MINA

“the world’s lightest blockchain”

Attività Progettuale di Calcolatori Elettronici M

Lorenzo Ziosi

Prof. Andrea Bartolini

Supervisors: Nicola Elia, Francesco Barchi

Agenda

- Introduction to Mina
- Introduction to zkApps
- Problems
- TicTacToe (Mina & EVM)
- Performance comparison

Introduction to Mina

Mina is an L1 blockchain based on zero-knowledge proofs (“ZKP”) with smart contracts written in TypeScript. It is the first cryptocurrency protocol with a succinct blockchain (22KB).

With Mina, the blockchain always remains a constant size–about 22KB (the size of a few tweets). It’s possible to verify the current consensus state of the protocol using this one recursive, 22KB zero-knowledge proof. This means participants can quickly sync and verify the current consensus state of the network.

Introduction to zkApps (1)

Mina's zero-knowledge smart contracts are referred to as "zkApps". zkApps provide powerful and unique characteristics such as unlimited **off-chain execution**, privacy for private data inputs that are never seen by the blockchain, the ability to write smart contracts in **TypeScript**, & more.

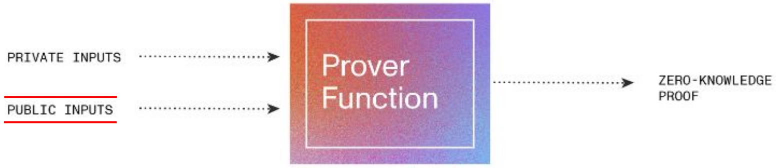
Because zkApps are based on **zero-knowledge proofs (zk-SNARKs)**, a zkApp developer writes what is called a "**circuit**". A circuit is the method from which a **prover function** and a corresponding **verifier function** are derived during the build process.

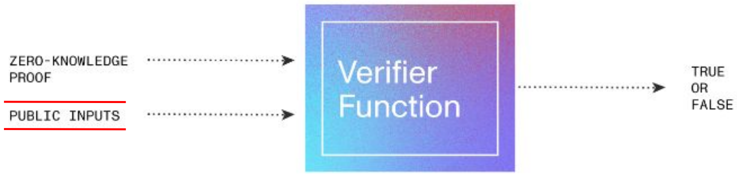
Introduction to zkApps (2)

The **prover function** is the function that executes a smart contract's custom logic.

Both **private inputs** and **public inputs** represent data that must be provided to the prover function [...] **Private inputs** are not needed again after that point.

The **verifier function** is the function that validates whether a zero-knowledge proof successfully passes all the constraints defined in the prover function. This *always* runs quickly and efficiently *irrespective of the prover function's complexity*.





Problems

- Slow and not always available funding - transactions may remain in pending state indefinitely
- Very large block time: 3 to 20 minutes per transaction
- Continuous contract's state fetching - before the execution and after the creation of each block
- Identical multiple transactions (due to a fast execution of the test script and the large block time) are rejected by the network
- Transactions may remain in pending state indefinitely - solvable by manually incrementing the next transaction's nonce
- Berkeley testnet gets regularly reset
- ...

TicTacToe (Mina)

Mina: https://github.com/o1-labs/zkapp-cli/tree/main/examples/tictactoe/ts/src
EVM: https://github.com/0xosas/tictactoe.sol/tree/master/contracts

zkApp written in **TypeScript** and tested on a **local** instance of the **blockchain**.

The main functions enable the start of a **new game** and the execution of a **new player move** from a particular player.

The final output of the function is a ZK-proof that will then be verified by the verifier nodes and thus inserted into the Blockchain. The **computations** for the function execution are carried out in the **client node**.

The test script has been modified to support the **Berkeley testnet,** which is the only network currently supporting zkApps.

TicTacToe (EVM)

Mina: https://github.com/o1-labs/zkapp-cli/tree/main/examples/tictactoe/ts/src
EVM: https://github.com/0xosas/tictactoe.sol/tree/master/contracts

Smart Contract written in **Solidity**.

The supported functions are nearly identical to Mina's. Some differences occur in the **state management**.

A user transaction requires the execution of a contract's function by **each one of the network nodes**; the execution implies the internal state change.

The test file has been built using the Truffle suite connected with the **Binance Smart Chain testnet**. This network is very performant thanks to its **low block time equal to about 3s**.

TicTacToe (comparison)

Mina: https://github.com/o1-labs/zkapp-cli/tree/main/examples/tictactoe/ts/src
EVM: https://github.com/0xosas/tictactoe.sol/tree/master/contracts

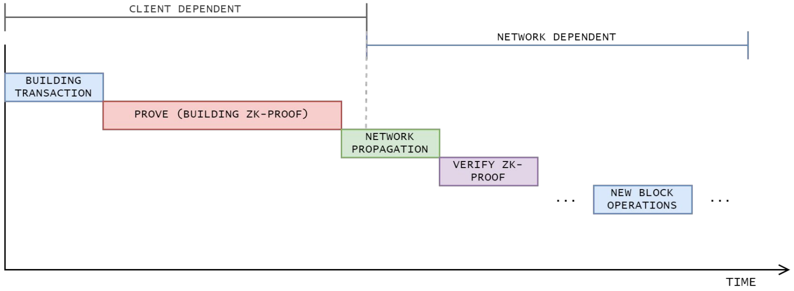
	Mina	EVM

State	@state(Field) board = State<Field>();
	@state(Bool) nextIsPlayer2 = State<Bool>();
	@state(Bool) gameDone = State<Bool>();
	@state(PublicKey) player1 = State<PublicKey>();
	@state(PublicKey) player2 = State<PublicKey>();
		uint256 gameBoard = 0;
		address internal playerOne;
		address internal playerTwo;

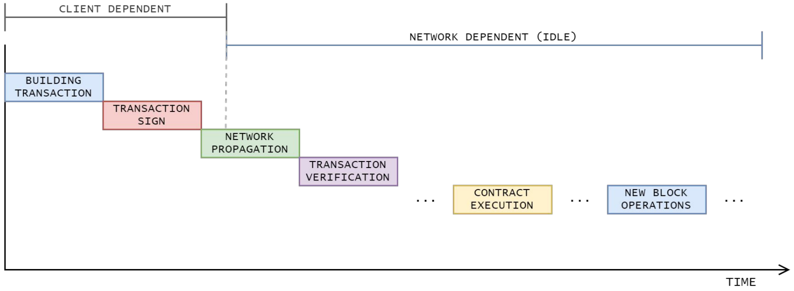
New game	@method startGame(player1: PublicKey, player2: PublicKey)	function newGame() external isPlayer(msg.sender) returns (uint256)
New move	@method play(pubkey: PublicKey, signature: Signature, x: Field, y: Field)	function move(uint256 _move) isPlayer(msg.sender) isTurn(msg.sender) moveIsValid(_move) external returns (uint256)

Performance comparison (1)

Mina



EVM



These charts show the chain of operations that each client makes on the two different blockchains.

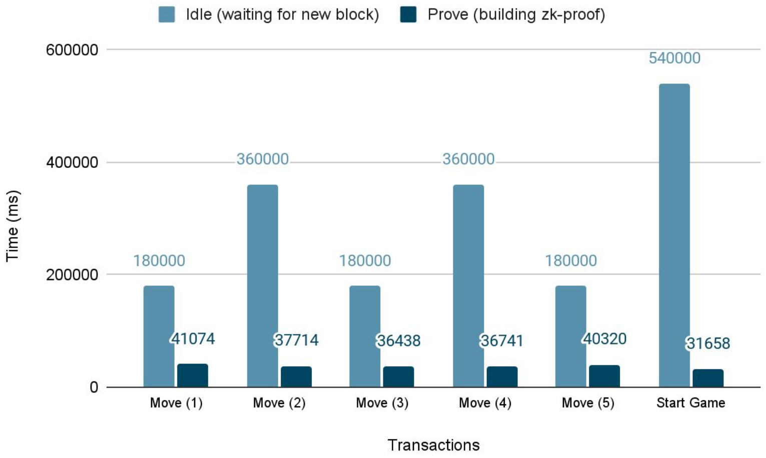
Some time (referred to as “client dependent”) will be spent by the clients to actually make computations, while some time (referred to as “network dependent” or “idle”) will be spent waiting for the network to process and validate the transactions

The comparison **takeaway** is that while in Mina the most computing-intensive tasks are carried out by the client machine, EVM leaves the actual execution to the network nodes.

Performance comparison (2)

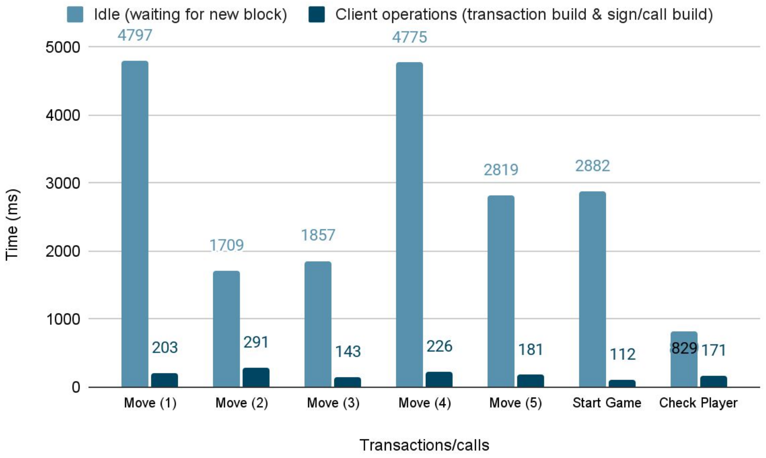
All the results are obtained on a HP 15-cx0997nl laptop, equipped with an Intel® Core™ i7-8750H (6 core, 2.2 to 4.1 GHz) and 16 GB DDR4-2666 SDRAM.

Mina



*Only the *Prove* function of client dependent operations has been indicated as it is the most computationally expensive.

EVM



Check Player is a read-only call. Therefore, it does not imply the execution of a transaction, nor the creation of any block. In this case, the idle time represents the response time of the remote node.