quezar

Mina Ecosystem 101

By Quezar

We will cover

- What are Zero Knowledge Proofs?
- Introduction to Mina
- Overview of zkApps
- And learn more by building a few zkApps

What are Zero Knowledge Proofs?

We will cover

- What are zero knowledge proofs?
- Why are they important?
- What problems do they solve?
- How do they work?
- SNARKs and STARKs
- zkVMs and zkEVMs

What are zero knowledge proofs?

They let you prove that you have certain **information or knowledge without actually revealing** what that information is.

Let's take an example.

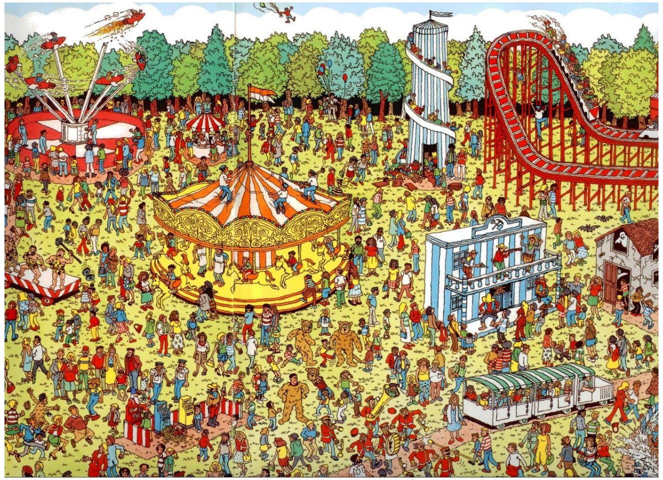
quezar

What are zero knowledge proofs?

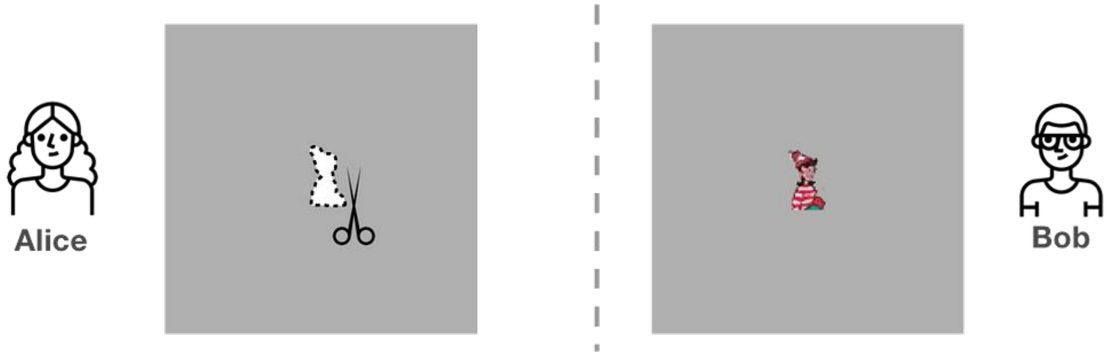
Alice needs to tell Bob

that she **knows** where Waldo is

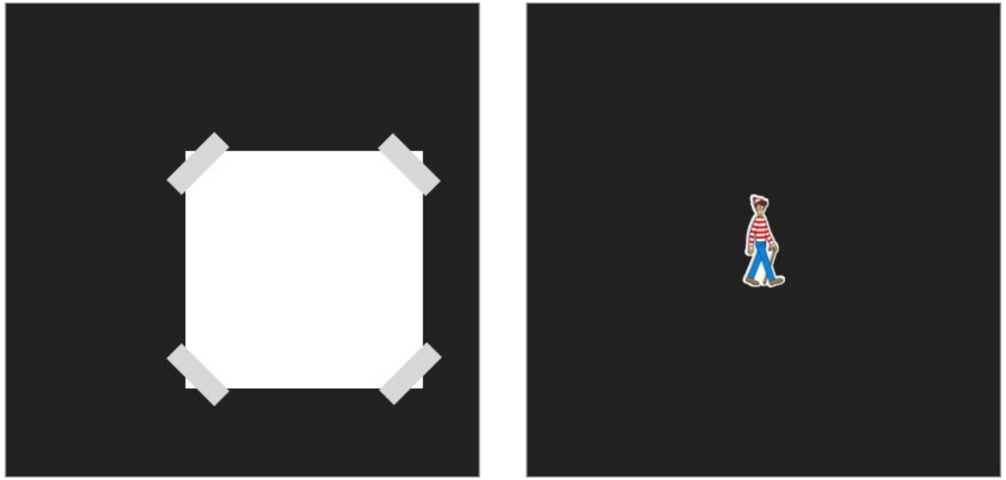
but **without revealing** where Waldo is.



What are zero knowledge proofs?

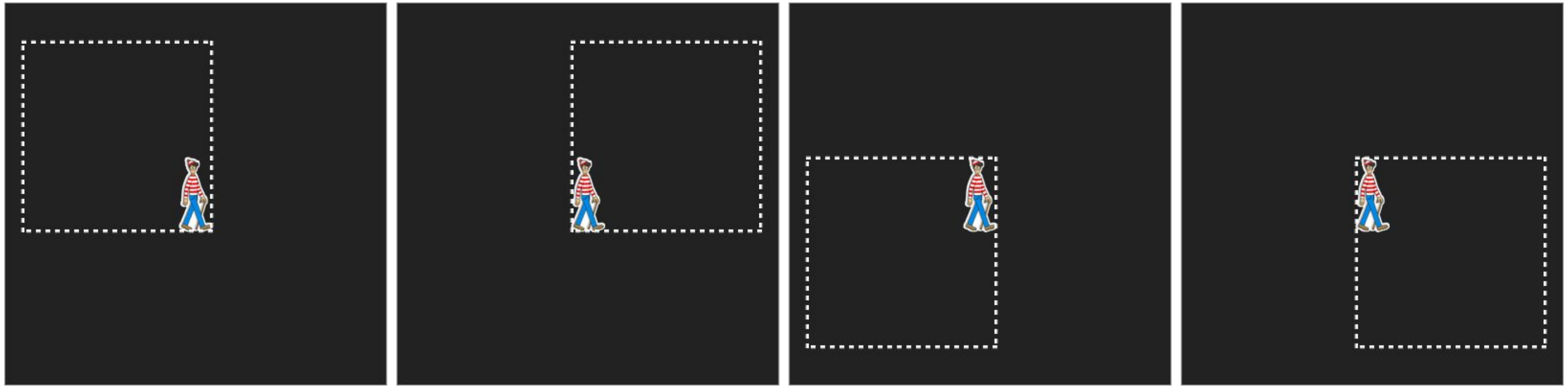


What are zero knowledge proofs?



quezar

What are zero knowledge proofs?



Why are ZKPs important?

Think of them like a **certificate**.

Certificate for **proving the authenticity** of something.

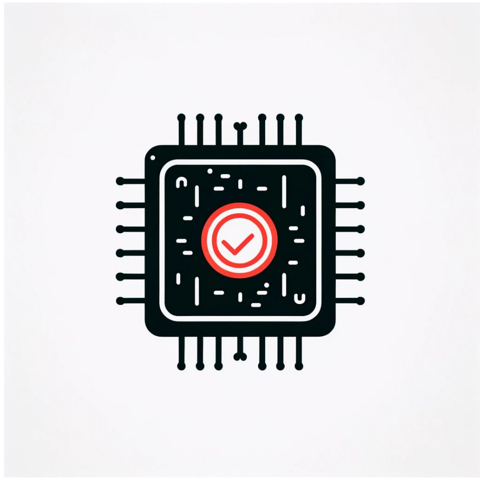
Something like a **computation** or some **data**.

Why are ZKPs important?

Verifiable compute

You can be sure that a computation has been performed without actually performing it.

This enables taking a lot of load off-chain, making blockchains more usable.



Why are ZKPs important?

Data privacy

You can be sure that some data is there without actually seeing it.

Most blockchains have their entire states public, which is problematic in some cases.

What problems do ZKPs solve?

Many use cases across different industries

- Finance and banking
- Cybersecurity and data privacy
- Blockchains & cryptocurrency
- Healthcare
- Others

What problems do ZKPs solve?

Finance and Banking

Privacy Enhanced Transactions

Ensuring transactional privacy in banking and cryptocurrencies without revealing sensitive details.

Compliance and Auditing

Proving compliance with regulatory requirements without exposing the underlying data.

Identity Verification

Securely verifying customer identity for Know Your Customer (KYC) processes without revealing personal information.

What problems do ZKPs solve?

Cybersecurity and Data Privacy

Secure Multi-party Computation

Facilitating collaborative computation among distrustful parties without exposing their private data.

Access Control

Validating user credentials without revealing or exchanging passwords.

Private Information Retrieval

Allowing users to retrieve data from a server without revealing what data is being retrieved.

quezar

What problems do ZKPs solve?

Blockchain and cryptocurrencies

Private Transactions

Enabling private transactions in public blockchains, such as in Zcash.

Smart Contract Privacy

Concealing certain aspects of smart contracts while still proving their correct execution.

Scalability Solutions

Improving blockchain scalability through succinct non-interactive proofs of computation.

quezar

What problems do ZKPs solve?

Healthcare

Confidential Health Records

Securely sharing patient data for research or treatment without exposing personal health information.

Insurance Claims Processing

Proving eligibility and claims without revealing sensitive health details.

What problems do ZKPs solve?

Others

Secure Electronic Voting

Ensuring voter anonymity while proving that votes are cast correctly and without double voting.

Supply Chain & Logistics

Verifying the authenticity and origin of products without revealing trade secrets or supplier information.

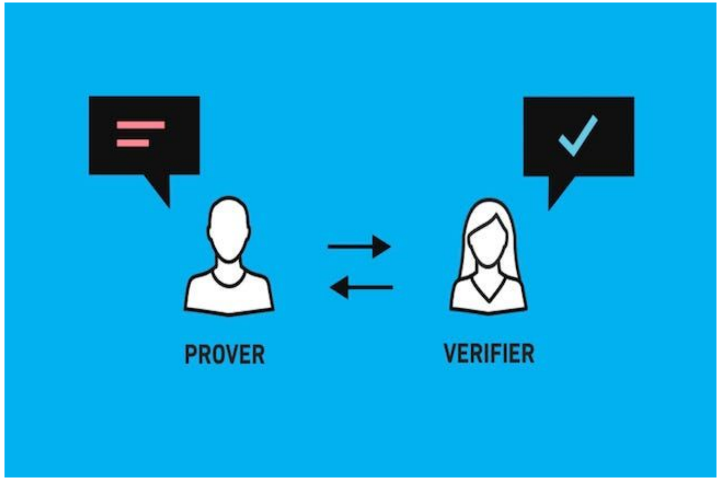
DRM / Content Access Control

Ensuring only authorized users access digital content without disclosing their identities or access patterns.

How do ZKPs work?

We'll cover this in three parts

- Key concepts
- Basic working mechanism
- Basis properties



How do ZKPs work?

Key Concepts

Prover

The entity that wants to prove the validity of a certain statement or possession of certain information

Verifier

The entity that needs to be convinced of the validity of the statement without learning any additional information.

Zero Knowledge

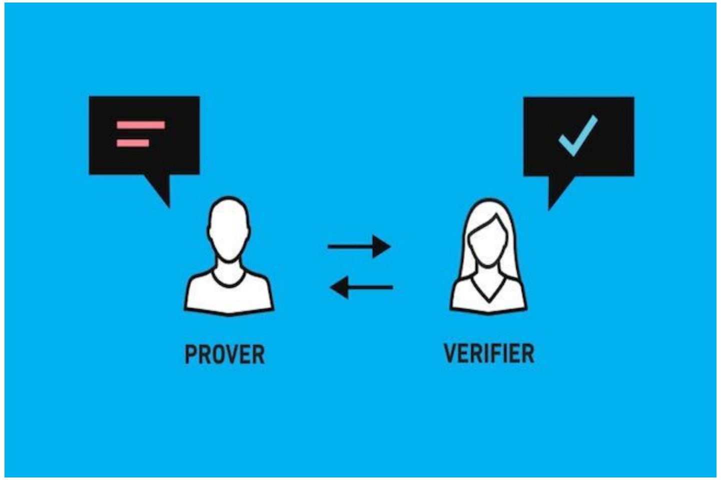
No knowledge (information) about the statement or the proof is transferred from the prover to the verifier, beyond the fact that the statement is indeed true.

How do ZKPs work?

Basic working mechanism

The entire process works in four steps

1. Initialization
2. Proof generation
3. Verification
4. Confirmation

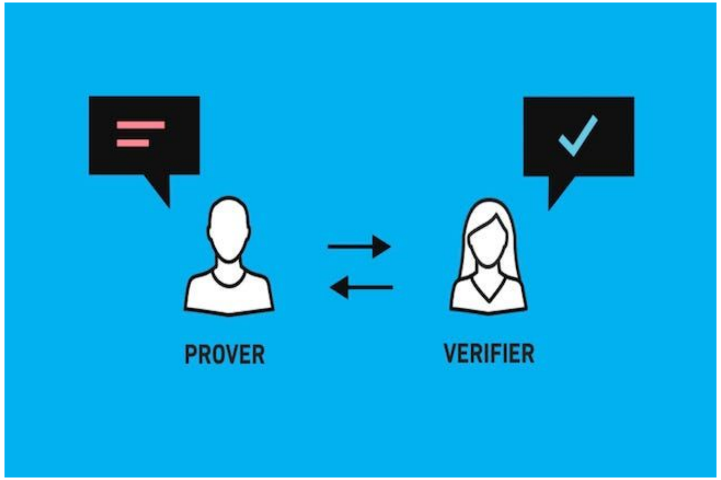


How do ZKPs work?

Basic working mechanism - 1. Initialization

Common Knowledge - Both the prover and the verifier agree on a common set of rules or a protocol that defines how the proof will be conducted.

Problem Statement - There is a statement or a problem that the prover wants to prove they know the solution to, without revealing the solution itself.

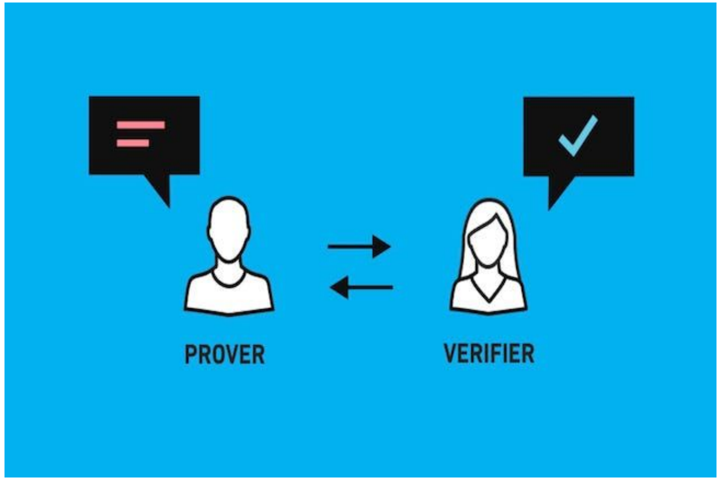


How do ZKPs work?

Basic working mechanism - 2. Proof Generation

Commitment - The prover computes a piece of evidence (a commitment) that is related to the secret knowledge they possess. This commitment must be constructed in such a way that it hides the actual secret.

Blinding Information - Often, the prover adds some random data or 'blinding' information to the commitment to ensure that the secret cannot be derived from the commitment alone.

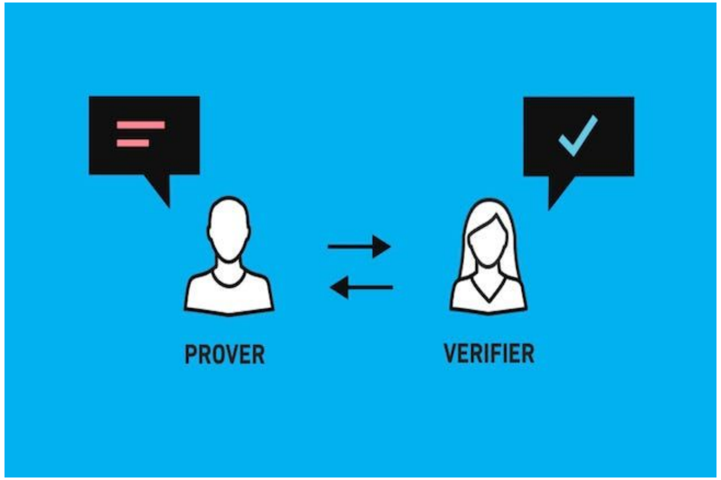


How do ZKPs work?

Basic working mechanism - 3. Verification

Challenge - The verifier sends a challenge to the prover. This challenge is a request for specific information that will help the verifier confirm the prover's knowledge without actually learning the secret.

Response - The prover prepares a response to the challenge, using their secret knowledge. This response must be crafted to satisfy the challenge without revealing the secret.

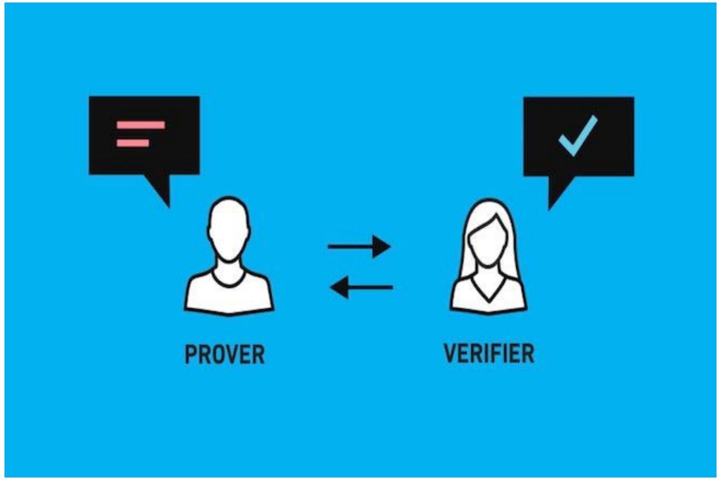


How do ZKPs work?

Basic working mechanism - 4. Confirmation

Verification - The verifier checks the prover's response. If the response is consistent with someone who possesses the secret knowledge, the verifier is convinced of the prover's knowledge.

Repeat - To increase confidence, this process can be repeated several times with different challenges.



How do ZKPs work?

Basic properties

ZKPs generally have these basic properties

1. Interactiveness
2. Soundness and Completeness
3. Non-transference

How do ZKPs work?

Basic properties - Interactiveness

Traditional ZKPs are interactive, involving multiple rounds of communication between the prover and the verifier.

In each round, the verifier challenges the prover, and the prover responds with evidence that supports the validity of their statement.

Importantly, these interactions do not leak any information about the proof itself.

How do ZKPs work?

Basic properties - Soundness and Completeness

Soundness - If the prover is dishonest, they cannot convince the verifier of a false statement.

Completeness - If the statement is true, an honest prover can always convince the honest verifier.

How do ZKPs work?

Basic properties - Non-transference

The proof cannot be transferred or reused by the verifier to convince another party.

What are SNARKs & STARKs?

Two most prominent zero knowledge proof systems.

SNARK stands for **S**uccinct **N**on-Interactive **Ar**gument of **K**nowledge

STARK stands for **S**calable **T**ransparent **Ar**gument of **K**nowledge

What are SNARKs & STARKs?

SNARKS are **succinct** and **non-interactive**.

Succinct means the **proof size is small** and can be quickly verified.

Non-interactive means it **doesn't require back-and-forth communication** between the prover and verifier.

What are SNARKs & STARKs?

STARKS are **scalable** and **transparent**.

Scalable means the **proof size scales slowly** with size of computation being proved. Hence better suited for larger computations and extensive datasets.

Transparent means they **don't rely on a trusted setup** enhancing their security and trustworthiness.

quezar

What are SNARKs & STARKs?

Each has its own strengths and weaknesses.

And hence are suitable for specific use cases.

Source: https://hacken.io/discover/zk-snark-vs-zk-stark/

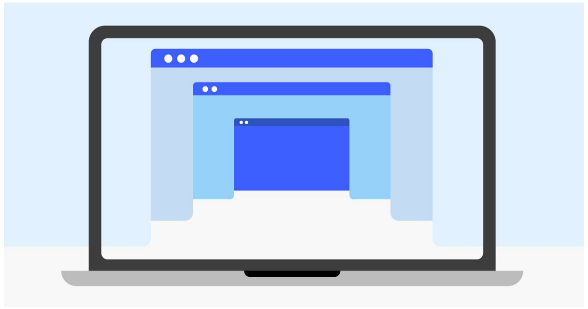
	ZK-SNARK	ZK-STARK
Proof Size	Low. Allow for EVM data availability	High. Driving the costs up
Trust Setup	Requires a trusted setup	Does not require a trusted setup
Verification Time	Fast verification times	Faster times only with large datasets
Quantum Security	Not quantum-resistant	Quantum-resistant
Transparency	Less transparent due to trusted setup	More transparent using public verifiable randomness
Scalability	Less scalable, linear increase	Highly scalable
Use Cases	Best for systems where proof size and speed are key	Best where transparency and quantum-resistance are priorities

What are zkVMs & zkEVMs?

VMs or Virtual Machines are programs that can other programs.

zkVMs are Zero Knowledge Virtual Machines

zkEVMs are Zero Knowledge Ethereum Virtual Machines



quezar

What are zkVMs & zkEVMs?

zkVMs or Zero Knowledge Virtual Machines

Incorporates zero-knowledge proofs to enable private computation.

It allows for the execution of programs or smart contracts while keeping the state and inputs of these computations private.

Examples include Cairo (by Starknet), Miden (by Polygon) and Risczero.

quezar

What are zkVMs & zkEVMs?

zkEVMs or Zero Knowledge Ethereum Virtual Machines

Specific type of zkVM designed to be compatible with Ethereum's virtual machine (EVM).

They enable the execution of smart contracts on the Ethereum blockchain using zero-knowledge proofs. This keeps transaction data and contract state private.

Examples include Polygon zkEVM, Scroll and zkSync.

quezar

Introduction to Mina

We will cover

- An overview of Mina protocol
- A brief walkthrough of protocol architecture
- How’s Mina different from other layer 1 protocols

Overview of Mina protocol

L1 blockchain based on **zero-knowledge proofs**

Succinct blockchain, only **22 KB in size**

Secured by proof of stake (PoS) consensus called **Ouroboros Samisika**

MINA

quezar

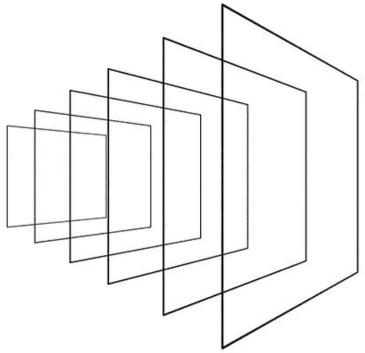
A brief walkthrough of protocol architecture

Let’s start with a TLDR of Mina’s protocol architecture:

Entire Mina blockchain is represented by a **single SNARK**

It keeps the blockchain size constant at **~22KBs**

This is achieved by “**Recursive Composition of SNARKs**”



A brief walkthrough of protocol architecture

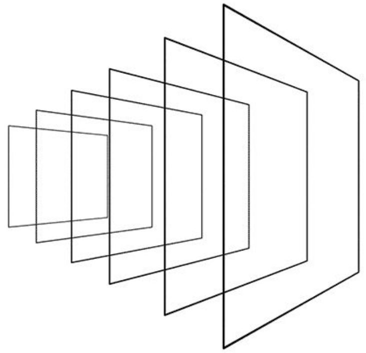
Okay, now let’s unpack it SNARK by SNARK. ;)

Problem with other blockchains is

Running a full node is **resource extensive**

And it **increases exponentially** with time

Limiting **scalability** and **decentralization** (and hence **security**)



quezar

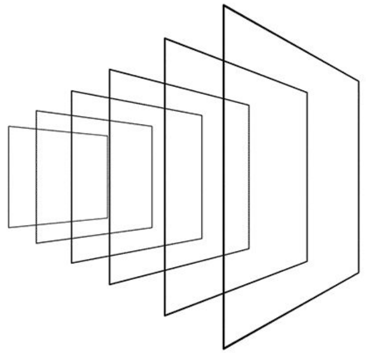
A brief walkthrough of protocol architecture

Okay, why is it resource extensive?

Because of **trust**.

The entire chain history is required to verify the current consensus state of the blockchains.

That’s how we establish trust on a decentralized network of anonymous node operators.



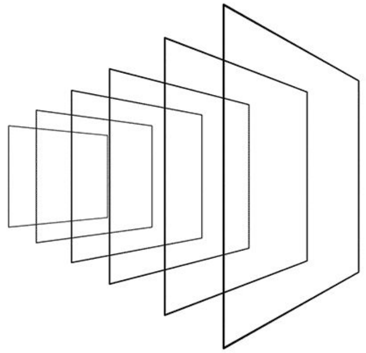
A brief walkthrough of protocol architecture

Hmm, then the root cause of the problem becomes?

The **verification mechanism**.

Let’s take an analogy to understand it better.

Imagine every time you spent a dollar, you have to check the entire history of how that dollar has been spent before you will spend it.



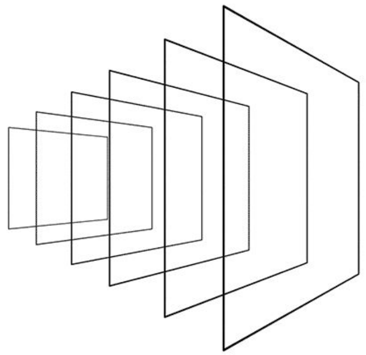
A brief walkthrough of protocol architecture

I see, so how do we solve this problem?

If only we had some way to certify that someone had already checked the history for us.

Like a certificate?

Sounds like something we've already heard.



A brief walkthrough of protocol architecture

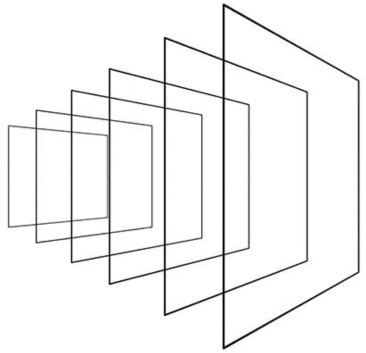
Enter zk SNARKs!

A certificate to prove that a computation was performed correctly.

Tiny (<1KB generally)

Easy to check (<10 seconds generally)

Versatile (works for any computation, no matter how complicated)



A brief walkthrough of protocol architecture

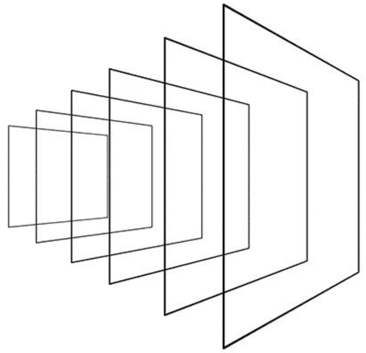
Great! How's it used in Mina?

Updating a blockchain means changing the state of accounts.

This is a computation.

And SNARKs can certify any computation.

So **a SNARK can certify each block update**



A brief walkthrough of protocol architecture

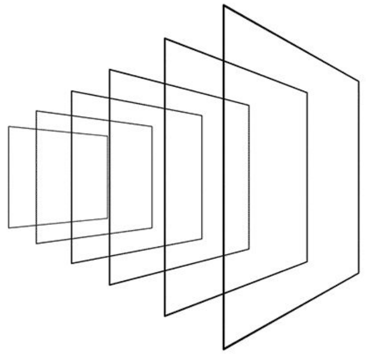
But a blockchain is a chain of block updates. So it should be a chain of SNARKs then?

Yes. But verifying previous SNARKs is a computation itself.

It can also be represented as a SNARK.

So entire blockchain can be represented by a single SNARK.

This is called **Recursive Composition of SNARKs**.



quezar

A brief walkthrough of protocol architecture

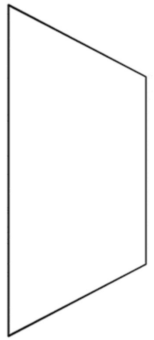
But a blockchain is a chain of block updates. So it should be a chain of SNARKs then?

Yes. But verifying previous SNARKs is a computation itself.

It can also be represented as a SNARK.

So entire blockchain can be represented by a single SNARK.

This is called **Recursive Composition of SNARKs**.



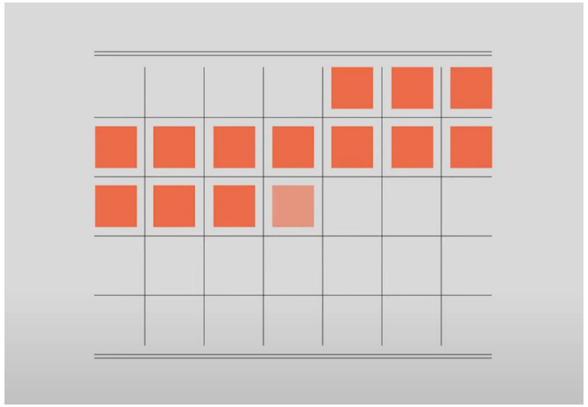
A brief walkthrough of protocol architecture

One more thing, the Ouroboros Samasika consensus mechanism.

A secure POS protocol that can resolve long-range forks without relying on third parties to provide history.

Uses VRFs to select winning block producers

No staking minimum, no slashing



How’s Mina different from other Layer 1s?

L1 blockchain based on zero-knowledge proofs

Enhanced Privacy and Security - Privacy by default. Private transactions and smart contracts with private state.

Efficient and Scalable Verification - Quick and efficient verification of the blockchain's state, regardless of the number of transactions in network's history.

How's Mina different from other Layer 1s?

Succinct blockchain, only 22 KB in size

Constant size blockchain - With ever growing amount of transactions and data, Mina stays at a constant size of 22KB as compared to other blockchains which require hundreds of GBs.

Higher Decentralisation - lightweight nature allows for easy full node operation, can even run on a smartphone or a browser. This lowers barrier to entry, enable broad participation in consensus.

quezar

Overview of zkApps

quezar

We will cover

- What are zkApps?
- How are they different from dapps?
- How does a zkApp work on Mina?

What are zkApps?

A zkApp

Is a **dApp**

That uses **zero knowledge proofs or ZKPs**

To enhance **security** and **privacy**

How are they different from dapps?

zkApps are different from dApps as they use model of

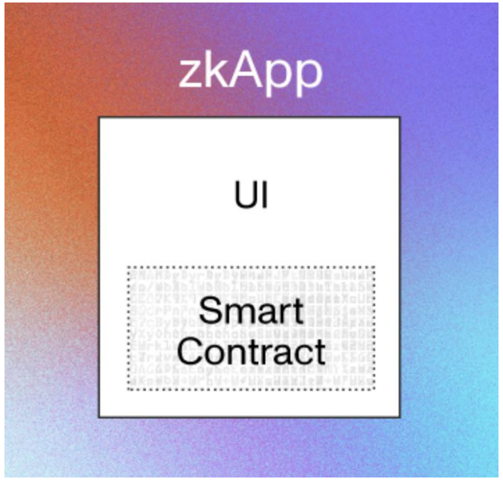
Off-chain execution - computation is done off-chain and is submitted in the form of a ZKP to the network for verification

Mostly off-chain state - very little storage is available for on-chain state, most of the state is kept off-chain

How does a zkApp work on Mina?

A zkApp consists of two parts:

1. A **smart contract** - code written with o1js
2. A **user interface (UI)** - for users to interact with the zkApp



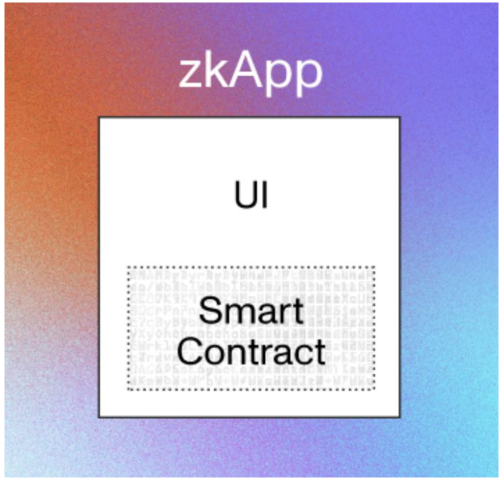
How does a zkApp work on Mina?

The smart contract

A provable code is written by using o1js

During build process it generates

- A prover function and
- A corresponding verifier function



How does a zkApp work on Mina?

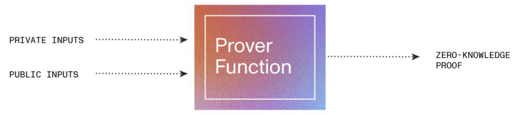
The prover function

Runs in user’s browser

Executes smart contract’s custom logic based on inputs by the user

Generates a proof of executed code

Each method in a o1js smart contract results in proof generation

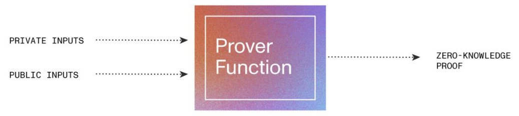


How does a zkApp work on Mina?

The prover function

Private inputs - the data is never seen by the blockchain

Public inputs - the data is stored on-chain or off-chain, depending on what the zkApp specified as required for a given use case.

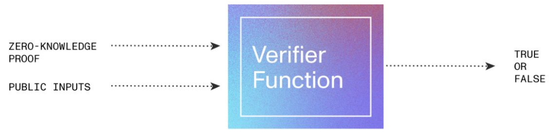


How does a zkApp work on Mina?

The verifier function

Validates whether a zero knowledge proof successfully passes all the constraints defined in the prover function using the public inputs

Mina acts as a verifier and runs the verifier function.



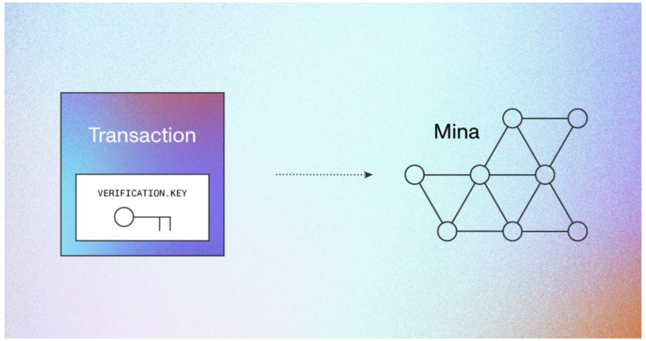
quezar

How does a zkApp work on Mina?

zkApp account

When a Mina address contains a verifier function (or verification key), it acts as a zkApp account.

Any transactions that do not pass the verifier function are rejected by the Mina network.

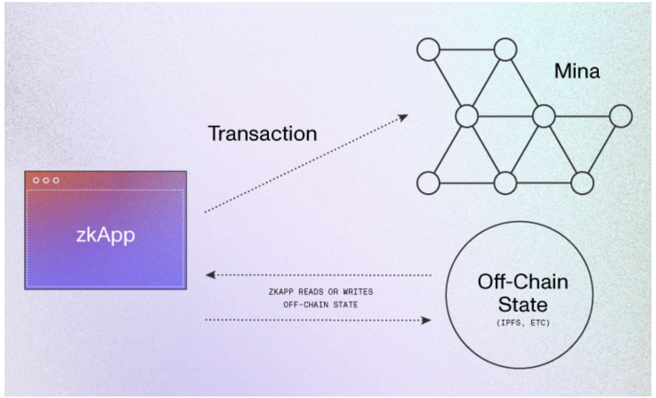


How does a zkApp work on Mina?

zkApp state

On-chain state - 8 fields of 32 bytes each available on Mina blockchain

Off-chain state - stored in an external storage like IPFS, root of Merkle tree or a similar data structure is stored on-chain



How does a zkApp work on Mina?

Updating the state of a zkApp on chain

Alongside a proof, prover function also outputs “account updates”

These are JSON plaintext describing how to update on-chain state

Passed as a public input for verifying integrity

If verification is successful, on-chain state is updated

How does a zkApp work on Mina?

Complete workflow of a zkApp

Smart contract is written using o1js

Build process creates prover and verifier functions

Prover function is deployed within the UI on a hosting service

Verifier function / verification key is deployed within the zkApp on Mina

quezar

How does a zkApp work on Mina?

Complete workflow of a zkApp

User inputs the data in UI

The inputs maybe public or private depending on the use case

Prover function generates a proof using these inputs

Prover function also creates “account update” detailing state changes

How does a zkApp work on Mina?

Complete workflow of a zkApp

The transaction is submitted by the user to the zkApp on Mina

This transaction contains the proof to be verified & public inputs.

The account updates are also passed with other public inputs.

How does a zkApp work on Mina?

Complete workflow of a zkApp

These are then verified by the verifier function in the zkApp

If the transaction is accepted, the state changes are applied to the zkApp

quezar

Example - Square

We will cover

- Some basics of o1js
- Set up development environment
- Write a basic zkApp
- Interact with it
- Deploy it on Testnet
- Compare it with a similar solidity dapp

Basics of o1js

Here’s a quick overview of some basic concepts in o1js

- **Fields** - basic unit of data in zero-knowledge proof programming, each variable is a Field
- **Smart Contract** - the base smart contract class
- **@method** - a decorator for creating smart contract functions
- **State** - a class used to create on-chain state
- **@state** - a decorator to create references to onchain state

Basics of o1js

One important thing about @method

- Under the hood, every @method defines a zk-SNARK circuit.
- A smart contract is a collection of circuits, all of which are compiled into a single prover & verification key.
- The proof means "I ran one of these methods, with some private input, and it produced this particular set of account updates".

Basics of o1js

One important thing about @method

- The account updates are public input.
- The proof will only be accepted on the network if it verifies against the verification key stored in the account.
- This ensures that same code that the zkApp developer wrote also ran on the user's device – thus, the account updates conform to the smart contract's rules.

Preparing the dev environment

For building zkApps on Mina, we'll need zkcli

We can install it by running: **npm install -g zkapp-cli**

You can check out more details here:

https://docs.minaprotocol.com/zkapps/install-zkapp-cli

Creating our project

Next, we will setup our project. Here we will

- Create a new project
  - **$ zk project square**
- Prepare it for writing our smart contracts
  - **$ zk file src/Square**
  - **$ touch src/main.ts**

Writing our first zkApp

Now we will write our first smart contract.

The logic is simple

- We’ll declare a variable aka Field with some initial value
- This value will get updated only if the new value is its square

quezar

Interacting with our first zkApp

Here we'll write a script to interact with our smart contract that

- Sets up a local blockchain
- Sets up a few accounts that we will need
- Sets up an instance of our zkApp
- And makes a few transactions to test our zkApp

Deploying our zkApp on Testnet

Awesome! Now that we have our zkApp ready, we'll deploy it on Mina testnet.

- Do required config
- Get testnet Mina
- Deploy contract on Berkley testnet

Comparison with an Ethereum dapp

Let’s a see how a similar dapp works on Ethereum.

And understand where zkApps are different from dapps.

- Entire code is deployed on-chain
- Execution happens on-chain
- All transactions are stored on-chain

quezar

Example - Private Input

We will cover

- Some new concepts
- Create our project
- Write our zkApp
- Interact with it

Some New concepts

In this part, we will learn more about

- Private input
- Poseidon hash function
- Salt argument

Some New concepts

Private input

- All inputs to a smart contract are private by default.
- Inputs are never seen by the blockchain unless you store those values as on-chain state in the zkApp account

Some New concepts

Poseidon hash function

- Optimized for fast performance inside zero knowledge proof systems.
- Takes in an array of Fields and returns a single Field as output

Some New concepts

Salt

- Adding salt as a second input to the contract code makes it harder for an attacker to reverse engineer the code and gain access to the contract.
- Salt makes the contract more secure and helps protect the data stored within it.
- For optimal security, the salt is known only to you and is typically random.

quezar

Creating our project

Okay, now we will setup our project. Like before, we will

- Create a new project
- Prepare it for writing our smart contracts

Writing our smart contract

Now we will write our smart contract. This time we will

- build a smart contract with a private state variable
- that can be modified only if a user knows the its value.

Only the result, updated hash, is revealed.

Because the salt and secret can't be deduced from their hash, they will remain private.

Interacting with our zkApp

Here we'll write a script to interact with our smart contract that

- Sets up a local blockchain
- Sets up a salt argument
- Sets up a few accounts that we will need
- Sets up an instance of our zkApp
- And makes a few transactions to test our zkApp

quezar

Example - Crowdfund

We will cover

- Some new concepts
- Create our project
- Write our zkApp
- Interact with it

Some New concepts

In this part, we will learn more about

- Account Updates
- Transactions

Some new concepts

Account updates

- The fundamental data structure that Mina transactions are built from.
- An account update always contains updates to one specific on-chain account.
    - For example, if you transfer MINA from one account to another, the balance on two accounts is updated – the sender and the receiver. Therefore, sending MINA requires two account updates.
- Can express all kinds of updates, events etc needed to develop smart contracts.

quezar

Some new concepts

Transactions

- Recall that smart contracts execute off-chain.
- The result of an off-chain execution is a transaction that can be sent to the Mina network to apply the changes made by the smart contract.
- A transaction contains multiple account updates.

Some new concepts

Transactions

- A transaction is a JSON object of the form: **{ feePayer, accountUpdates: [...], memo }**
- **feePayer -** is a special account update containing a fee field, for specifying the transaction fee.
- **accountUpdates -** is a list of normal account updates that make up the transaction.
- **memo -** is an encoded string that can be used to attach an arbitrary short message.

quezar

Creating our project

Okay, now we will setup our project like before. We will

- Create a new project
- Prepare it for writing our smart contracts

Writing our zkApp

In this example we will

- Create a zkApp that can collect funds
- Send some Mina tokens to our zkApp

Interacting with our zkApp

Here we'll write a script to interact with our smart contract that

- Sets up a local blockchain
- Sets up a few accounts that we will need
- Sets up an instance of our zkApp
- And makes a transaction to our zkApp

quezar

Example - Lottery

We will cover

- Some new concepts
- Create our project
- Write our zkApp
- Interact with it

Some new concepts

In this part, we'll learn more about

- Onchain values
- Events

Some new concepts

Onchain values

- Similar to the onchain state, we can also access onchain values in a zkApp.
- **Network values** include the current timestamp, block height, total Mina in circulation, and other network state
- **Account values** include fields and properties of the zkApp account, such as balance, nonce, and delegate

quezar

Some new concepts

Events

- Are public arbitrary information that can be passed along with a transaction
- Are not stored on-chain.
    - Only events from the most recent couple of transactions are retained by consensus nodes. After that, the events are discarded, but are still accessible on archive nodes
- Events are not meant for use within proofs directly, as they can't be predicated on inside proofs.
- Events are used to signal to UIs

Creating our project

Okay, now we will setup our project like before. We will

- Create a new project
- Prepare it for writing our smart contracts

Writing our zkApp

In this example we will combine our understanding from previous two examples

- We will create a zkApp containing a secret key & a lottery prize
- If someone is able to guess this key correctly, they win the prize

Interacting with our zkApp

Here we'll write a script to interact with our smart contract that

- Sets up a local blockchain
- Sets up a few accounts that we will need
- Sets up an instance of our zkApp
- And makes a few transactions to our zkApp

Example - Sudoku

We will cover

- Some new concepts
- Overview of our zkApp
- Walkthrough of our zkApp

Some new concepts

In this part, we'll learn about

- Structs - custom data types, just like other programming languages
- Provables - akin to arrays

Overview of our zkApp

We will cover a sudoku example code.

- In this zkApp, the creator will set a sudoku solution.
- People can then submit a solution and check if their solution is right or not

Walkthrough of our zkApp

There are three important scripts in this zkApp

- Sudoku.ts
  - Our main sudoku smart contract
- Sudoku-lib.js
  - Helper code for creating and solving sodoku
- Main.ts
  - Script to interact with our smart contract

quezar

UI

We will cover

- Setup Wallet
- Project setup & overview
- Walkthrough of the code
- Demo

Setup Wallet

Install Auro wallet - https://www.aurowallet.com/

Get testnet Mina - https://faucet.minaprotocol.com/

Project Setup & Overview

Setup project using: **$ zk project <project_name>**

Project will contain:

- Contracts
  - from the Square example
- UI
  - based on NextJS
  - add helper files - zkappWorker.ts, zkappWorkerClient.ts

Demo

We'll understand how to deploy a complete zkApp (contract + UI).

- Contract
  - Setup, build & deploy
- UI
  - Configure zkApp
  - Test UI locally

quezar

What Next?

You might want to...

- Check out the official docs to know about Mina in more detail.
    - https://docs.minaprotocol.com/

- Join the community!
    - X, Discord, Forums, Telegram etc
    - https://minaprotocol.com/community
    - https://docs.minaprotocol.com/participate/online-communities

You might want to...

- Discover projects in the ecosystem
    - https://minaprotocol.com/mina-ecosystem-grants
- Join programs to contribute to ecosystem
    - https://docs.minaprotocol.com/participate/grants-and-programs

quezar

You might want to...

- Dive deeper into ZKPs
  - ZKPs MOOC by Berkley: Youtube Playlist
  - Awesome ZKPs: Github Repo

by Quezar

Thank You!

Website: https://quezar.xyz/

Twitter: https://twitter.com/quezarHQ

Github repo:
https://github.com/QuezarHQ/mina-ecosystem-101