

Lesson 15 - Identity Solutions / zkML

Lesson 13 : Stark Theory

Lesson 14 : Cryptographic Alternatives

Lesson 15 : Identity solutions / zkML

Lesson 16 : Research and review

Today's topics

- Identity Solutions
 - zkML
 - Oracles
 - Akworks
 - Halo2
-

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.

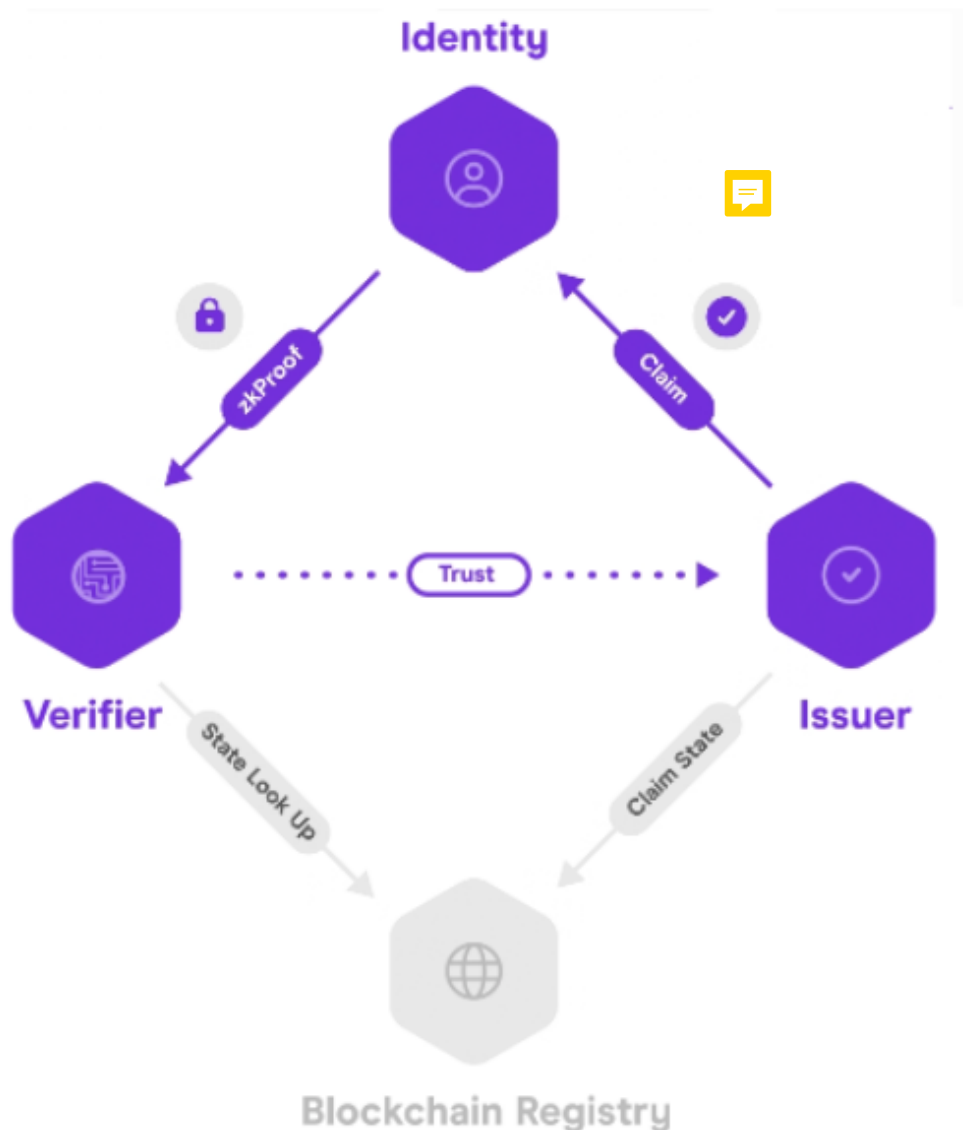
There is an ID client toolkit to facilitate onboarding

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.

Setup

The semaphore CLI will set up a hardhat project

```
npx @semaphore-protocol/cli@latest create  
my-app
```

Example [contract](#)

See [Worldcoin](#)



World ID is a digital passport that lets a user prove they are a unique and real person while remaining anonymous.

This happens through zero knowledge proofs and other privacy-preserving cryptographic mechanisms.

It works with a project as follows

1. The user gets their World ID in a compatible wallet 🗨️
2. The user receives credentials in their World ID. The flagship credential is biometric verification, currently available by using

the [Orb](#). The user can also verify their phone number to obtain the respective credential.

3. A Project integrates with WorldID
4. The user connects their World ID to authenticate, and optionally prove they are a unique human doing something only once. The user's wallet will generate a zero knowledge proof to accomplish this.
5. The project verifies the proof either by using the [API](#) or by verifying it [on-chain](#).

Clique

[See](#)



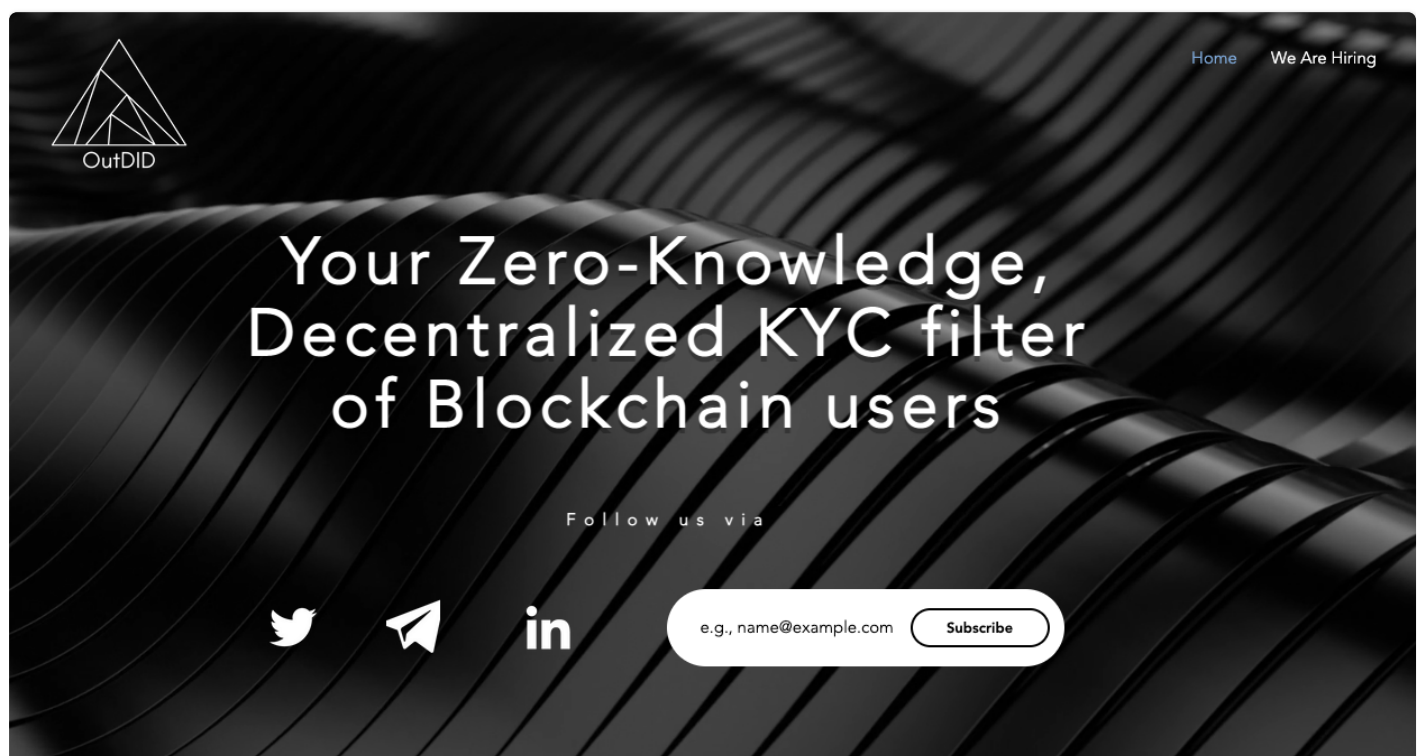
Clique builds identity-oracles for web2 user behaviour data. We provide legacy data pipelines for web3 protocols, so that they can incentivise and engage users on existing web2 platforms.

Clique uses the standardized Twitter and Discord O-Auth tokens to get public information about an account.

None of the private account information is revealed to the protocols.


OutDID.io

[See docs](#)



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

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 guarantee 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.

They also issue a [zkID card](#).

zkID Card

zkID Card Center empowers the issuance and management of W3C-compatible credentials, enabling versatile digital applications and fostering personal data control.

Operating on the W3C standard DID and VC, the zkID Card Center stands as a premier credential issuance platform. It facilitates organizations in issuing and managing digital credentials in line with global standards. These credentials find utility in diverse Web3 scenarios, from memberships to DeFi KYC. They not only streamline identity management for organizations but also empower individuals to harness their personal data while safeguarding their privacy.



Verifiable

Each zkID Card is designed to be validated for its authenticity, offering verifiers an additional layer of confidence.



Interoperable

zkID Cards aim to work in different settings, whether off-chain or on EVM-compatible chains, to help facilitate a smoother identity verification process for users.



Flexible

We provide an option for issuers to customize their templates, so that zkID cards can be better suited to meet diverse real-world needs.


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.

ZK-ML

Machine Learning background

Machine learning, a branch of artificial intelligence, focuses on creating and utilizing algorithms that allow computers to independently learn and adapt from data. Through iterative processes, this leads to the optimization of performance. Large-scale language models such as GPT-4 and Bard epitomize the latest advances in natural language processing, using extensive training data to craft text that closely resembles human language. 

In a neural network, each node functions as a linear regression model, accepting an input and possessing its own error term, or bias term, along with weights.

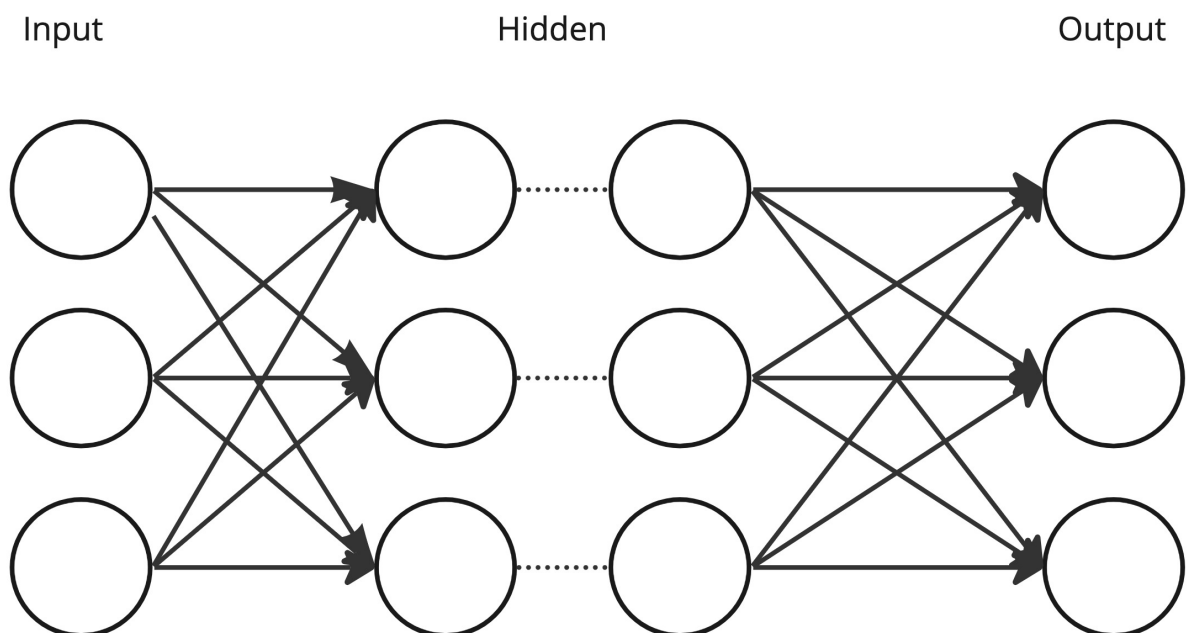
These weights are essential as they define the significance that a specific node assigns to a particular segment of the input.

The node's output is then determined, and an activation function applied to this output ascertains if the neural network will activate or

"fire."

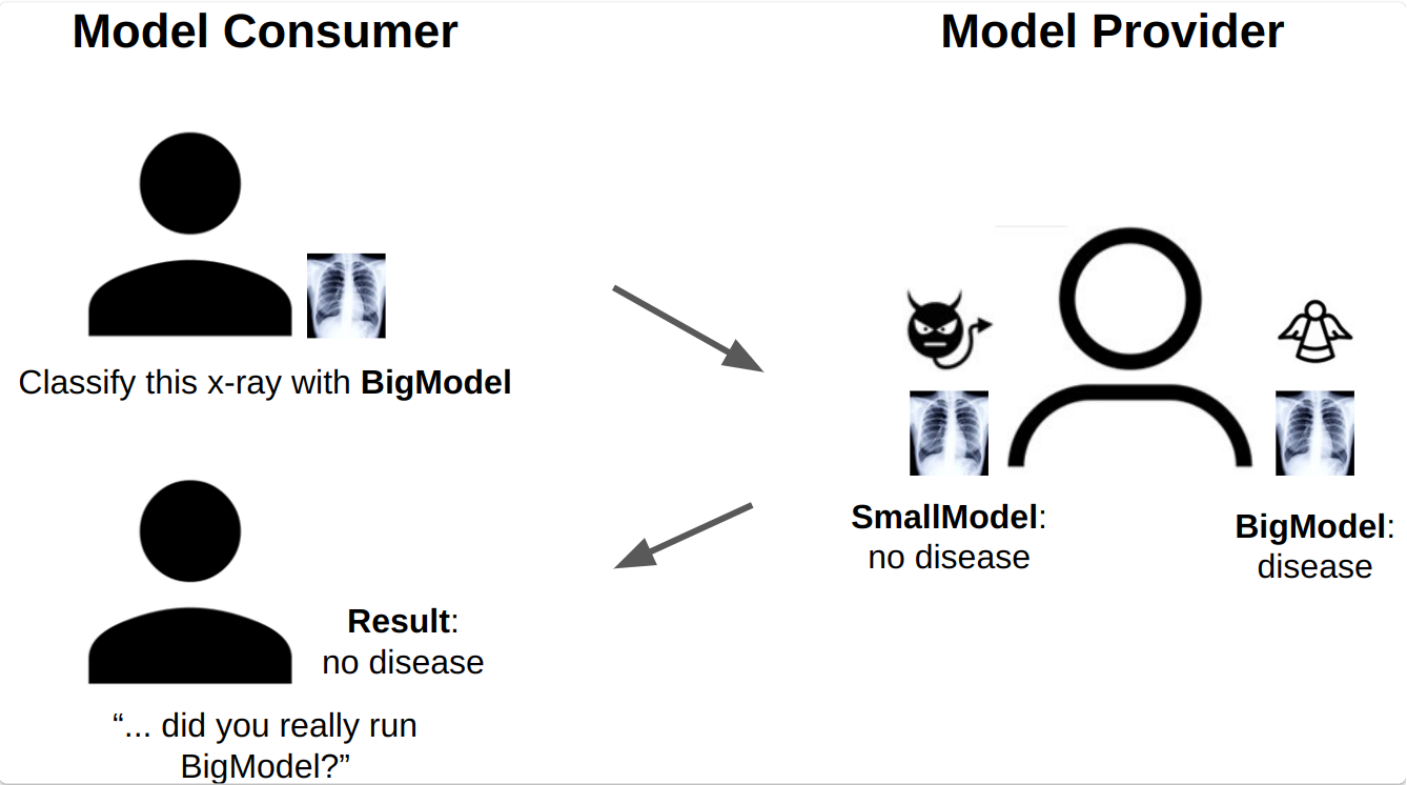
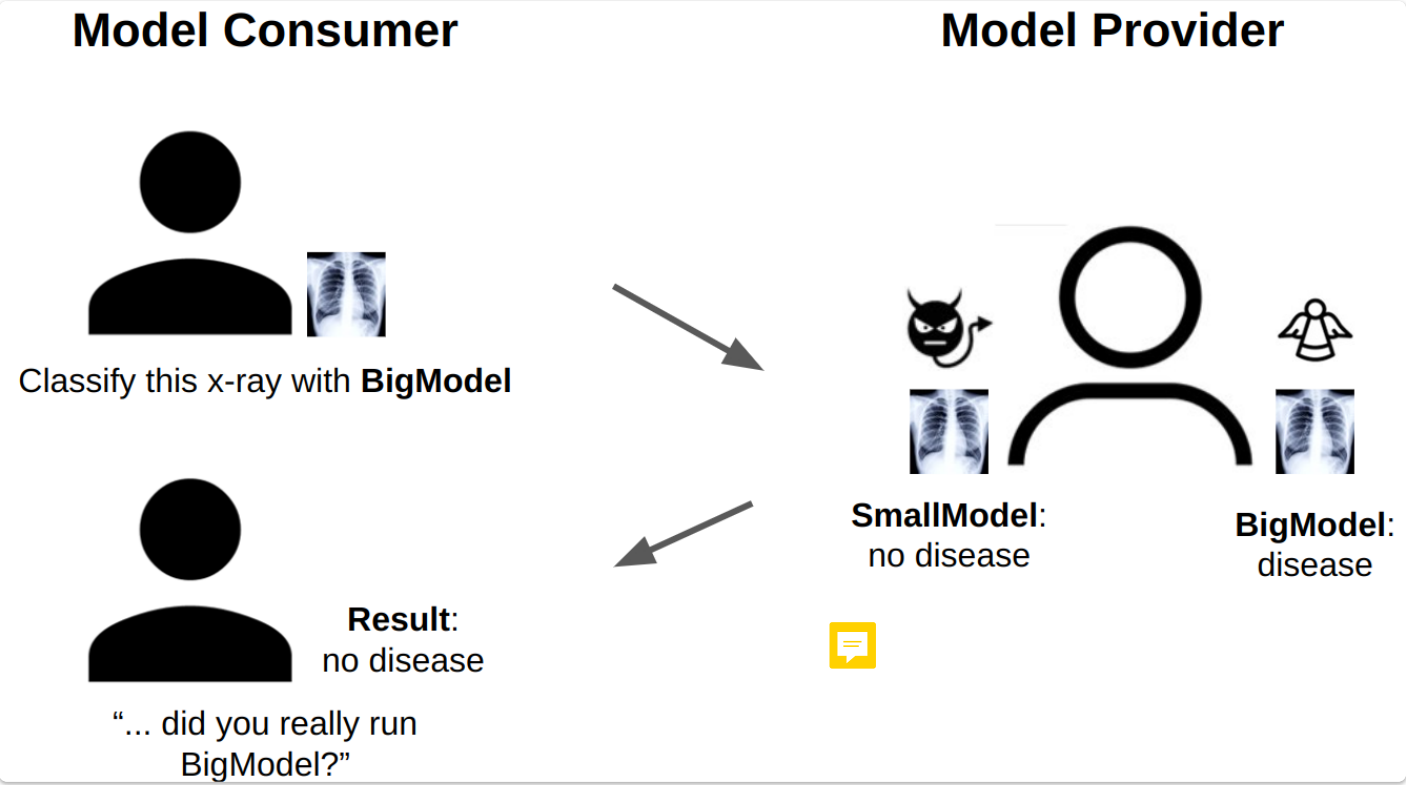
Activation occurs if the output value surpasses a set threshold.

When a node activates or "fires," its output is then used as the input for another node in the network. This process continues with nodes from one layer transmitting data to the succeeding layer, culminating in a chain of data flow. This sequential data transfer from one layer to the next is what gives rise to the concept known as a "feedforward" network.




Training of the network involves 'back propagation' as is many times more expensive than 'feed forward' or inference.

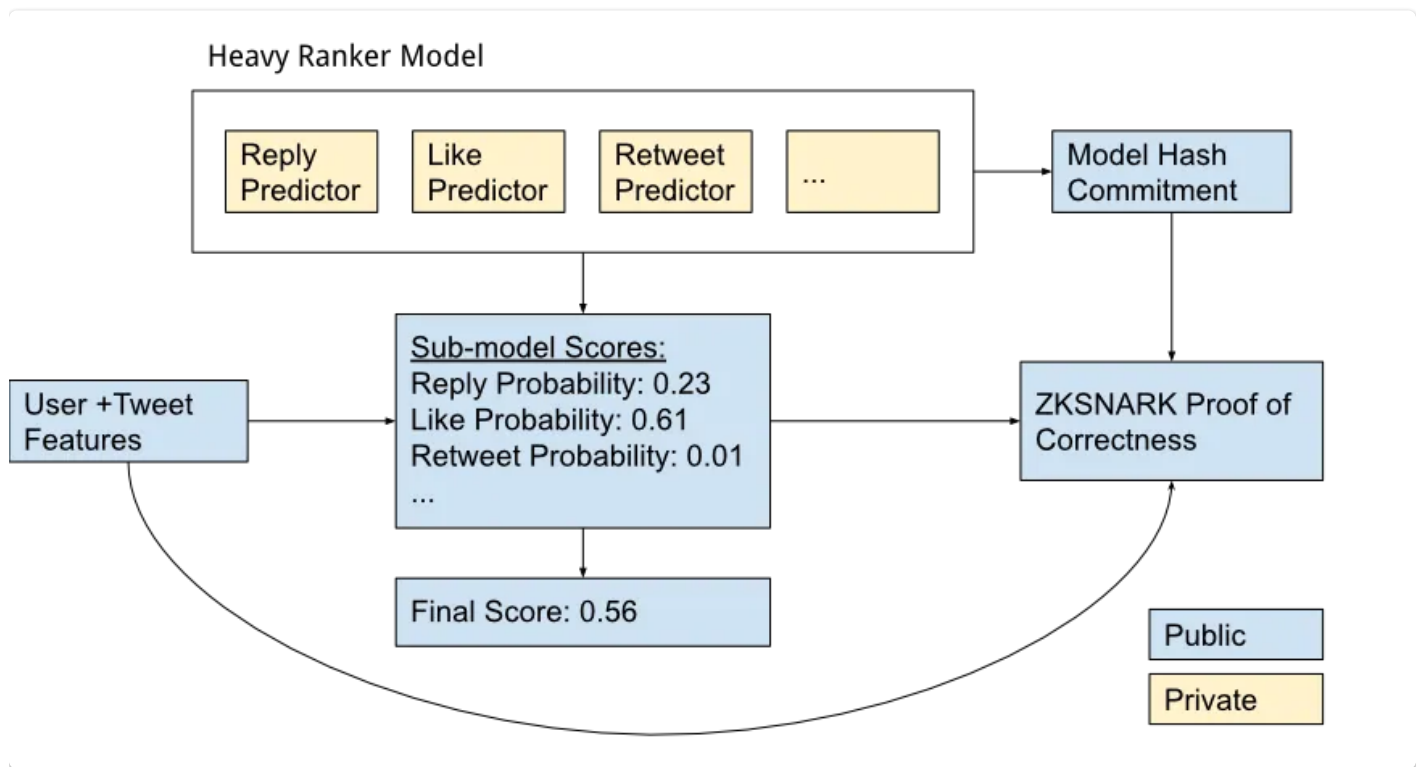
Example use case



Example verifying twitter feeds

See [blog](#)

Twitter uses a ML model to select the tweets shown to you, this project uses zkML to confirm that this is running correctly. 



Unfortunately using zkML for this is impractical
Proving the Twitter model currently takes
6 *hours* to prove for a *single example* using [ezkl](#).
Verifying the tweets published in one second
(~6,000) would cost *~\$88,704* on cloud
compute hardware.

Oracle Solutions

Mina zkOracles

See the notes in the Mina lesson.

Pragma Oracle formerly Empiric

Decentralized & Composable Data Infrastructure

In strategic partnership with  STARKWARE

 ETH/USD

💰 1569.39500

🕒 6:43 min



 BTC/USD

💰 22385.24000

🕒 6:43 min




 SOL/USD

💰 21.31740

🕒 6:43 min



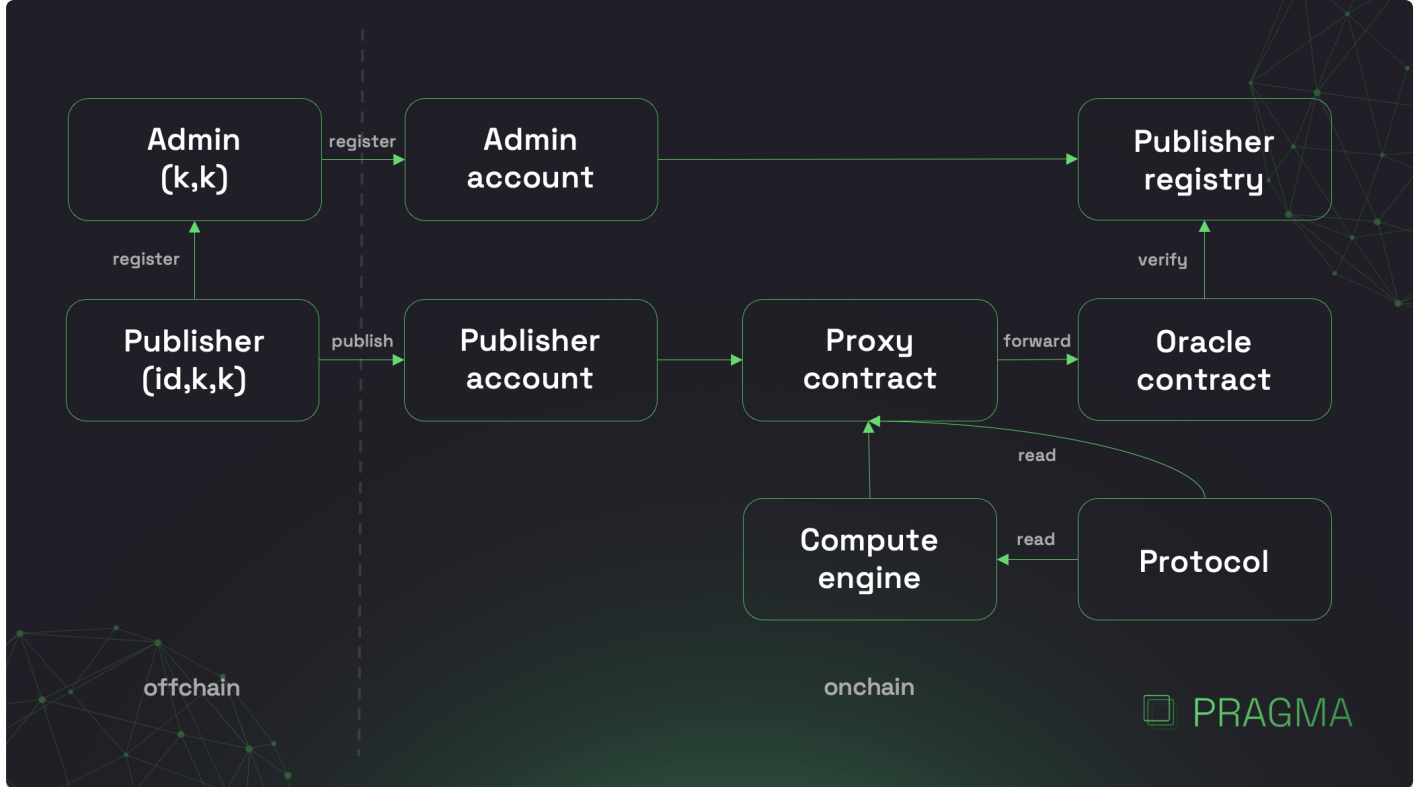
Pragma is available on Starknet and coming soon to Consensys zkEVM and already integrated with leading protocols such as ZKlend, Magnety, Serity, CurveZero, Canvas, and FujiDAO. 

Assets supported on Starknet

- 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](#)



Also see [this video](#) about V2

Code snippet

```
use pragma::oracle::oracle::
{IOracleABIDispatcher,
IOracleABIDispatcherTrait};
use pragma::entry::structs::
{AggregationMode, DataType,
PragmaPricesResponse};
use starknet::ContractAddress;
use
starknet::contract_address::contract_addresses_const;
```

```
const KEY :felt252 = 18669995996566340; //
felt252 conversion of "BTC/USD", can also
write const KEY : felt252 = 'BTC/USD';
```

```
fn get_asset_price_median(oracle_address:
ContractAddress, asset : DataType) -> u128
{
    let oracle_dispatcher =
IOracleABIDispatcher{contract_address :
oracle_address};
    let output : PragmaPricesResponse=
oracle_dispatcher.get_data(asset,
AggregationMode::Median(()));
    return output.price;
}
```

```
//USAGE
```

```
let oracle_address : ContractAddress =
contract_address_const::
<0x620a609f88f612eb5773a6f4084f7b33be06a6fe
d7943445aebce80d6a146ba>();
let price =
```

```
get_asset_price_median(oracle_address,  
DataType::SpotEntry(KEY));
```

Pragma SDK

See [Repo](#)

Installation

```
pip install empiric-network
```

Pragma API



Computational Feeds

You can compose and program data with Cairo, in order to get the right computed data for your protocol.

Empiric has designed compute engines that use the same raw market data underlying our price feeds, but calculate different metrics to produce

feeds of processed data.

For example

- Realised Volatility see [Docs](#)
 - Yield Curve , see [Docs](#)
-



A novel privacy-preserving oracle protocol, created by students and faculty at [IC3](#).

Deco

- works with *modern TLS versions*
- requires *no trusted hardware*
- requires *no server-side modifications*

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.

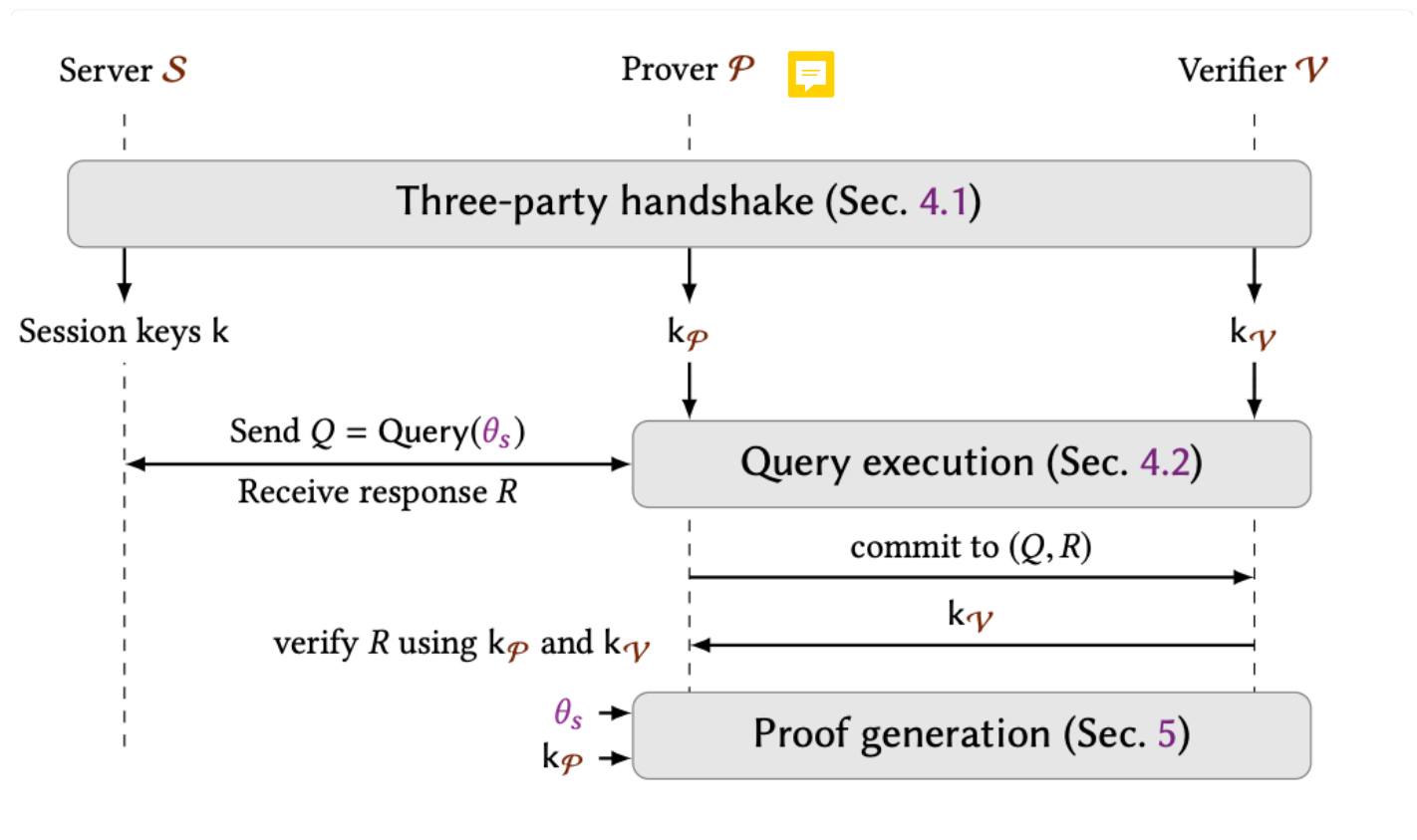
Such credentials are signed by entities called issuers that can authoritatively associate claims with users such as citizenship, occupation, college degrees, and more. DECO allows any existing web server to become an issuer and provides key-sharing management to back up accounts, as well as a privacy-preserving 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.

Process

Diagram from the white paper



Three party handshake

Essentially, P and V jointly act as a TLS client. They negotiate a shared session key with S in a secret shared form

Query Execution

Since the session keys are secret-shared, as noted, P and V execute an interactive protocol to construct a TLS message encrypting the query. P then sends the message to S as a standard TLS client.

P commits to the session data before receiving V's key share, making the commitment unforgeable. Then P can verify the integrity of the response, and prove statements about it.

Proof Generation

With unforgeable commitments, if P opens the commitment to the messages completely (i.e., reveals the encryption key) then V could easily verify the authenticity of the messages by checking MACs on the decryption.

Revealing the encryption key for the messages, however, would breach privacy: it would reveal all session data exchanged between P and S. In theory, P could instead prove any statement about the messages in zero knowledge (i.e., without revealing the encryption key).

Generic zero-knowledge proof techniques, though, would be prohibitively expensive for many natural choices of the statement.

DECO instead introduces two techniques to support efficient proofs for a broad, general class of statement, namely selective opening of a TLS session transcript, see the white paper for details.

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

DECO end-to-end performance depends on the available TLS ciphersuites, the size of private data, and the complexity of application specific proofs.

It takes about 13.77s to finish the protocol, which includes the time taken to generate

unforgeable commitments (0.50s), to run the first stage of two-stage parsing (0.30s), and to generate zero-knowledge proofs (12.97s).

Roadmap

- [Chainlink](#) plans to do an initial PoC of DECO, with a focus on decentralized finance applications such as [Mixicles](#).

For more details see their [blog](#).

Arkworks

`arkworks` is a Rust ecosystem for zkSNARK programming. Libraries in the `arkworks` ecosystem provide efficient implementations of all components required to implement zkSNARK applications, from generic finite fields to R1CS constraints for common functionalities.

[Tutorial](#) includes [Exercises](#) for

1. Merkle Tree
2. Validating a single transaction
3. Writing a rollup circuit

Halo2

See [tutorial](#)

See [documentation](#)

See [Halo2 Book](#)

Halo 2 is a proving system that combines the [Halo recursion technique](#) with an arithmetisation based on [PLONK](#), and a [polynomial commitment scheme](#) based around the Inner Product

Argument

History



Chips

Using our API, we define chips that "know" how to use particular sets of custom gates. This creates an abstraction layer that isolates the

implementation of a high-level circuit from the complexity of using custom gates directly.

Example Simple Circuit

```
trait NumericInstructions<F: Field>:language-rust
  Chip<F> {
    /// Variable representing a number.
    type Num;

    /// Loads a number into the circuit
    as a private input.
    fn load_private(&self, layouter: impl
  Layouter<F>, a: Value<F>) ->
  Result<Self::Num, Error>;

    /// Loads a number into the circuit
    as a fixed constant.
    fn load_constant(&self, layouter:
  impl Layouter<F>, constant: F) ->
  Result<Self::Num, Error>;

    /// Returns  $c = a * b$ .
    fn mul(
      &self,
      layouter: impl Layouter<F>,
      a: Self::Num,
      b: Self::Num,
    ) -> Result<Self::Num, Error>;
```

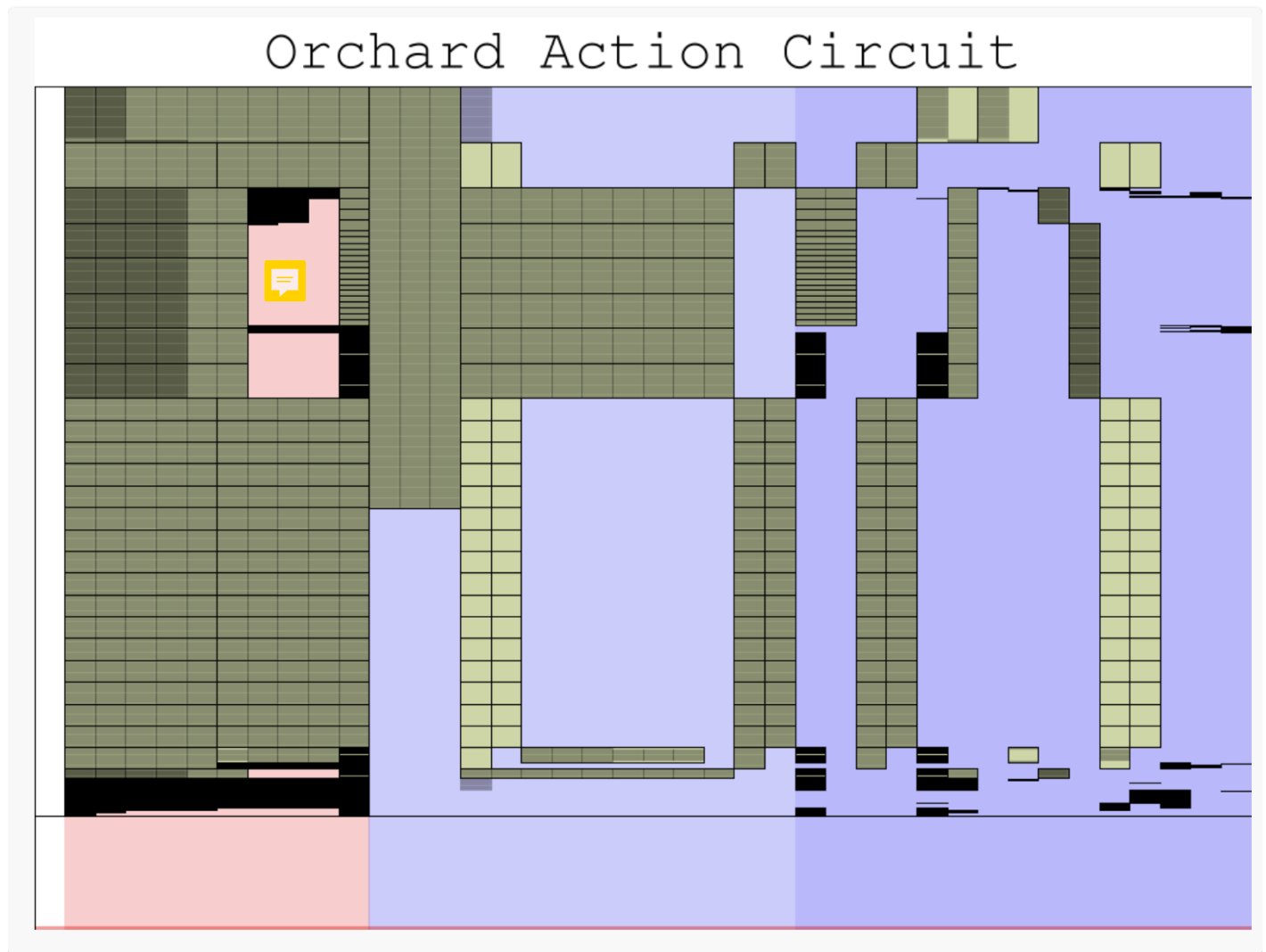
```
    /// Exposes a number as a public
input to the circuit.
    fn expose_public(
        &self,
        layouter: impl Layouter<F>,
        num: Self::Num,
        row: usize,
    ) -> Result<(), Error>;
}
```



Halo 2 circuits are two-dimensional: they use a grid of "cells" identified by columns and rows, into which values are assigned.

Constraints on those cells are grouped into "gates", which apply to every row simultaneously, and can refer to cells at relative rows.

To enable both low-level relative cell references in gates, and high-level layout optimisations, circuit developers can define "regions" in which assigned cells will preserve their relative offsets.



In the example circuit layout pictured, the columns are indicated with different backgrounds.

The instance column in white;

advice columns in red;

fixed columns in light blue; and

selector columns in dark blue.

Regions are shown in light green, and assigned cells in dark green or black.

- Instance columns contain per-proof public values, that the prover gives to the verifier.
- Advice columns are where the prover assigns private (witness) values, that the verifier learns zero knowledge about.
- Fixed columns contain constants used by every proof that are baked into the circuit.
- Selector columns are special cases of fixed columns that can be used to selectively enable gates.

