On Deployed vOPRFs, Nullifiers, and the World ID Infrastructure

Roman Walch

[Don't] share your data.

Privacy and Identity

- Privacy brings challenges for identity
- Authenticate ...
  - ... without deanonymization
  - ... without linking between services
- Limit actions despite privacy (e.g., just one airdrop per ID)
- Protect secret keys linked to identity
  - Allow multiple keys, e.g., for multiple devices
  - Allow changing keys

$\Rightarrow$ OPRF based solution for anonymous identity

- In process of being deployed for World ID
- Case study for remaining presentation

World ID Infrastructure

- World ID
  - Digital identity linked to individuals
  - Unique identifier for humans
    - No duplicate signups
    - No bots, AI, ect.
    - Enforced with scan of human iris
- This talk: Authenticate to services using World ID
  - Anonymity
    - Only reveal that you have any World ID
  - Unlinkability between actions
    - ... but still limit one time actions, such as air drops

© World

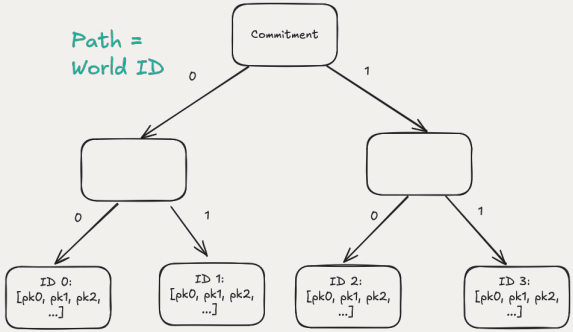
Semaphore

- Semaphore is usually the go-to solution for such a problem
  - Audited and well known by the community
- Two part protocol:
  - A users key pair is part of an allowlist (Merkle tree)
    - Authenticates the user is allowed to perform action
  - Produces a nullifier $H(sk, \text{scope})$
    - Same action would produce same nullifier again

    $\Rightarrow$ Limits the action to be a one-time action
- Zero knowledge proof:
  - Proofs user owns sk
  - Proofs $C(sk)$ is part of the allowlist (Merkle proof)
  - Proofs correct computation of the nullifier

World ID

- Why not just use Semaphore?
- World ID will be linked to multiple public keys
  - Are allowed to change
  - Action with any *sk* must produce same nullifier



$\Rightarrow$ Just use $H(id, \text{scope})$?

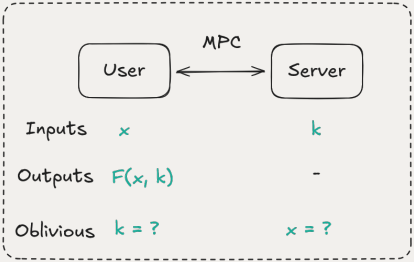
- Too little entropy
- For a given scope, I can brute force *id* from nullifier
- 17 million World IDs $\Rightarrow$ Deanonymize on a laptop

Threshold OPRF

OPRF

- We need another source of entropy in the nullifier

⇒ Use an OPRF

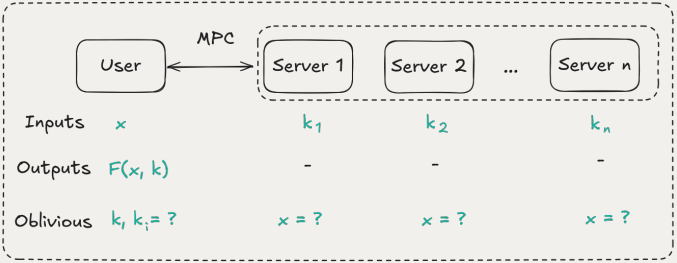


- Problem: Server can still brute force x from the OPRF output
  - Can collude with app provider to deanonymize users

Threshold OPRF

- Solution: Have multiple servers
  - Ideally different organizational and legal entities in different jurisdiction
- Each server has secret share of k
  - Shamir secret sharing: n servers, any t are enough to finish protocol

$\Rightarrow$ t colluding servers required to deanonymize users using brute force



OPRF and World ID

World ID using OPRF

- Nullifier for action for different app providers (RPs):

$$n \leftarrow \mathrm{OPRF}(H(id_u, id_{rp}, action), k_{rp})$$

- Remember Semaphore?
  - User authentication and zero knowledge proofs still missing
- Final ZK proof:
  - User knows sk (proven by verifying a signature)
  - pk corresponding to sk is part of the World ID registry at index id_u
  - id_u has a valid credential signature
  - OPRF has been computed correctly

$\Rightarrow$ Requires verifiable OPRF

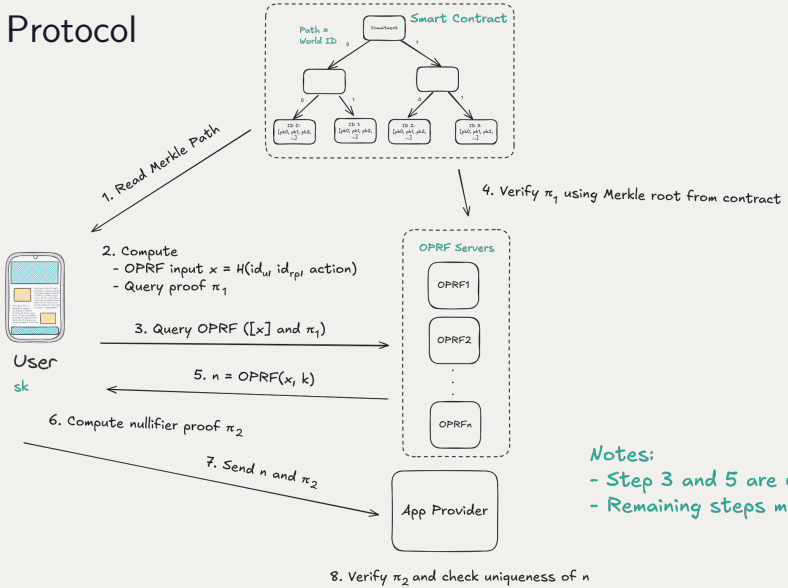
OPRF Authentication

- There is still a brute force problem...
- Malicious actor can brute force id_u by repeatedly querying OPRF

  $\Rightarrow$ User authentication required
- Query OPRF ZK proof:
  - User knows sk (proven by verifying a signature)
  - pk corresponding to sk is part of the World ID registry at index id_u
  - id_u has a valid credential signature
  - OPRF input of the user is $H(id_u, id_{rp}, action)$

$\Rightarrow$ Subset of the final nullifier ZK proof

Full Protocol

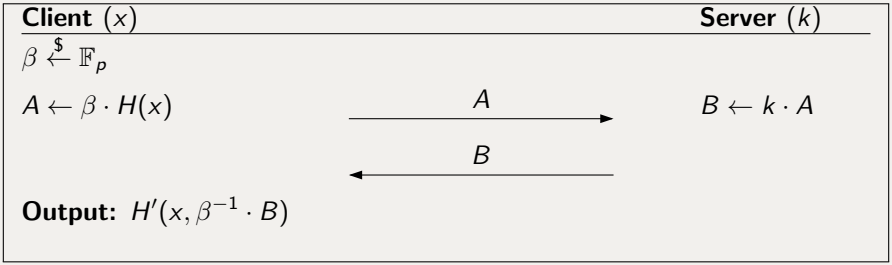


Notes:
- Step 3 and 5 are use-case independent
- Remaining steps mostly user authentication

Building a vOPRF

OPRF Instantiation

- We use the common TwoHashDH OPRF:



- Simple and efficient
- Easy to thresholdize

Verifiability

- Verifiability via discrete logarithm equality proof (Chaum-Pedersen proof)
  - Requires public key $K = k \cdot G$
  - Proves K and B share the same DLog with bases G and A respectively

Algorithm 1 DLog Equality Proof

Input: k, A, B, K, G
Output: e, s

$r \xleftarrow{\$} \mathbb{F}_p$
$R_1 \leftarrow r \cdot G$
$R_2 \leftarrow r \cdot A$
$e \leftarrow H(A, B, K, G, R_1, R_2)$
$s \leftarrow r + e \cdot k$
return (e, s)

Threshold DLog Equality Proof

- Literature: Usually one individual proof per Server
  - We want recursively verify valid DLog equality proof
  - Proof size would scale with number of parties
  - $\Rightarrow$ MPC variant to create just one proof
- DLog equality proof is very similar to Schnorr proof and EdDSA
  - Threshold DLog equality very similar to threshold EdDSA signatures
  - Our variant based on Frost3
  - $\Rightarrow$ 2 round protocol, no communication between servers
- OPRF key generation also inspired by threshold signatures

Threshold DLog Equality Proof

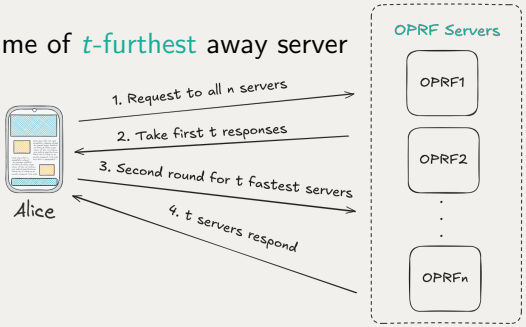
- Literature: Usually one individual proof per Server
  - We want recursively verify valid DLog equality proof
  - Proof size would scale with number of parties

  $\Rightarrow$ MPC variant to create just one proof
- DLog equality proof is very similar to Schnorr proof and EdDSA
  - Threshold DLog equality very similar to threshold EdDSA signatures
  - Our variant based on Frost3

  $\Rightarrow$ 2 round protocol, no communication between servers
- OPRF key generation also inspired by threshold signatures

OPRF Performance

- Implemented in Rust using BabyJubJub curve
- Compute on each OPRF server (m7a.4xlarge, singlethreaded):
  - 2.6 ms, where 1.9 ms is ZK proof verification for authentication
- Latency:
  - 2x round-trip-time of *t*-furthest away server



Zero Knowledge Proofs

- ZK proofs in Circom
  - Proofs over BN254
  - Poseidon2 for every hash function

Function	R1CS Constraints	$\lceil \log_2(\cdot) \rceil$
Semaphore (d=32)	9383	14
EdDSA Poseidon2 Verifier	5618	13
DLog Equality Verifier	12293	14
Oprf Query (d=32)	23190	15
Oprf Nullifier (d=32)	39889	16

- Mobile phone proving time $< 1\,s$

Deployment

- Protocol has been audited (OPRF, Circom, smart contracts)
- Currently 3 OPRF nodes hosted by highly reputable parties (e.g., universities)
  - Planned to increase number to 10+ in next year
  - Worldwide deployment
- In process of deployment for World ID infrastructure
  - Verifiable threshold key generation via smart contract
- Core protocol is generic
  - Replace user authentication for your use case and use our OPRF

Further Reading

Repository

Technical writeup

Questions

?

TACEO
Team Experience

From cryptographic foundations to large-scale deployments, our team has proven expertise. We designed the Poseidon hash function, a **cornerstone of ZK systems**, and built the **world's largest MPC**, securing the data of over 11 million people.

We deliver world-leading technology across the stack.

We've bridged academia, the web2 & the web3 industries.

We are trusted and backed by:

Archetype
POLYMORPHIC CAPITAL
cyber•Fund
World
CSX alózcrypto
WAGMI VENTURES
a_capital
Aztec
DAEDALUS

We were students and professors at:

KU LEUVEN
DTU
TU Graz
ENS

200 papers | 20+ together | 4 PhDs

We got experience from:

NXP
CSX alózcrypto
Microsoft
Google
DFINITY
WORLDLINE

TACEO

Subscribe for updates

hello@TACEO.io TACEO.io 𝕏: @TACEO_IO