Lesson 8 - Account Abstraction

Newton

Useful learning resource

Latest Changes to Starknet

See docs

Planned Updates to Cairo

Cairo v 1.0 - available in ~ 2 months
Focus - Developer happiness

Sierra

A new intermediate level representation
Transactions should always be provable
Asserts are converted to if statements, if it returns false we don't do any modifications to storage
Contracts will count gas
Still needs to be low level enough to be efficient
So the process would be
Cairo (new) => Sierra => Cairo bytecode
Sierra bytecode

- cannot fail
- counts gas
- compiles to Cairo with virtually no overhead

Events continued

Documentation

Events for transactions that have a state of at least PENDING will have event information in their transaction receipt.

```
"events":_[
____{
________"data":_[
____________"0x0",
____________"0x10e1"
________],
________"from_address":_"0x14acf3b7e92f97adee4d5359a7de3d673582f0ce03d338
________"keys":_[
____________"0x3db3da4221c078e78bd987e54e1cc24570d89a7002cefa33e548d6c72c
________]
____}
]
```

the starknet CLI gives you access to the transaction receipt

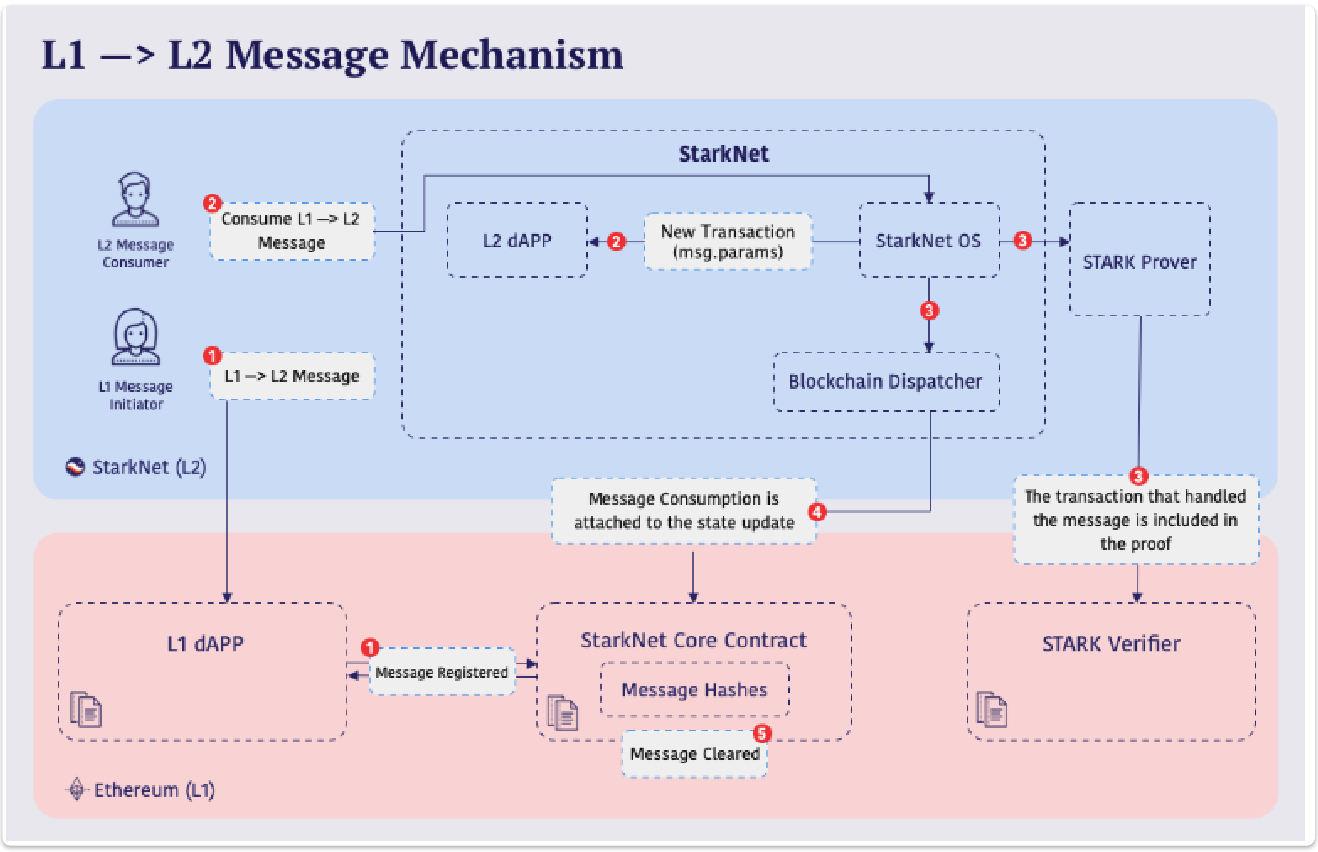
```
starknet get_transaction_receipt --hash ${TRANSACTION_HASH}
```

Support for querying events is limited.

Apibara

This project is similar to the Graph and can be used to index starknet events.
There is a tutorial here to show how to index events for an NFT

L1 to L2 Messaging

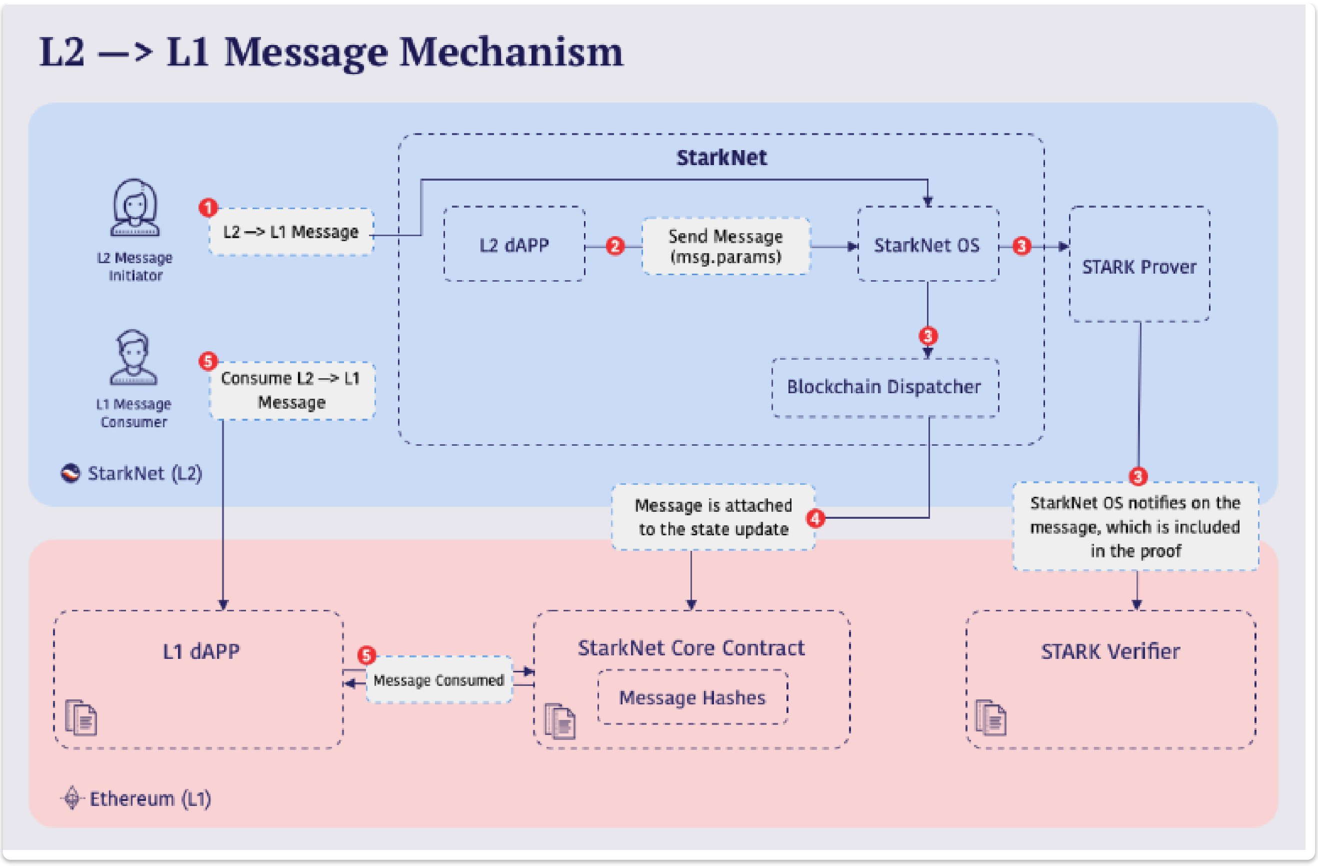


1. The L1 contract calls (on L1) the `send_message()` function of the StarkNet core contract, which stores the message. In this case the message includes an additional field - the “selector”, which determines what function to call in the corresponding L2 contract.
2. The StarkNet Sequencer automatically consumes the message and invokes the requested L2 function of the contract designated by the “to” address.

This direction is useful, for example, for “deposit” transactions.

Note that while honest Sequencers automatically consume L1 -> L2 messages, it is not enforced by the protocol (so a Sequencer may choose to skip a message). This should be taken into account when designing the message protocol between the two contracts.

L2 to L1 Mesaging



Messages from L2 to L1 work as follows:

1. The StarkNet (L2) contract function calls the library function `send_message_to_l1()` in order to send the message. It specifies:
    1. The destination L1 contract ("to"),
    2. The data to send ("payload")

   The StarkNet OS adds the "from" address, which is the L2 address of the contract sending the message.

2. Once a state update containing the L2 transaction is accepted on-chain, the message is stored on the L1 StarkNet core contract, waiting to be consumed.

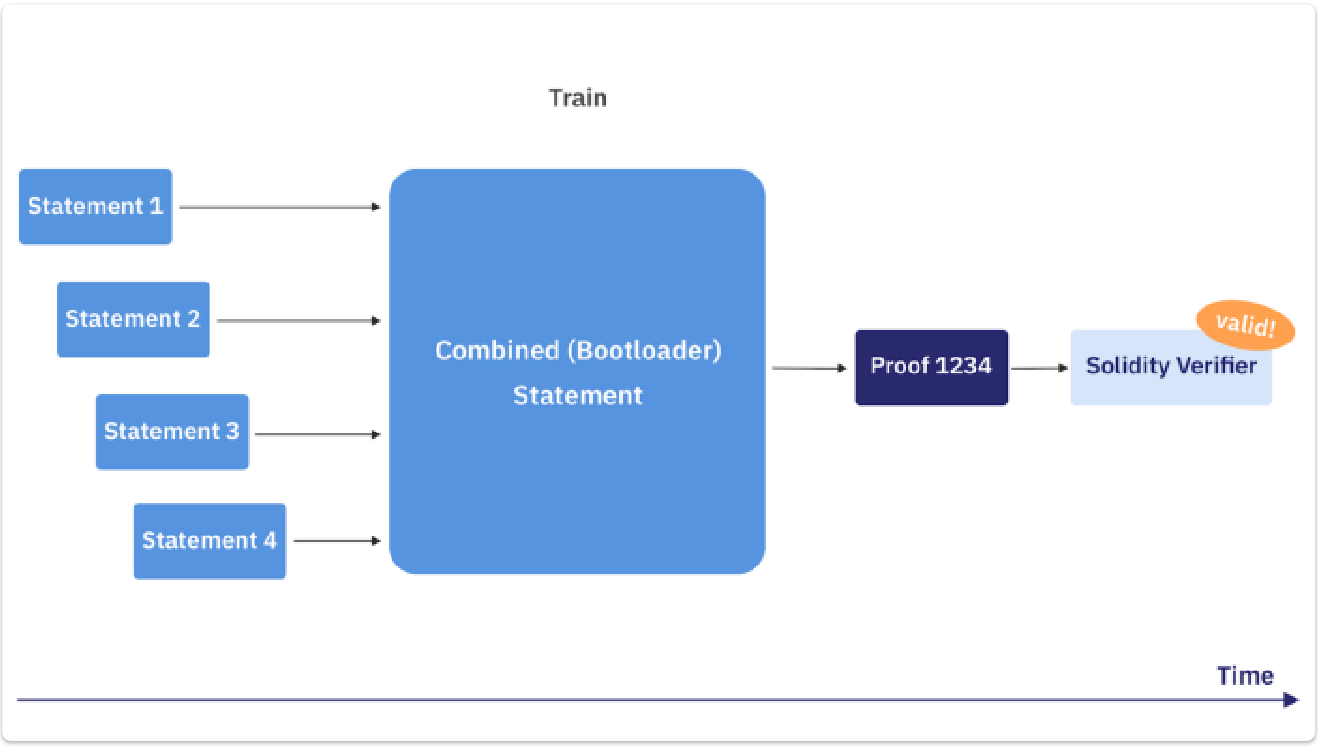
3. The L1 contract specified by the "to" address invokes the `consumeMessageFromL2()` of the StarkNet core contract.

Note: Since any L2 contract can send messages to any L1 contract it is recommended that the L1 contract check the "from" address before processing the transaction.

Recursive STARKS

See post

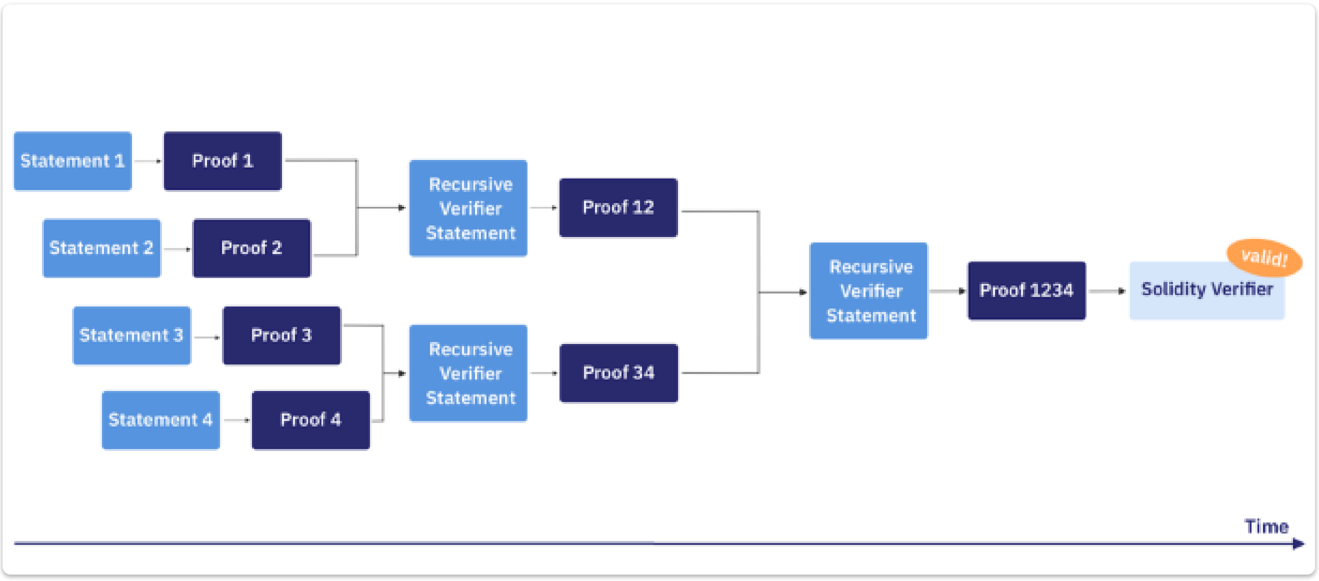
Initially SHARP (The shared prover) would process proofs from applications sequentially, oncce a threshold number of tranactions had arrived, a proof would be generated for all of them.



The amount of memory needed to to generate the proof was a limiting factor.

STARKs have roughly linear proving time and log validation time.

Recursive proofs



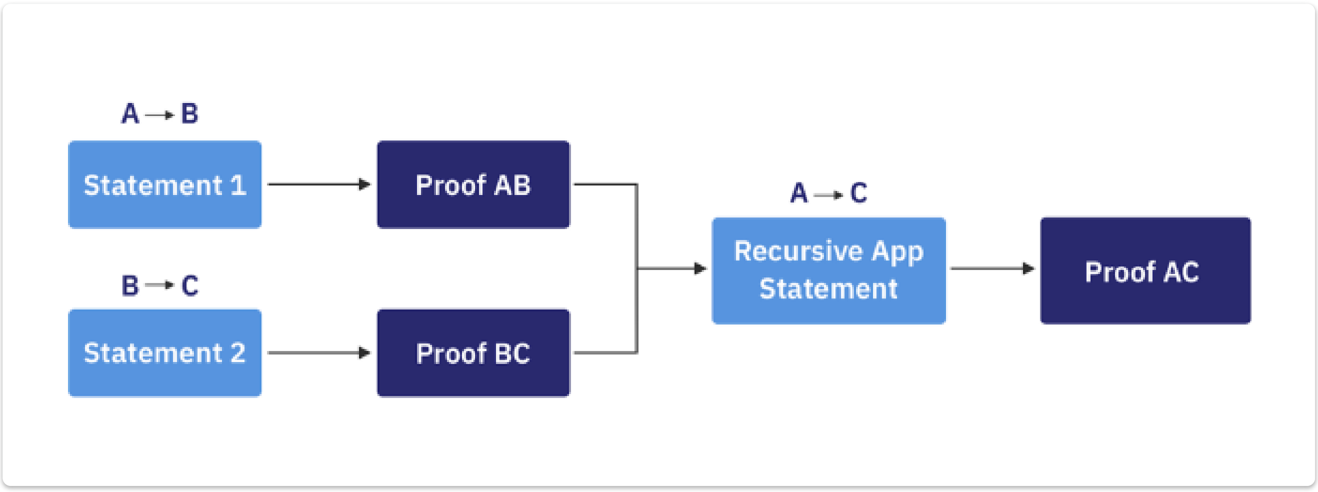
Here the proofs are calculated in parallel, then combined in pairs and a proof created and so on.

This results in

1. Reduced on chain cost, and memory requirements
2. Reduced latency since the proofs can be computed in parallel and we don't need to wait for the final proof to arrive.

Application recursion

Each STARK proof attests to the validity of a statement applied to some input
STARK recursion compresses two proofs with *two* inputs into *one* proof with two inputs. In other words, while the number of proofs is reduced, the number of inputs is kept constant. If the recursive statement is allowed to be *application-aware*, i.e. recognizes the semantics of the application itself, it can both compress two proofs into one *as well as* combine the two inputs into one.



Paying fees on testnet

To get some testnet L2 ETH
you can

1. Use the Starknet Faucet
2. Transfer ETH from L1 Goerli via the starkgate [bridge]
   ((https://goerli.starkgate.starknet.io/).

Account Abstraction

See docs from Nethermind
Also this blog from Ethereum and this blog from Gnosis

Why Account Abstraction matters

The shift from EOAs to smart contract wallets with arbitrary verification logic paves the way for a series of improvements to wallet designs, as well as reducing complexity for end users. Some of the improvements Account Abstraction brings include:

- Paying for transactions in currencies other than ETH
- The ability for third parties to cover transaction fees
- Support for more efficient signature schemes (Schnorr, BLS) as well as quantum-safe ones (Lamport, Winternitz)
- Support for multisig transations
- Support for social recovery

Previous solutions relied on centralized relay services or a steep gas overhead, which inevitably fell on the users' EOA.

EIP-4337 is a collaborative effort between the Ethereum Foundation, OpenGSN, and Nethermind to achieve Account Abstraction in a user-friendly, decentralized way.

Account Abstraction on Starknet

An account is a contract address.
With abstract accounts, it's the who (address) that matters, not the how (signature).

A user, will still have an address that can be shared publicly so that someone can send a token to that address.
The user still has a wallet with private keys that are used to sign transactions.

The difference is that the address will be a contract. The contract can contain any code. Below is an example of a stub contract that could be used as an account. The user deploys this contract and calls `initialize()`, storing the public key that their wallet generated.

```
A minimalist Account contract ###

# This function is called once to set up the account contract.

@external
func initialize(public_key):
    store_public_key(public_key)
    return ()
end

# This function is called for every transaction the user makes.
@external
func execute(destination, transaction_data, signature):
    # Check the signature used matches the one stored.
    check_signature(transaction_data, signature)
    # Increment the transaction counter for safety.
    increase_nonce()
    # Call the specified destination contract.
    call_destination(destination, transaction_data)
    return ()
end
```

Then to transfer a token, they put together the details (token quantity, recipient) and sign the message with their wallet. Then `execute()` is called, passing the signed message and intended destination (the address of the token contract).

A user can define what they want their account to "be". For many users, an account contract will perform a signature check and then call the destination.

However, the contract may do anything:

- Check multiple signatures.
- Receive and swap a stable coin before paying for the transaction fee.
- Only agree to pay for the transaction if a trade was profitable.
- Use funds from a mixer to pay for the transaction, separating the deposit from withdrawal.
- Send a transaction on behalf of itself - a DAO.

If you wish to explore this more, this is a useful workshop explaining account abstraction. There are account libraries available from Open Zeppelin

Multicalls

A `multicall` aggregates the results from multiple contract calls. This reduces the number of seperate API Client or JSON-RPC requests that need to be sent. In addition it acts as an `atomic` invocation where all values are returned for the same block.

Popular wallet providers like Argent use this design to implement account contracts on StarkNet to accomodate a multicall or a single call with one scheme.

There are many implementations of multicall that allow the caller flexibility in how they distribute and batch their transactions.

Game development on Starknet

"I define the on-chain gaming thesis as this: on-chain games will be the frontier for technological advancement of blockchain for the next five years. Web3 game developers like MatchboxDAO are building the toolkit for future consumer-facing Dapps."

See Matchbox DAO blog
Matchbox DAO repo

Solve2Mint

Solve 2 Mint is a framework for NFT emission where each NFT maps to a unique solution to an equation or puzzle.

Topology

Topology are also very involved in this area.
Their philosophy is that usually the game developers are 'gods' but with blockchain and zkps, we can prevent this and allow a common exploration of problems.
They have maxims such as "Compute more, store less" which makes economic sense on a layer 2.
They also believe that humans are at a disadvantage to bots, since humans need to go through devices to interact with a game.
They think that games should be designed around NP-hard problems so that an algorithm cannot be written to solve it.
They also advise to add randomness from the user, because if the game is deterministic, then someone could simulate theh future off chain.

Game interaction has been helped by account abstraction and the use of session keys for example with the Argentx wallet.

Game design talk : Video from Topology

Fountain

A 2-dimensional physics engine written in Cairo
repo
See tweet

Roll Your Own

Roll Your Own - A Dope Wars open universe project.

A modular game engine architecture for the StarkNet L2 roll-up, where computation is cheap and new game styles are being explored. Roll your own module and join the ecosystem.

Influence

Influence is an immersive, persistent space strategy game where players compete through multiple avenues, including: mining, building, trading, researching, and fighting, and interact in a 3D universe through third-person control of their ships and the installations they build.

Advantages using starknet

1. High Scalability & Low Fees
   Moving to Starknet, they expect a reduction of 100X in fees
2. Instant Actions
   Actions are regarded as final once they are submitted to the sequencer.
3. More complex transactions at the same price

Loot Realms

Advantages

1. Cheap Fees
2. Heavy computation possible
3. Composability of game items and liquidity

Dope War

Design

Physics Engine on Starknet

Video

Also Physics puzzle

Dark Forest (Not on Starknet)

Dark Forest is a massively multiplayer online real-time strategy (MMORTS) space conquest game build on top of the Ethereum blockchain.

From a medium article describing the game

Dark Forest game V0.3 uses zero-knowledge proof technology to prove 2 operations regarding the planet’s location:
1/ planet initialization (init)
2/ planet movement (move).
The Circuit logic is implemented in darkforest-v0.3/circuits/.
They are implemented with circom, and the circuit proof uses Groth16.

0xParc

0xParc is a research organisation focusing on crypto primitives (such as ZK), tooling, infrastructure & education.

PROJECTS

- dfdao which has extentended and adapted the original DF code base with new games types. They experimented with bots, in game guilds and new tactics: See twitter thread
- Mud is onchain gaming framework based on the Entity Component System architecture pattern. Comprised of:
  > - Solec: Solidity entity component system.
  > - Recs: Reactive entity component system in typescript.
  > - Network: Typescript lib for synchronising contract & client state.
- OpCraft Onchain version of minecraft not ZK but proof of concept for mud. Runs on it's own optimism chain build with OPstack.

AREAS OF RESEARCH:

- Gameplay mechanics: Onchain gaming is more about social dynamics, game theory, building tools, bots, new clients & interfaces than the passive gameplay of traditional games.
- Membership proofs. Prove you own one of the private keys from a list of public keys without revealing identity. 0xParc
- Witness encryption: Unlock encryption by solving some puzzle or meeting some set of criteria. Provide some proof that you completed some tasks to decrypt.
- Fully Homomorphic Encryption: Computation without seeing the data. Example: a onchain poker game where we can prove our hand wins without revealing our cards.
- Multiparty Computation: Compute some solution across multiple actors while keeping all inputs secret.
- Indistinguishability Obfuscation: obfuscation hides the implementation of a program while still allowing users to run it.

See: ZKPs and "Programmable Cryptography"

Non technical sidequest: Writings about philosophy of onchain gaming/Motivation behind onchain gaming as low risk, open space for experiment: Moving Castle

Devcon6 talk: Dark Forest: Lessons from 3 Years of On-Chain Gaming