Anonymous signaling on Ethereum

Why we need anonymity, and how we can achieve it

Andy Product Owner, Privacy and Scaling Explorations

Invisibility is a superpower

"I don't know why people are so keen to put the details of their private life in public; they forget that invisibility is a superpower."

-Banksy

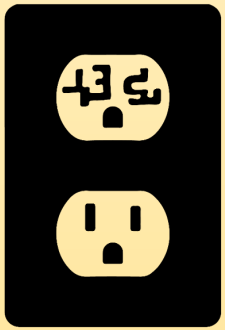
Anonymity can help us to:

- **Limit power**: knowledge is power, protecting it makes us stronger.
- **Promote freedom of speech**: knowing that our data and identity are safe encourages us to think freely.
- **Safeguard reputation**: our ideas should not be judged based on who we are, but rather on what we have to say.

Drawbacks

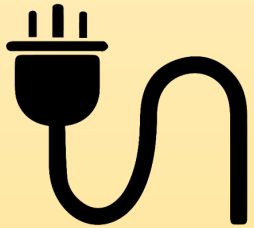
Complexity

- Still too niche technologies
- Lack of practical development tools



Indifference

- Lack of awareness
- Still few education resources



Solutions

Privacy by default

Privacy and cryptography should be the backbone of the Internet infrastructure.

Education

People should be more aware of the technological and social complexity of the world we live in.

Developer experience

Developers need to be able to rely on robust, easy-to-use tools.

Semaphore

Semaphore is a zero-knowledge protocol that lets users prove their membership in a **group** and send **signals** such as votes or endorsements without revealing the user's original **identity**.

And additionally, it provides a simple mechanism to prevent **double-signaling**.

Identities

Each identity is made up of:

- Two secret values: **Trapdoor** and **Nullifier**
- One public value: **Commitment**

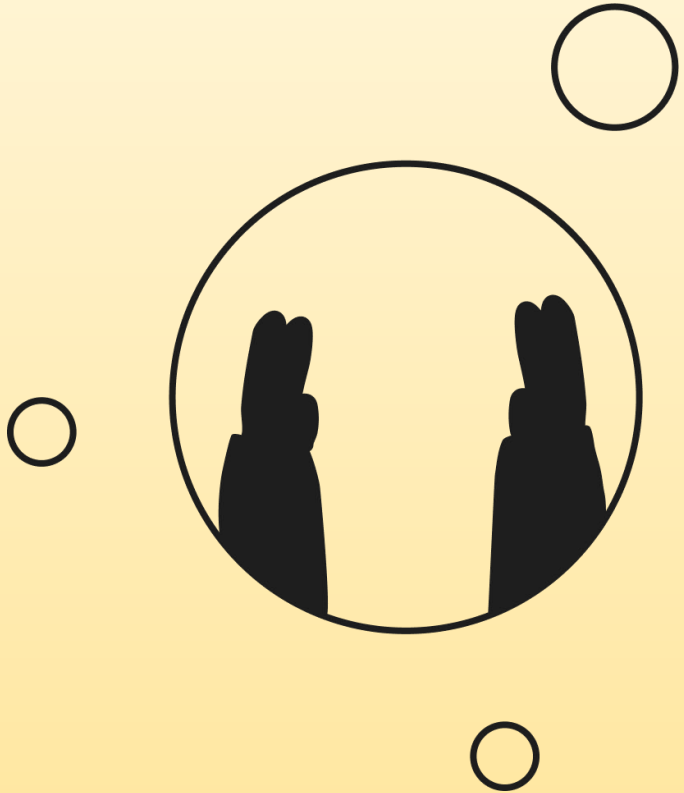
```
import { Identity } from "@semaphore-protocol/identity"

// Create a random identity
const { trapdoor, nullifier, commitment } = new Identity()

// Create a deterministic identity
const identity = new Identity("secret-message")

// Export your identity
identity.toString()
```

https://semaphore.appliedzkp.org/docs/guides/identities

Groups

Groups can be thought of as **anonymity sets**. They are a way to establish necessary **trust** among participants.

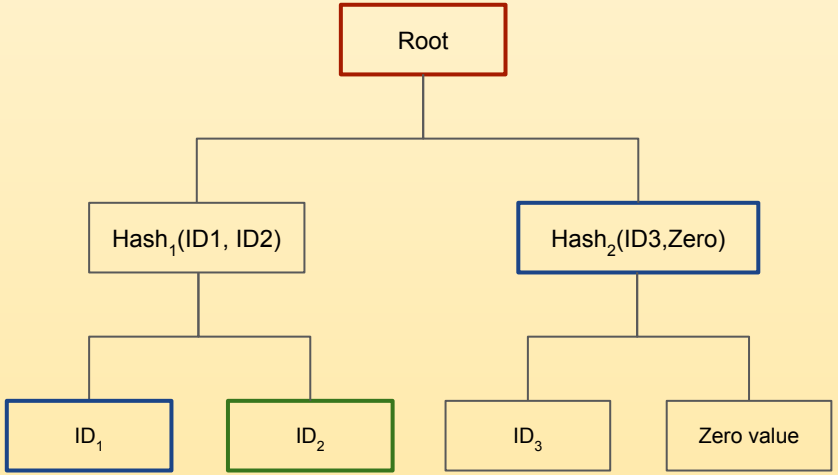
Semaphore groups are **binary Merkle trees**, in which the leaves are identity commitments and all the other nodes in the tree are hashes of their two child nodes.



Groups

In particular, Semaphore uses **incremental binary Merkle trees**. Every new leaf will have as its index the index of the previous leaf + 1.

To prove that a leaf is part of the tree, it will be sufficient to share the **path between the leaf and the root**, so that everyone can calculate the hashes and check whether the roots match.



Proof of membership:
Leaf: ID_2
Sibling nodes: [ID_1, $Hash_2$]
Path indices: [0, 1]

Groups

Semaphore groups can be created off-chain with a JavaScript library, or on-chain with the Semaphore contracts.

```
import { Group } from "@semaphore-protocol/group"

const group = new Group()

group.addMember(commitment)

group.updateMember(0, commitment2)

group.removeMember(0)
```

```
contract Greeter {
    ISemaphore public semaphore;
    uint256 public groupId;

    constructor(address semaphoreAddress, uint256 _groupId) {
        semaphore = ISemaphore(semaphoreAddress);
        groupId = _groupId;

        semaphore.createGroup(groupId, 20, 0, address(this));
    }
}
```

https://semaphore.appliedzkp.org/docs/guides/groups

ZK-Proofs

After creating their identity and joining a group users can anonymously prove that they are members of that group and send **signals**, such as votes, endorsements or any message.

To generate a valid proof we also need an **external nullifier**. The hash of this value and the identity nullifier is the **nullifier hash**, which can be used to avoid double-signaling.

ZK-Proofs

Zero-knowledge proofs can be generated off-chain with a JavaScript library. They can then be verified both on-chain and off-chain.

```
import { generateProof, verifyProof } from "@semaphore-protocol/proof"

const externalNullifier = 42n
const greeting = "Hello world"

const fullProof = await generateProof(identity, group, externalNullifier, greeting)

const verificationKey = JSON.parse(fs.readFileSync("./semaphore.json", "utf-8"))

await verifyProof(verificationKey, fullProof)
```

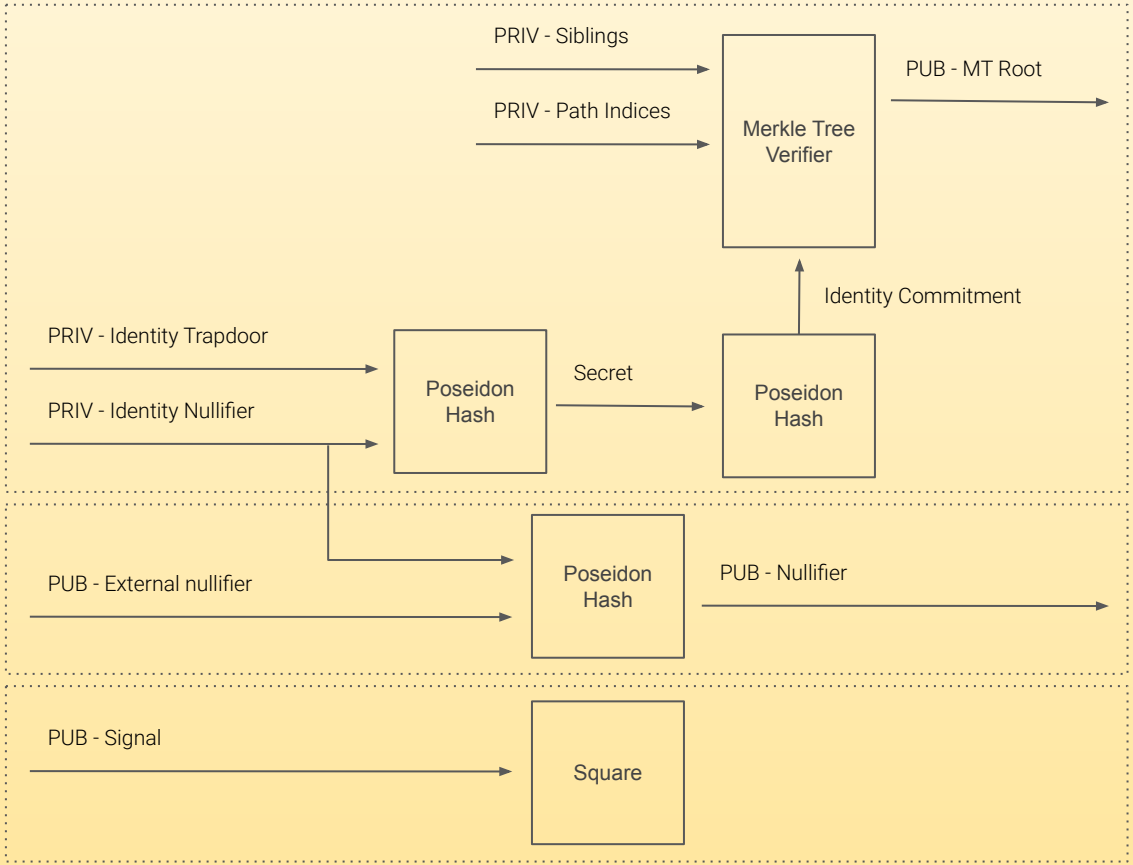
```
contract Greeter {

    function greet(
        bytes32 greeting,
        uint256 merkleTreeRoot,
        uint256 nullifierHash,
        uint256[8] calldata proof
    ) external {
            semaphore.verifyProof(
                    groupId,
                    merkleTreeRoot,
                    greeting,
                    nullifierHash,
                    groupId,
                    proof
            );
}
```

https://semaphore.appliedzkp.org/docs/guides/proofs

Circuits

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

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

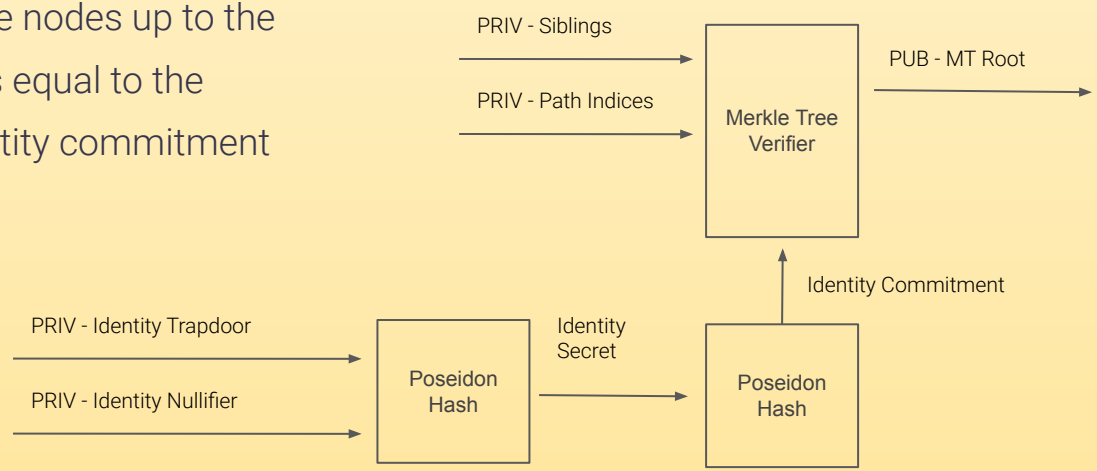


https://semaphore.appliedzkp.org/docs/technical-reference/circuits

Circuits

The identity secret values are hashed to obtain the identity commitment, which is a leaf of the tree.

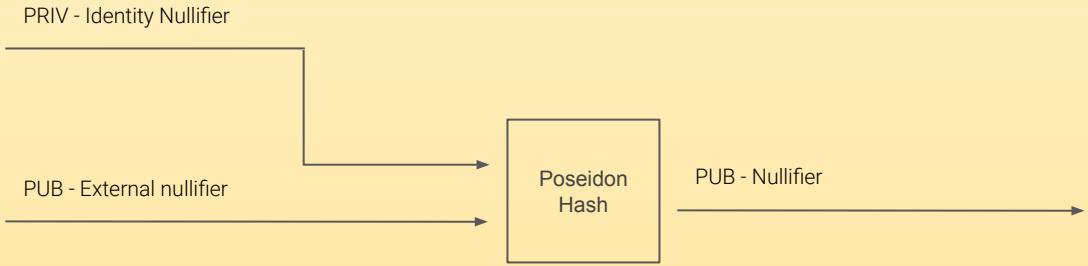
Starting from the leaf, the hash of the tree nodes up to the root is calculated. If the calculated root is equal to the public value passed to the proof, the identity commitment belongs to the tree.



https://semaphore.appliedzkp.org/docs/technical-reference/circuits

Circuits

The identity nullifier and the external nullifier are hashed to obtain the public nullifier, which can be used to prevent double signaling externally.



https://semaphore.appliedzkp.org/docs/technical-reference/circuits

Circuits

The signal square is a simple constraint used to prevent tampering. In fact, adding a dummy constraint is the Circom recommended way to solve the issue of unused public inputs.

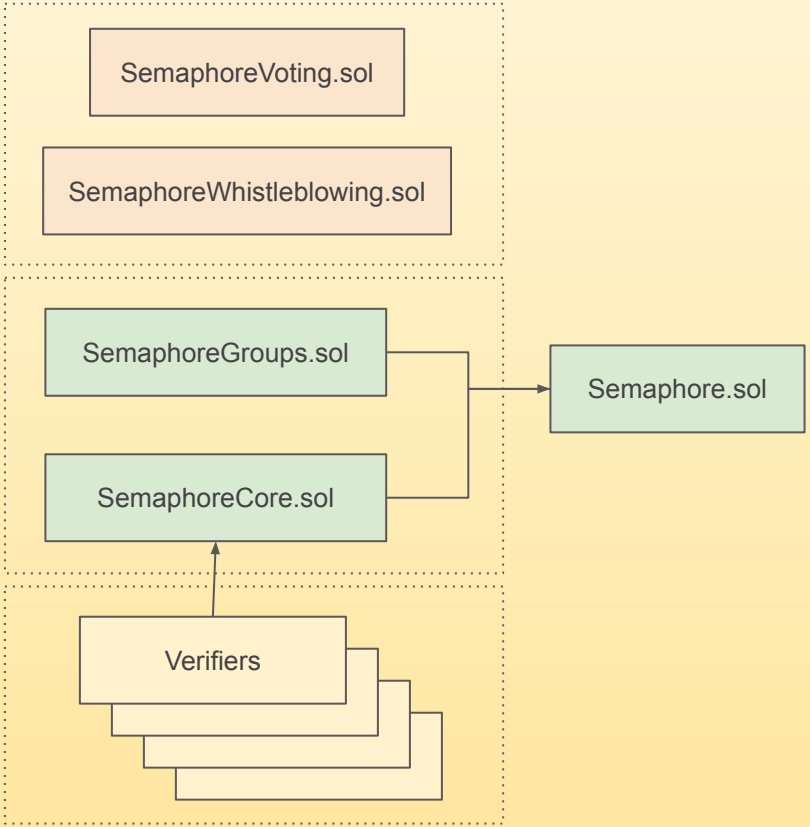
Thus, attackers cannot update the value of a signal without invalidating the proof.

https://semaphore.appliedzkp.org/docs/technical-reference/circuits

Contracts

Semaphore includes three types of contracts:

- **Verifiers**
- **Base contracts**
  - SemaphoreGroups.sol
  - SemaphoreCore.sol
- **Extension contracts**
  - SemaphoreVoting.sol
  - SemaphoreWhistleblowing.sol



https://semaphore.appliedzkp.org/docs/technical-reference/contracts

Contracts

Semaphore.sol inherits the base contracts and provides a ready-to-use solution for developers.

It includes a mechanism to prevent double signaling with the proof nullifiers and it is currently available on Goerli and Arbitrum One.

```
contract ISemaphore {

    function verifyProof(...) external;
    function createGroup(...) external;
    function updateGroupAdmin(...) external;
    function addMember(...) external;
    function addMembers(...) external;
    function updateMember(...) external;
    function removeMember(...) external;

}
```

https://semaphore.appliedzkp.org/docs/technical-reference/contracts

Semaphore in use today

Unirep

Unirep is a protocol which allows anonymous members of a group to give, receive, and prove reputation without revealing their identity.

https://docs.unirep.io

ZKitter

Anonymous social network where people can post and chat without losing their real-life reputation.

https://zkitter.com

TAZ apps

Experimental Semaphore applications to learn through experience about privacy and anonymity at Devcon VI.

https://taz.appliedzkp.org

Semaphore in use today

Interep

Interep is an **anti-sybil system** which provides special Semaphore groups that can be used by DApps or services to verify users' reputations without exposing their identities.

Users can join those groups based on their Web2 reputation and Interep only checks whether they meet the right criteria (e.g. number of Github stars), without storing any sensitive data.

https://interep.link

Future plans

Semaphore will continue to be developed and improved over time. Some potential future directions include:

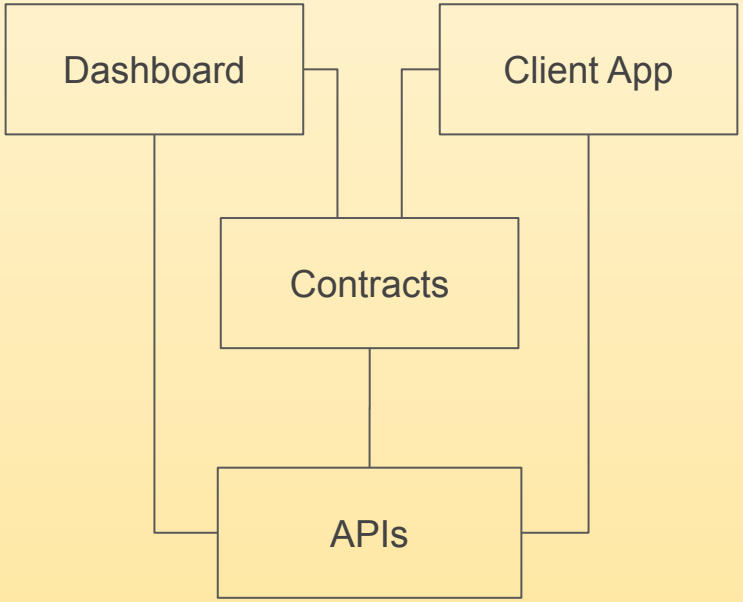
- **ZK-Groups**: an infrastructure to manage groups
- **V3 Audits:** internal and external for major versions.
- **Boilerplate App** https://boilerplate-semaphore.vercel.app/
- Investigate other zero-knowledge technologies and **new proving systems**
- Continue improving the **developer experience** and **documentation**
- Create a **strong community**

ZK-Groups

ZK-Groups provides a comprehensive cloud or self-hosted infrastructure to allow anyone to create their own groups:

- **Permissioned groups**: admins decides who to add to a group and how
- **Permissionless groups**: users can join groups themselves, based on certain attributes they have

Developers can use a Dashboard to manage groups, and integrate them into their applications with the APIs.



https://github.com/semaphore-protocol/zk-groups

Thank you!

Semaphore Discord

Semaphore website

Andy

Product Owner, Privacy and Scaling Explorations

andres.guzto@gmail.org @AndyGuzmanEth