Lesson 15 - Identity Solutions / zkML / Oracles

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.

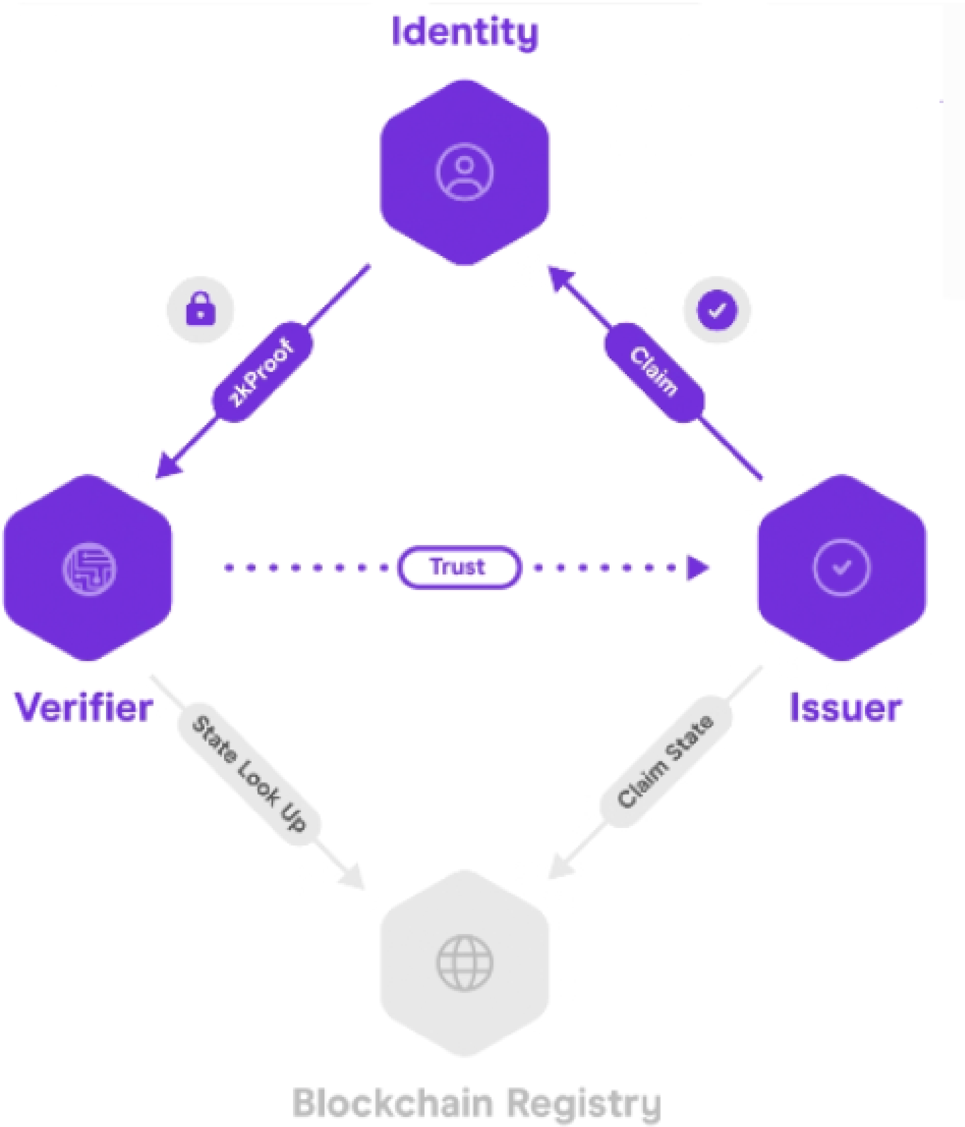
There is an ID client toolkit to facilitate onboarding
On chain verification uses zkProof Request Language, which allows applications to specify which requested private attributes a user needs to prove.

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

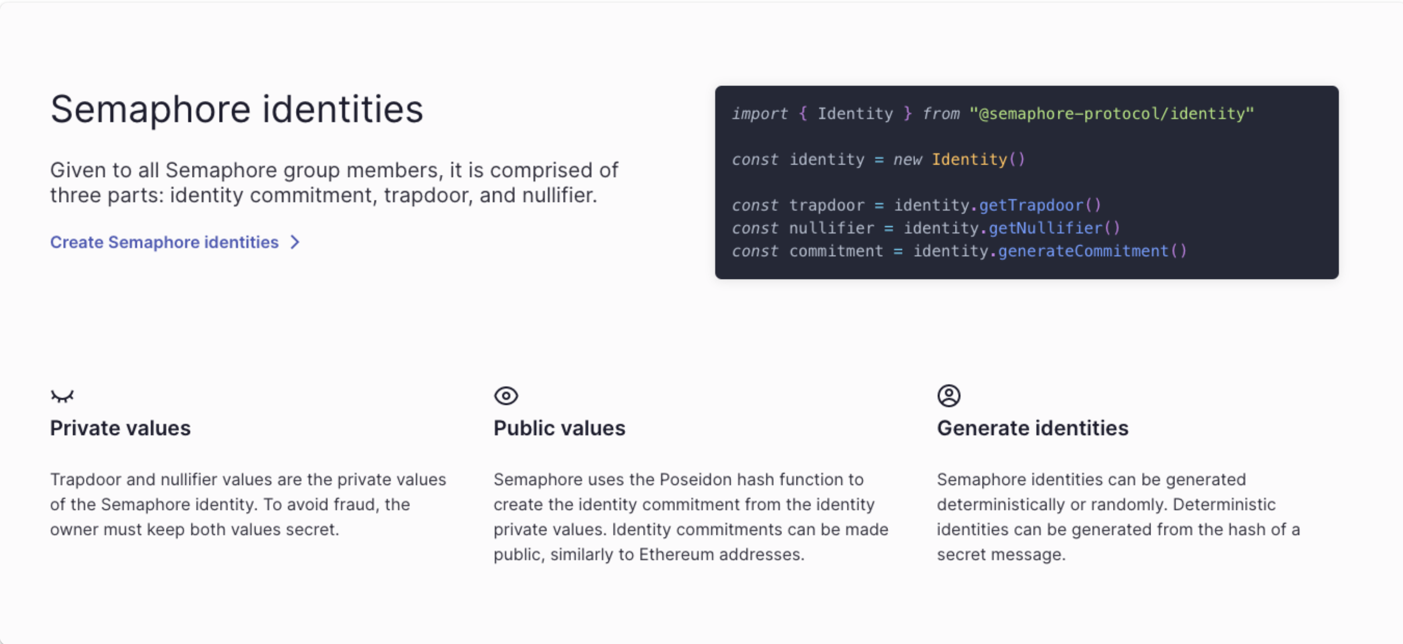
comes from an Issuer.

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.



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

WorldID

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] (https://clique.social/)

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.

Axiom

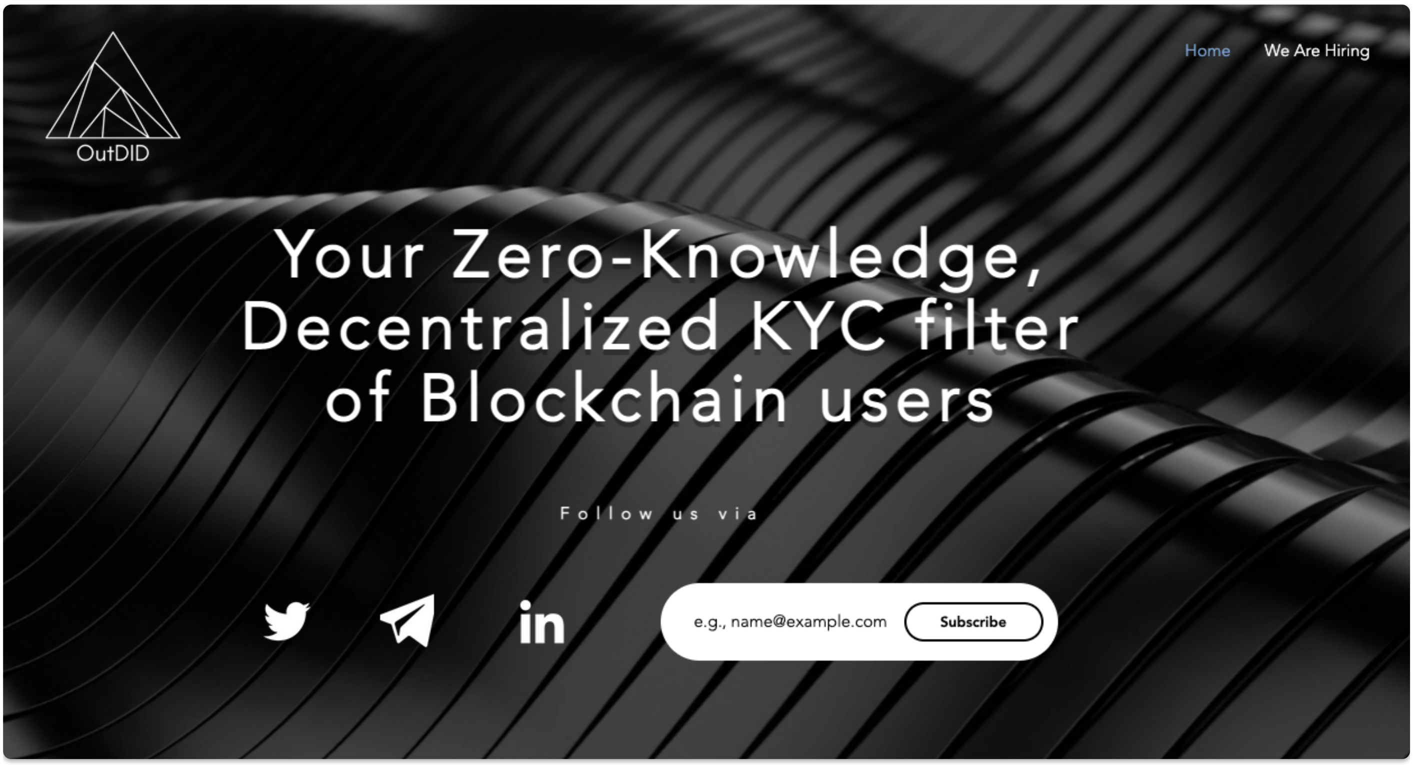
See Demo
Axiom is a ZK coprocessor designed for Ethereum. It allows smart contracts to access on-chain data in a trustless manner and perform various computations on that data. Developers can submit queries to Axiom and utilise the ZK-verified results directly in their smart contracts.

Axiom operates in three steps:

1. Read: Axiom employs ZK proofs to securely retrieve data from block headers, states, transactions, and receipts in past Ethereum blocks. Since all on-chain data is stored in one of these forms, Axiom can access any information available to archive nodes.
2. Compute: Once the data is obtained, Axiom applies verified compute operations on top of it. These operations range from basic analytics like sum, count, max, and min, to cryptographic tasks such as signature verification and key aggregation, as well as machine learning algorithms like decision trees, linear regression, and neural network inference. Each compute operation's validity is confirmed through a ZK proof.
3. Verify: Axiom provides a ZK validity proof with the result of each query, ensuring two things: (1) the input data was accurately fetched from the chain, and
(2) the compute operations were correctly executed. This ZK proof is then verified on-chain within the Axiom smart contract, making the final result securely available for use by other smart contracts downstream.

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.

ZK-ML

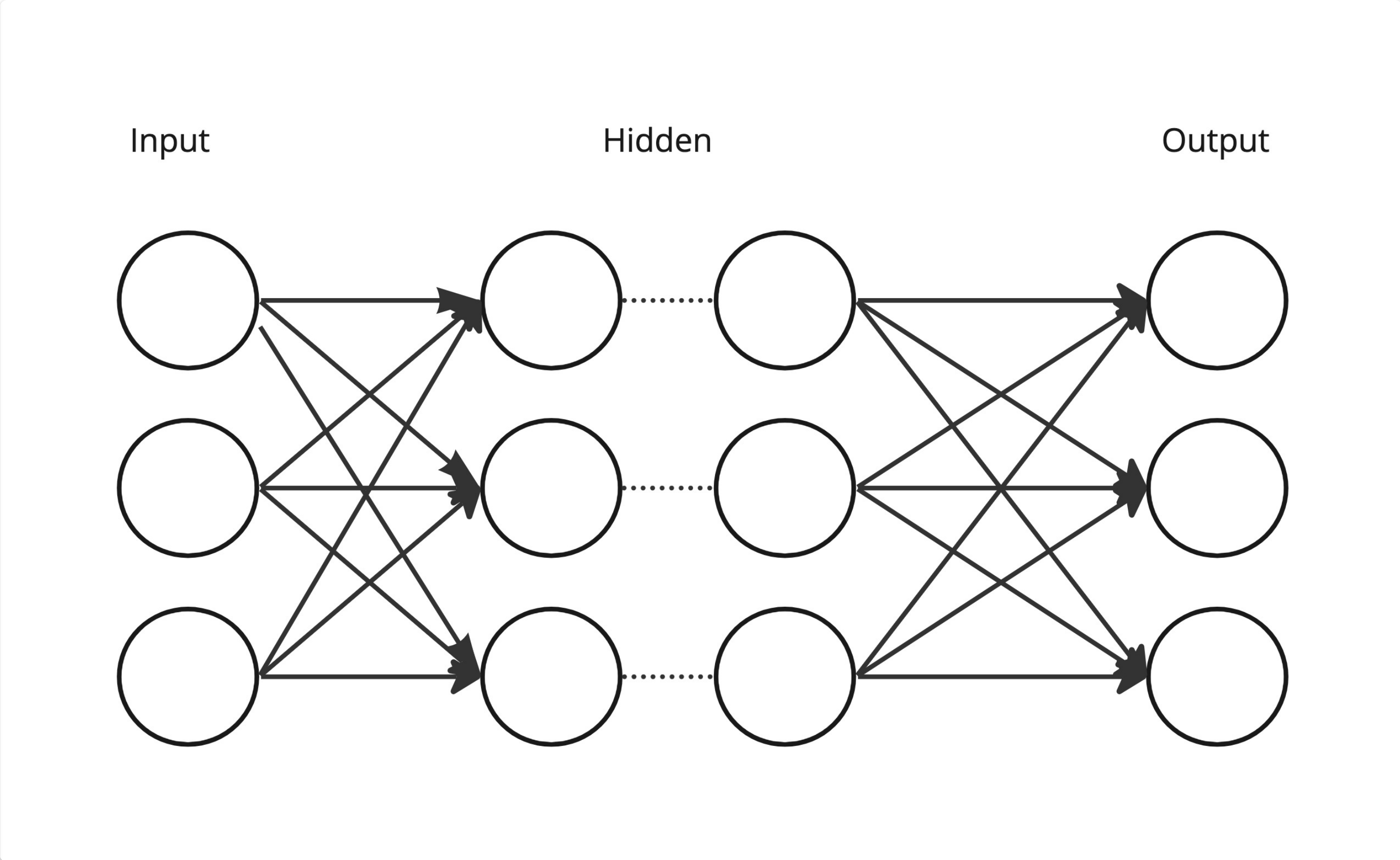
Machine Learning background

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.

The fast-paced progress in machine learning methodologies offers enormous potential to tackle intricate problems in various fields such as healthcare, finance, and transportation.

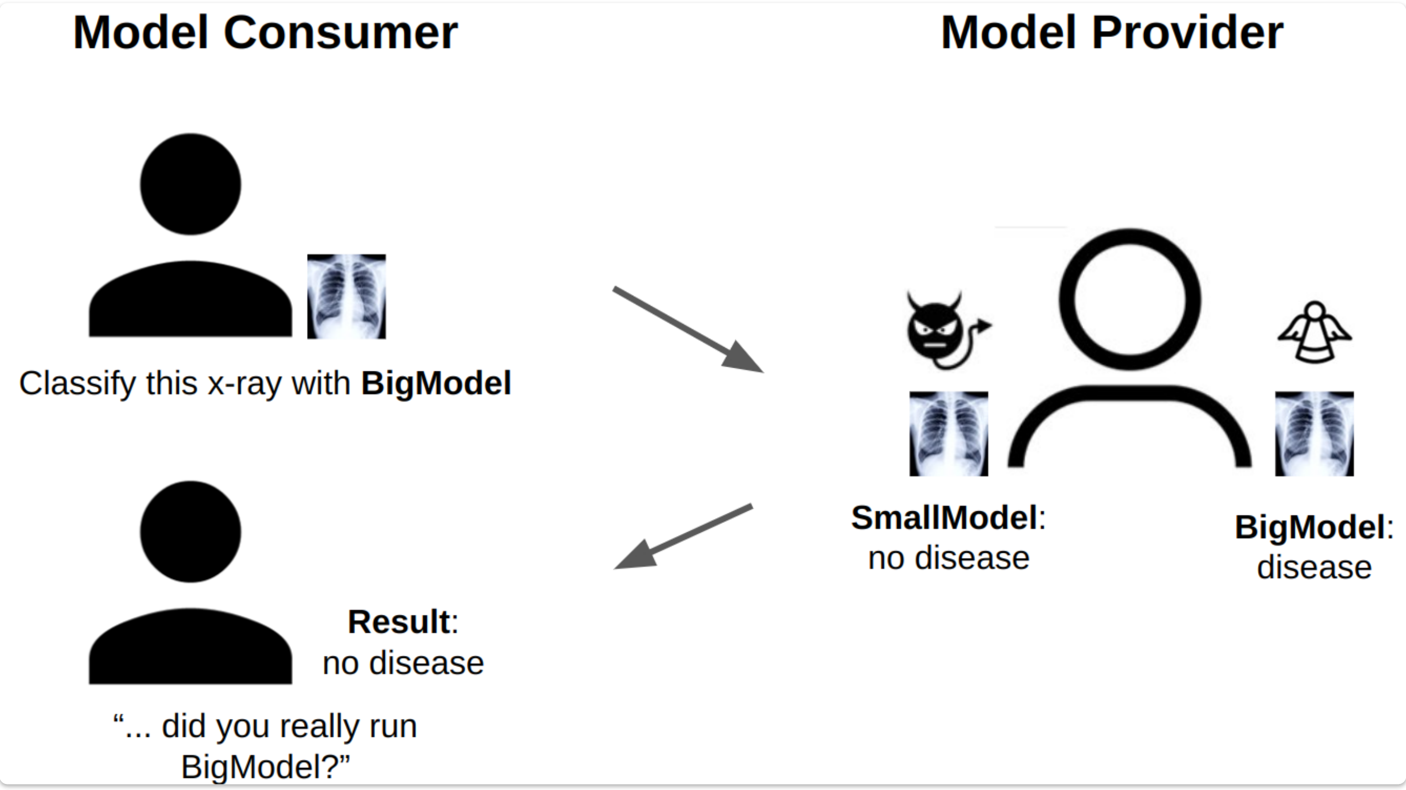
In a neural network, each node functions as a linear regression model, accepting an input and possessing its own error term, or bias term, along with weights. These weights are essential as they define the significance that a specific node assigns to a particular segment of the input.
The node's output 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.

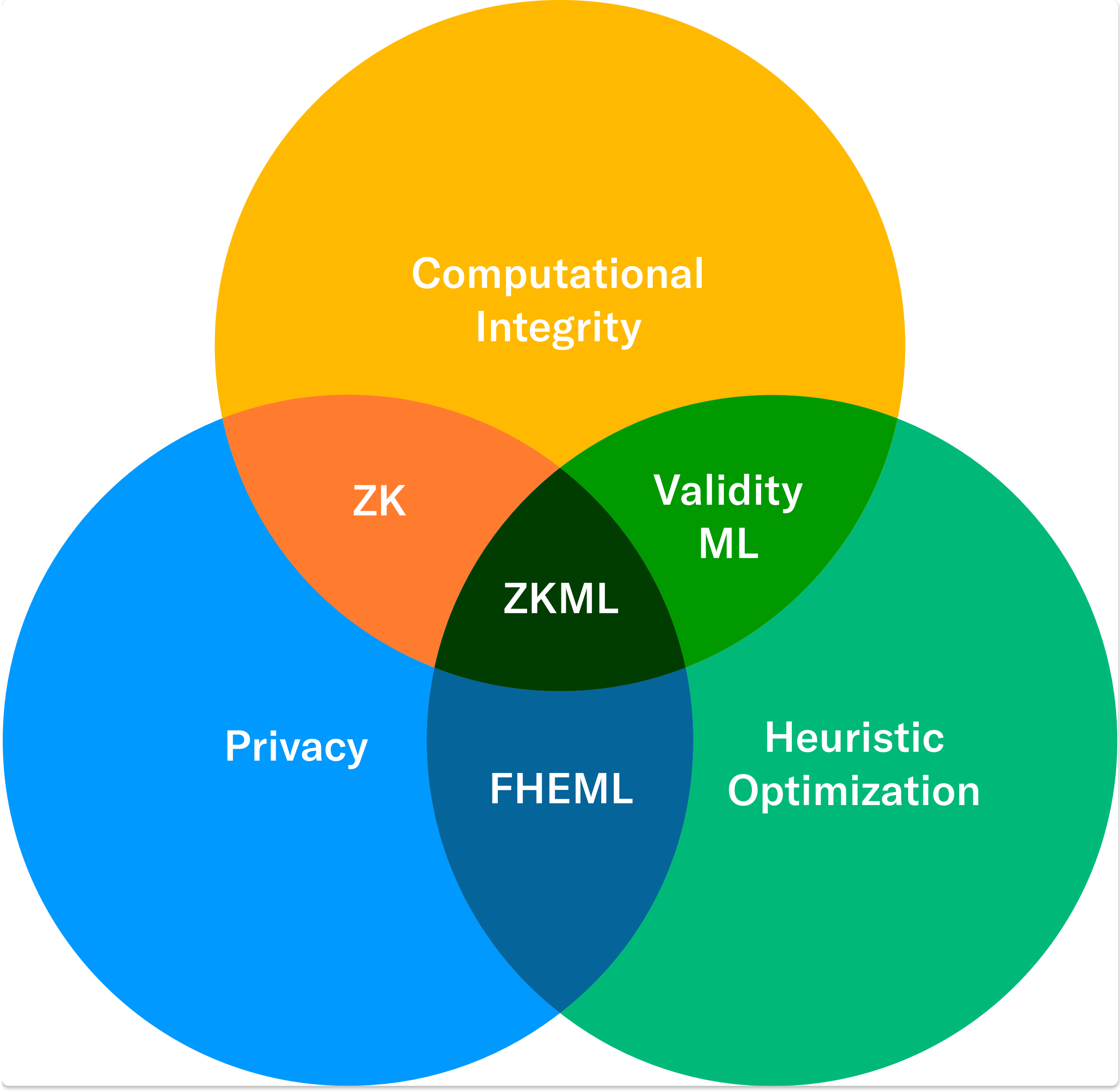


Training of the network involves 'back propagation' as is many times more expensive than 'feed forward' or inference.

Example use case



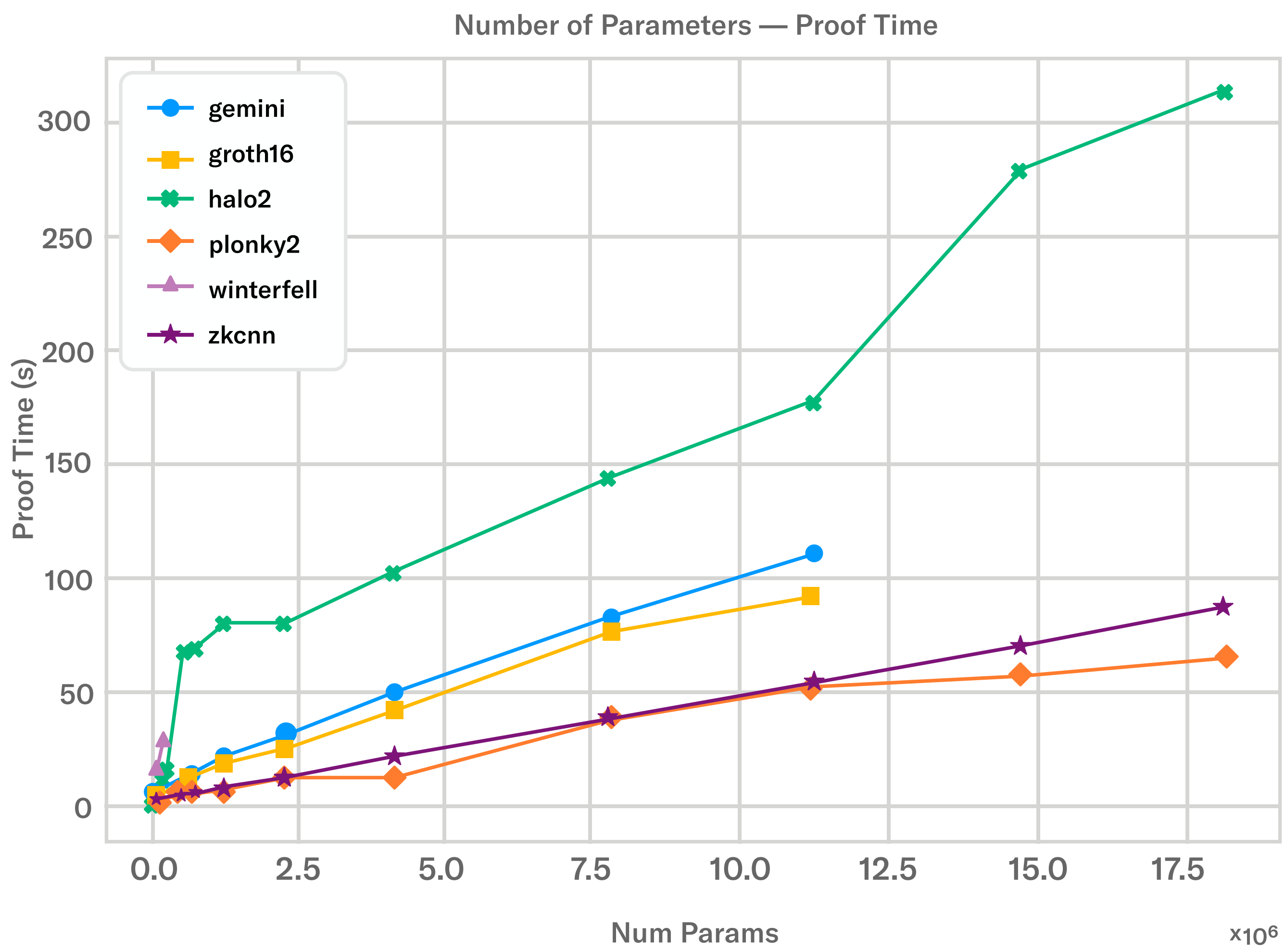
From introduction [blog]



The current state of zero-knowledge systems, even with high-performance hardware, is insufficient to handle large language models. But progress is being made with smaller models.

The Modulus Labs team recently released a paper titled "The Cost of Intelligence", where they benchmark existing ZK proof systems against a wide range of models of different sizes. It is currently possible to create proofs for models of around 18M parameters in about 50 seconds running on a powerful AWS machine using a proving system like plonky2.

Figure 1 illustrates the scaling behaviour of different proving systems as the number of parameters of a neural network are increased:"

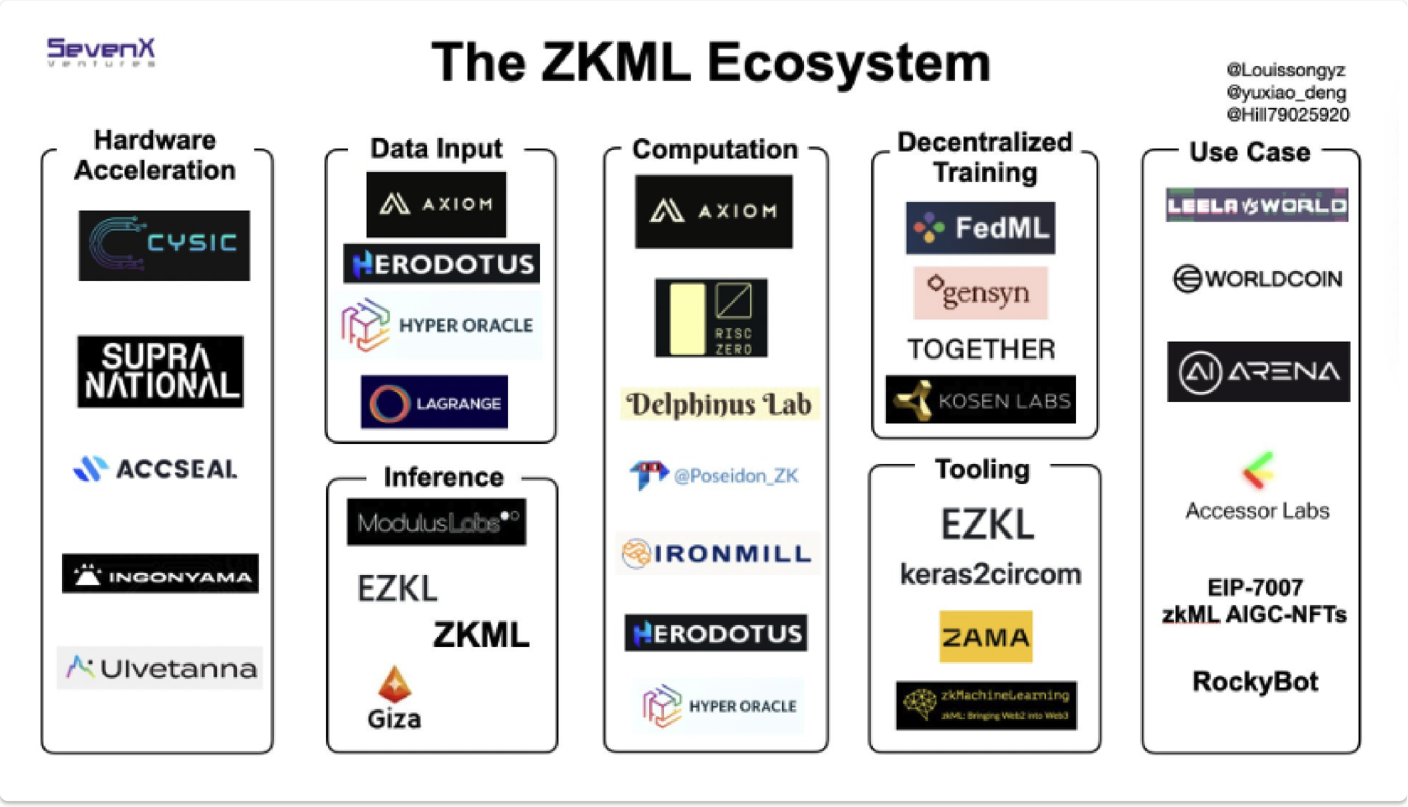


Modulus Labs

Modulus Labs are working on a number of use cases :

- On-chain verifiable ML trading bot - RockyBot
- Blockchains that self-improve vision
- Enhancing the Lyra finance options protocol AMM with intelligent features
- Creating a transparent AI-based reputation system for Astraly (ZK oracle)

Ecosystem



Approaches to modelling a neural network

From article by Pratyush Ranjan Tiwari

1. Naive implementation using addition/multiplication gates: The circuit would be very simple and any proof protocol could be utilised but the size of the circuit is very large $O(n^2 \cdot w^2)$ for image matrix being $n \times n$ and the weight matrix being $w \times w$. Making this method inefficient for any powerful CNN with more than a couple layers.
2. Compute convolutions using FFT: This was demonstrated by an S&P (Oakland) '20 paper ZXZS20 and results in a circuit of size $O(n^2 \log n)$ with $O(\log n)$ depth.
   Another variant of this approach was followed by the zkCNN paper by utilizing a sumcheck protocol for FFT, which is asymptotically optimal with prover time $O(n^2)$.
3. Convolution $\approx$ Polynomial multiplications: This approach was demonstrated by the vCNN paper. It uses the elegant approach of first computing the result of the convolution outside the circuit and then given the result as input, it checks equality of polynomial multiplication.
   The equality check is done by checking equality and random points, the security comes from the Schwartz-Zippel Lemma. This is the most efficient

approach as the zk proof circuit evaluating polynomials at random points is of size $O(n^2 + w^2)$.

A major problem with these approaches is the need to convert floating point numbers to integers for the proofs.
Similarly efficiently proving the correctness of matrix multiplication is difficult.

Some possible workarounds are 'quantise' the weights so that they can be represented as elements in a finite field. This still has the problem of the modular nature of the field.

Weightless networks

See article
An approach from the zero gravity team winners of the ZK Lisbon hackathon '23 was use a neural net design that didn't use weights.
From their description

"We propose a different approach: let's go back to a time before the NN paradigm was settled, to a time when a greater variety of neural nets roamed the earth, and let's find a machine learning model more amenable to ZKP. One such model is the “Weightless Neural Network”.

Weightless means no weights, no floating point arithmetic, and no expensive linear algebra, let alone non-linearities

A Weightless Neural Network (WNN) is entirely combinatorial. Its input is a bitstring (e.g. encoding an image), and their output is one of several predefined classes, e.g. corresponding to the ten digits. It learns from a dataset of (input, output) pairs by remembering observed bitstring patterns in a bunch of devices called RAM cells, grouped into “discriminators” that correspond to each output class. RAM cells are so called since they are really just big lookup tables, indexed by bitstring patterns, and storing a 1 when that pattern has been observed in an input string that is labeled with the class of this discriminator.

Zero Gravity is a system for proving an inference run (i.e. a classification) for a pre-trained, public WNN and a private input. In Zero Gravity, the prover claims to know an input bitstring x such that the public model classifies it as class y. The input x can be treated as a private input, in which case the system is zero-knowledge: although inference does reveal something about x to the verifier (namely its corresponding output class y), this information is already contained in the statement being proved.

EZKL

See Repo

`ezkl` is a library and command-line tool for doing inference for deep learning models and other computational graphs in a zk-snark (ZKML). It enables the following workflow:

1. Define a computational graph, for instance a neural network (but really any arbitrary set of operations), as you would normally in pytorch or tensorflow.
2. Export the final graph of operations as an .onnx file and some sample inputs to a `.json` file.
3. Point `ezkl` to the `.onnx` and `.json` files to generate a ZK-SNARK circuit with which you can prove statements such as:
   Halo2 is used as a backend.

ddkang/zkml

This is a project for constructing proofs of ML model execution in ZK-SNARKs.
See paper and Blog post Trustless verification

> ## Giza
>
> ### Applications
>
> - **Smart Contracts Enhanced by AI**: StarkNet smart contracts can now incorporate machine learning (ML) elements into their operational logic.
> - **Gaming with AI Agents**: Integrate AI-driven characters that exist and interact entirely on-chain within games developed on StarkNet.
> - **Machine Learning Inference on Ethereum's L1 Layer**: Execute inferences on StarkNet, sending the outcomes to L1 for decision-making within L1 contracts using results from the implemented models.
> - **Web2 Inference Delivery via Oracle**: Deploy a robust, fully accessible, censorship-resistant model that is both traceable and efficient for inference services in web2 solutions, facilitated by an oracle.
>
> ### Targeted Initiatives

Orion: ONNX Runtime

ONNX Runtime serves as a multi-platform accelerator for machine learning models, designed with a versatile interface to allow for integration with hardware-specific libraries. It's compatible with various frameworks including PyTorch, Tensorflow/Keras, TFLite, scikit-learn, and more.

With Orion, a new Cairo 1.0-based ONNX runtime is introduced, aiming to supply a verifiable execution environment for ML model inferences utilizing STARKs.

Collaboration with Kleros

Orion's ability to translate machine learning languages into a certifiable smart contract language enables Giza to bring simple ML models on-chain, thereby significantly enhancing the functionality of smart contracts.

Structured as a Schelling-point coordination game, Kleros Court calls upon random token holders to serve as jurors. Their votes on the given case are guided by specific rules and evidence. Inconsistent and wrong decisions are minimized through a sturdy appeals process and crypto-economic incentives, leading to the majority vote forming the case's final ruling.

Applications of Kleros have expanded into areas such as DeFi insurance claims, eCommerce dispute resolution, DAO governance, identity verification, data curation, and social recovery of assets.

Humanity Verification Process

Kleros has innovated a decentralized method for authenticating user-submitted videos, in which jurors wager on the authenticity of submissions—whether they are fraudulent or genuine.

The utilization of on-chain ML can markedly boost efficiency and lower the expenses related to human judgment, thus facilitating the large-scale implementation of social authentication layers. By integrating supplementary data sources like wallet transaction history and image recognition, ML models are able to generate scores that guide human decision-making where applicable.

Axiom

The ZK Coprocessor for Ethereum.

Axiom is a ZK coprocessor for Ethereum which provides smart contracts trustless access to all on-chain data and arbitrary expressive compute over it. Queries into Axiom are trustlessly fulfilled with ZK-verified results on-chain.
To fulfill queries, Axiom performs three steps:

- Read: Axiom uses ZK proofs to trustlessly read from block headers, states, transactions, and receipts in any historical Ethereum block. All Ethereum on-chain data is encoded in one of these forms, meaning that Axiom can access anything an archive node can.
- Compute: Once data has been ingested, Axiom applies verified compute primitives on top. This includes diverse operations from basic analytics (sum, count, max, min) to cryptography (signature verification, key aggregation) and machine learning (decision trees, linear regression, neural network inference). The validity of each computation is verified in a ZK proof.
- Verify: Axiom accompanies each query result with a ZK validity proof that (1) the input data was correctly fetched from the chain and (2) the compute was correctly applied. This ZK proof is verified on-chain in the Axiom smart contract, and the final result is then trustlessly available for use in your smart contract.

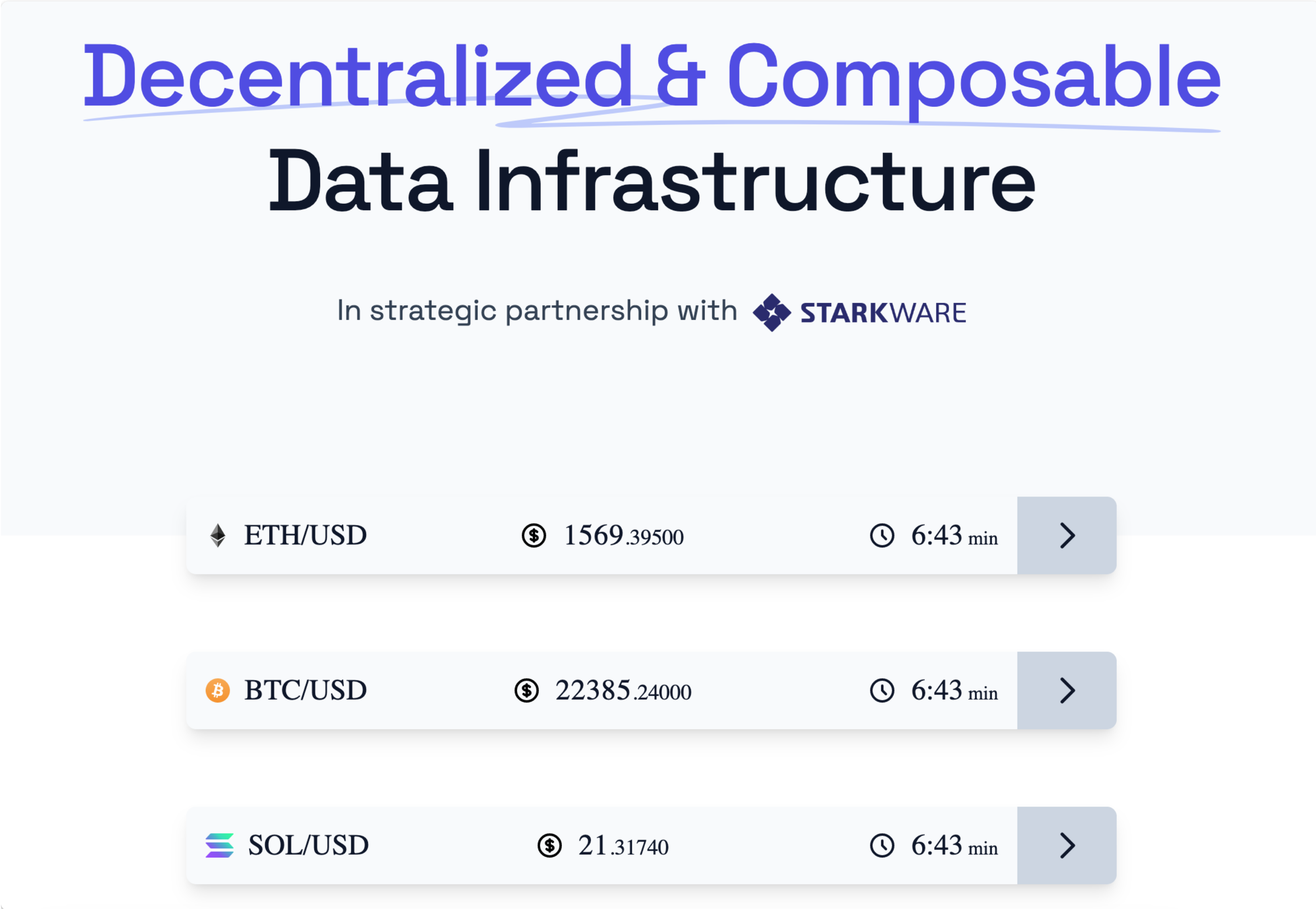
Ingonyama - Hardware for zkML

See Site

> ### zkML Resources
>
> dcbuilder has many resources
> Introduction to zkML from Worldcoin
> Zero Knowledge Podcast
> Image redacting using zkSNARKS paper
> ZK ML community [calls](- ZKML community call #0) on telegram
> Research from PSE team at EF

Oracle Solutions

Empiric



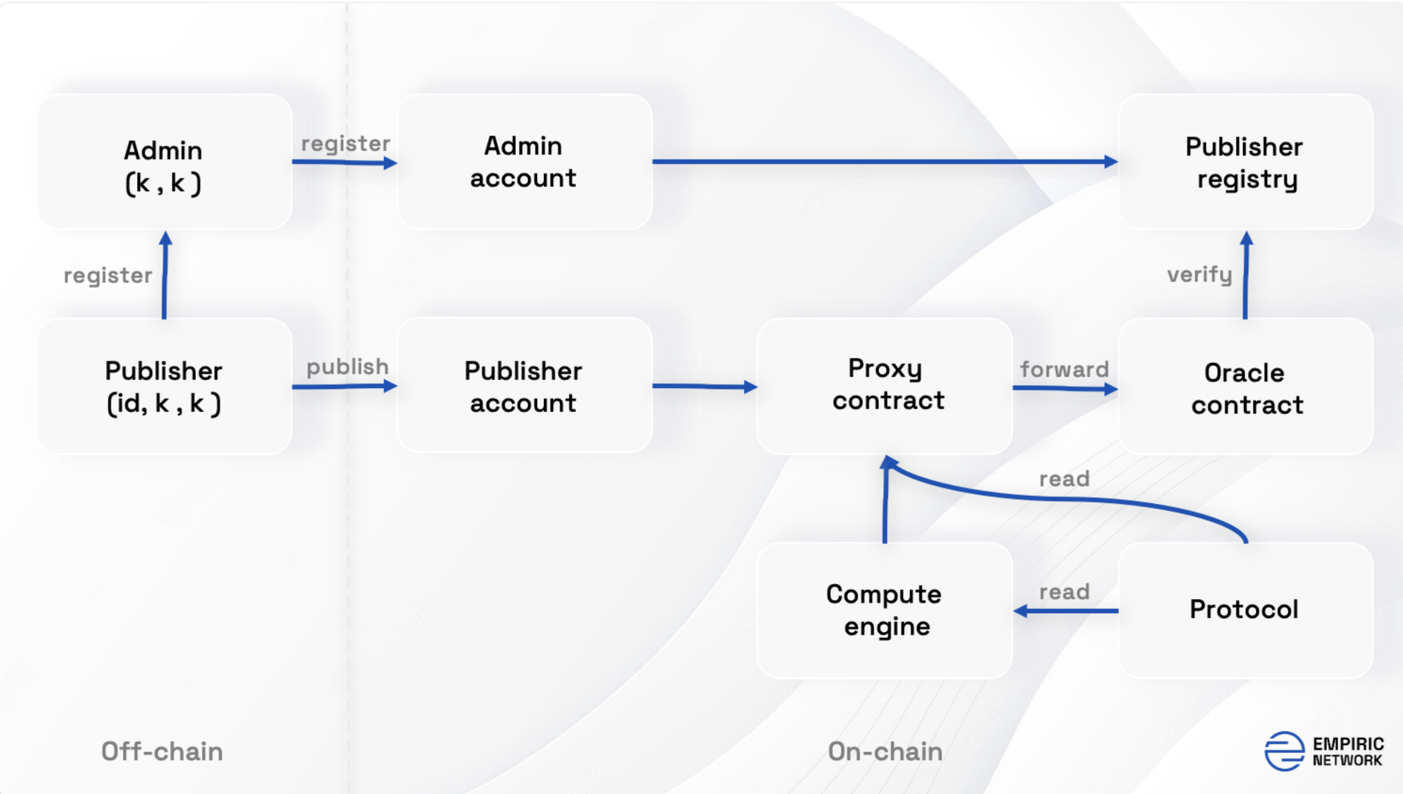
Empiric is available on Starknet and coming soon to Consensys zkEVM and already integrated with leading protocols such as ZKLend, Magnety, Serity, CurveZero, Canvas, and FujiDAO.

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



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
```

Empiric SDK

Installation

`pip install empiric-network`

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

DECO

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.

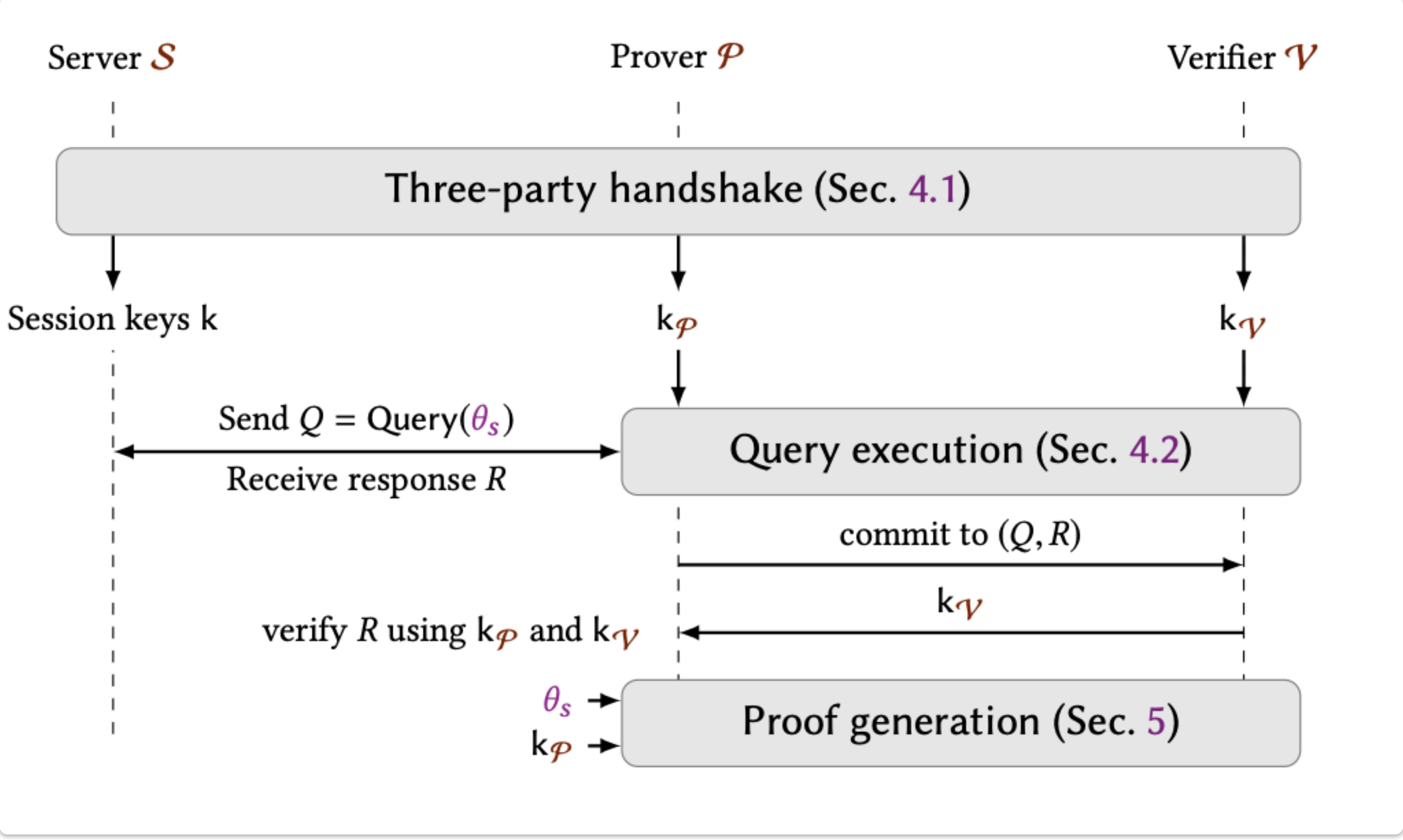
DECO can also be used to power the creation of decentralized identity (DID) protocols such as CanDID, where users can obtain and manage their own credentials, rather than relying on a centralized third party.
Such credentials are signed by entities called issuers that can authoritatively associate claims 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

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.

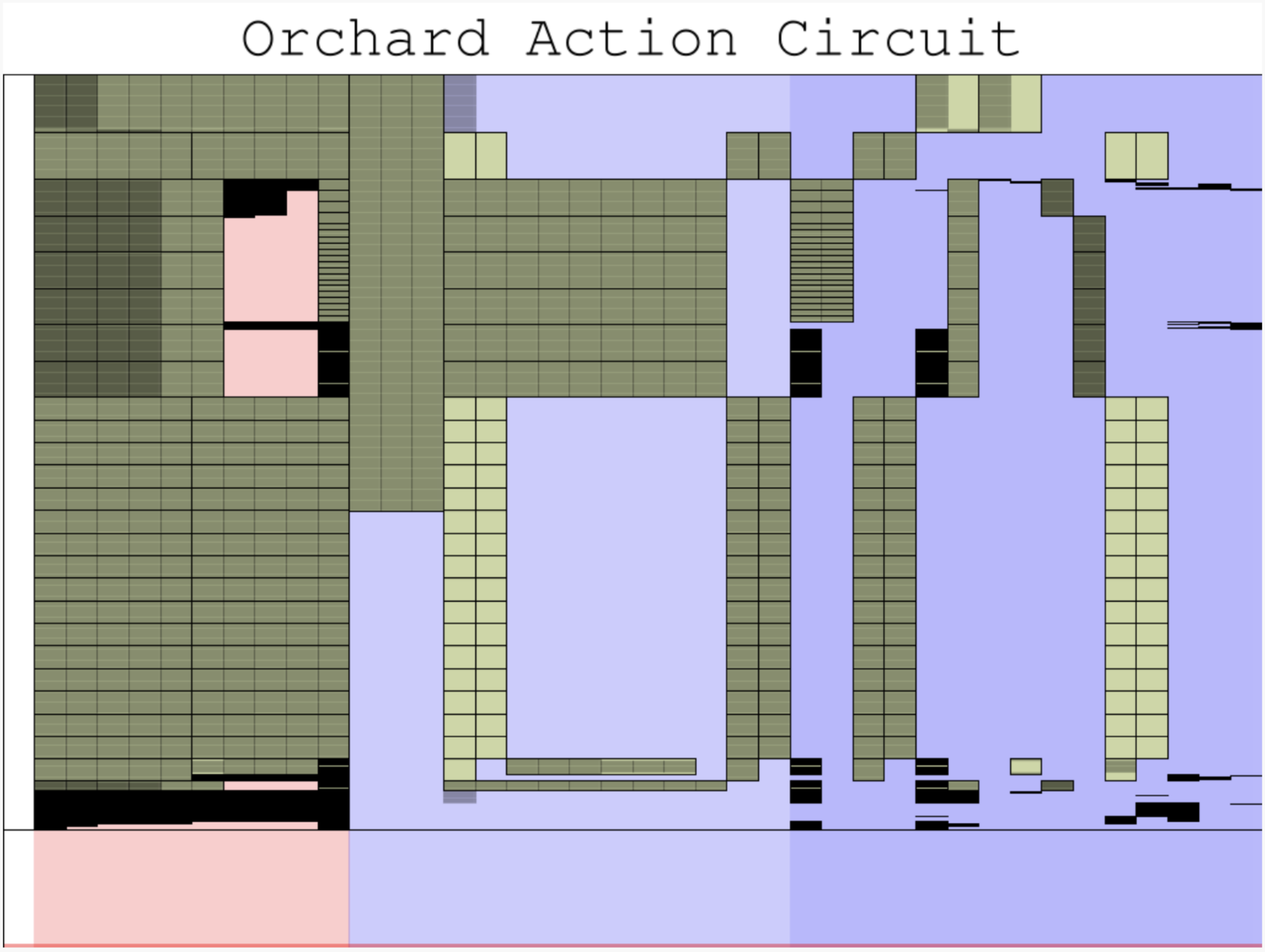
Example Simple Circuit

```
trait NumericInstructions<F: Field>: 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.

Example from ZCash



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.

Column Types

- 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.