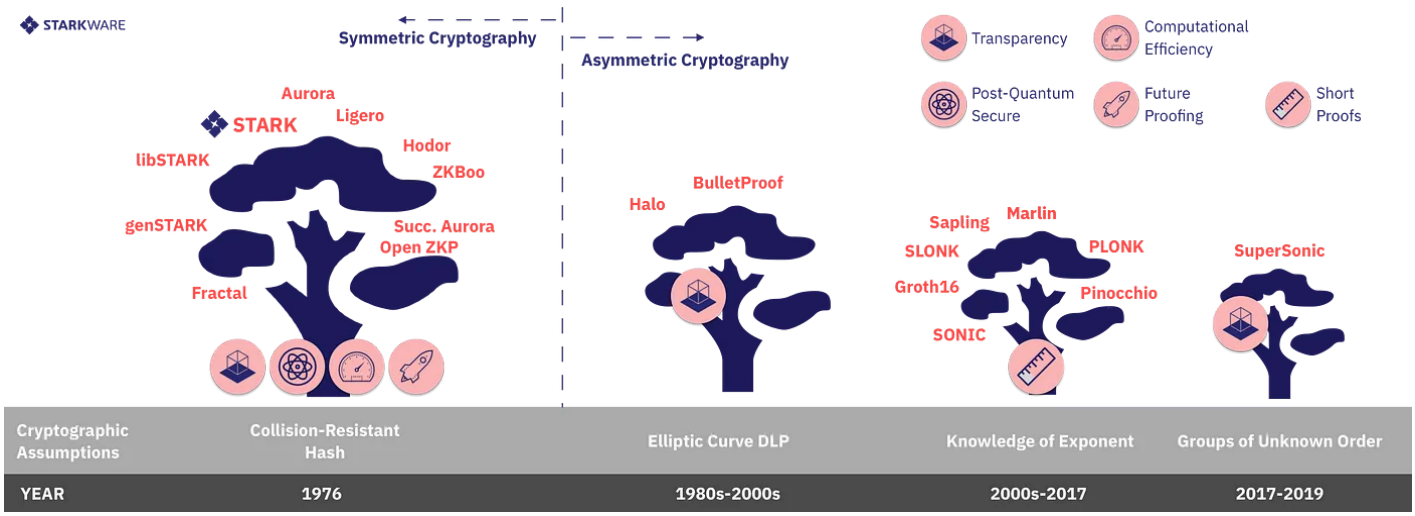
Lesson 3

Lesson topics

- ZKP System comparison
- ZKP Use cases

ZKP Comparison

Real Life ZKP choices



	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	Strong 😒	Collision resistant hashes 😄	Discrete log 😒

SNARKS / STARKS are general purpose systems for creating proofs.

We don't always need SNARKS or STARKS other techniques may be more efficient for our use case.

Other useful techniques

- Blind signatures
- Accumulators
- Pedersen commitments
- Sigma protocols

Advantages of SNARKS

- Small proof size
- Fast verification
- Generic approach

Drawbacks to SNARKS

- Require a trusted setup
- Very long proving keys
- Proof generation is impractical on constrained devices
- Strong security assumptions haven't been well tested.
- Theoretical background is difficult to understand

Advantages of Sigma Protocols

- Very economical for small circuits
- Do not require a trusted setup
- Security assumptions are weak and are well understood
- Maths is fairly easy to understand

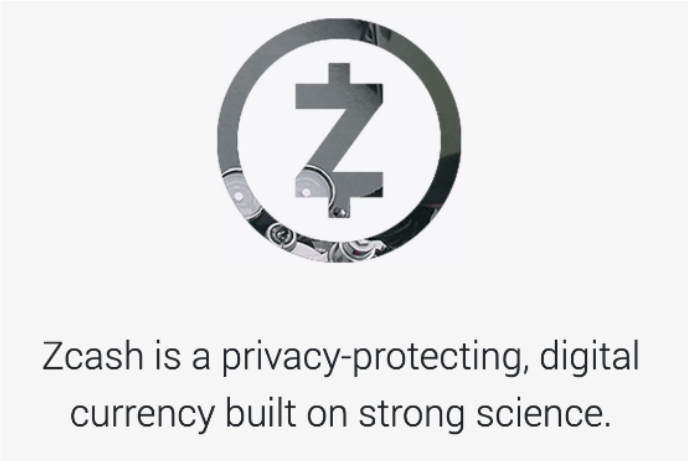
Disadvantages of Sigma Protocols

- Sigma protocols do not scale well, proof size, proof generation and verification size are linear in the complexity of the statement
- Cannot easily handle generic computation, they are better suited to algebraic constructions.

See the excellent blog post by Alex Pinto

ZKP Use Cases

Privacy preserving cryptocurrencies



Also Nightfall , ZKDai

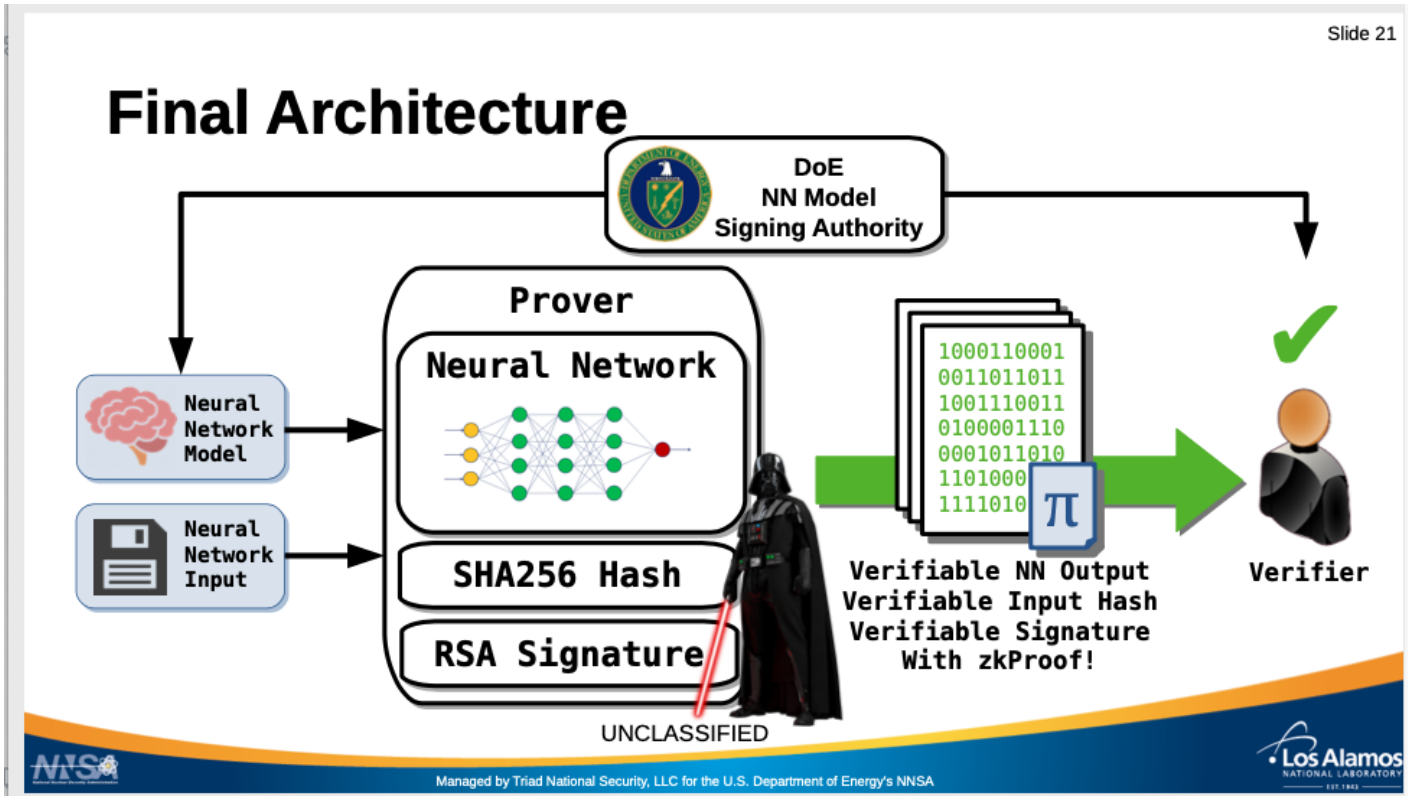
We will cover ZCash in more detail in a later lesson.

Nuclear Treaty Verification

LA-UR-20-20260
Approved for public release; distribution is unlimited.

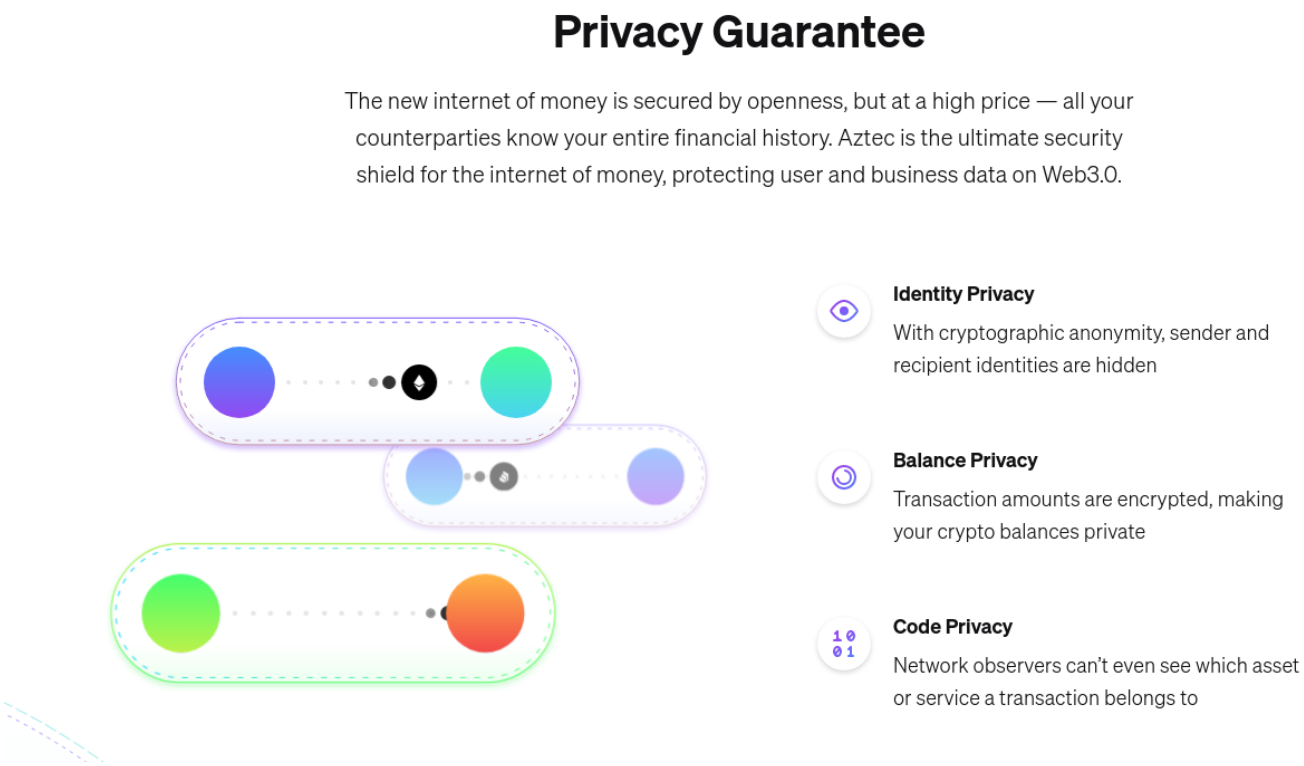
Title: SNNzkSNARK An Efficient Design and Implementation of a Secure Neural Network Verification System Using zkSNARKs

Author(s): DeStefano, Zachary Louis



Privacy Preserving Financial Systems

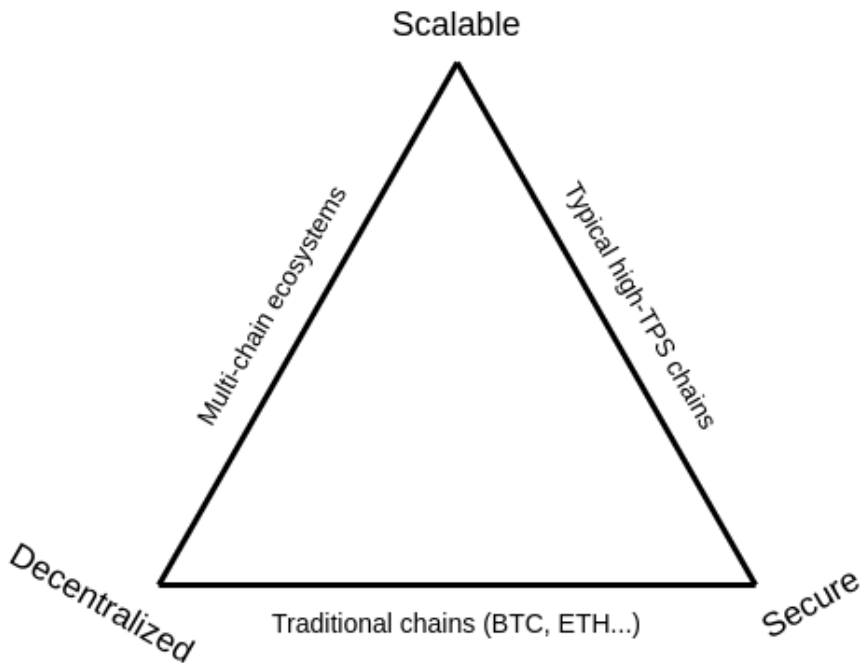
Aztec



Their work has prompted the Ethereum Standard for Confidential Tokens : https://github.com/ethereum/EIPs/issues/1724

Scalability Introduction

The scalability trilemma



"The decentralization of a system is determined by the ability of the weakest node in the network to verify the rules of the system." - Georgios Konstantopoulos

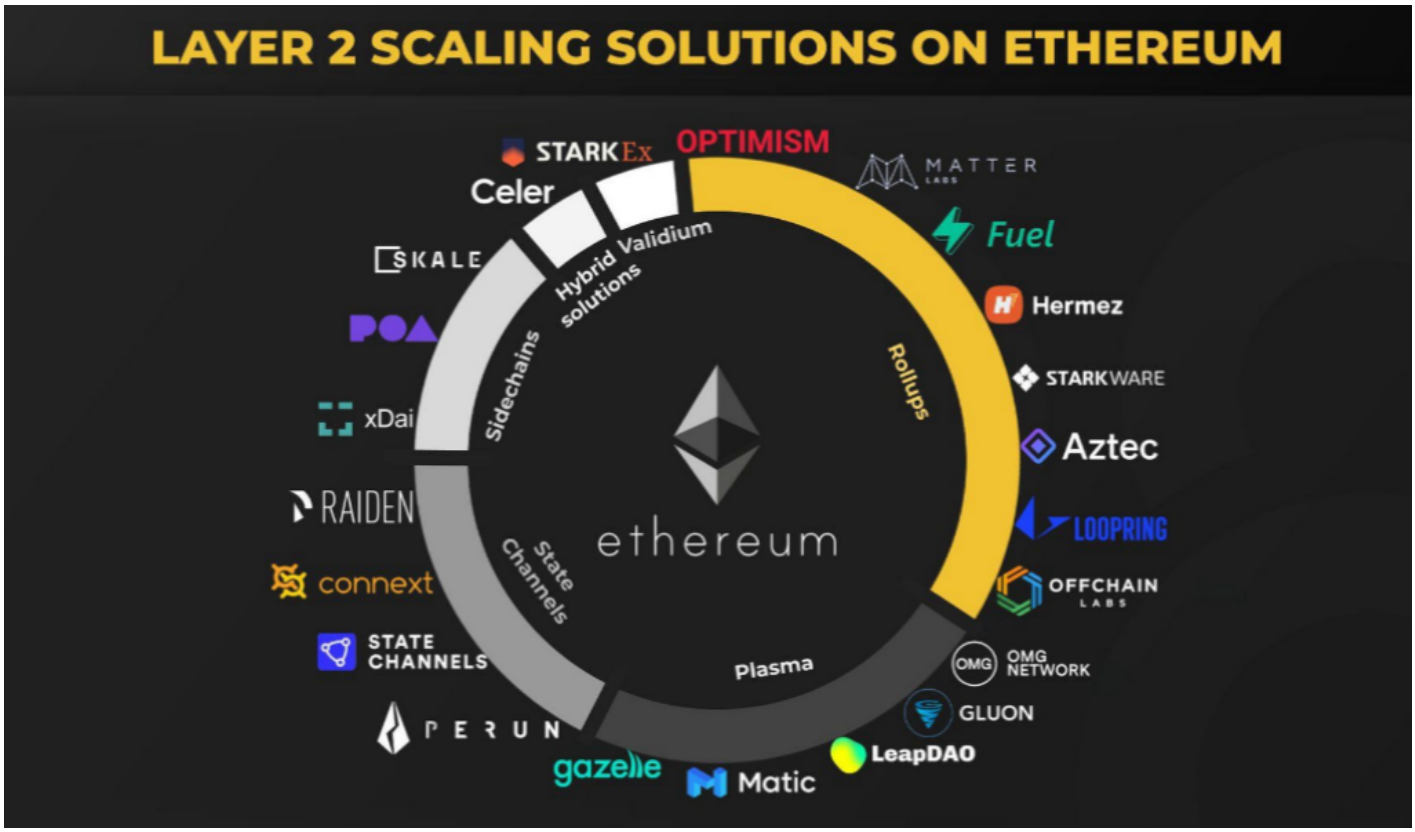
"**For a blockchain to be decentralized, it's crucially important for regular users to be able to run a node, and to have a culture where running nodes is a common activity**." - Vitalik article

On Ethereum there is a goal to keep the hardware requirements low.
There is a useful guide to scalability in their documentation

Off chain Scaling

Generally speaking, transactions are submitted to these layer 2 nodes instead of being submitted directly to layer 1 (Mainnet). For some solutions the layer 2 instance then batches them into groups before anchoring them to layer 1, after which they are secured by layer 1 and cannot be altered.
A specific layer 2 instance may be open and shared by many applications, or may be deployed by one project and dedicated to supporting only their application.

Early Layer 2 Solutions on Ethereum



Rollups

Rollups are solutions that have

- transaction execution outside layer 1
- data or proof of transactions is on layer 1
- a rollup smart contract in layer 1 that can enforce correct transaction execution on layer 2 by using the transaction data on layer 1

The main chain holds funds and commitments to the side chains
The side chain holds state and performs execution
There needs to be some proof, either a fraud proof (optimistic) or a validity proof (zk)

Rollups require "operators" to stake a bond in the rollup contract. This incentivises operators to verify and execute transactions correctly.

There are currently 2 types of rollups

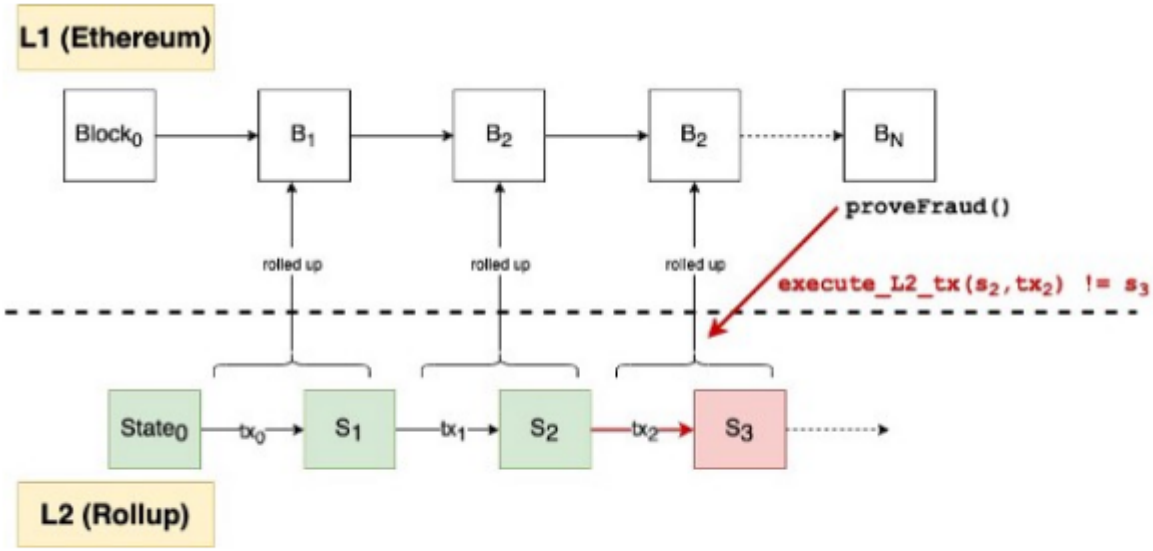
- Zero Knowledge Proof rollups
- Optimistic rollups

ZKP or validity Rollups

These rollups rely on a proof of the correctness of the execution that produces the rollup block state transition being supplied to a validator contract on L1.
The state transition on L2 will not be regarded as valid unless this proof has been validated.
Although we use a zero knowledge proof, the zero knowledge aspect is usually ignored, the inputs and data involved is usually public, the focus is on the correctness of computation. For this reason some people prefer the term validity proof.

Optimistic Rollups

The name Optimistic Rollups originates from how the solution works. 'Optimistic' is used because aggregators publish only the bare minimum information needed with no proofs, assuming the aggregators run without commiting frauds, and only providing proofs in case of fraud. 'Rollups' is used because transactions are commited to main chain in bundles (that is, they are rolled-up).

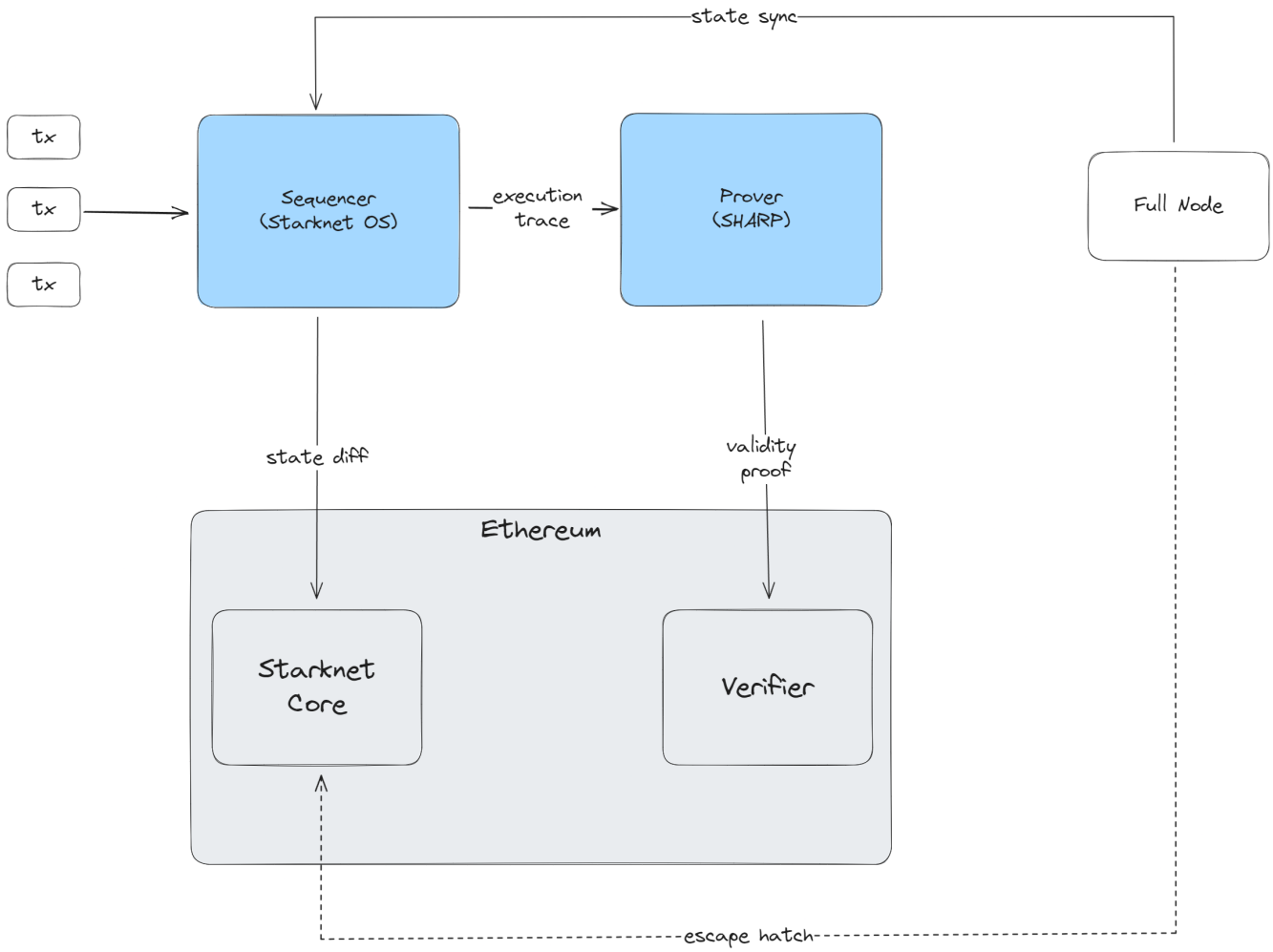


Optimistic execution scales because L2 transactions can be replayed on L1 — but only when necessary!

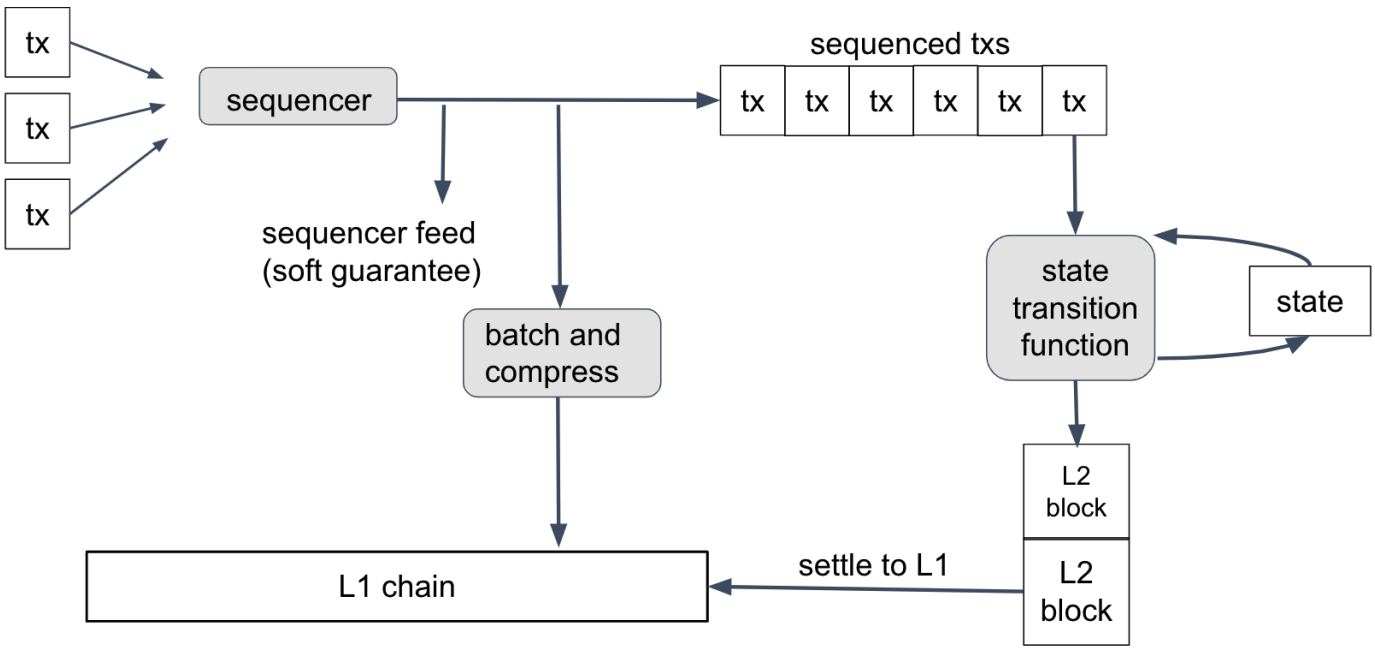
See this article for further discussion of the differences between these types of rollups.

Typical Rollup Architecture

This is an overview of the Starknet architecture



Here is Arbitrum's design



Zero Knowledge Proof Rollups

See Ethworks Report

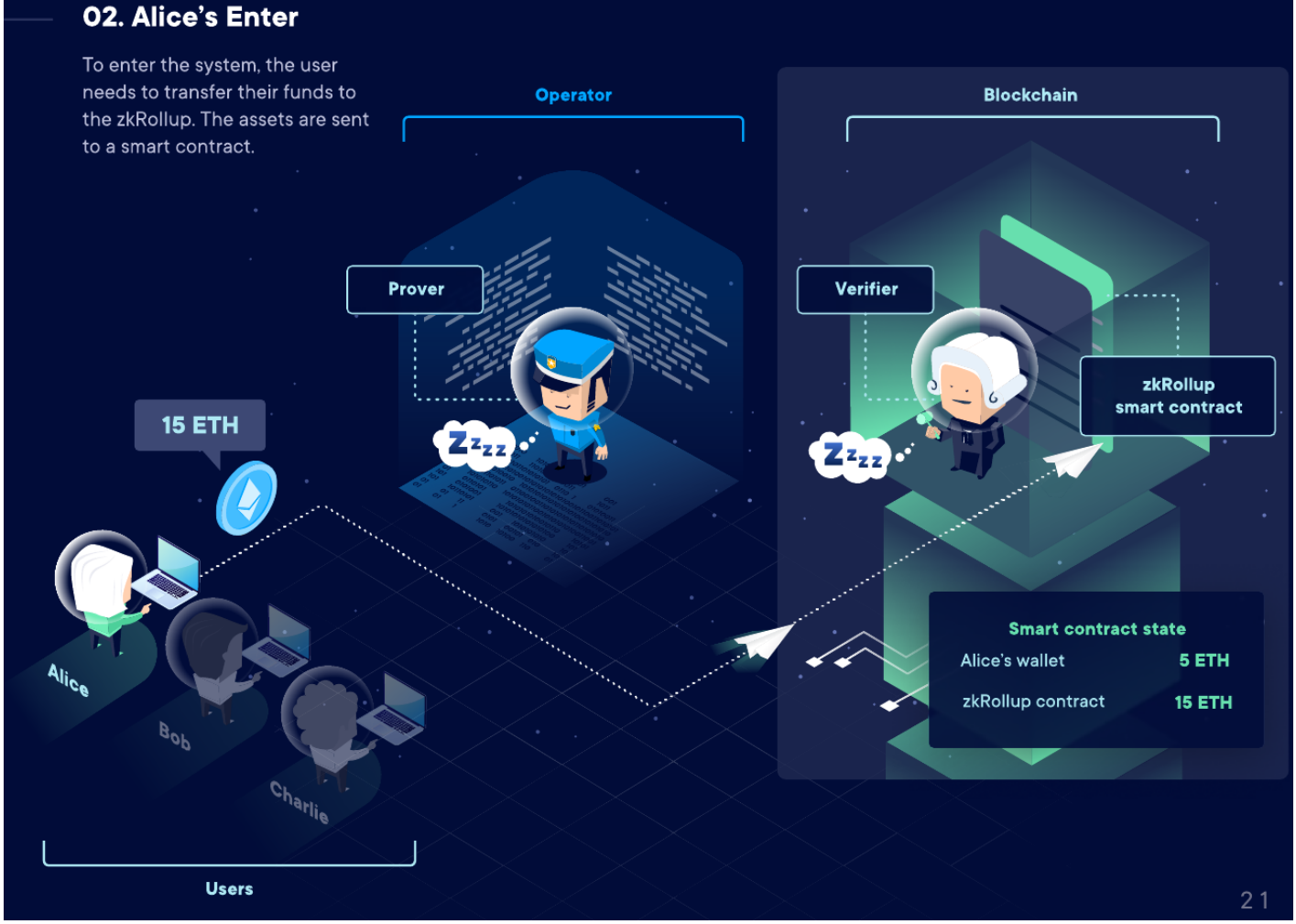
Note that in this context the proofs produced are often referred to as validity proofs since the zero knowledge aspect is not required.

ZK Rollup Process

From Ethworks

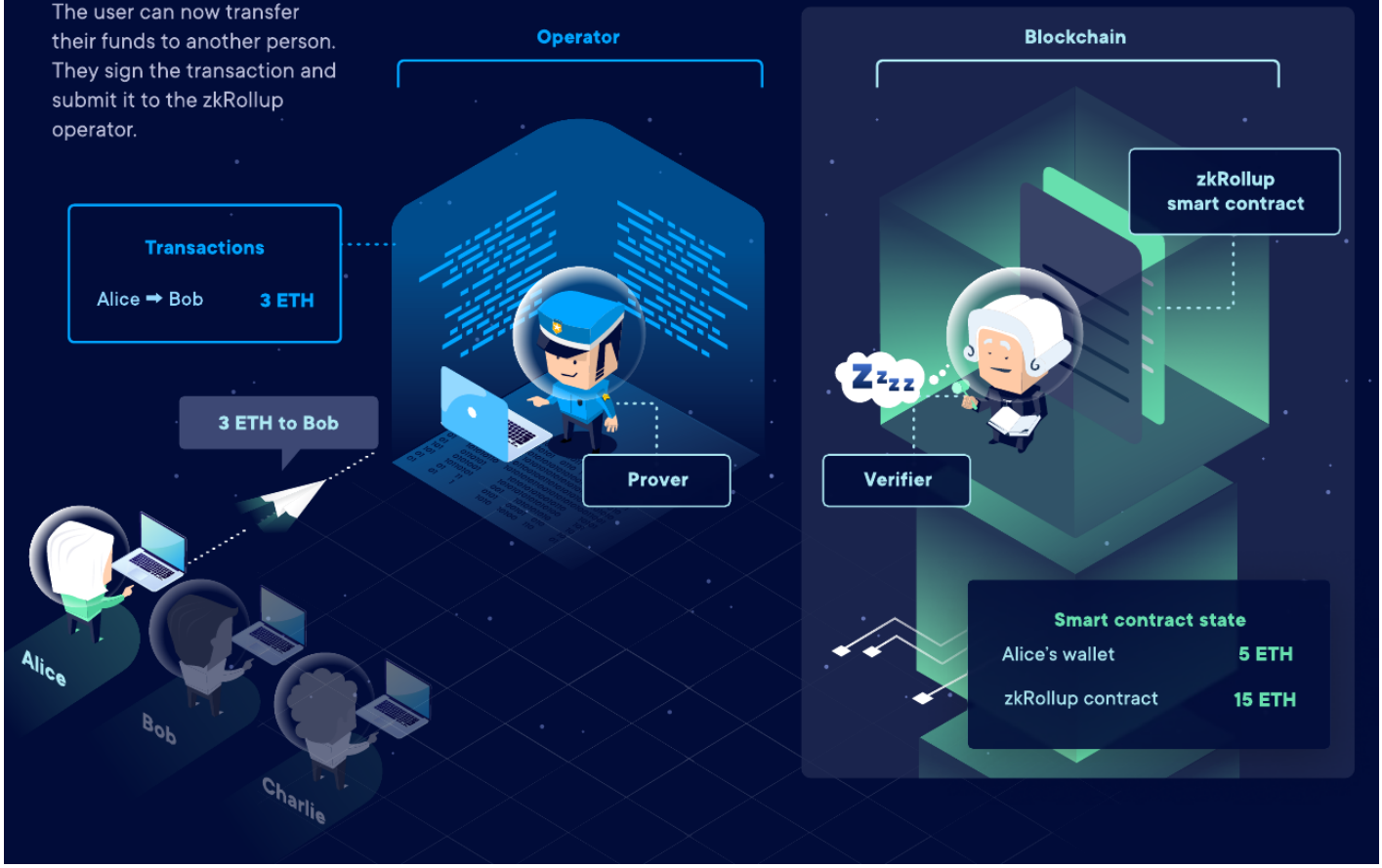
02. Alice's Enter

To enter the system, the user needs to transfer their funds to the zkRollup. The assets are sent to a smart contract.

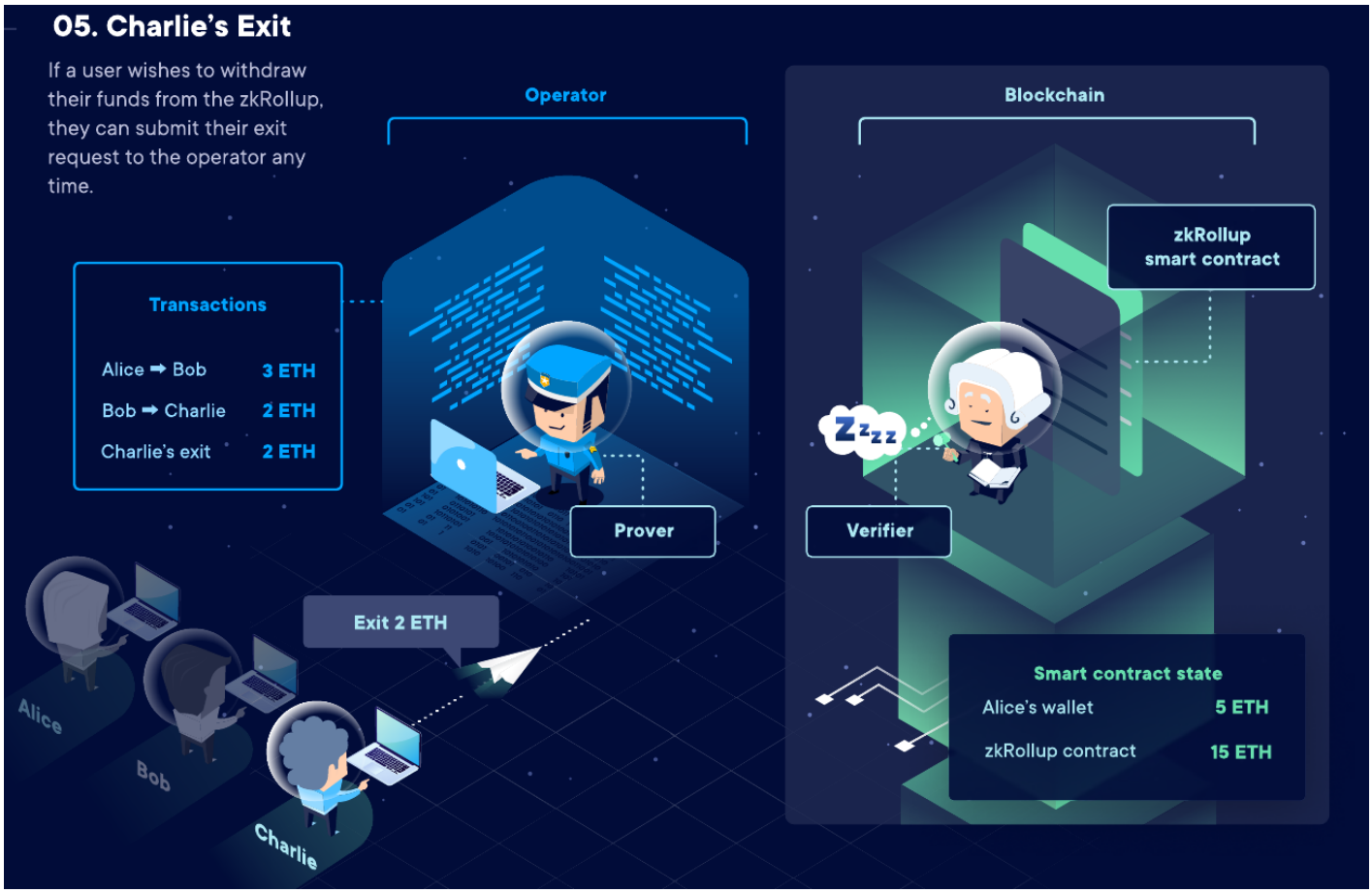


03. Alice's Transfer

The user can now transfer their funds to another person. They sign the transaction and submit it to the zkRollup operator.




04. Bob's Transfer
Operator
Blockchain
zkRollup
smart contract
Transactions
Alice ➡ Bob 3 ETH
Bob ➡ Charlie 2 ETH
Prover
Verifier
2 ETH to Charlie
Alice
Bob
Charlie
Smart contract state
Alice's wallet 5 ETH
zkRollup contract 15 ETH


05. Charlie's Exit
If a user wishes to withdraw their funds from the zkRollup, they can submit their exit request to the operator any time.
Operator
Blockchain
zkRollup
smart contract
Transactions
Alice ➡ Bob 3 ETH
Bob ➡ Charlie 2 ETH
Charlie's exit 2 ETH
Prover
Verifier
Exit 2 ETH
Alice
Bob
Charlie
Smart contract state
Alice's wallet 5 ETH
zkRollup contract 15 ETH

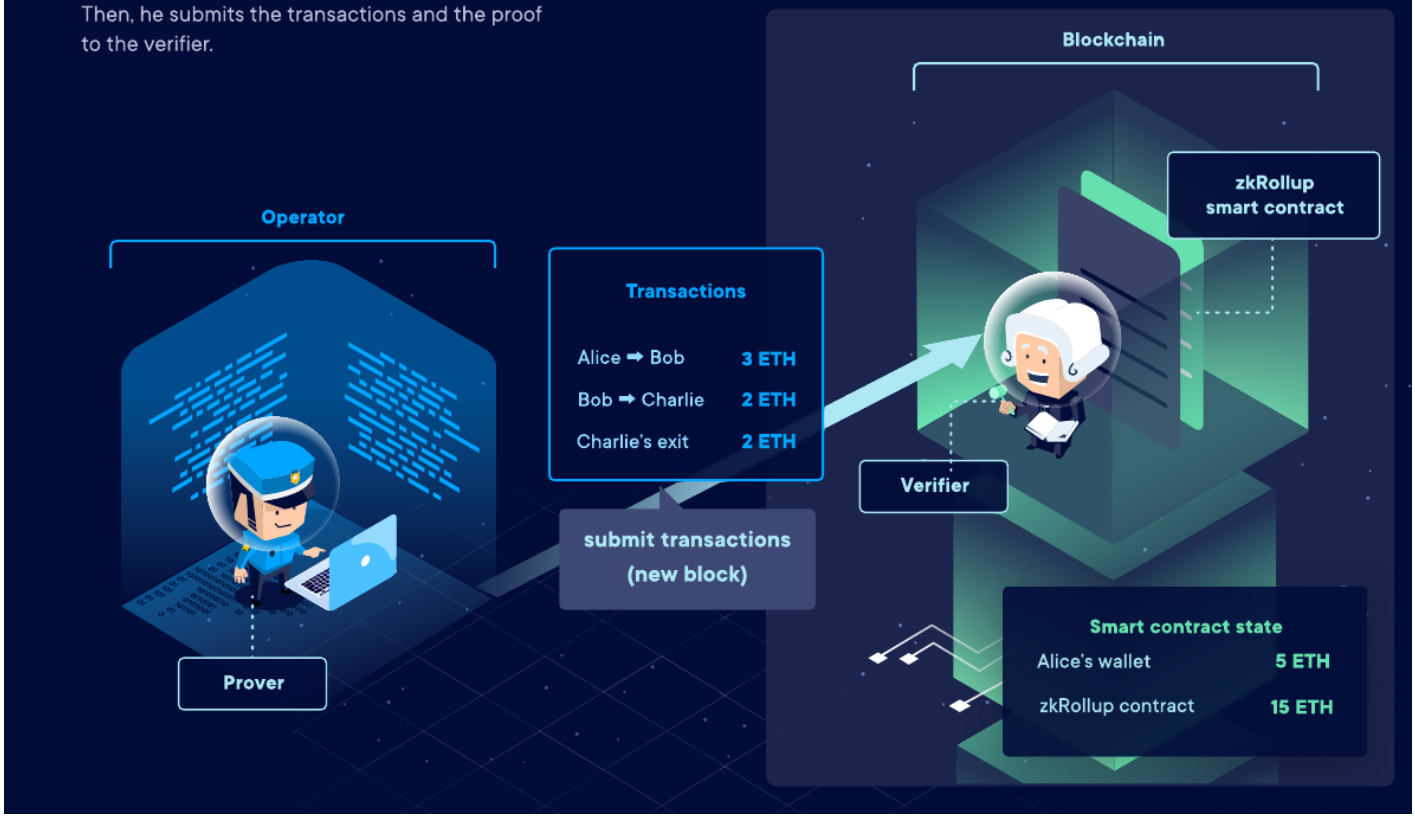
06. Collecting Transactions

In the meantime, the operator collects transactions and exit requests from many users.

* Note that even if Bob and Charlie didn't have any funds on the zkRollup, they could still receive transfers from other users.

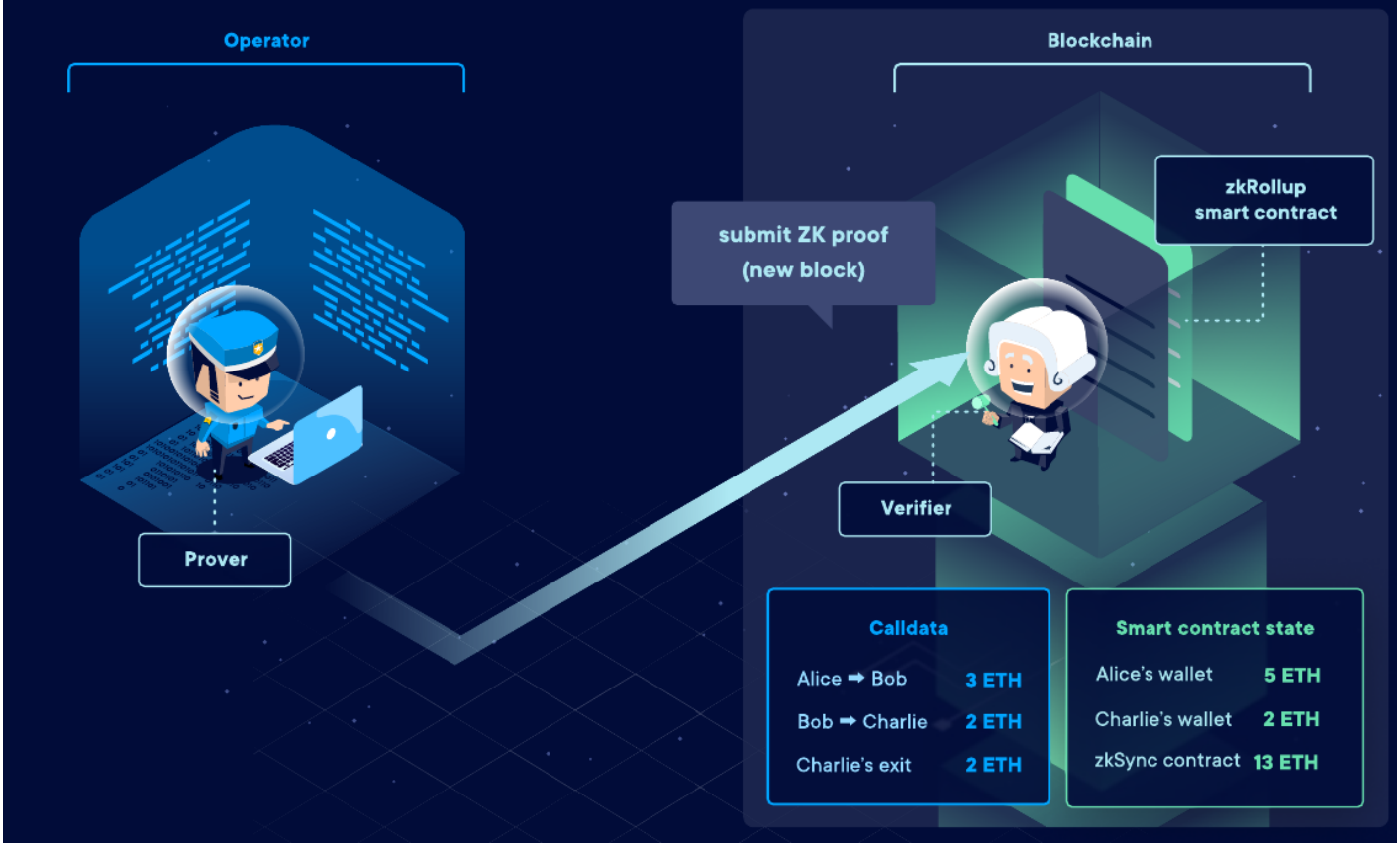
07. Submitting Transactions

Once in a while, the operator bundles the collected transactions together and generates a ZK proof. Then, he submits the transactions and the proof to the verifier.



08. Submitting ZK Proof

The smart contract verifies the transactions and the proof. Once it's done, the transactions are finalized.



Transaction Compression

How does compression work?

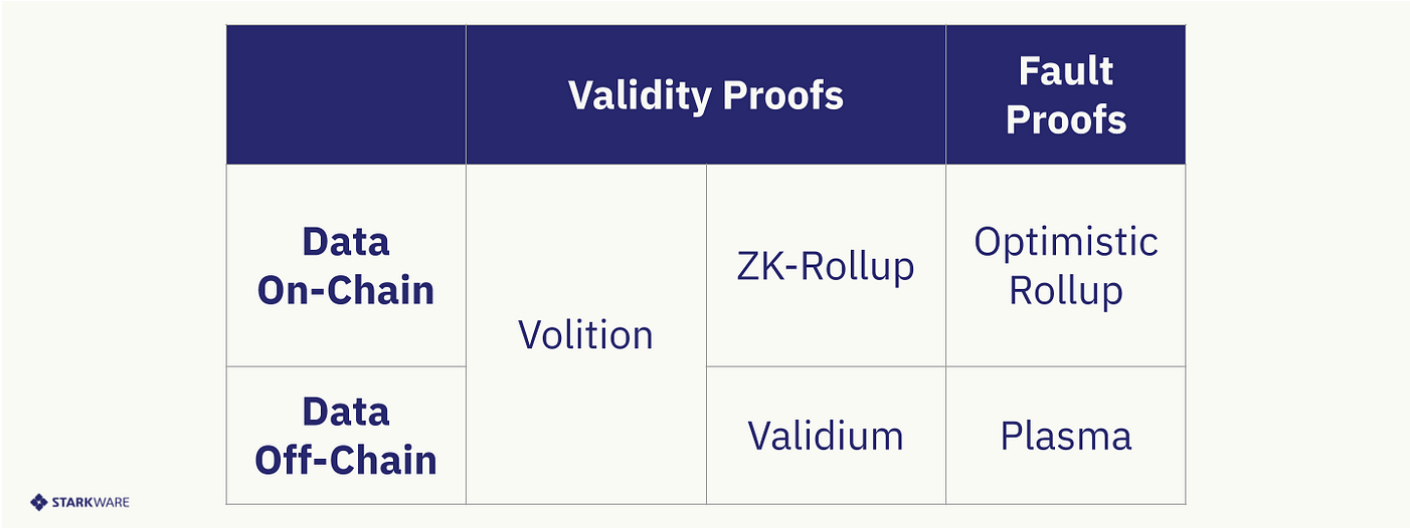
A simple Ethereum transaction (to send ETH) takes ~110 bytes. An ETH transfer on a rollup, however, takes only ~12 bytes:

Parameter	Ethereum	Rollup
Nonce	~3	0
Gasprice	~8	0-0.5
Gas	3	0-0.5
To	21	4
Value	~9	~3
Signature	~68 (2 + 33 + 33)	~0.5
From	0 (recovered from sig)	4
Total	~112	~12

Part of this is simply superior encoding: Ethereum's RLP wastes 1 byte per value on the length of each value. But there are also some very clever compression tricks that are going on:

Data Availability

In order to re create the state, transaction data is needed, the data availability question is where this data is stored and how to make sure it is available to the participants in the system.



See Docs

StarkNet is currently in ZK-Rollup mode (see above). This means that upon the acceptance of a state update on-chain, the state diff between the previous and new state is sent as calldata to Ethereum.

This data allows anyone that observes Ethereum to reconstruct the current state of StarkNet. Note that to update the StarkNet state on L1, it suffices to send a valid proof — without information on the transactions or particular changes that this update caused. Consequently, more information must be provided in order to allow other parties to locally track StarkNet's state.

Other projects

Filecoin

Proof of replication

In a Proof of Replication, a storage miner proves that they are storing a physically unique copy, or *replica*, of the data. Proof of Replication happens just once, at the time the data is first stored by the miner.

Whereas Proof of Replication is run once to prove that a miner stored a physically unique copy of the data at the time the sector was sealed, Proof of Spacetime (PoSt) is run repeatedly to prove that they are continuing to dedicate storage space to that same data over time.

Both the Proof of Replication and Proof of Spacetime processes in Filecoin use zk-SNARKs for compression.

The process of creating Filecoin's zk-SNARKs is computationally expensive (slow), but the resulting end product is small and the verification process is very fast. Compared to the original proofs, zk-SNARKs are tiny, making them efficient to store in a blockchain. For example, a proof that would have taken up hundreds of kilobytes on the Filecoin chain can be compressed to just 192 bytes using a zk-SNARK.

See zkSNARKS for the world

For storage to be verified on Filecoin, two proofs are involved: *Proof of Replication (PoRep)* and *Proof of Spacetime (PoSt).* In PoRep, a storage provider proves that they are storing a unique copy of a piece of data or information. PoRep happens just once, when the initial storage deal between client and provider happens and the data is first stored by the miner. Each PoRep that goes on-chain includes 10 individual SNARKs, which together prove that the process was done correctly through probabilistic challenges.

PoSt, on the other hand, serves to prove that the storage provider *continues* to store the original data over time without manipulation or corruption. When a storage provider first agrees to store data for a client, they must put down collateral in the form of FIL. If at any point during the agreement, the provider fails to prove PoSt, they are penalized and can lose all or some of their posted FIL collateral.

The result of an on-chain interaction in which the *prover* and *verifier* agree that data has been stored and maintained in an appropriate manner is a *proof.* As mentioned above, without a solution to make these proofs small and efficient, they would take up a tremendous amount of the network’s bandwidth and deliver high operational costs to both storage providers and miners. By using zk-SNARKs to generate the proofs, however, the resulting proofs are small and the verification process is extremely fast (and thus, cheap). For example, proofs that typically would require hundreds of kilobytes to verify can instead be compressed to just 192 bytes with zk-SNARKs. As mentioned above, each PoRep includes 10 SNARKs, meaning 1920 bytes in each (10 * 192 bytes).

Filecoin is the largest deployed zk-SNARK network to date

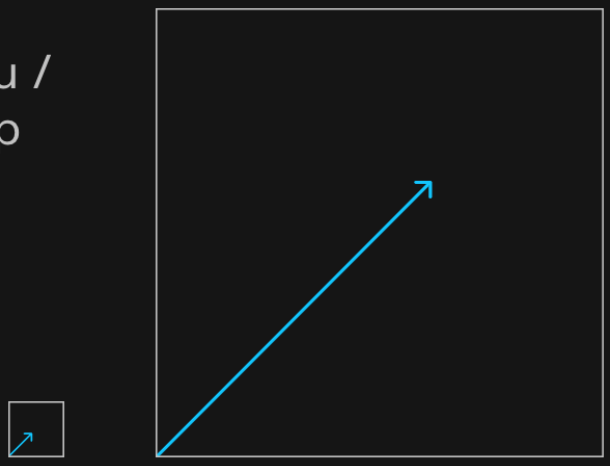
As far as we know, Filecoin represents the largest zk-SNARK deployment to date — in several dimensions:

- Our trusted setup enables circuits of up to 2^27 = ~134M constraints.
- Our large individual circuits have > 100M constraints.
- To satisfy our security requirements, some proofs bundle as many as 10 individual zk-SNARKs into a single large proof.
- We have also extended and deployed research on zk-SNARK aggregation to allow compression of thousands of individual proofs into a single proof.

All of the above contribute to Filecoin's ability to prove more information than the rest of the world has ever proved in production.

The baseline

Powers of Tau / Trusted Setup



Maximum Constraints

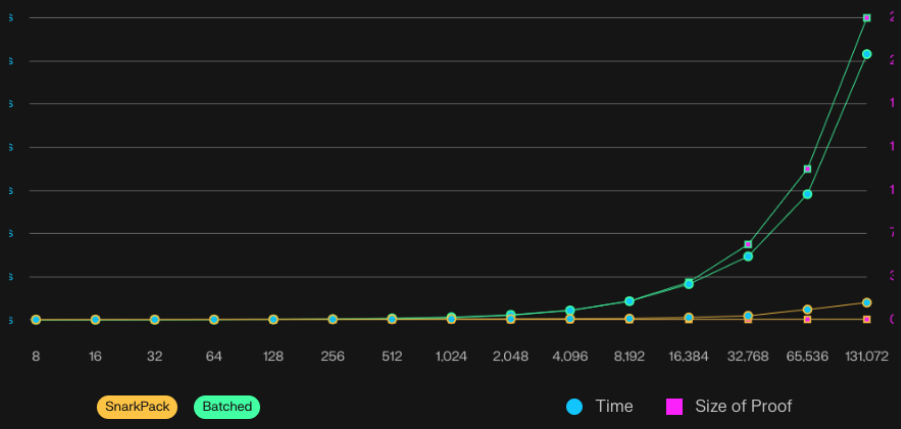
Zcash 2,097,152 Filecoin 134,217,728

To support the amount of constraints needed for Filecoin, we ran a new Powers of Tau Ceremony, increasing the supported number by a factor of 64, over the Ceremony Zcash had run. This allows us to generate proofs of over 100 million constraints, limited only by the size of the parameters which must be distributed.

To support the Phase 2 (circuit-specific) trusted setup for our large circuits, we implemented techniques to dramatically decrease RAM usage, allow for parallelism, and reduce I/O overhead — in order to allow many parties using practical hardware to participate during the 7 weeks the ceremony took place.

Pack 'em tighter

SnarkPack



Even though Batch Verification helped, we needed faster verification, so we implemented SnarkPack. This allows us to aggregate many zk-SNARKs into a single combined proof. Not only does this optimization reduce verification time by a factor of more than 10x at scale — it also reduces chain bandwidth by reducing the average bytes-per-proof which must be submitted to the chain.

In order to accomplish this, we built on the research on the Inner Product Argument — and collaborated with the authors to extend it to support our needs without requiring a new trusted setup. We accomplished this by adapting the techniques to securely apply using two existing Powers of Tau trusted setups. This is a great example of how we have historically had to pick our way through obstacles to the practical realization of groundbreaking scale.

Tornado Cash

To process a deposit, Tornado.Cash generates a random area of bytes, computes it through the Pederson Hash (as it is friendlier with zk-SNARK), then send the token and the hash to the smart contract. The contract will then insert it into the Merkle tree.

To process a withdrawal, the same area of bytes is split into two separate parts: the **secret** on one side and the **nullifier** on the other side.
The nullifier is hashed.
This nullifier is a public input that is sent on-chain to get checked with the smart contract & the Merkle tree data. It avoids double spending for instance.

Thanks to a zk-SNARK, it is possible to prove the hash of the initial commitment and of the nullifier without revealing any information.
Even if the nullifier is public, privacy is sustained as there is no way to link the hashed nullifier to the initial commitment.
Besides, even if some has the information that the transaction is present in the Merkle root, the information about the exact Merkle path, thus the location of the transaction, is still kept private.

Deposits are simple on a technological point of view, but expensive in terms of gas as they need to compute the hash and update the Merkle tree. At the opposite end, the withdrawal process is complex, but cheaper as gas is only needed for the nullifier hash and the zero knowledge proof.

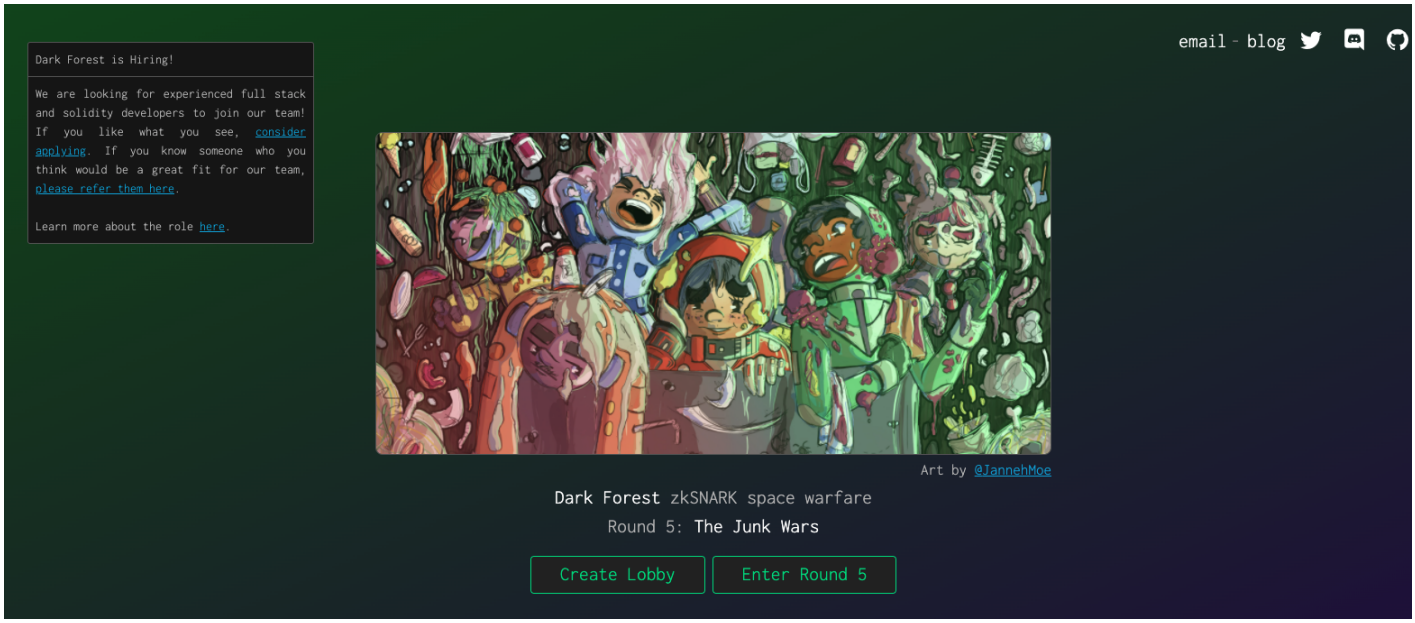
TORNADO CASH Verifier Contract

Contract

Zero Knowledge Lottery based on Tornado Cash

See article

Dark Forest



How does Dark Forest use SNARKs?

A central mechanic in Dark Forest is that the cryptographic “fog of war.” The fog of war ensures that you don’t automatically know where all players, planets, and other points of interests are in the universe; you have to spend computational resources to discover them. This mechanic is secured by zkSNARKs.

In a universe with a fog of war, the locations of all players are private and hidden from each other. This means that players don’t upload the coordinates of their planets to the Ethereum blockchain, which can be publicly inspected. Instead, each player uploads the hash of their location to the blockchain. This ensures that players stay “committed” to a specific location, but also that the location can’t be determined from inspection of the Ethereum data layer.

Without zkSNARKs, there’s an obvious attack vector - if a player uploads a random string of bytes that doesn’t correspond to a real and valid location, and the integrity of the game is broken. To prevent this, Dark Forest requires players to submit zkSNARKs with every move to ensure that players are indeed submitting hashes corresponding to valid coordinates that they have knowledge of.

When players make moves, they’re also required to submit ZK proofs that their moves are “valid” - you can’t move too far or too fast. Without zkSNARKs, a malicious player could make illegal “teleport” moves by claiming that the hash they are moving from is next to the hash they’re moving to, even if the two locations are actually on opposite sides of the universe. Once again, requiring ZK proofs keeps players honest. To use a chess analogy, the required ZK proofs basically tell the contract, “I’m moving my knight; I’m not going to tell you where I moved my knight from, or where I moved it to, but this proof proves that it did in fact move in a legal L-shape.”

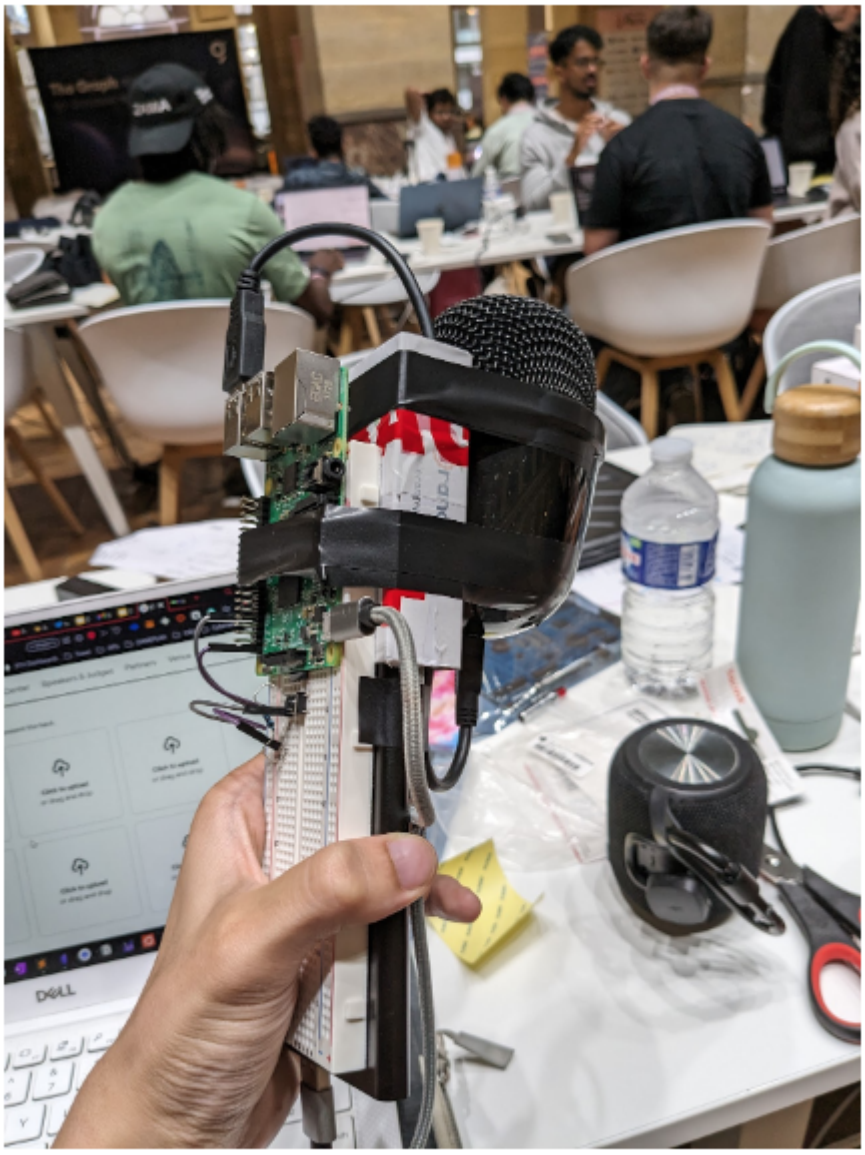
Authenticating audio

See recent project from ETH Global

ZK Microphone

🎙️🔒 ZK Microphone: Trusted audio in the age of deepfakes 🔒🎙️ Generative AI is a threat to society. It enables disinformation, manipulation, and political subversion. We've built the world's first attested microphone and used ZK-SNARKs to protect authenticity and privacy.

Source Code



Also
https://www.youtube.com/watch?v=PyipRqeQykA&list=PLj80z0cJm8QFnY6VLVa84nr-21DNvjWH7&index=21
Hardware attested audio

Other existing / suggested applications

- Verifying sufficient credit score - using range proofs

- Replacing username / passwords M-Pin
- Supply Chain Transparency Origin Trail
- Software Verification Paper
- Digital Forensics Paper
- Identity Sovrin
- Verifying Qualification TiiQu
- KYC ING
- Tax Qedit
- Legal evidence integrity Stratuum

we will cover many other examples