Lesson 15 - Stark Theory / Oracles

STARK Theory

Polynomial Recap

A basic fact about polynomials and their roots is that if $p(x)$ is a polynomial, then $p(a) = 0$ for some specific value a,

if and only if there exists
a polynomial $q(x)$ such that
$(x - a)q(x) = p(x)$,

and therefore

$q(x) = \frac{p(x)}{(x-a)}$

and $deg(p) = deg(q) + 1$.

This is true for all roots

Schwarz-Zippel lemma

See wikipedia

We use this for polynomial identity testing
This is to determine whether two multivariate polynomials are identical.

If we have two non-equal polynomials of degree at most d, they can intersect at no more than d points.

STARK Process Overview

We are interested in Computational Integrity (CI), for example knowing that the Cairo program you wrote was computed correctly.

As with SNARKS we need to go through a number of transformations from the *trace* of our program, to the proof.
The first part of this is called arithmetisation, it involves taking our trace and turning it into a set of polynomials.

Our problem then becomes one where the prover that attempts to convince a verifier that the polynomial is of low degree.

The verifier is convinced that the polynomial is of low degree if and only if the original computation is correct (except for an infinitesimally small probability).

Use of randomness

The prover uses randomness to achieve zero knowledge, the verifier uses randomness when generating queries to the prover, to detect cheating by the prover.

Succinctness and performance

Much of the work that is done in creating a proof is ensuring that it is succinct and that it can be produced and verified in a reasonable time.

Arithmetisation

There are two steps

1. Generating an execution trace and polynomial constraints
2. Transforming these two objects into a single low-degree polynomial.

In terms of prover-verifier interaction, what really goes on is that the prover and the verifier agree on what the polynomial constraints are in advance.
The prover then generates an execution trace, and in the subsequent interaction, the prover tries to convince the verifier that the polynomial constraints are satisfied over this execution trace, unseen by the verifier.

The execution trace is a table that represents the steps of the underlying computation, where each row represents a single step

The type of execution trace that we're looking to generate must have the special trait of being succinctly testable — each row can be verified relying only on rows that are close to it in the trace, and the same verification procedure is applied to each pair of rows.

For example imagine our trace represents a running total, with each step as follows

Step	Amount	Total
0	0	0
1	5	5
2	2	7
3	2	9
4	3	12
5	6	18

If we represent the row as i , and the column as j , and the values as $A_{i,j}$
We could write some contraints about this as follows

$A_{0,2} = 0$

$\forall 1 >= i <= 5 : A_{i,2} - A_{i,1} - A_{i-1,2} = 0$

$A_{5,2} = 18$

These are linear polynomial constraints in $A_{i,j}$
Note that we are getting some succinctnesss here because we could represent a much larger number of rows with just these 3 constraints.

We want a verifier to ask a prover a very small number of questions, and decide whether to accept or reject the proof with a guaranteed high level of accuracy.
Ideally, the verifier would like to ask the prover to provide the values in a few (random) places in the execution trace, and check that the polynomial constraints hold for these places.
A correct execution trace will naturally pass this test.

However, it is not hard to construct a completely wrong execution trace (especially if we knew beforehand which points would be tested) , that violates the constraints only at a single place, and, doing so, reach a completely far and different outcome.
Identifying this fault via a small number of random queries is highly improbable.

But polynomials have some useful properties here
Two (different) polynomials of degree *d* evaluated on a domain that is considerably larger than *d* are different almost everywhere.
So if we have a dishonest prover, that creates a polynomial of low degree representing their trace (which is incorrect at some point) and evaluate it in a large domain, it will be easy to see that this is different to the *correct* polynomial.

Our plan is therefore to

1. Rephrase the execution trace as a polynomial
2. extend it to a large domain, and
3. transform that, using the polynomial constraints, into yet another polynomial that is guaranteed to be of low degree if and only if the execution trace is valid.

A more complex example

See article
Imagine our code calculates the first 512 Fibonacci sequence
1,1,2,3,5 ...

If we decide to operate on a finite field with max number 96769
And we have calculated that the 512th number is 62215.

Then our constraints are

$A_{0,2} - 1 = 0$
$A_{1,2} - 1 = 0$

$\forall 0 >= i <= 510 : A_{i+2,2} = A_{i+1,2} + A_{i,2}$

$A_{511,2} - 62215 = 0$

Starknet Events

Empiric are very active in this space.
This article is a useful overview about events

Starknet Indexer

This is a WIP but will

- Gather event data
- Persists and links the data
- Make the data available for SQL / GraphQL / http queries

EXAMPLE CONSOLE

See console

Thoth - Cairo/Starknet bytecode analyser, disassembler and decompiler

See Repo
A multi purpose project that can decompile bytecode from a network.
Also has some static analysers

Oracle Solutions

Useful overview

Stork

- Stork is an oracle with on-chain price computation. This means that median prices are calculated on StarkNet based on price quotes provided by individual publishers rather than being aggregated off-chain
- Allowlisted publishers can publish and sign data on-chain
- Right now 2 publishers are live publishing price updates on the testnet (Dexterity and Stork)
- Has 5 price tickers available (a more updated list can be found here) : BTC / USD, ETH/USD, DAI/USD, BUSD/USDT
- The median time between price updates was 180 seconds

Empiric

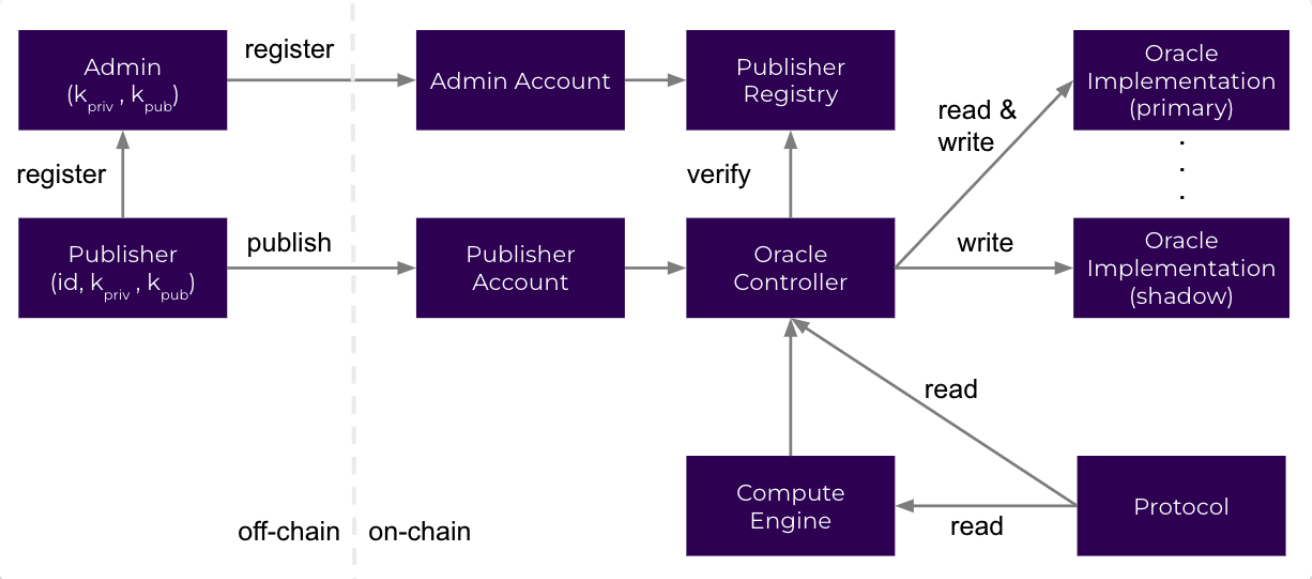
See article

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

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


@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 ();
}
```

DECO

See paper

DECO Short for decentralised 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 decentralised identity (DID) protocols such as CanDID, where users can obtain and manage their own credentials, rather than relying on a centralised 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 monetise 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 monetise 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