Lesson 9 - Mina

What is Mina?

Mina is the first cryptocurrency protocol with a succinct blockchain. With Mina, no matter how much the usage of the network grows, the blockchain always stays the same size - about 22kb.

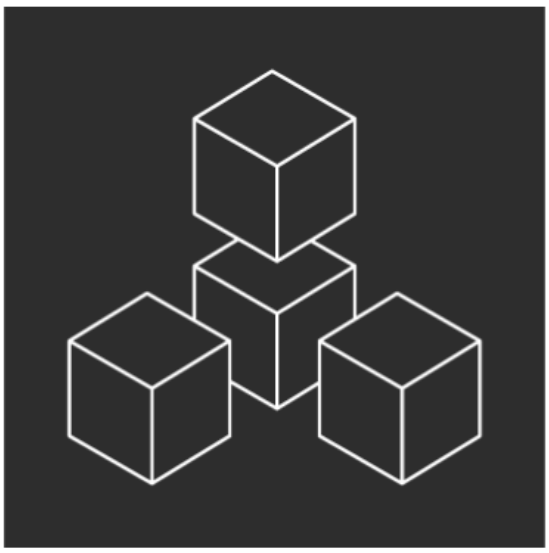
What data is needed for usable representation of the blockchain?

- A clear, usable representation of the basic data being queried of a state - accounts balances
- Data that a node needs in order to verify that this state is real in a trustless manner
- The ability to broadcast transactions on the network to make a transfer

Roles in Mina

Block producer

**



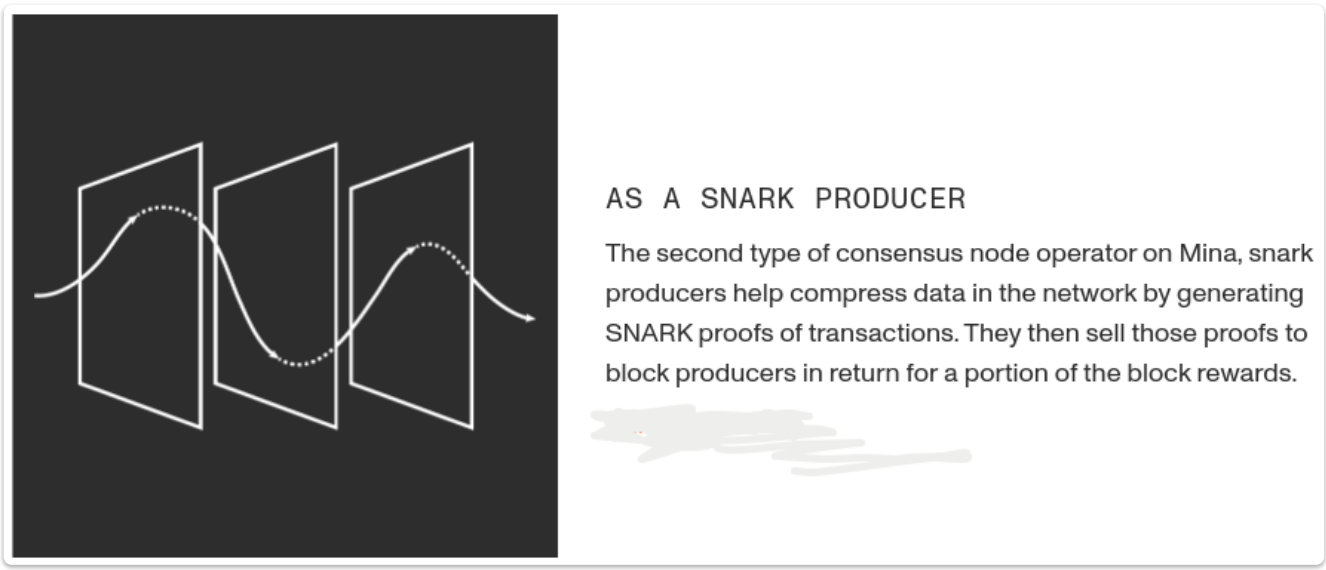
A block in Mina is constituted of:

- Protocol state
  - Genesis state hash
  - Blockchain state
  - Consensus state
  - Consensus constants
- Protocol state proof
- Staged ledger diff

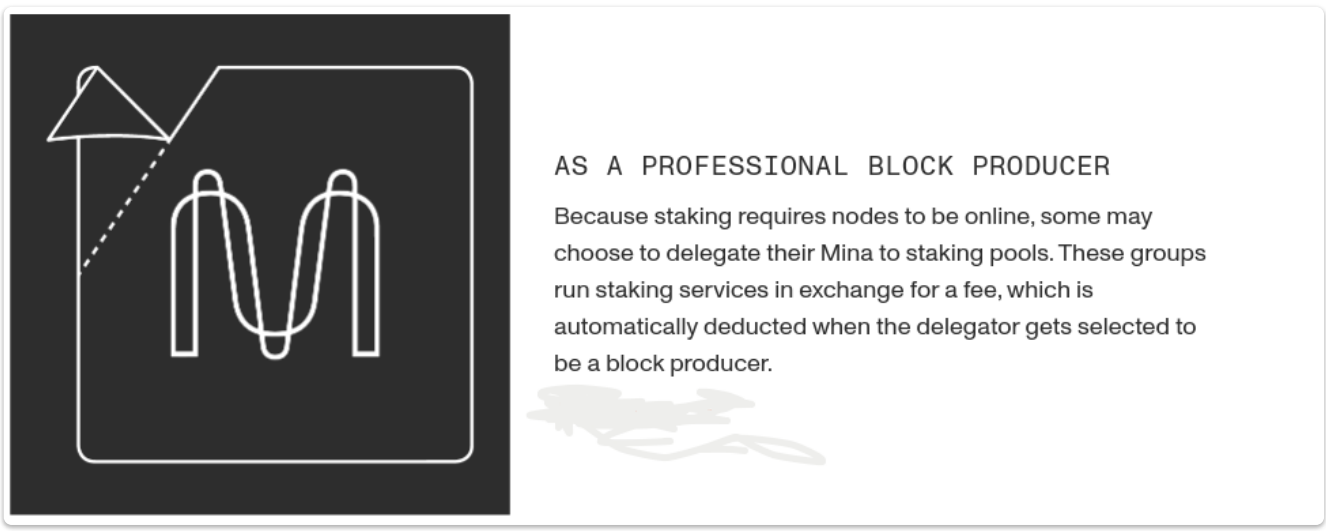
- Delta transition chain proof
- Current protocol version
- Proposed protocol version

Snark worker

**



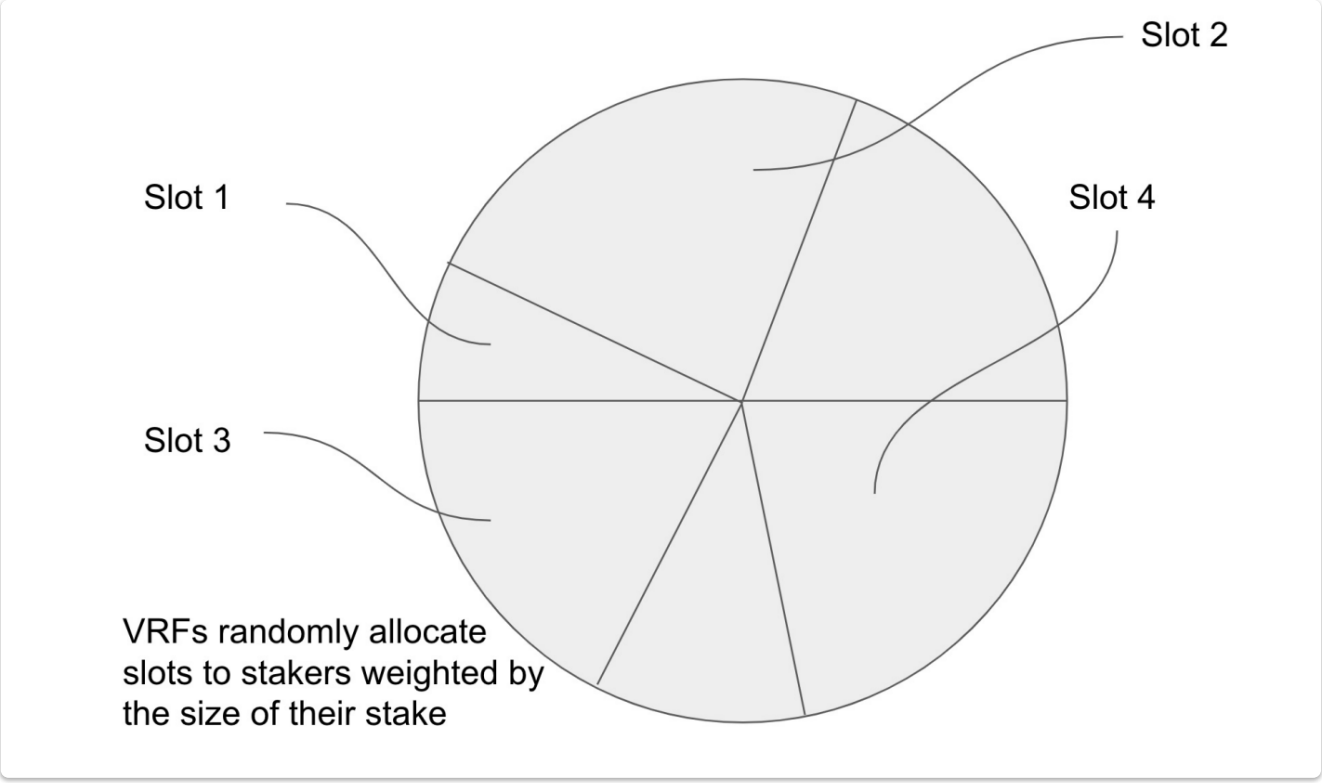
Professional block producer



Mina's consensus mechanism

Mina's consensus mechanism is an implementation of Ouroboros Proof-of-Stake. Due to Mina's unique compressed blockchain, certain aspects of the algorithm have diverged from the Ouroboros papers, and the version Mina uses is called **Ouroboros Samisika** VRFs are used to decide whether to produce a block, the probability is

proportional to the producer's stake.

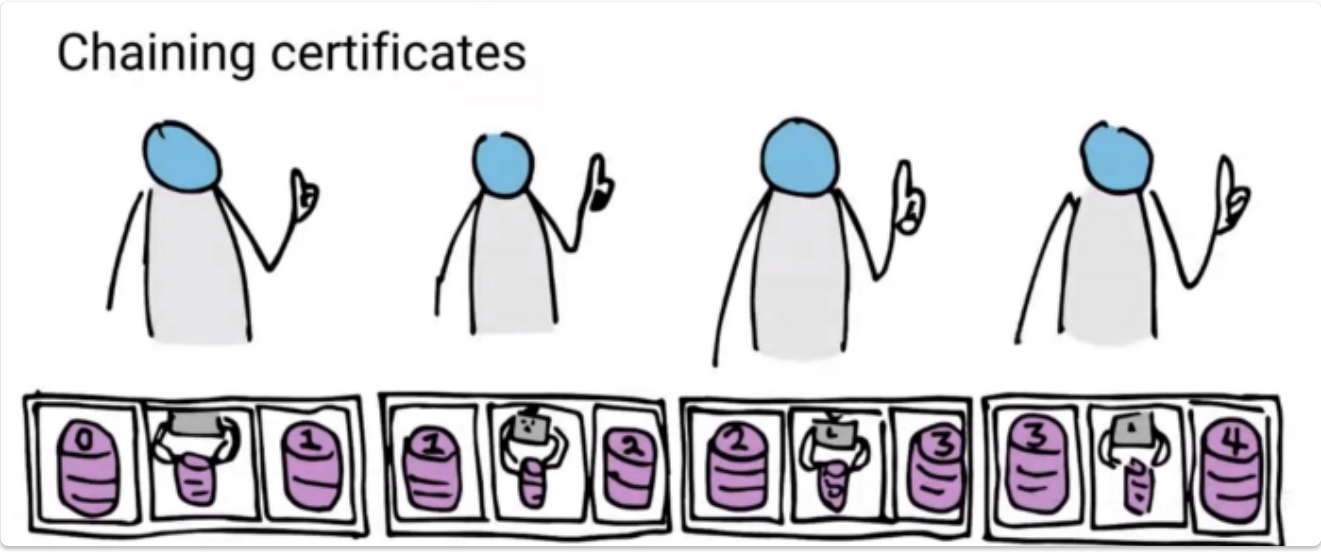


RECURSIVE COMPOSITION OF PROOF (SUCCINCTNESS)

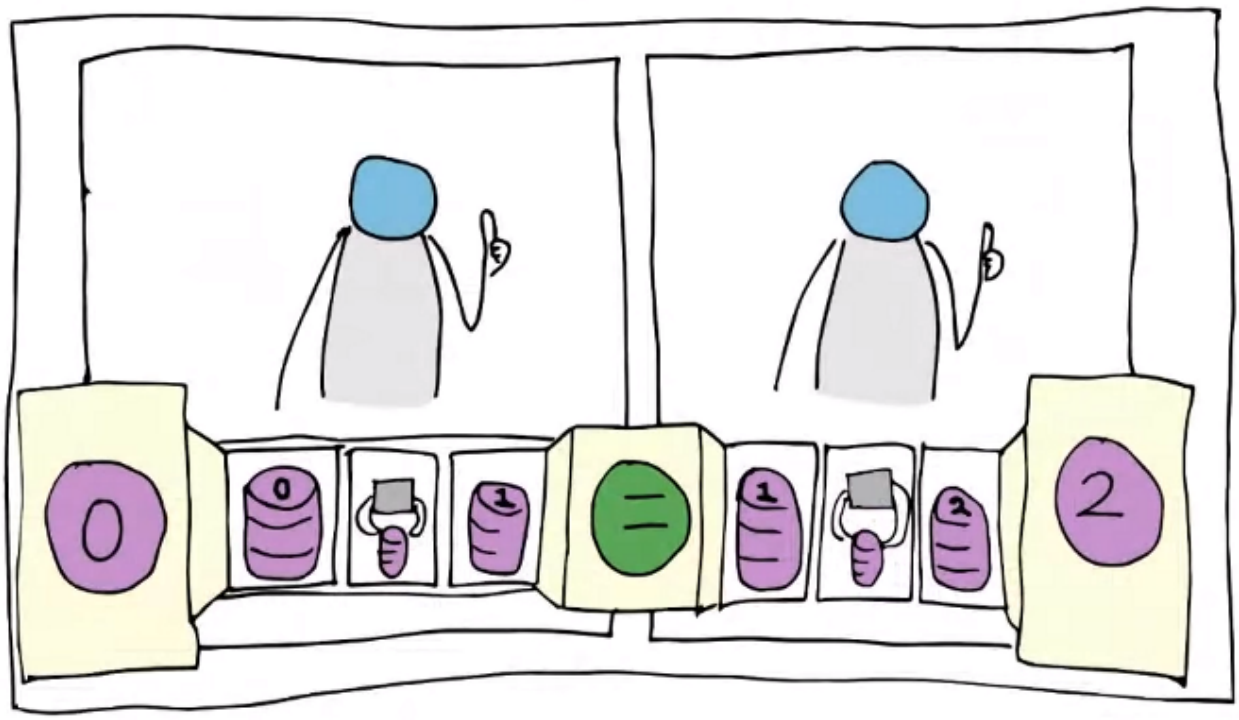
The blockchain is dynamic and new blocks keep getting added to it. However, we would like to ensure succinctness at any given point in time. Therefore, as the blockchain "grows", we compute a new SNARK proof that not only validates the new blocks, but also the existing SNARK proof itself. The notion of a SNARK proof that attests to the verifiability of another SNARK proof is the notion of "incrementally-computable SNARK"

from Mina: Decentralized Cryptocurrency at Scale (whitepaper)

https://www.youtube.com/watch?v=qCVACpgQSjo

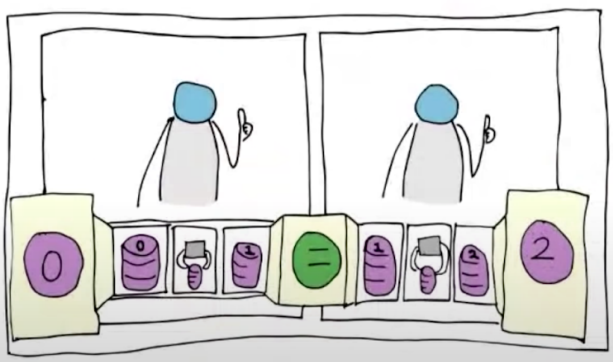


Thus, “You can get from **DB 0** to **DB 2**”

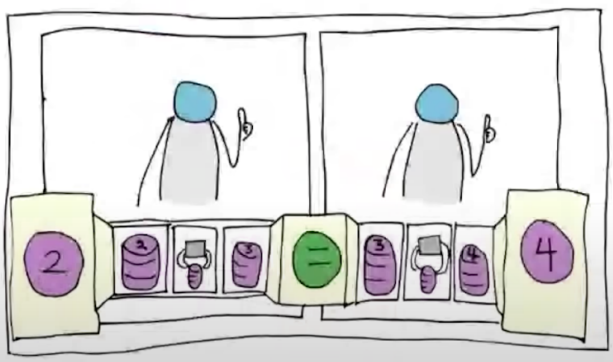


It’s a SNARK, so still ~1 kB

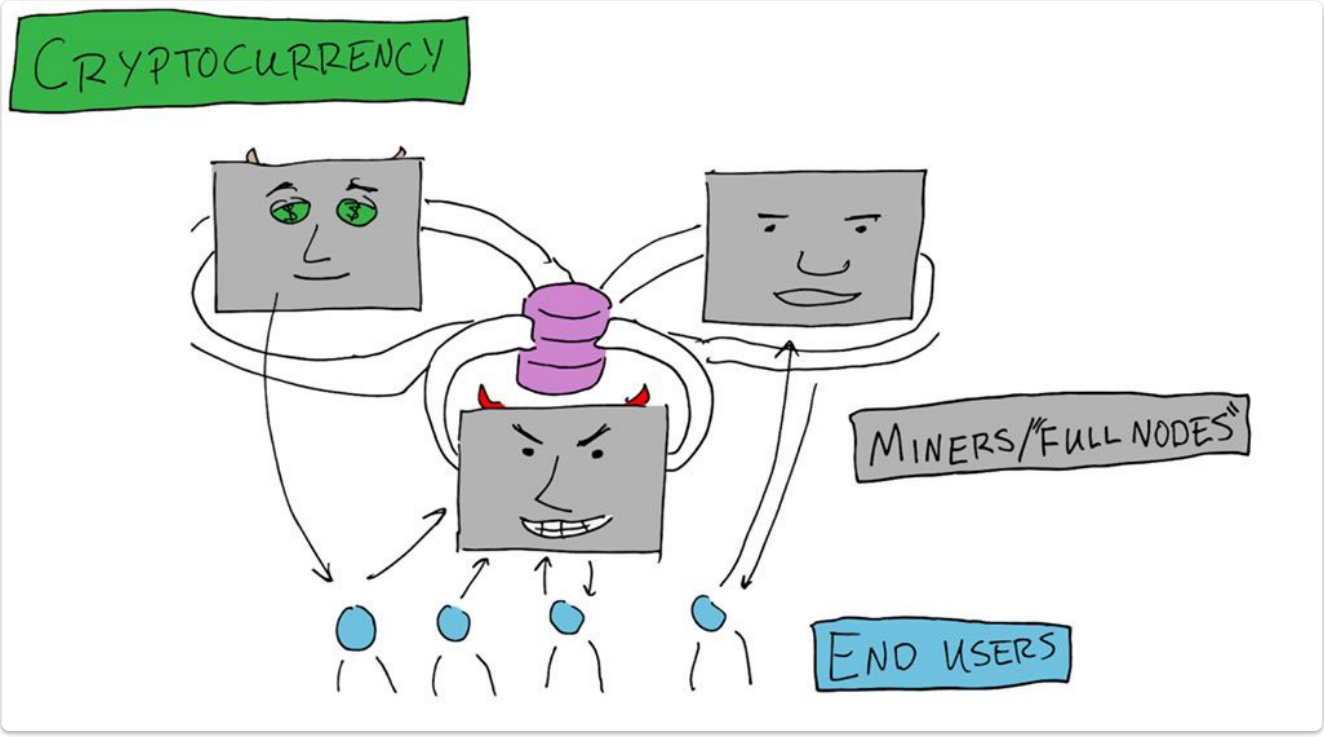
“You can get from
DB 0 to **DB 2**”



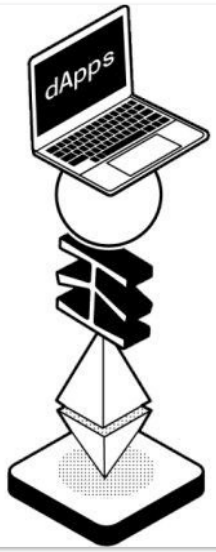
“You can get from
DB 2 to **DB 4**”




0
=
4


CRYPTOCURRENCY
MINERS/"FULL NODES"
END USERS

**



Community

https://awesome.mina.tools/
https://minacrypto.com/

awesome-mina

Awesome Mina

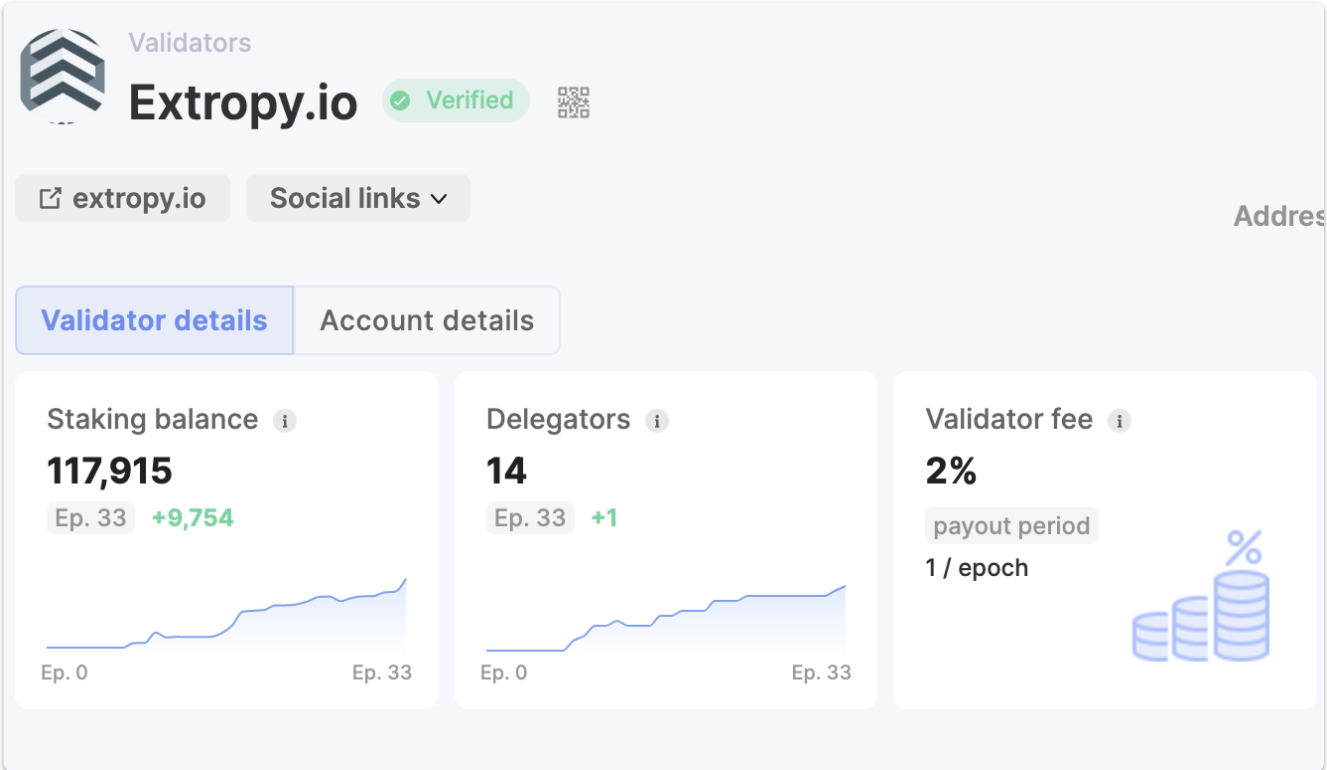
awesome

A curated list of Mina resources, software and tools for Mina Protocol. This list is open-source and you can contribute to improve it by creating a PR or submitting an issue on the Github repository by following the guidelines.

List of content:

- Official Links
- Wallets
- Explorers
- Tools
- Staking pools
- Monitoring dashboards
- Node scripts
- Ledger Projects
- Guides, news and articles
- Payout scripts
- Others
- International communities

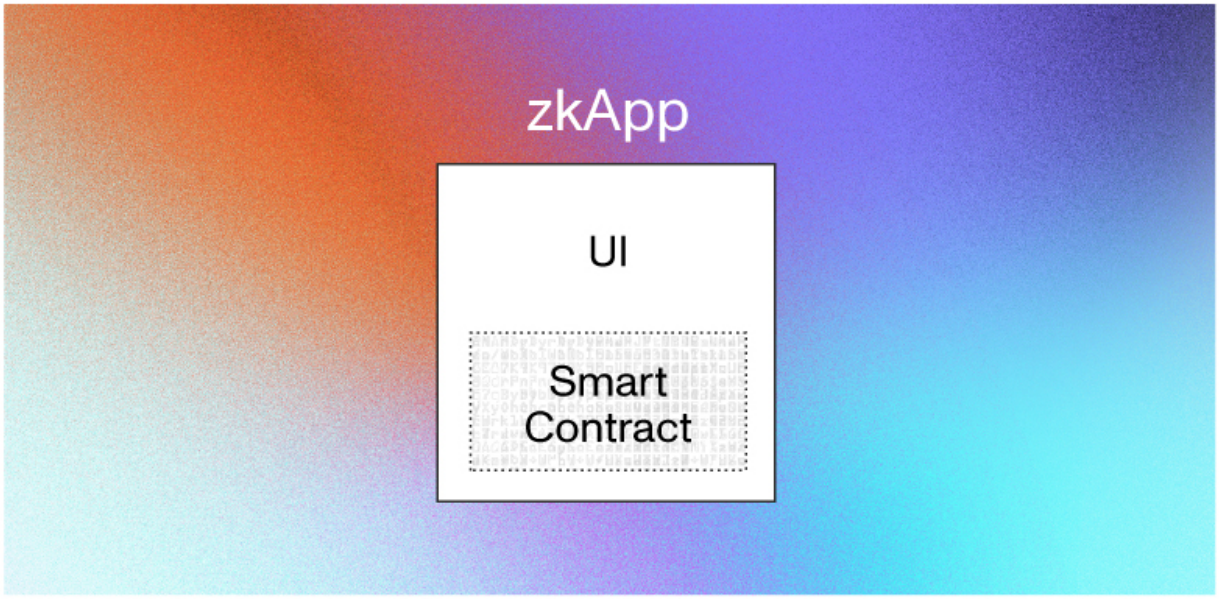
Staking with Extropy



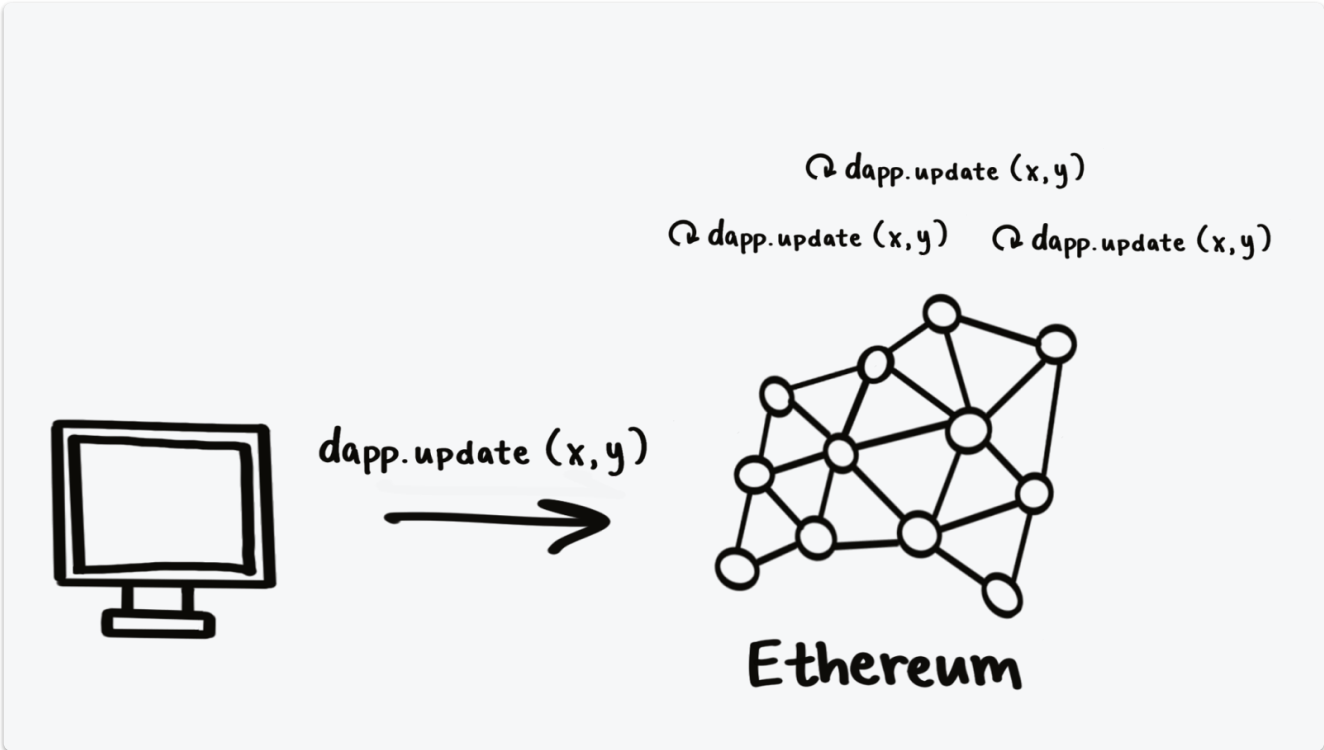
https://extropy-io.medium.com/staking-on-mina-fa540ad06811

What are zkApps?

zkApps ("zero-knowledge apps") are Mina Protocol's
smart contracts that use an off-chain execution and mostly off-chain state model.
This allows for private computation and state that can be either private or public.

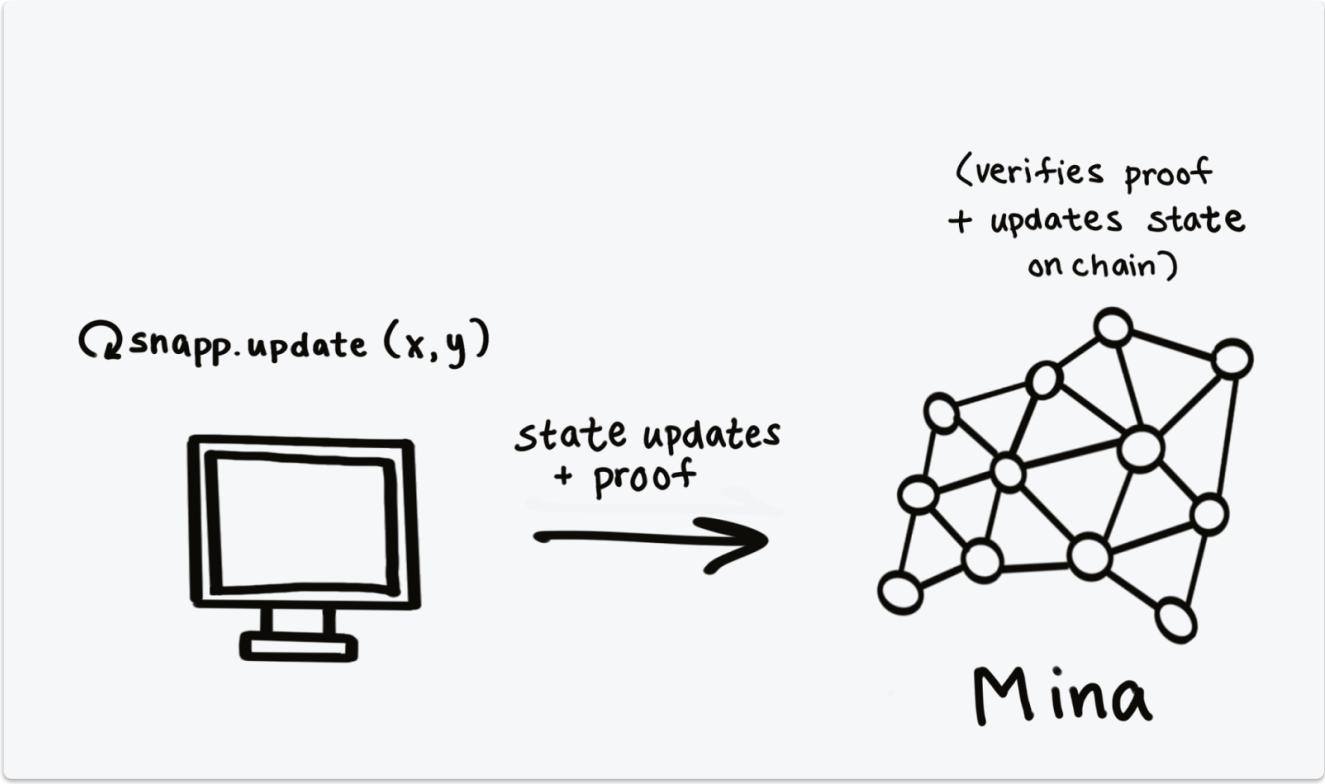


Ethereum dapps

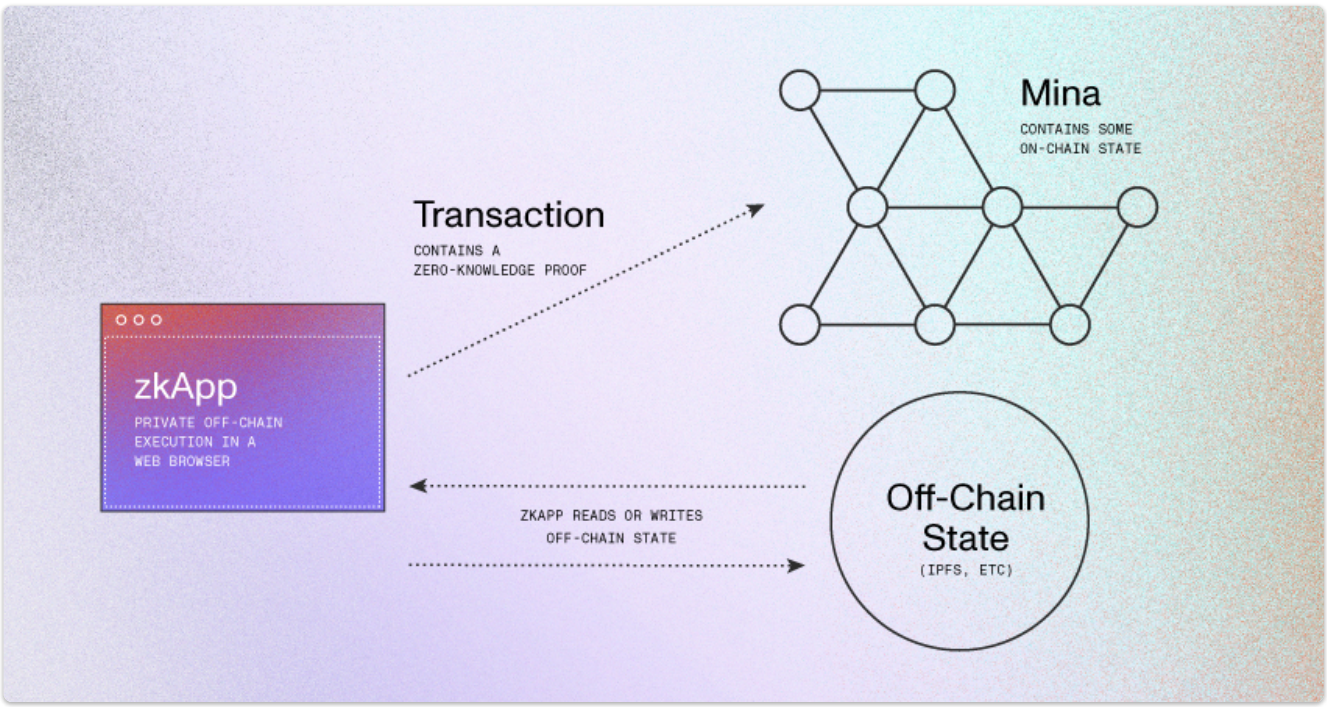


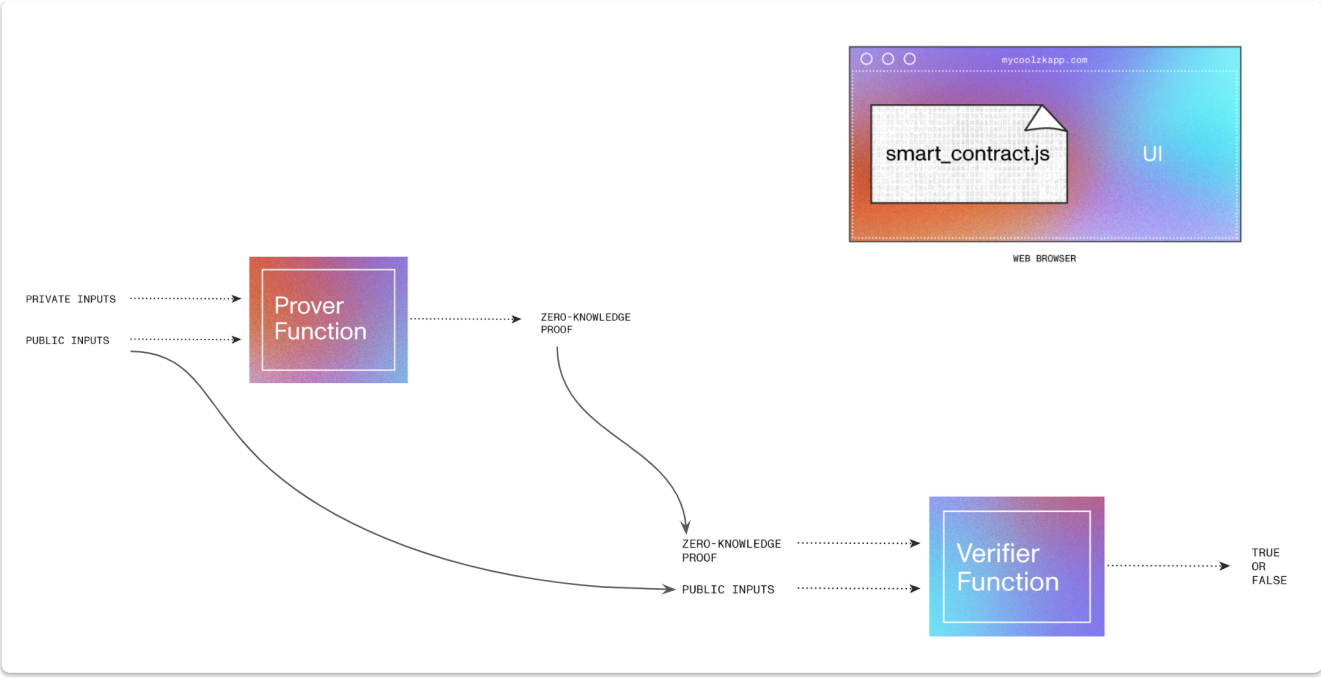
Ethereum uses **on-chain computation**.

Mina zkApps



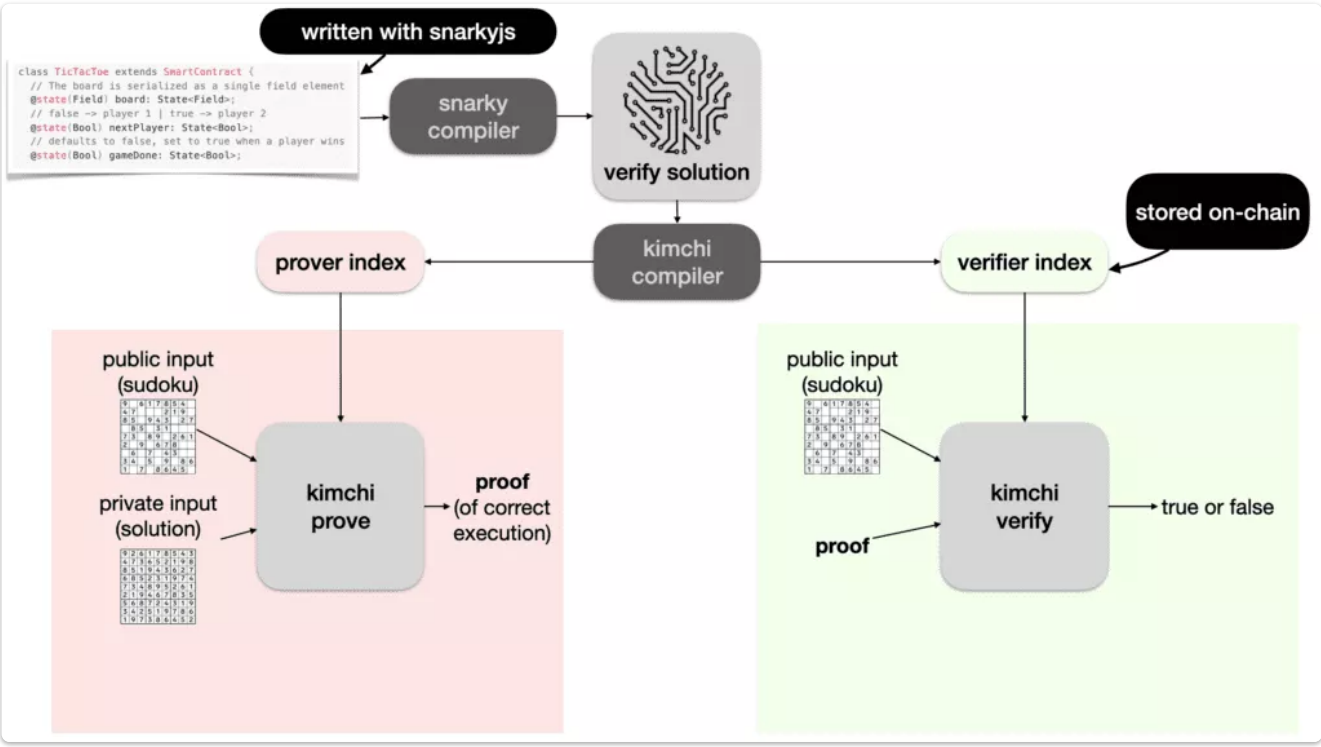
Mina uses **off-chain computation & on-chain verification**.





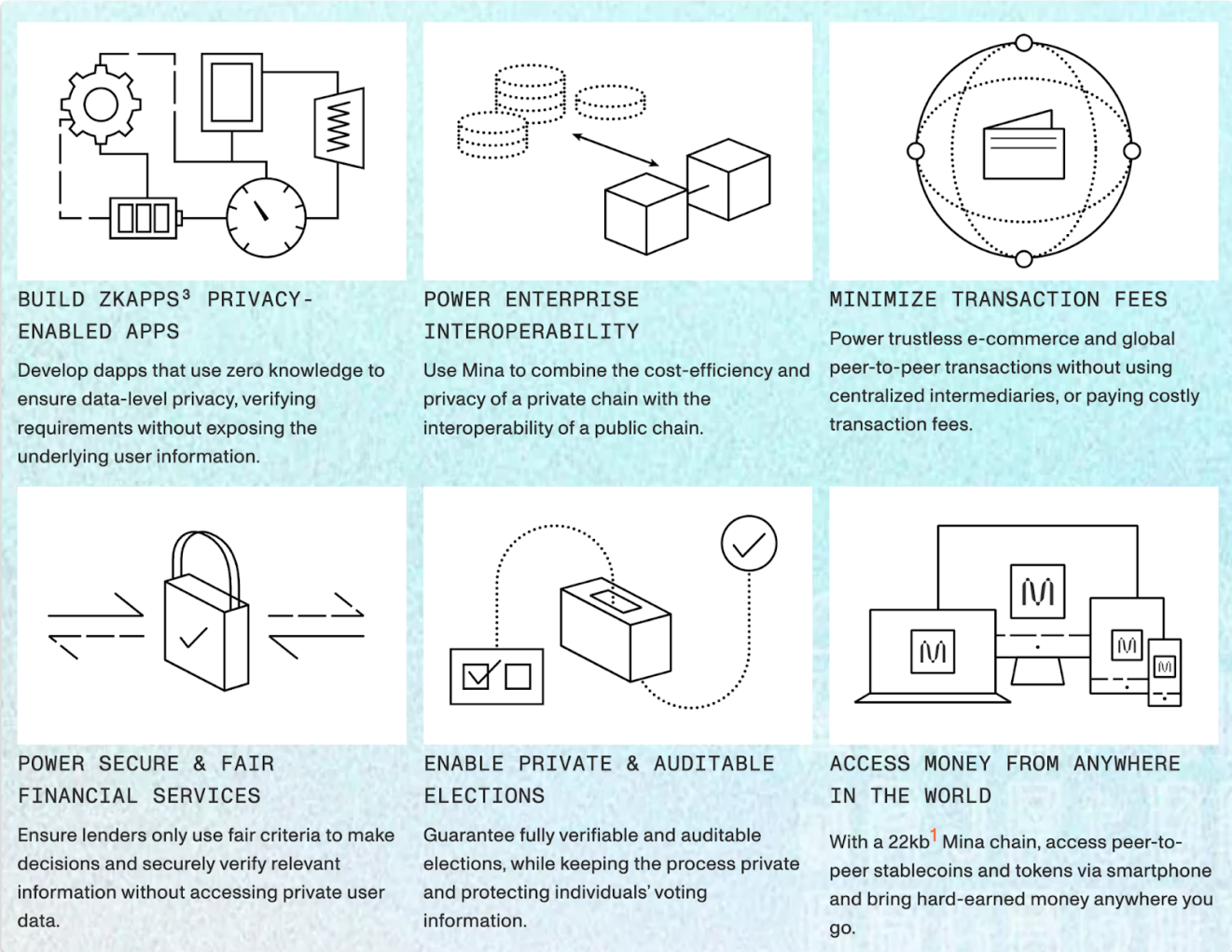
High level view of process a non interactive proof

https://extropy-io.medium.com/why-is-mina-so-tasty-part-1-kimchi-5cc8603a4e69



https://minaprotocol.com/blog/kimchi-the-latest-update-to-minas-proof-system
https://github.com/o1-labs/snarky
https://medium.com/minaprotocol/meet-pickles-snark-enabling-smart-contract-on-coda-protocol-7ede3b54c250

zkApps use cases



zkBridge ETH & WMina example

The bridge is currently only one way- i.e MINA state can be read on Ethereum , but not the other way around.

How does the bridge works ?

- Retrieve MINA state proof and the verification key.
- Generate an Auxiliary proof for the state received.
- Post the Auxiliary proof to Ethereum blockchain , where a smart contract will update only if valid.

[ETH] A Wrapped MINA contract on ETH will be created which will be the minter. This contract has a MINA address where a user will deposit funds into.

[MINA] User deposits funds into address above on MINA blockchain.

[MINA] Deposit transaction gets committed.

[ETH] Auxiliary Proof of MINA state is submitted to ETH.

[ETH] MINA state proof is validated and confirmed.

[ETH] Wrapped MINA Contract is called which uses the proof and validates the balance and mints Wrapped MINA on Ethereum.

SnarkyJS

zkApps are written in TypeScript using SnarkyJS.

SnarkyJS is a TypeScript library for writing smart contracts based on zero-knowledge proofs for the Mina Protocol. It is included automatically when creating a new project using the Mina zkApp CLI.

Primitive data types

Field elements are the basic unit of data in zero-knowledge proof programming. Each field element can store a number up to almost 256 bits in size.

```
new Bool(x); // accepts true or false

new Field(x); // accepts an integer,or a numeric string

new UInt64(x); // accepts a Field - useful for constraining numbers to 64 bits

new UInt32(x); // accepts a Field - useful for constraining numbers to 32 bits

PrivateKey, PublicKey, Signature; // useful for accounts and signing

new Group(x, y); // a point on our elliptic curve, accepts two
Fields/numbers/strings

Scalar; // the corresponding scalar field (different than Field)
```

In typical programming, you might use:

`const sum = 1 + 3;`

In SnarkyJS, you would write this as:

`const sum = new Field(1).add(new Field(3));`

This can be simplified as:

`const sum = new Field(1).add(3);`

Conditionals (Important !)

Traditional conditional statements are not supported by

SnarkyJS:

```
// this will NOT work
if (foo) {
x.assertEquals(y);
}
```

Instead, use SnarkyJS' built-in Circuit.if() method, which is a ternary operator:

```
const x = Circuit.if(y.equals(Field.zero)), a, b); // behaves like `foo ? a :
b`
```

Common methods

```
let x = new Field(4); // x = 4

x = x.add(3); // x = 7

let hash = Poseidon.hash([x]); // takes array of Fields, returns Field

x = x.sub(1); // x = 6

let privKey = PrivateKey.random(); // create a private key

x = x.mul(3); // x = 18

let pubKey = PublicKey.fromPrivateKey(privKey); // derive public key

x = x.div(2); // x = 9

let msg = [hash];

x = x.square(); // x = 81

let sig = Signature.create(privKey, msg); // sign a message

x = x.sqrt(); // x = 9

let b = x.equals(8); // b = Bool(false)

b = x.greaterThan(8); // b = Bool(true)

b = b.not().or(b).and(b); // b = Bool(true)

b.toBoolean(); // true

sig.verify(pubKey, msg); // Bool(true)
```

Setup

DEPENDENCIES:

NodeJS 16+ (or 14 using node --experimental-wasm-threads)
NPM 6+

Git 2+
IDE (VS Code recommended)

zkapp-cli

```
npm install -g zkapp-cli

zk project my-proj-name

npm run build && node ./build/src/index.js
```

Useful links for snarkyJS

Crowdcast

Resource Kit