Lesson 16 - Oracles / Identity / Review

Oracle Solutions

Empiric

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

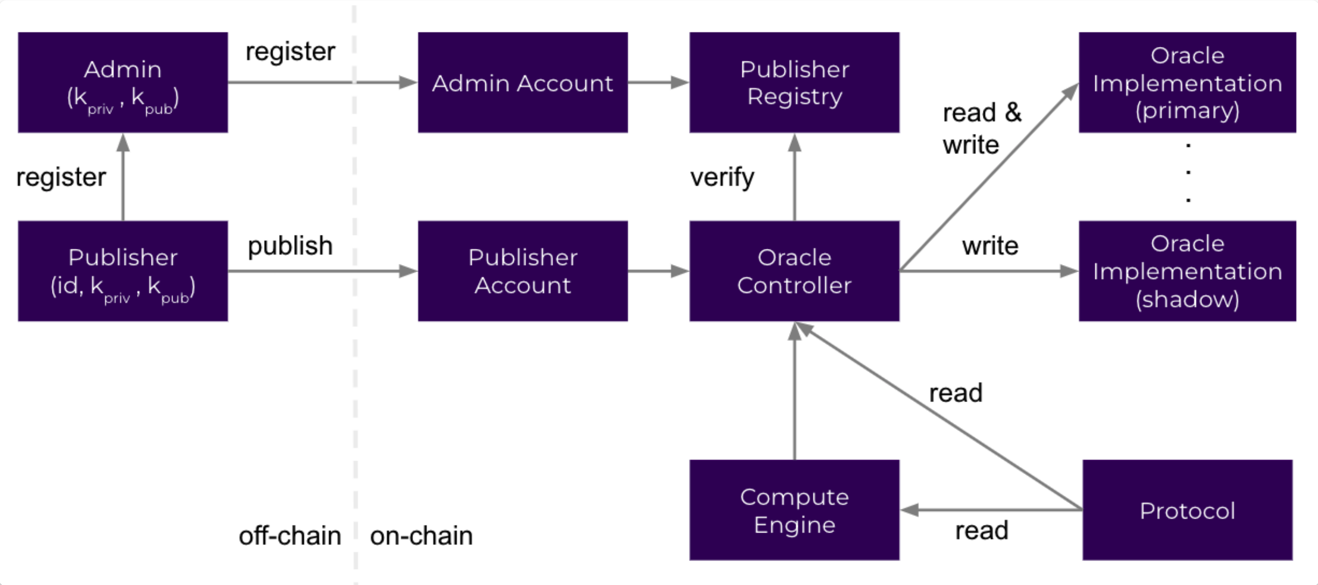
Empiric has been live in stealth for the last few months (on StarkNet testnet) and already integrated with leading protocols such as ZKLend, Magnety, Serity, CurveZero, Canvas, and FujiDAO.

Assets supported

- BTC/USD, key: 27712517064455012
- BTC/EUR, key: 27712517063406962
- ETH/USD, key: 28556963469423460
- ETH/MXN [Experimental], key: 28556963468900462
- SOL/USD, key: 32492132765102948
- AVAX/USD, key: 7022907837751063396
- DOGE/USD, key: 7237116810493260644
- SHIB/USD, key: 8316012582363558756
- BNB/USD, key: 27705915699721060
- ADA/USD, key: 27413441311765348
- XRP/USD, key: 33902823363408740
- MATIC/USD, key: 2017717457628037477220
- USDT/USD, key: 8463218574934504292
- DAI/USD, key: 28254602066752356
- USDC/USD, key: 8463218501920060260
- TUSD/USD, key: 8391740354804937572

- BUSD/USD, key: 7094703662122234724

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.

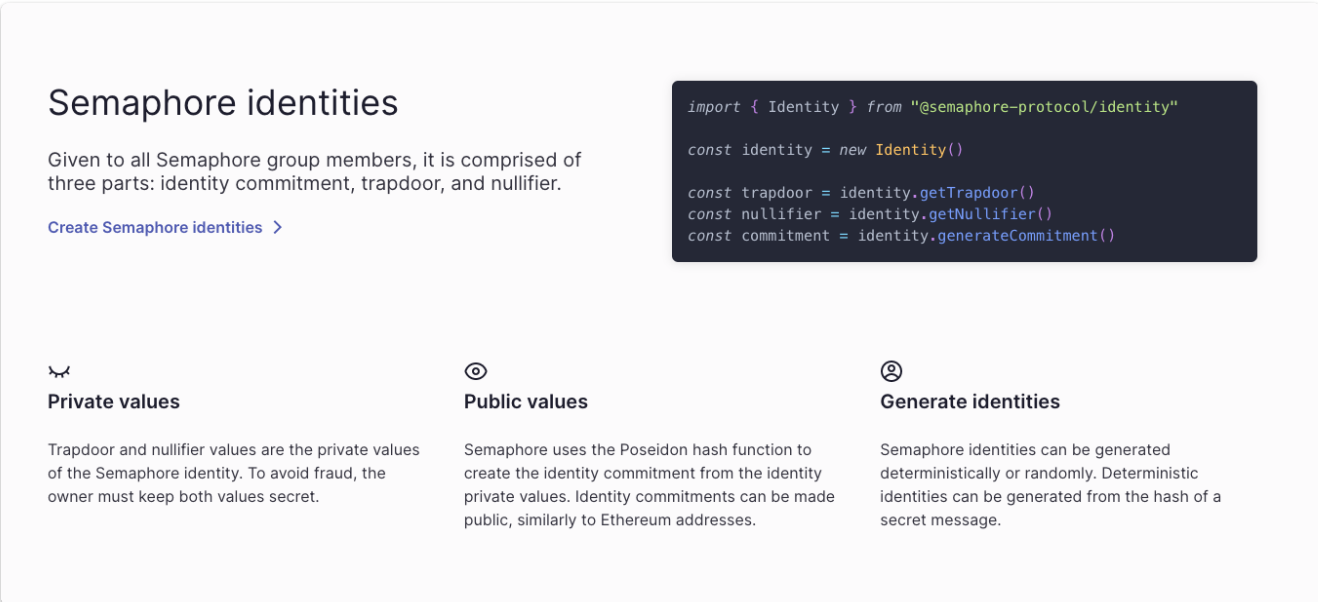
- Thus a single instance of DECO could enable anyone to become an oracle for any website
- Importantly, DECO does not require trusted hardware, unlike alternative approaches that could achieve a similar vision

---

Identity Solutions

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.

---

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.

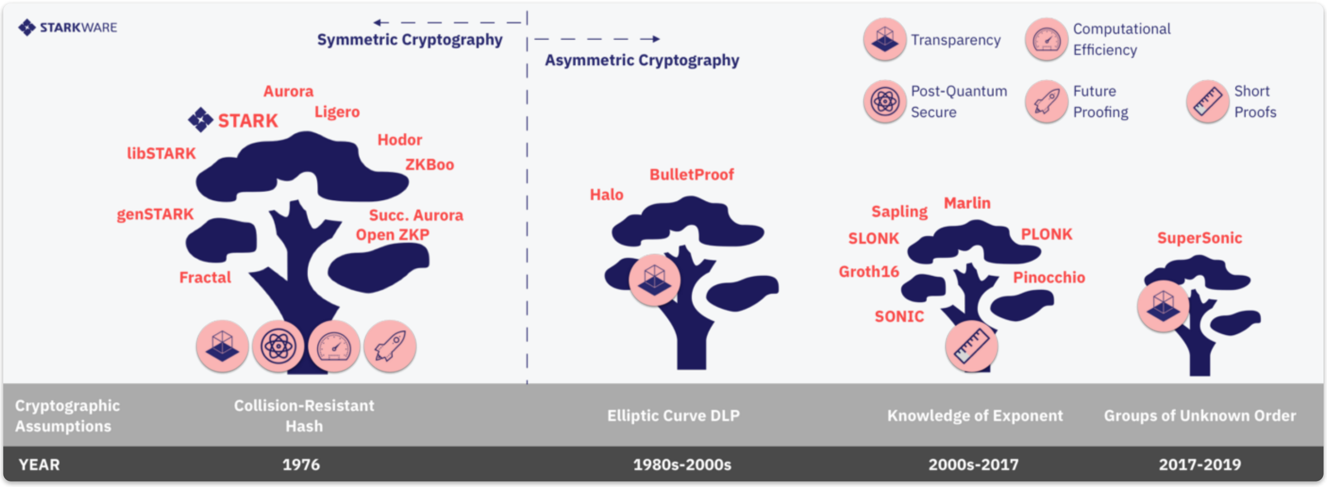
zCloak uses the Distaff VM
Distaff is a zero-knowledge virtual machine written in Rust.
For any program executed on Distaff VM, a STARK-based proof of execution is automatically generated. This proof can then be used by anyone to verify that a program was executed correctly.

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

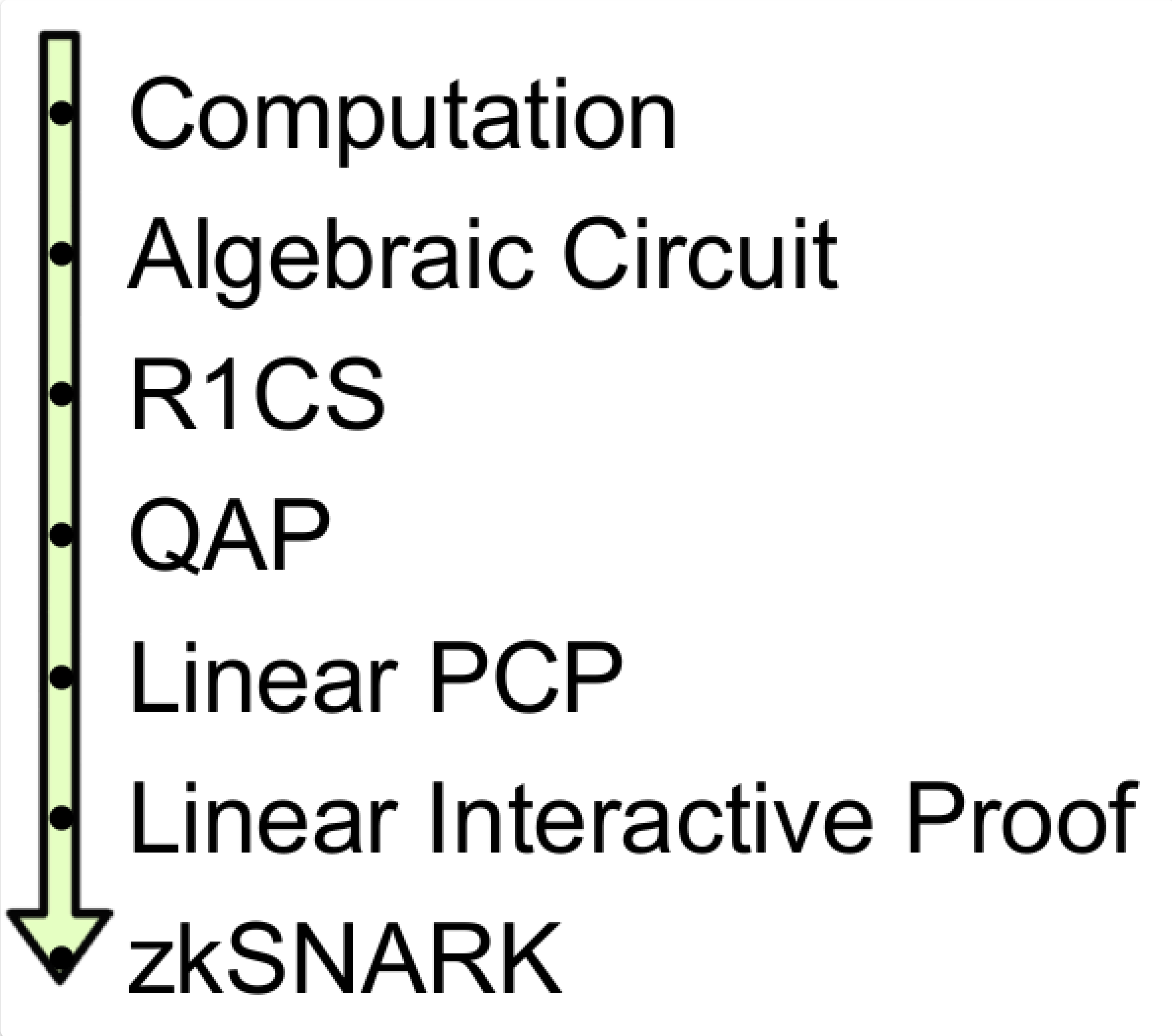
Course Review

Lesson 1 , 2 - ZKP Theory
Lesson 3 - Use cases
Lesson 4 - Zokrates / ZCash
Lesson 5 - Starknet
Lesson 6,7,8 - Cairo
Lesson 9,10 - Mina
Lesson 11 - Aztec
Lesson 12 - PLONK / Scroll
Lesson 13 - STARK theory
Lesson 14 - zkEVM
Lesson 15 - Cryptographic alternatives
Lesson 16 - Oracles / Identity / Review



	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	Strong 😒	Collision resistant hashes 😄	Discrete log 😏

zkSNARKS

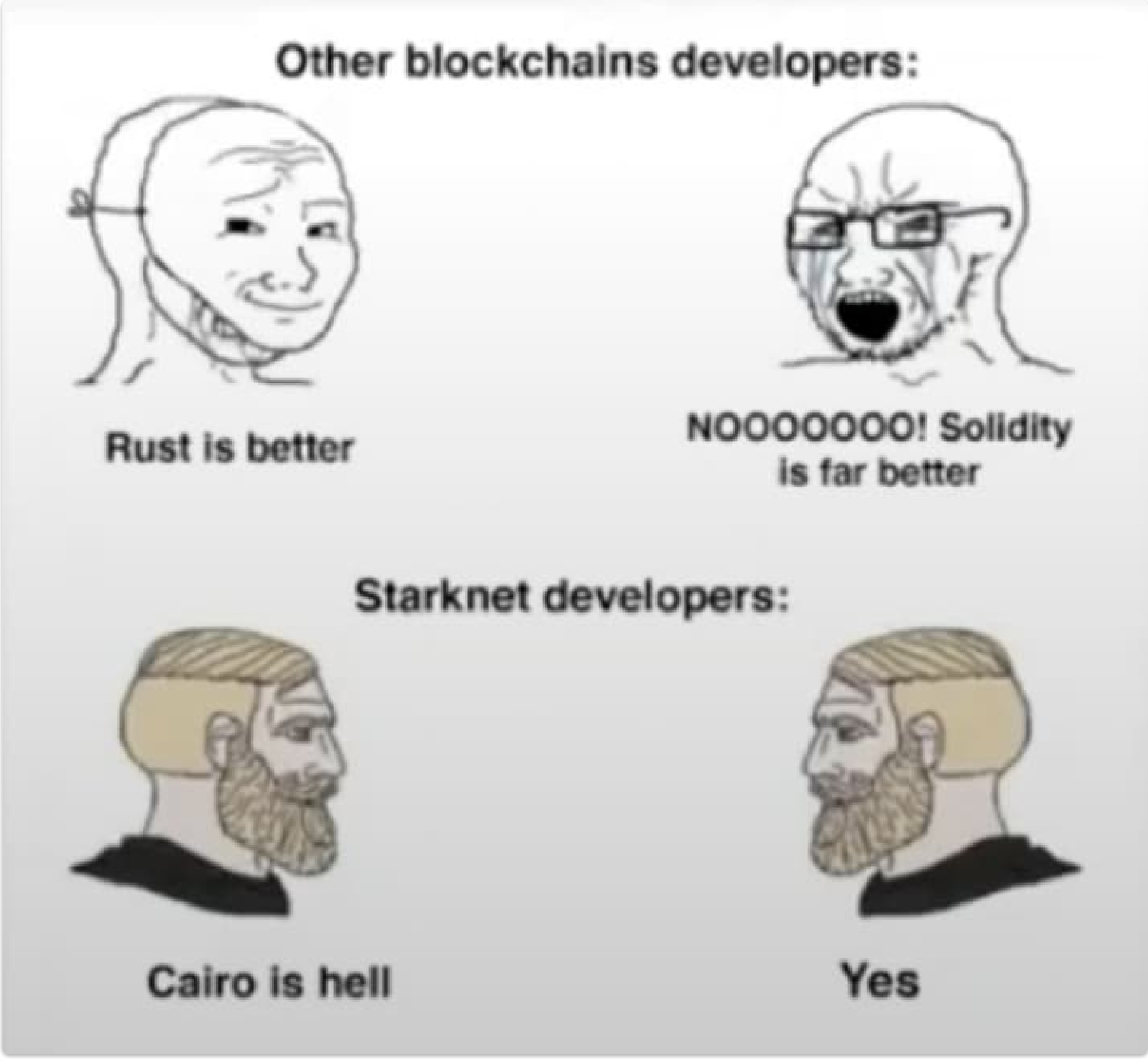


1. Trusted Setup
2. A High Level description is turned into an arithmetic circuit
3. Further Mathematical refinement
   R1CS, and then a series of formulae called a Quadratic Arithmetic Program (QAP).
4. Blind evaluation of a polynomial using Homomorphic Hiding
5. Non interactive proof

STARKS

1. Arithmetisation
   1. Generating an execution trace and polynomial constraints
   2. Transforming these two objects into a single low-degree polynomial.
2. Low degree testing - Fast Reed — Solomon Interactive Oracle Proof of Proximity

Cairo

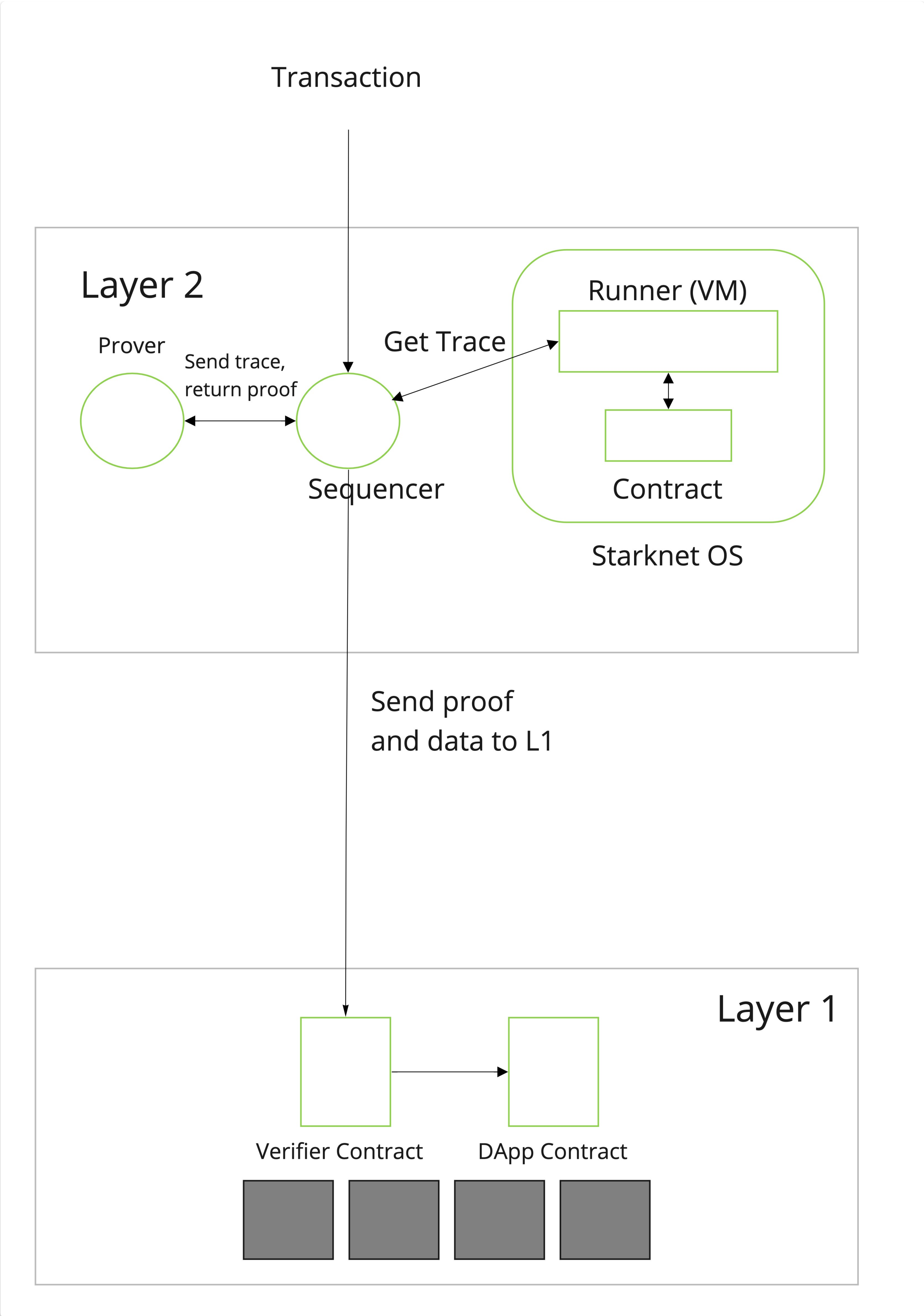


- Cairo programs (stateless) / contracts

- Private values supplied in hints

Useful tools

- Cairo playground
- Protostar
- Nile


Transaction
Layer 2
Prover
Send trace,
return proof
Get Trace
Runner (VM)
Sequencer
Contract
Starknet OS
Send proof
and data to L1
Layer 1
Verifier Contract
DApp Contract

WARP

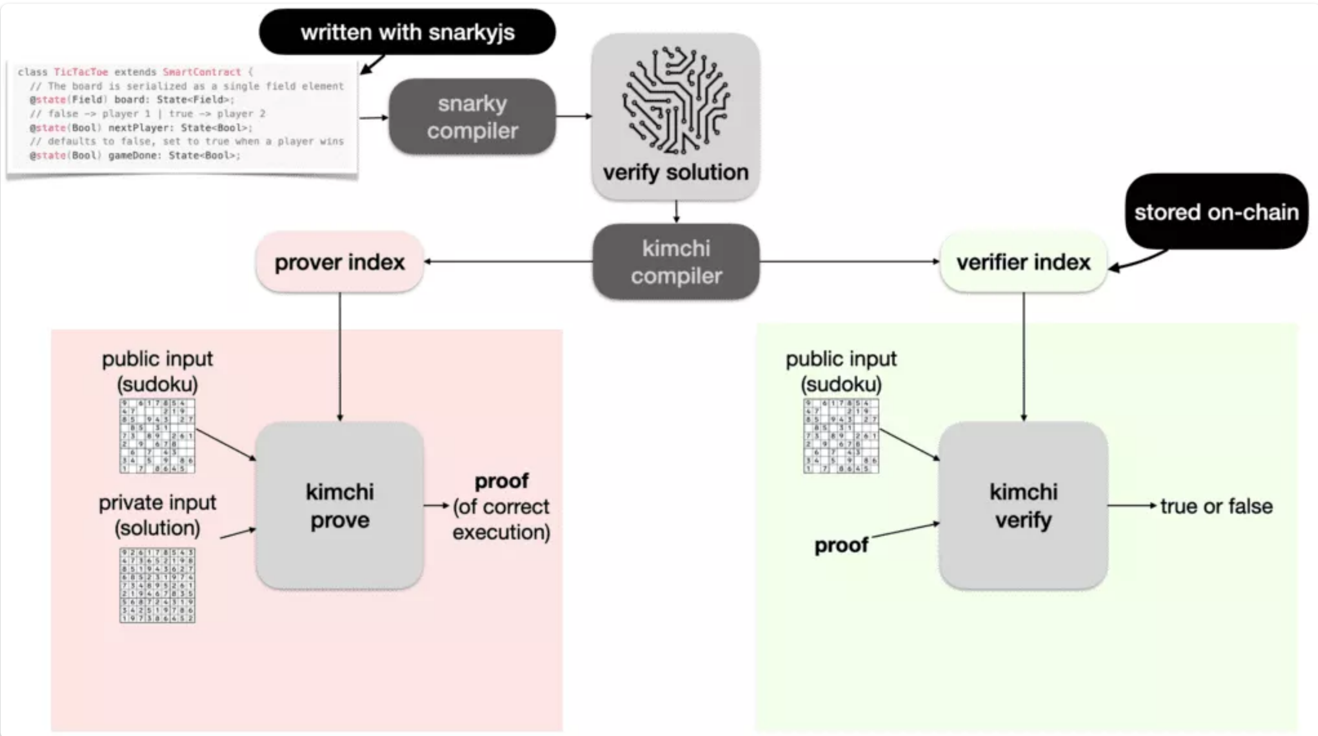
Bringing Ethereum smart contracts to StarkNet

See the Github repo

Transpiling Solidity to Cairo contracts


MINA PROTOCOL
Blockchain
22KB
FIXED SIZE

zkApps


written with snarkyjs
class TicTacToe extends SmartContract {
// The board is serialized as a single field element
@state(Field) board: State<Field>;
// false -> player 1 | true -> player 2
@state(Bool) nextPlayer: State<Bool>;
// defaults to false, set to true when a player wins
@state(Bool) gameDone: State<Bool>;
snarky compiler
verify solution
stored on-chain
prover index
kimchi compiler
verifier index
public input (sudoku)
private input (solution)
kimchi prove
proof (of correct execution)
public input (sudoku)
kimchi verify
proof
true or false

AZTEC

Scroll Pre-alpha Testnet

polygon Zero

World's fastest ZK scaling technology

zkSync