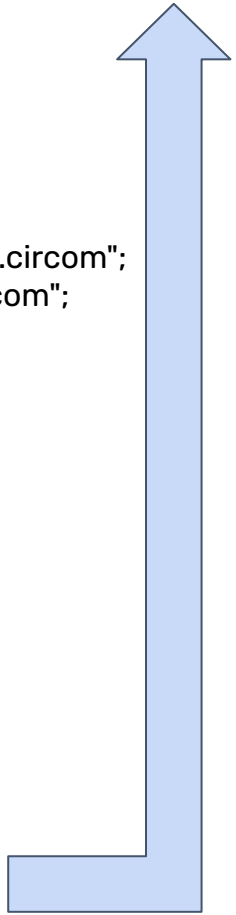
Merkle tree



There's More 👉

MT Inclusion Proof

```
pragma circom 2.0.0;

include "../node_modules/circomlib/circuits/poseidon.circom";
include "../node_modules/circomlib/circuits/mux1.circom";

template MerkleTreeInclusionProof(nLevels) {
    signal input leaf;
    signal input pathIndices[nLevels];
    signal input siblings[nLevels];

    signal output root;

    component poseidons[nLevels];
    component mux[nLevels];

    signal hashes[nLevels + 1];
    hashes[0] <== leaf;
```



```
    for (var i = 0; i < nLevels; i++) {
        pathIndices[i] * (1 - pathIndices[i]) === 0;

        poseidons[i] = Poseidon(2);
        mux[i] = MultiMux1(2);

        mux[i].c[0][0] <== hashes[i];
        mux[i].c[0][1] <== siblings[i];

        mux[i].c[1][0] <== siblings[i];
        mux[i].c[1][1] <== hashes[i];

        mux[i].s <== pathIndices[i];

        poseidons[i].inputs[0] <== mux[i].out[0];
        poseidons[i].inputs[1] <== mux[i].out[1];

        hashes[i + 1] <== poseidons[i].out;
    }

    root <== hashes[nLevels];
```

You can find more information on these links:

[Link 1 >]