Lesson 15 - Identity / Oracles / Cryptographic tools

Identity Solutions

Polygon ID

See site

Polygon ID has the following properties:

- Blockchain-based ID for decentralised and self-sovereign models
- Zero-knowledge native protocols for ultimate user privacy
- Scalable and private on-chain verification to boost decentralised apps and DeFi
- Open to existing standards and ecosystem development

It uses the Iden3 and Circom toolkit

In comparison with NFTs and VCs

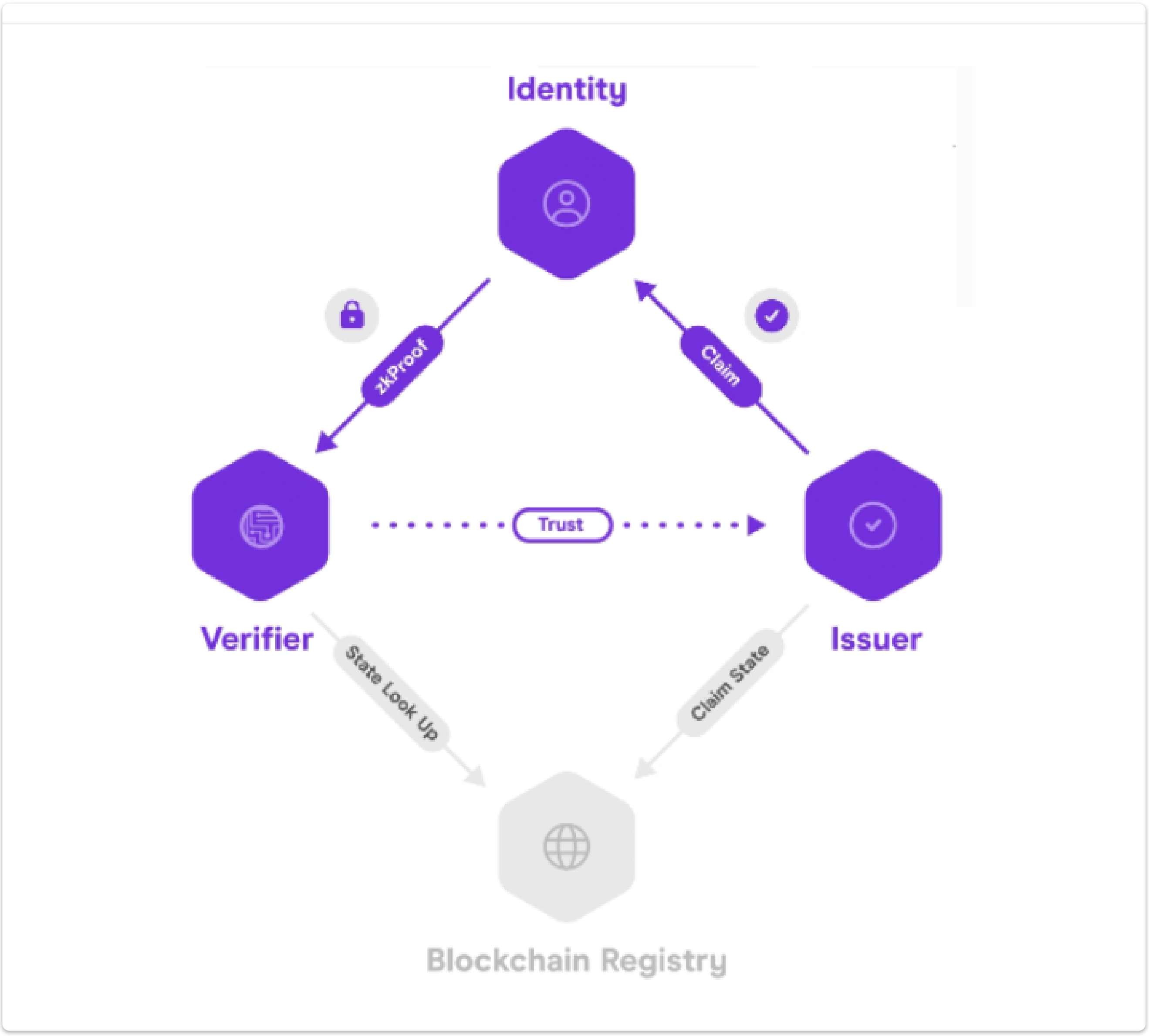
NFTs are not private, and have high minting costs.
While VCs offer some degree of privacy with selective disclosure and ZK add-on, their limitations are in the expressibility and composability, which are required for applications.
Verifying VCs on-chain is prohibitively expensive.

There is an ID client toolkit to facilitate onboarding
On chain verification uses zkProof Request Language, which allows applications to specify which requested private attributes a user needs to prove.

Roles in Polygon ID

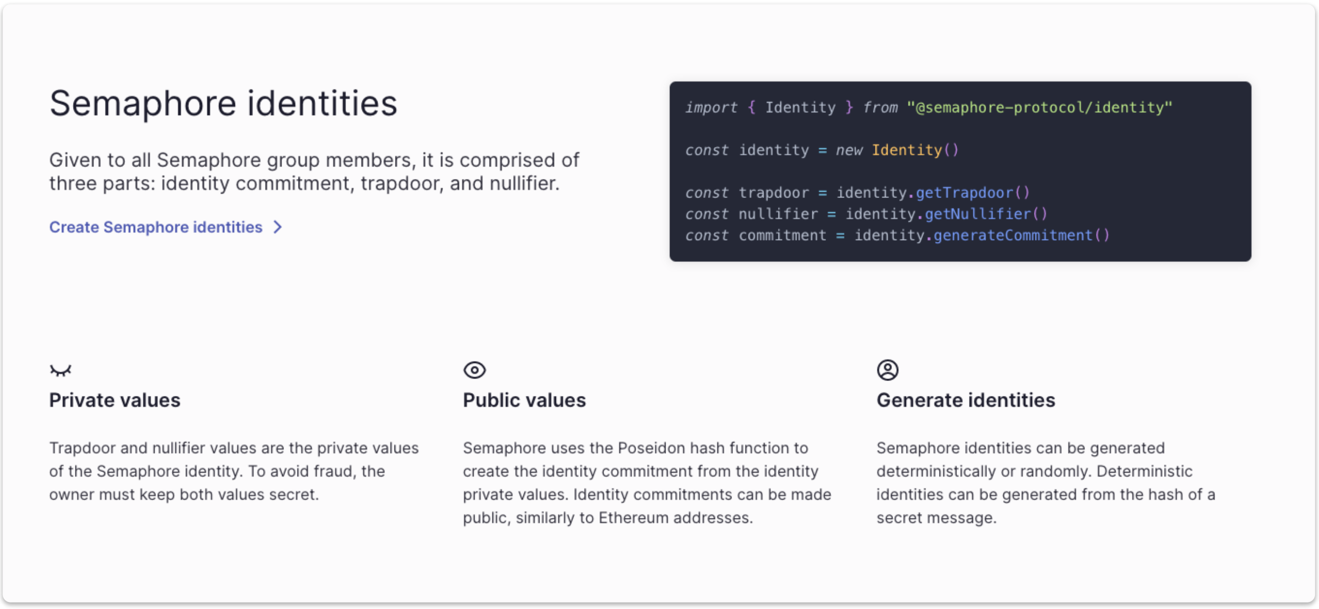
1. Identity Holder: An entity that holds claims in its Wallet. A claim is issued by an Issuer to the Holder. The Identity holder creates the zero-knowledge proofs of the claims issued and presents these proofs to the Verifier (which verifies the correctness and authenticity of the claim). A Holder is also called Prover as it needs to prove to the Verifier that the credential it holds is authentic and matches specific criteria.
2. Issuer: An entity (person, organisation, or thing) that issues claims to the Holders. Claims are cryptographically signed by the Issuer. Every claim comes from an Issuer.
3. Verifier: A Verifier verifies the claims presented by a Holder. It requests the Holder to send proof of the claim issued from an Issuer and on receiving the zero-knowledge proofs from the Holder, verifies it. The verification process includes checking the veracity of the signature of the Issuer. The simplest real-world

examples of a Verifies can be a recruiter that verifies your educational background or a voting platform that verifies your age.



Semaphore

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



Semaphore identities

Given to all Semaphore group members, it is comprised of three parts: identity commitment, trapdoor, and nullifier.

Create Semaphore identities >

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

const identity = new Identity()

const trapdoor = identity.getTrapdoor()
const nullifier = identity.getNullifier()
const commitment = identity.generateCommitment()
```

Private values

Trapdoor and nullifier values are the private values of the Semaphore identity. To avoid fraud, the owner must keep both values secret.

Public values

Semaphore uses the Poseidon hash function to create the identity commitment from the identity private values. Identity commitments can be made public, similarly to Ethereum addresses.

Generate identities

Semaphore identities can be generated deterministically or randomly. Deterministic identities can be generated from the hash of a secret message.

Circuit

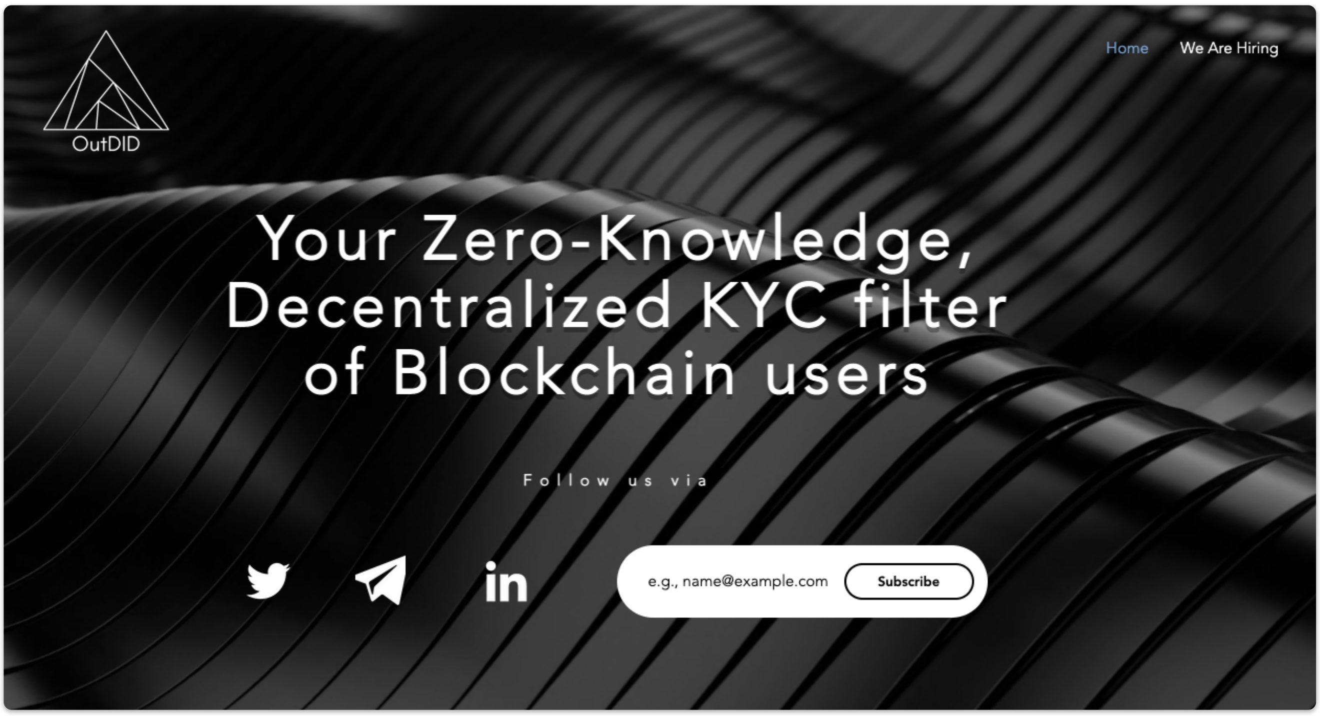
The Semaphore circuit is the heart of the protocol and consists of three parts:

- Proof of membership
- Nullifier hash
- Signal

They provide tools to create and verify proofs.

OutDID.io

See docs



Providing KYC and Identity using Circom with for example passport data.

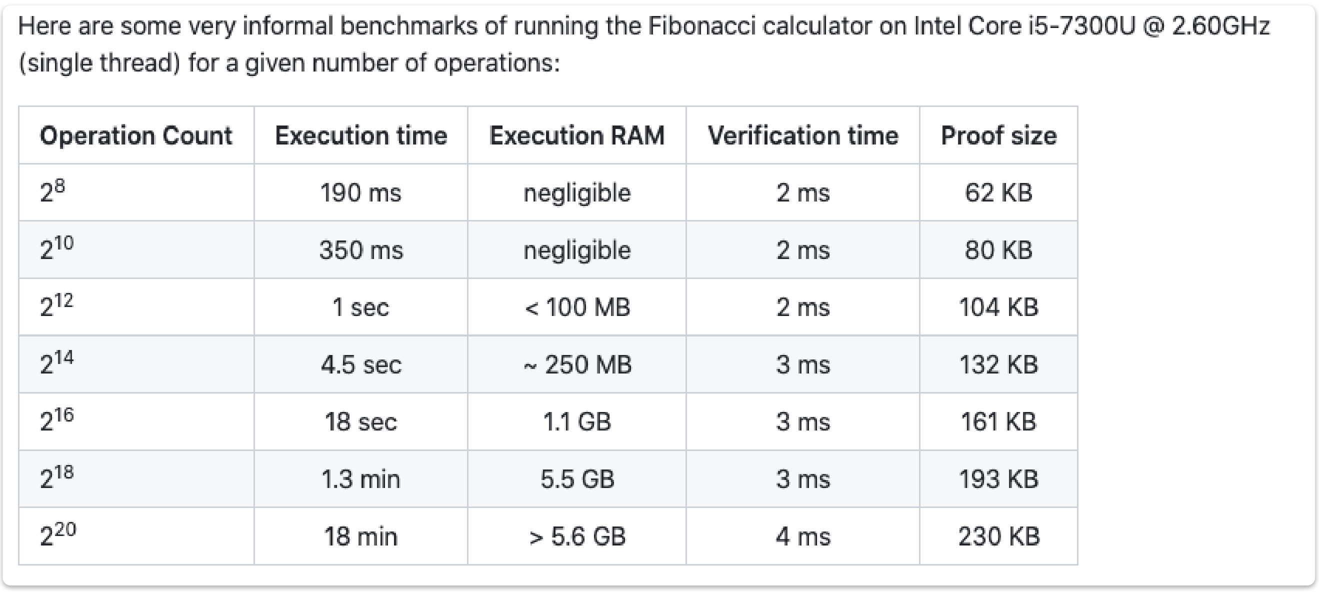
zCloak Network

zCloak Network provides Zero-Knowledge Proof as a Service based on the Polkadot Network

In the 'Cloaking Space' you control your own data and you can run all sorts of computation without sending your data away.
Note that the data stored in the Cloaking Space is not just some arbitrary data on your device, but they are attested by some credible network/organization to garantee its authenticity.
The type of computation can range from

- a regular state transition of a blockchain,
- a check of your income for a bank loan to
- an examination of your facial features to pass an airport checkpoint.

zCloak uses the Distaff VM
Distaff is a zero-knowledge virtual machine written in Rust.
For any program executed on Distaff VM, a STARK-based proof of execution is automatically generated. This proof can then be used by anyone to verify that a program was executed correctly.



Here are some very informal benchmarks of running the Fibonacci calculator on Intel Core i5-7300U @ 2.60GHz (single thread) for a given number of operations:

Operation Count	Execution time	Execution RAM	Verification time	Proof size
2^{8}	190 ms	negligible	2 ms	62 KB
2^{10}	350 ms	negligible	2 ms	80 KB
2^{12}	1 sec	< 100 MB	2 ms	104 KB
2^{14}	4.5 sec	~ 250 MB	3 ms	132 KB
2^{16}	18 sec	1.1 GB	3 ms	161 KB
2^{18}	1.3 min	5.5 GB	3 ms	193 KB
2^{20}	18 min	> 5.6 GB	4 ms	230 KB

Secret Sharing / Multiparty Computation

The goal in secret sharing is to divide secret S into n pieces of data $S_i..S_n$ in such a way that:

Knowledge of any k or more S_i pieces makes S easy to compute. That is, the complete secret S can be reconstructed from any combination of k pieces of data.
Knowledge of any $k-1$ or fewer S_i pieces leaves S completely undetermined, in the sense that the possible values for S seem as likely as with knowledge of 0 pieces.

A naive splitting of a key would just make a brute force attack easier.

Shamir Secret Sharing

Based on the fact the k points are required to define a polynomial of degree $k-1$
With our points being elements in a finite field $\mathbb{F}$ of size P where $0 < klen < P; S < P$ and P is a prime number.

Choose at random $k-1$ positive integers $a_1..a_{k-1}$ with $a_i < P$ and let $a_0 = S$

The person splitting the secret builds a polynomial where the secret is the constant term a_0

$f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{k-1}x^{k-1}$

Let us construct any n points out of it, for instance set $i = 1..n$ to retrieve $(i, f(i))$.
Every participant is given a point (a non-zero integer input to the polynomial, and the corresponding integer output) along with the prime which defines the finite field to use.
Given any subset of k of these pairs, we can find the coefficients of the polynomial using interpolation. The secret is the constant term a_0

Properties of Shamir's (k, n) threshold scheme are:

- Secure: Information theoretic security.
- Minimal: The size of each piece does not exceed the size of the original data.
- Extensible: When k is kept fixed, D_i pieces can be dynamically added or deleted without affecting the other pieces.
- Dynamic: Security can be easily enhanced without changing the secret, but by changing the polynomial occasionally (keeping the same free term) and constructing new shares to the participants.
- Flexible: In organizations where hierarchy is important, we can supply each participant different number of pieces according to their importance inside the organization. For instance, the president can unlock the safe alone, whereas 3 subordinates are required together to unlock it.

Additive Secret Sharing

Given a secret $s \in F$, the dealer D selects $n-1$ random integers
$R = r_1, r_2, r_{n-1}$ uniformly from F.

D then computes

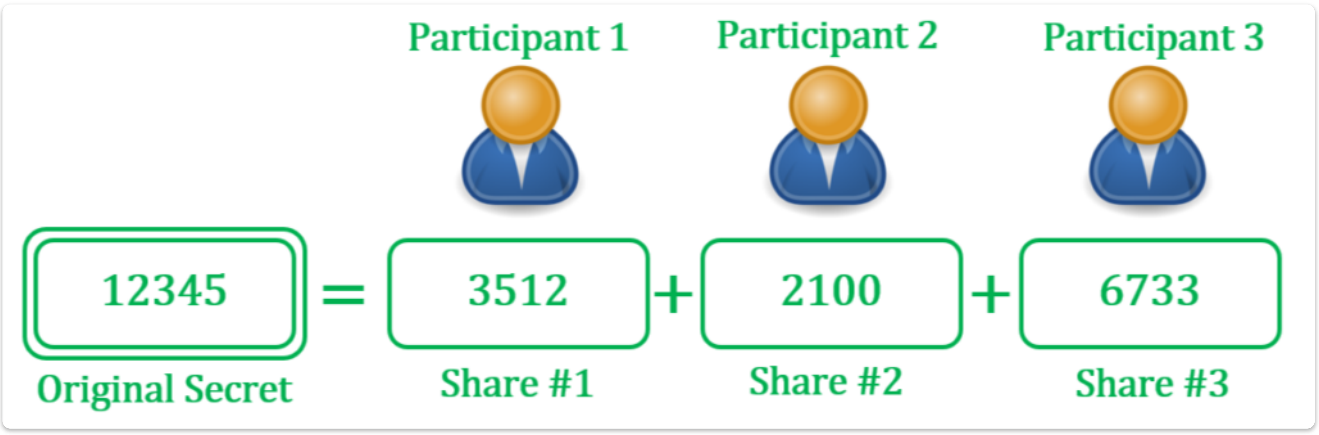
s_n = $s - \sum_{i=1}^{n-1} r_i \; mod \; F$

D sends each player P_i $1 \leq i \leq n-1$ the share $s_i = r_i$, and the share s_n is sent to P_n.

The reconstruction of secret $s \in F$ is trivial; simply add all of the shares together:

s = $\sum_{i=1}^{n} s_i \; mod \; F$

The above additive secret sharing scheme requires all participants to contribute their shares in order to reconstruct the secret.
If one or more of the participants are missing, no information about the original secret can be recovered; such a scheme is known as a perfect secret sharing scheme.



Multiparty computation overview

A key point to understand is that MPC is not a single protocol but rather a growing class of solutions that differ with respect to properties and performance.
However, common for most MPC systems are the three basic roles:

- The Input Parties delivering sensitive data to the confidential computation.
- The Result Parties receiving results or partial results from the confidential computation.
- The Computing Parties jointly computing the confidential computation

In an multi party computation,
a given number of participants, $p_1, p_2, \ldots p_n$,
each have private data, respectively $d_1, d_2, \ldots d_n$.

Participants want to compute the value of a public function on that private data:
$f(d_1, d_2 \ldots d_n)$ while keeping their own inputs secret.

Most MPC protocols make use of a secret sharing scheme such as Shamir Secret Sharing.
The function $f(d_1, d_2 \ldots d_n)$ is used to define an arithmetic circuit over a finite field which consists of addition and multiplication gates.

In the secret sharing based methods, the parties do not play special roles. Instead, the data associated with each wire is shared amongst the parties, and a protocol is then used to evaluate each gate.

Two types of secret sharing schemes are commonly used;

1. Shamir secret sharing
2. Additive secret sharing
   In both cases the shares are random elements of a finite field that add up to the secret in the field; intuitively, security is achieved because any non-qualifying set of shares looks randomly distributed.

Oracle Solutions

Empiric

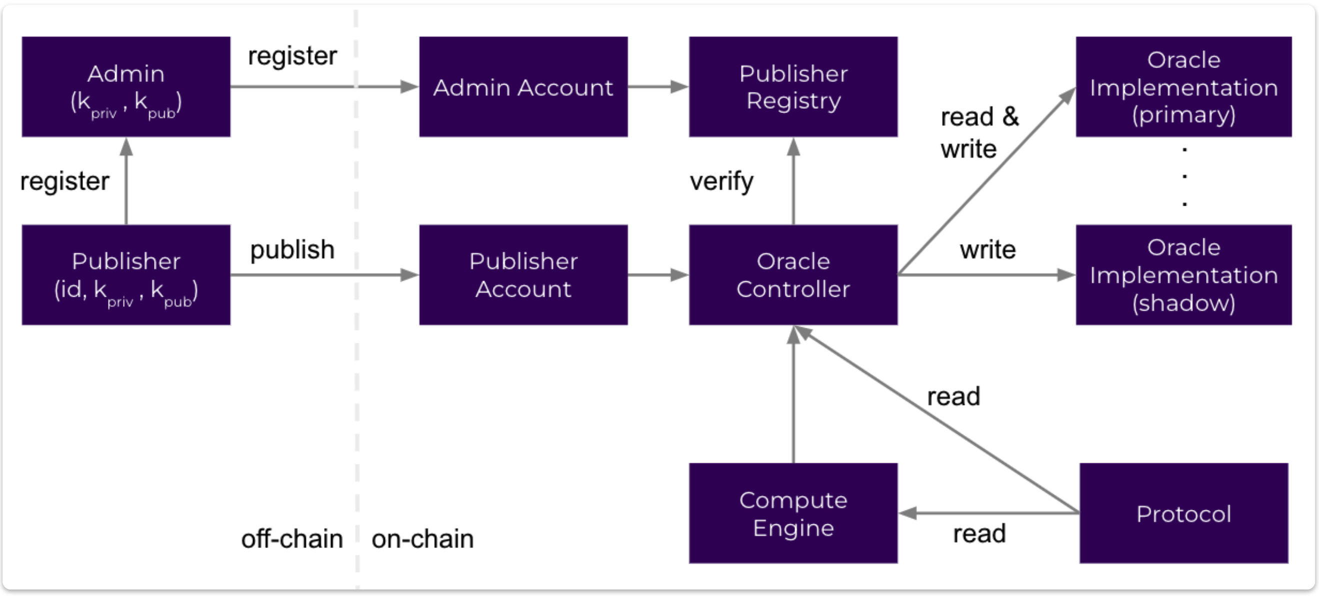
Empiric's Founding Data Partners include Jane Street, Alameda Research, Gemini, FTX, CMT Digital and Flow Traders.

Empiric has been live in stealth for the last few months (on StarkNet testnet) and already integrated with leading protocols such as ZKLend, Magnety, Serity, CurveZero, Canvas, and FujiDAO.

Assets supported

- BTC/USD,
- BTC/EUR,
- ETH/USD,
- ETH/MXN
- SOL/USD,
- AVAX/USD,
- DOGE/USD,
- SHIB/USD,
- BNB/USD,
- ADA/USD,
- XRP/USD,
- MATIC/USD,
- USDT/USD,
- DAI/USD,
- USDC/USD,
- TUSD/USD,
- BUSD/USD,

Contracts



Code snippet

```
%lang starknet

from starkware.cairo.common.cairo_builtins import HashBuiltin

# Oracle Interface Definition
const EMPIRIC_ORACLE_ADDRESS =
0x012fadd18ec1a23a160cc46981400160fbf4a7a5eed156c4669e39807265bcd4
```

```
const KEY = 28556963469423460  # str_to_felt("eth/usd")
const AGGREGATION_MODE = 120282243752302  # str_to_felt("median")

@contract_interface
namespace IEmpiricOracle:
    func get_value(key : felt, aggregation_mode : felt) -> (
        value : felt,
        decimals : felt,
        last_updated_timestamp : felt,
        num_sources_aggregated : felt
    ):
    end
end

# Your function
@view
func my_func{
    syscall_ptr : felt*,
    pedersen_ptr : HashBuiltin*,
    range_check_ptr
}() -> ():
    let (eth_price,
        decimals,
        last_updated_timestamp,
        num_sources_aggregated) = IEmpiricOracle.get_value(
            EMPIRIC_ORACLE_ADDRESS, KEY, AGGREGATION_MODE
        )
    # Your smart contract logic!
    return ()
end
```

DECO

See paper

DECO Short for decentralized oracle, DECO is a new cryptographic protocol that enables a user (or oracle) to prove statements in zero knowledge about data obtained from HTTPS-enabled servers. DECO consequently allows private data from unmodified web servers to be relayed safely by oracle networks. (It does not allow data to be sent by a prover directly on chain.)
DECO has narrower capabilities than Town Crier, but unlike Town Crier, does not rely on a trusted execution environment.

DECO can also be used to power the creation of decentralized identity (DID) protocols such as CanDID, where users can obtain and manage their own credentials, rather than relying on a centralized third party.
Such credentials are signed by entities called issuers that can authoritatively associate claims with users such as citizenship, occupation, college degrees, and more. DECO allows any existing web server to become an issuer and provides key-sharing management to back up accounts, as well as a privacy-preserving form of Sybil resistance based on definitive unique identifiers such as Social Security Numbers (SSNs).

ZKP solutions like DECO benefit not only the users, but also enable traditional institutions and data providers to monetize their proprietary and sensitive datasets in a confidential manner.
Instead of posting the data directly on-chain, only attestations derived from ZKPs proving facts about the data need to be published.
This opens up new markets for data providers, who can monetize existing datasets and increase their revenue while ensuring zero data leakage. When combined with Chainlink Mixicles, privacy is extended beyond the input data executing an agreement to also include the terms of the agreement itself.

A web server itself could assume the role of an oracle, e.g., by simply signing data. However, server-facilitated oracles would not only incur a high adoption cost, but also put users at a disadvantage: the web server could impose arbitrary constraints on the oracle capability.

- Thus a single instance of DECO could enable anyone to become an oracle for any website
- Importantly, DECO does not require trusted hardware, unlike alternative approaches that could achieve a similar vision