

## *Lesson 9 : Mina / Aleo*

Lesson 10 : zkEVM Solutions

Lesson 11 : Risc Zero / Circom

Lesson 12 : zkSNARKS Theory

### **zkOracles**

See [video](#)

See [tutorial](#)

The idea behind an oracle is to allow a zkApp to get data from any HTTPS data source.

This is a similar idea to that of [DECO] (See [paper](#)) or [Pragma Oracle](#)

The tutorial creates an oracle that

- Fetches data from the desired source
- Signs it using a Mina-compatible private key
- Returns the data, signature, and public key associated with the private key
- Allows the signature to be verified by the zkApp

## Off chain storage

There are a number of community projects in this area looking to provide simple solutions using existing decentralised storage providers such as IPFS and Filecoin.

[Tutorial 6](#) walks through an example that uses a merkle tree to store data

For validation the merkle root is stored on chain, the tree is stored off chain.

---

## Advanced Concepts

### Recursion

See [Tutorial Documentation](#)

Recursion brings use cases such as

- high-throughput applications through rollups,
- Proofs of complex computations,
- Multi-party proofs.

We generally think of one prover and one verifier, multi party proofs allow multiple provers to recursively update a proof off chain and then send it to be verified on chain. We will look at the idea of collaborative SNARKS more generally in a later lesson.

### Concurrency

One difficulty with off chain computation is synchronising state updates.

In Mina these are handled via [Actions and a Reducer](#)

See [Docs](#) for an example.

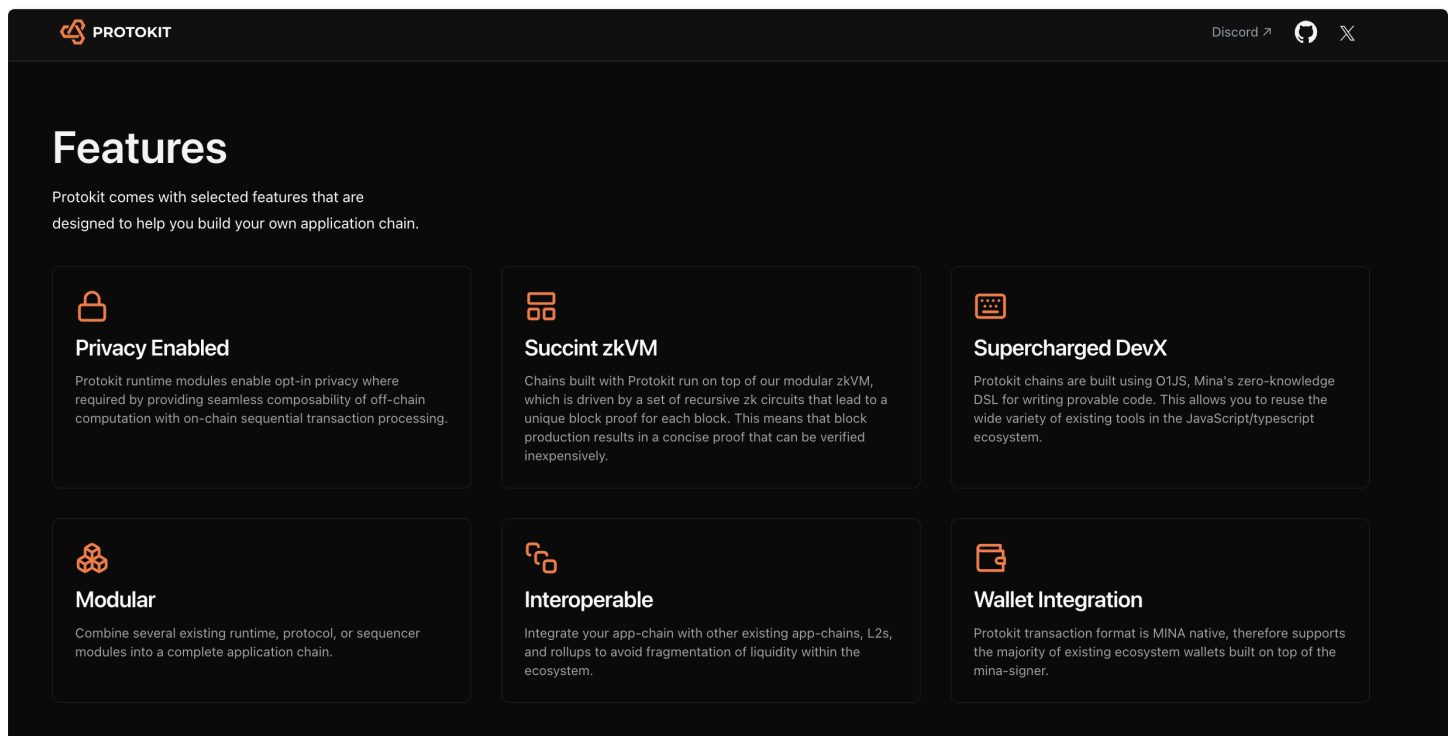
---

# Protokit

See [Docs](#)

Protokit is a project that came out of a recent Mina builder program.

Protokit enables developers to build zero-knowledge, interoperable and privacy preserving application chains with a minimal learning curve.



The screenshot shows the Protokit website with a dark theme. At the top, there's a navigation bar with the Protokit logo, a 'Discord' link, and social media icons for GitHub and Twitter. The main heading is 'Features', followed by a subtext: 'Protokit comes with selected features that are designed to help you build your own application chain.' Below this, there are six feature cards arranged in a 2x3 grid. Each card has an icon, a title, and a brief description.

Icon	Feature Name	Description
	Privacy Enabled	Protokit runtime modules enable opt-in privacy where required by providing seamless composability of off-chain computation with on-chain sequential transaction processing.
	Succinct zkVM	Chains built with Protokit run on top of our modular zkVM, which is driven by a set of recursive zk circuits that lead to a unique block proof for each block. This means that block production results in a concise proof that can be verified inexpensively.
	Supercharged DevX	Protokit chains are built using O1JS, Mina's zero-knowledge DSL for writing provable code. This allows you to reuse the wide variety of existing tools in the JavaScript/typescript ecosystem.
	Modular	Combine several existing runtime, protocol, or sequencer modules into a complete application chain.
	Interoperable	Integrate your app-chain with other existing app-chains, L2s, and rollups to avoid fragmentation of liquidity within the ecosystem.
	Wallet Integration	Protokit transaction format is MINA native, therefore supports the majority of existing ecosystem wallets built on top of the mina-signer.


They have a [developer workshop](#) this week.



# Developer Workshop

## 25th of October 2023

### Protokit Developer Workshop #1

 Hosted by Matej & Raphael Panic

OCT

25

OCT

25

Wednesday, 25 October

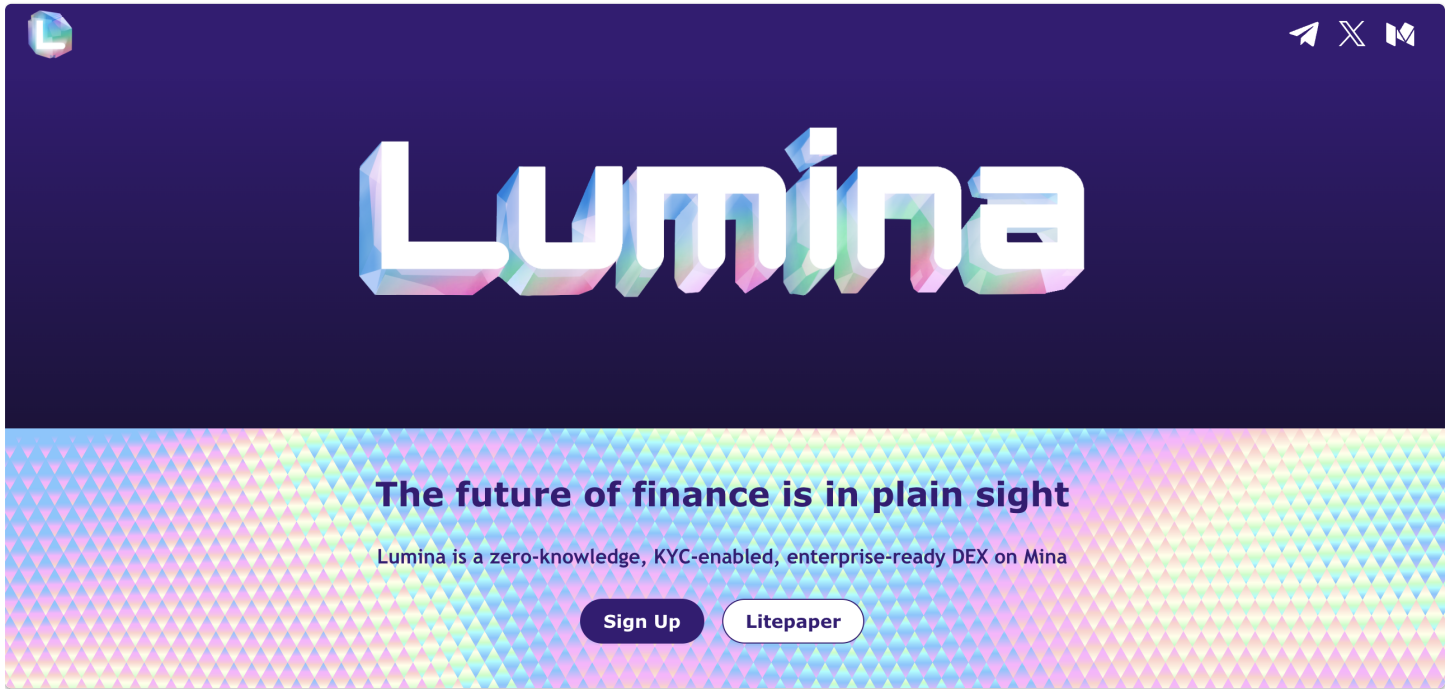
17:00 to 18:30 BST



Google Meet

# Decentralised Exchanges on Mina

Lumina



See [Site](#)

See [Litepaper](#)

Kaupang Dex

This was built using protokit :

[Kaupang Dex](#)

## Upcoming Mina Projects

### Collaboration with Etonect

A Mina Foundation grant will enable etonec to build a state-of-the-art Zero-Knowledge Proof-ID (zkp-ID) solution with KYC and AML functionality within Lumina DEX, a decentralized exchange built on the Mina Protocol

### Community projects / libraries

- **o1js-elgamal** A partially homomorphic encryption library for o1js based on Elgamal encryption: [GitHub](#) and [npm](#)
  - **o1js-pack** A library for o1js that allows a zkApp developer to pack extra data into a single Field. [GitHub](#) and [npm](#)
-

## Aleo Introduction

### An all-in-one ZK platform

#### Leo

Rust-based DSL with syntax that abstracts low-level cryptography, making it easy to express logic in zero-knowledge.

#### snarkOS

Permissionless and scalable network for ZK powered smart contracts fueled by our novel consensus protocol, AleoBFT.

#### snarkVM

A powerful virtual machine for zero-knowledge execution featuring a custom immediate representation (IR), unlimited runtime, and efficient proof generation.



## Consensus Mechanism

Aleo uses a hybrid Pos / Pow / BFT mechanism called AleoBFT

This [article](#) provides an overview.

## General concepts

### SnarkVM

This is the virtual machine responsible for running programs. Computation is done off chain and verified on chain.

### Transactions

Transactions are used to hold transitions which will update the state of the ledger. They are signed by an owner and require a fee.

### Records

These are the data structure responsible for holding state, they are created and consumed by transitions.

In concept they have a similarity with Notes in ZCash and Aztec. The record has an owner and can be encrypted for privacy.

### Programs



Programs hold Aleo instructions (an IR).

Program inputs can be private, the state that the program operates on is supplied in records.

The output of the program also includes a proof that the computation proceeded correctly.

---

## Development on Aleo

See [Docs](#)

Leo language

See [Docs](#)

Leo is a functional, statically-typed programming language built for writing private applications.

It is similar to Rust in syntax, and like other zkp DSLs it has a restricted set of datatypes, but supports

- Integers of different sizes
- Booleans
- Addresses
- Field, Group and Scalar elements
- Signatures

Leo code compiles to Aleo instructions which are an intermediate representation, which then compile to bytecode.

The bytecode consists of AVM codes, which run on the Aleo virtual machine.

Leo installation

See [Instructions](#)

Leo Playground

Available at [playground](#)

The Leo wallet

See [Wallet](#)

See [Adapter Docs](#)