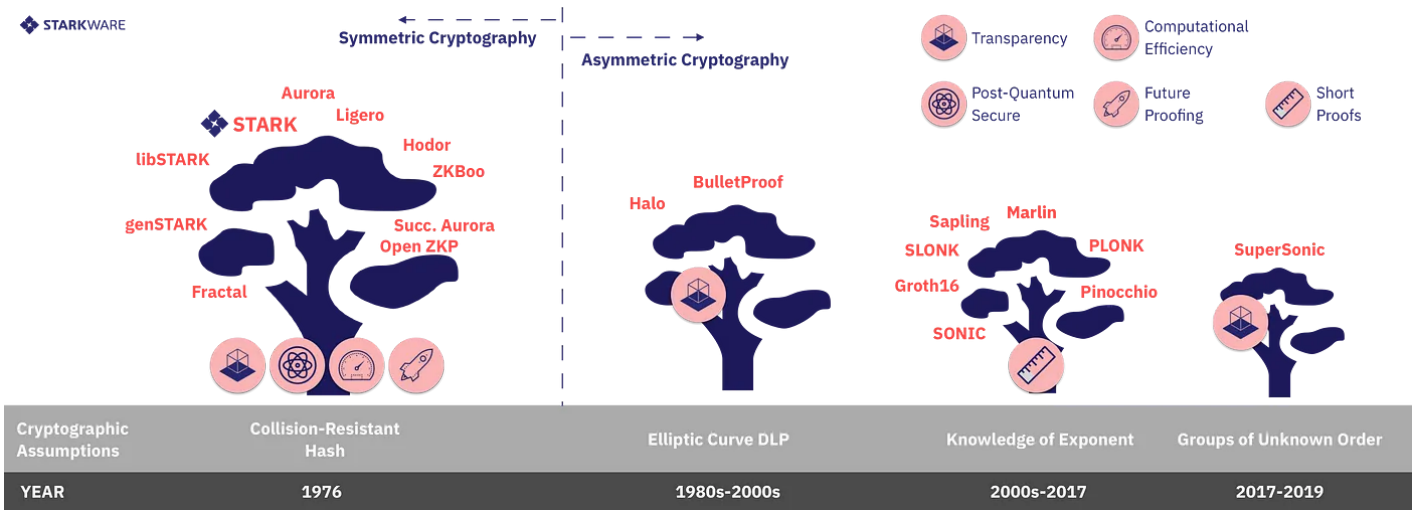
Lesson 3

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

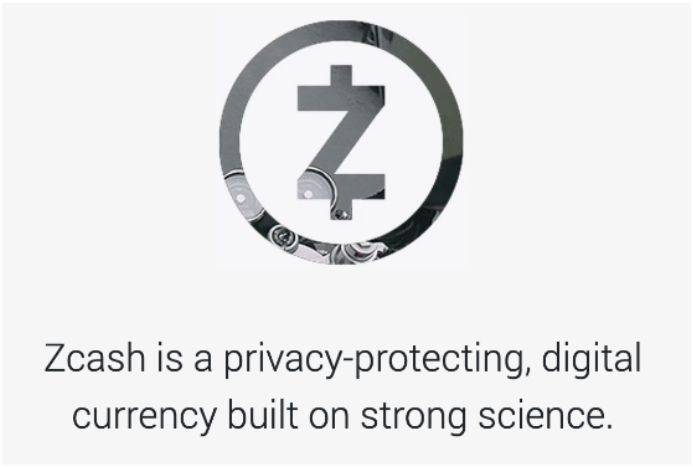
Disadvantages of Sigma Protocols

- 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

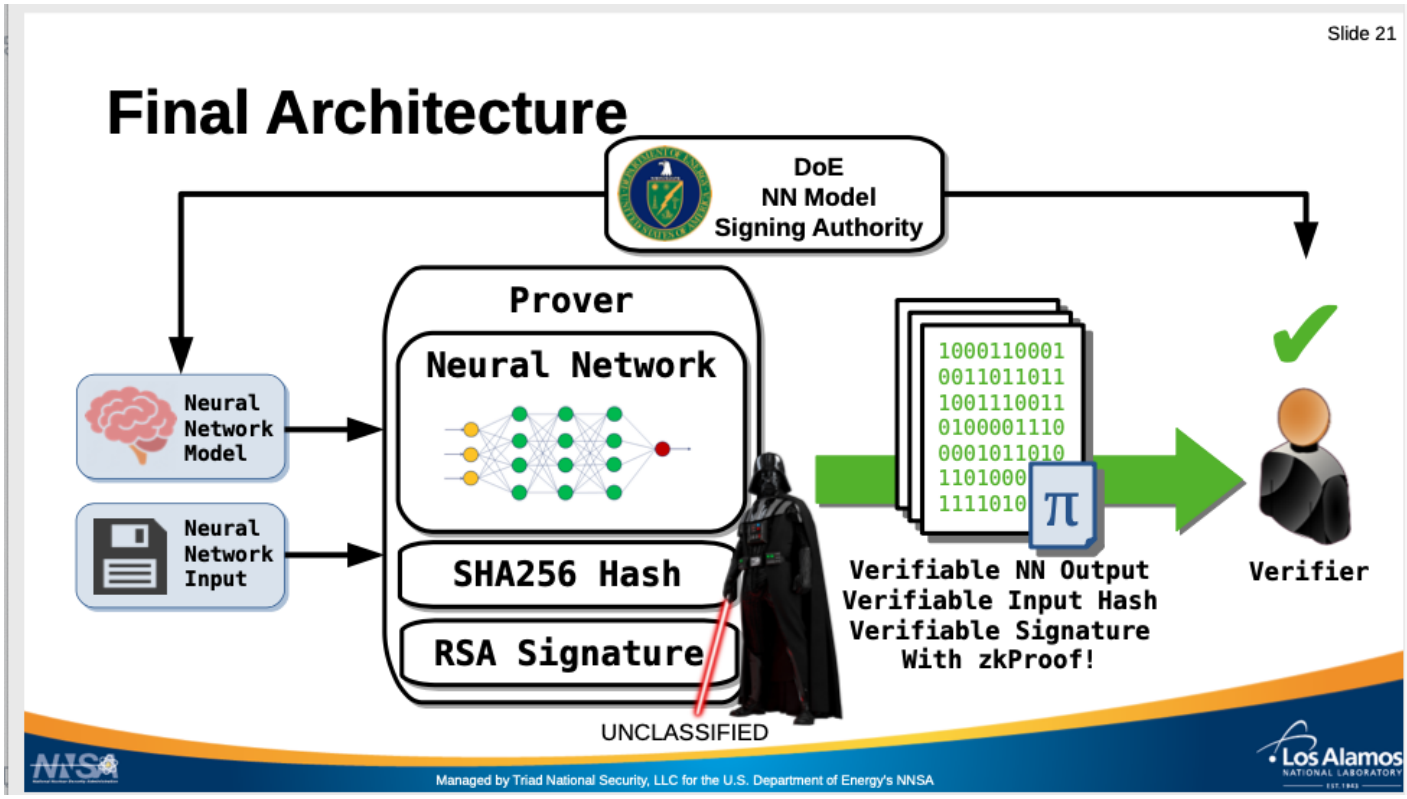
We will cover ZCash in more detail in a later lesson.

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

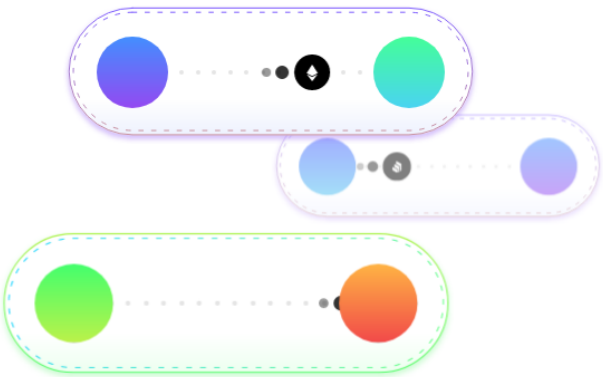


Privacy Preserving Financial Systems

Aztec

Privacy Guarantee

The new internet of money is secured by openness, but at a high price — all your counterparties know your entire financial history. Aztec is the ultimate security shield for the internet of money, protecting user and business data on Web3.0.



Identity Privacy

With cryptographic anonymity, sender and recipient identities are hidden

Balance Privacy

Transaction amounts are encrypted, making your crypto balances private

Code Privacy

Network observers can't even see which asset or service a transaction belongs to

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

Scalability Introduction

The scalability trilemma

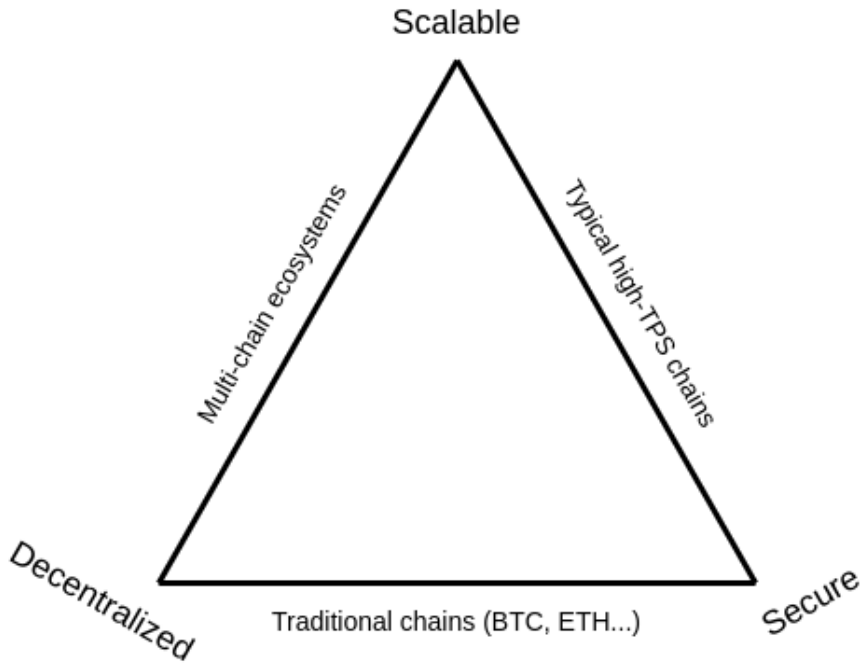


FIGURE 2. Taxonomy and comparison of blockchain scalability solutions.

From Scaling Blockchains: A Comprehensive Survey by Hafid et al.

"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

In Ethereum there is a goal to keep the hardware requirements low.

Solutions

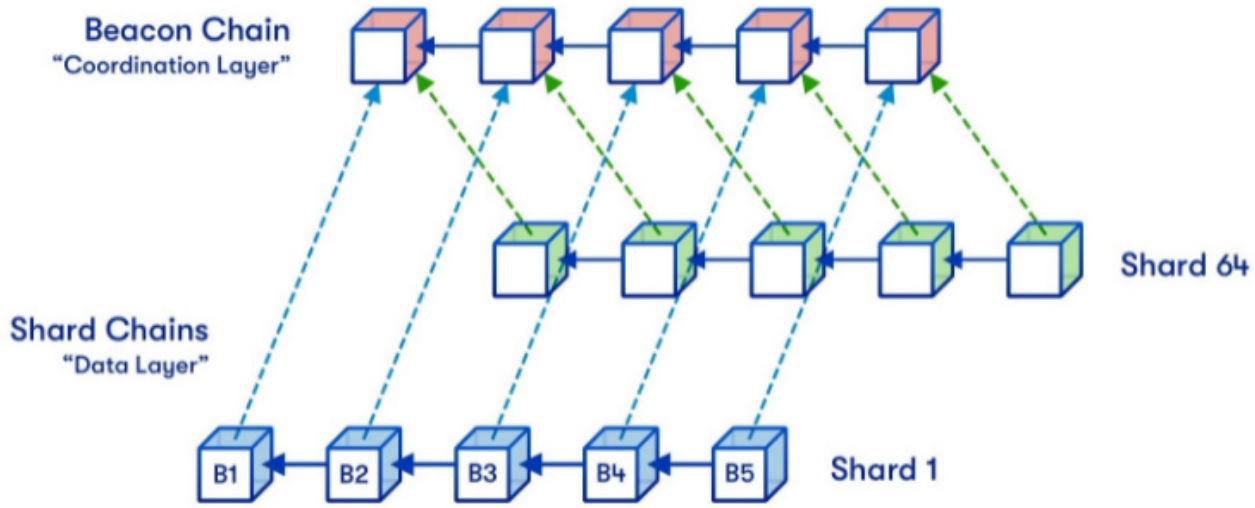
On chain Scaling (Layer 1)

Changing the Consensus Mechanism

Using DPoS - EOS
For example moving from Proof of Work to Proof of Stake - Ethereum

Sharding

Ethereum plans to introduce 64 new shard chains, to spread the network load.
Vitalik's overview
Introduction



This will follow the merge of Mainnet with the Beacon Chain, probably in 2022.

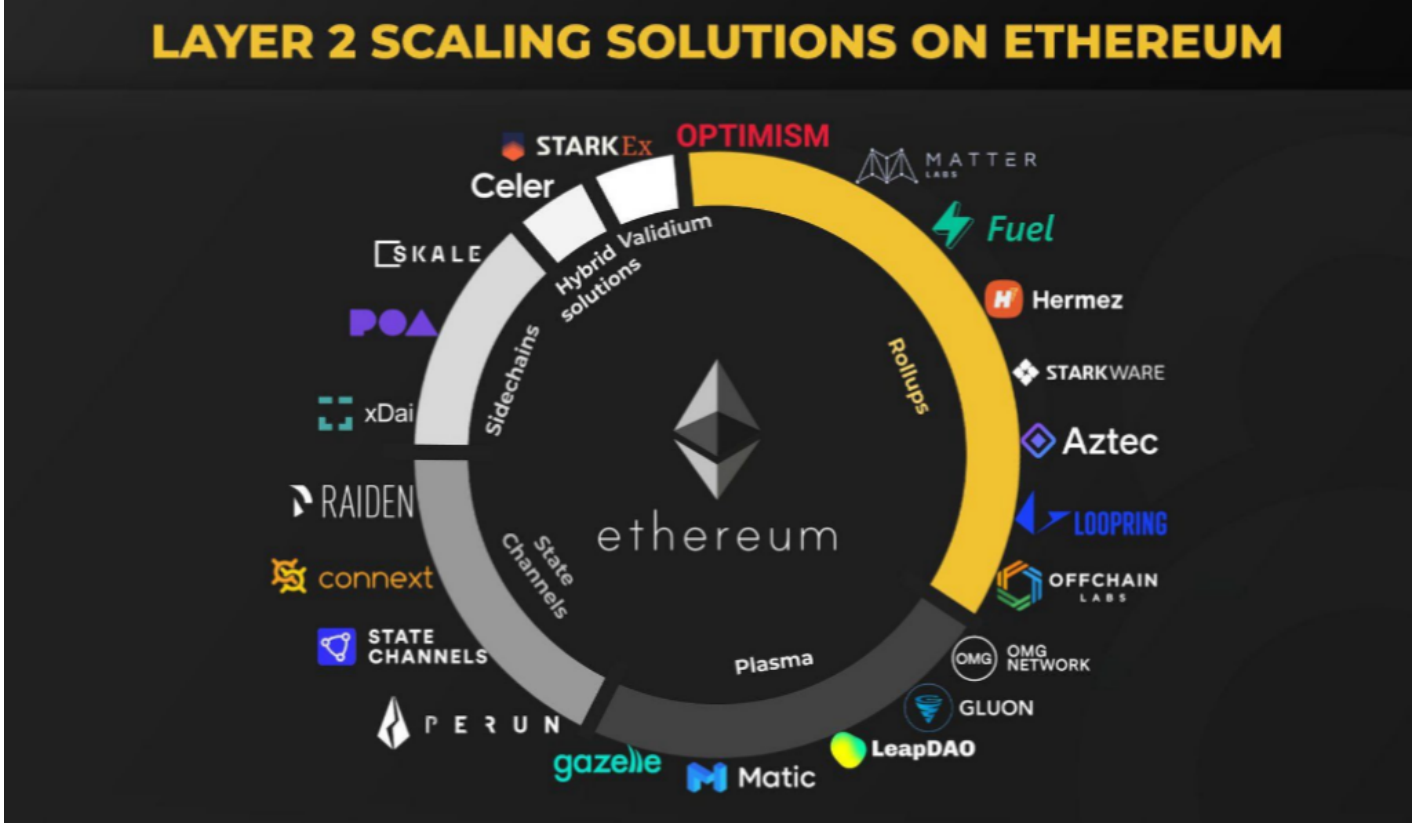
Introduction of Sharding

Vitalik sees 3 options

- Shards remain as data depots
- A subset of the 64 shards will allow smart contracts
- Wait until increased use of ZKPs allows private transactions

Off chain Scaling (Layer 2)

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.



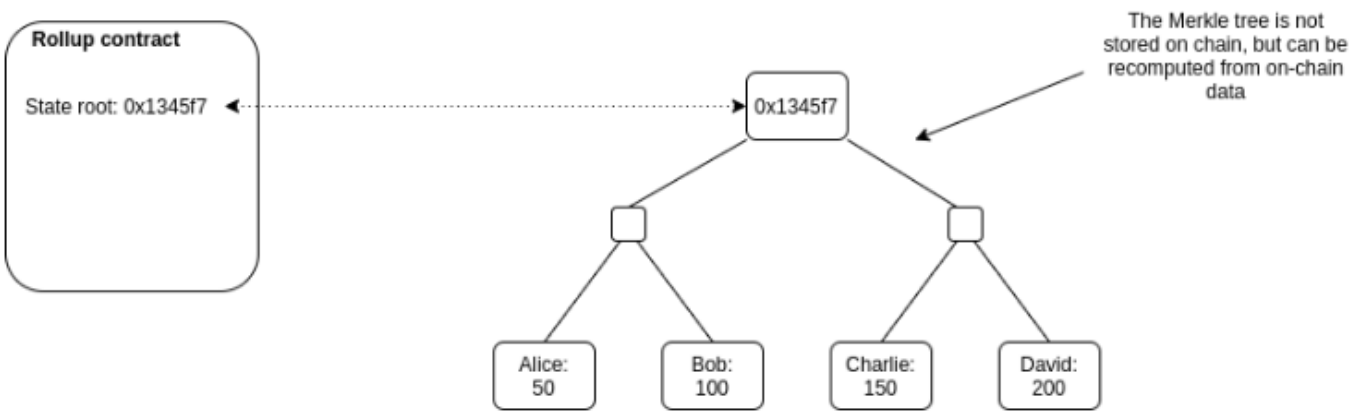
Rollups

Rollups are solutions that have

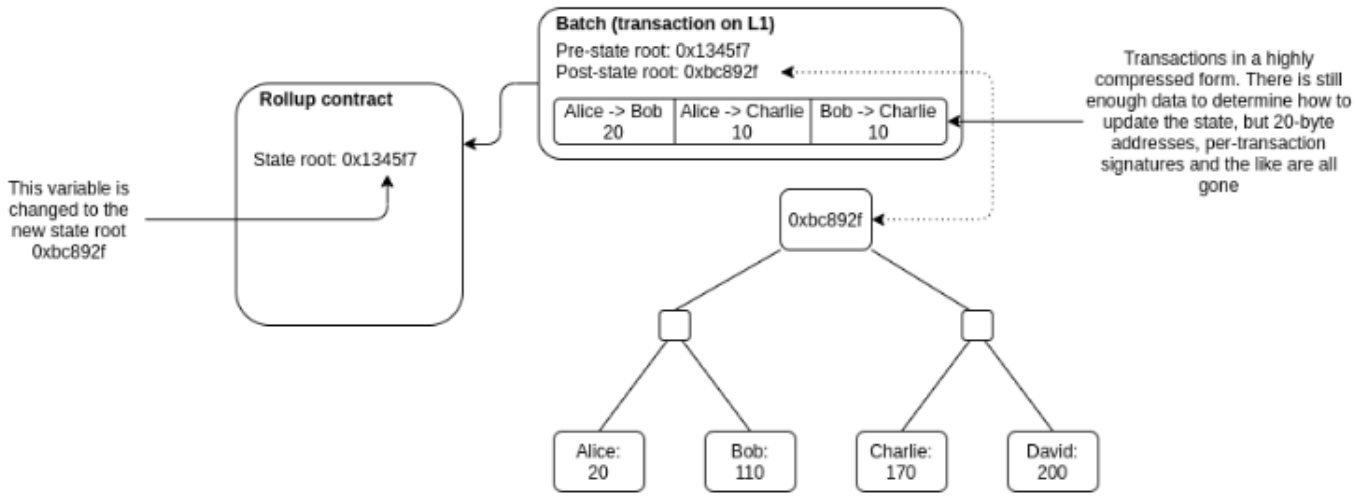
- transaction execution outside layer 1
- data or proof of transactions is on layer 1
- a rollup smart contract in layer 1 that can enforce correct transaction execution on layer 2 by using the transaction data on layer 1

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.



Anyone can publish a batch, a collection of transactions in a highly compressed form together with the previous state root and the new state root (the Merkle root after processing the transactions). The contract checks that the previous state root in the batch matches its current state root; if it does, it switches the state root to the new state root.

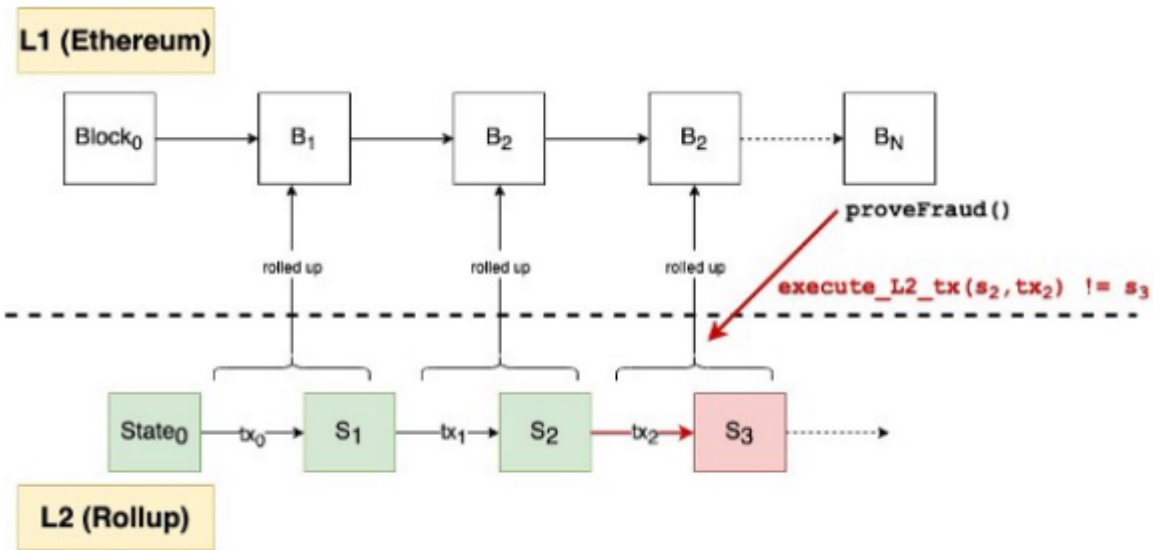


There are currently 2 types of rollups

- Zero Knowledge Proof rollups
- Optimistic rollups

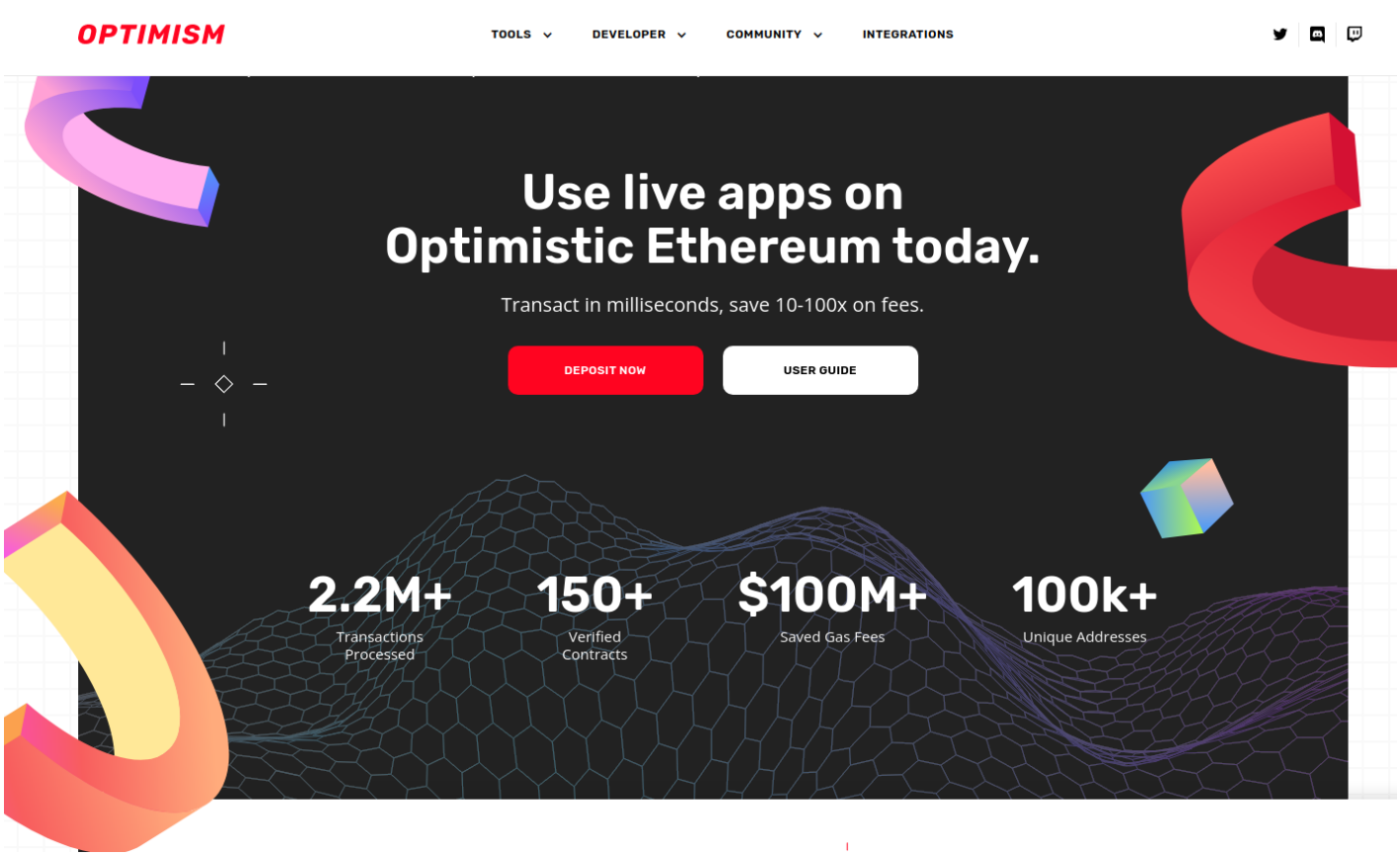
Optimistic Rollups

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!

Example Projects



Process

- Developer sends transaction off-chain to a bonded aggregator
- Anyone with a bond may become an aggregator.
- There are multiple aggregators on the same chain.
- Fees are paid however the aggregator wants (account abstraction / meta transactions).

- Developer gets an instant guarantee that the transaction will be included or else the aggregator loses their bond.
- Aggregator locally applies the transaction & computes the new state root.
- Aggregator submits an Ethereum transaction (paying gas) which contains the transaction & state root (an optimistic rollup block).
- If anyone downloads the block & finds that it is invalid, they may prove the invalidity with verify_state_transition(prev_state, block, witness) which:
    - Slashes the malicious aggregator & any aggregator who built on top of the invalid block.
    - Rewards the prover with a portion of the aggregator's bond.

Zero Knowledge Proof Rollups

See Ethworks Report

An overview from Ethereum
The ZK-Rollup scheme consists of two types of users: transactors and relayers.

- Transactors create their transfer and broadcast the transfer to the network. The transfer data consists of an indexed "to" and "from" address, a value to transact, the network fee, and nonce. A shortened 3 byte indexed version of the addresses reduces processing resource needs. The value of the transaction being greater than or less than zero creates a deposit or withdrawal respectively. The smart contract records the data in two Merkle Trees; addresses in one Merkle Tree and transfer amounts in another.
- Relayers collect a large amount of transfers to create a rollup. It is the relayers job to generate the SNARK proof. The SNARK proof is a hash that represents the delta of the blockchain state. State refers to "state of being." SNARK proof compares a snapshot of the blockchain before the transfers to a snapshot of the blockchain after the transfers (i.e. wallet values) and reports only the changes in a verifiable hash to the mainnet.

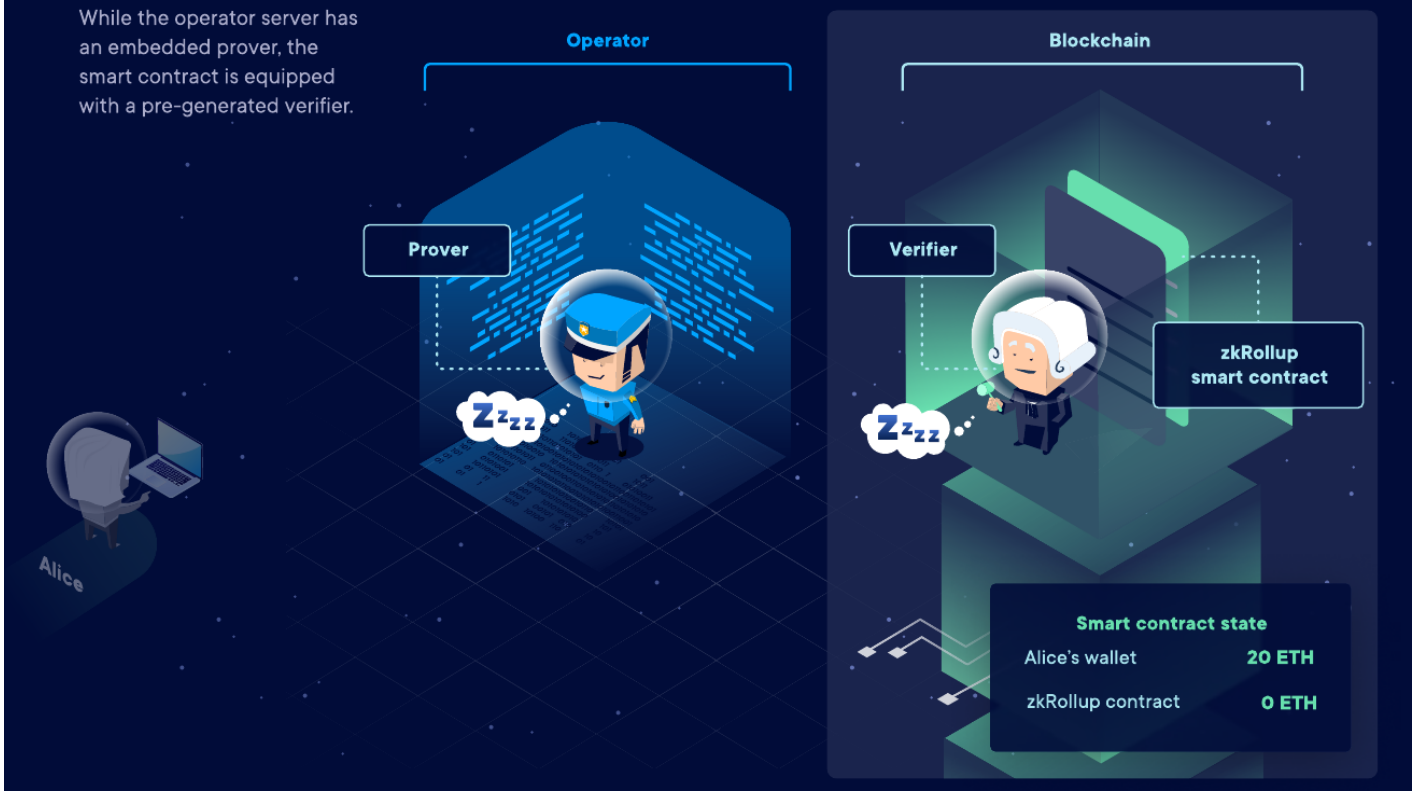
It is worth noting that anyone can become a relayer so long as they have staked the required bond in the smart contract. This incentivises the relayer not to tamper with or withhold a rollup.

ZK Rollup Process

From Ethworks

01. Start

While the operator server has an embedded prover, the smart contract is equipped with a pre-generated verifier.



02. Alice's Enter

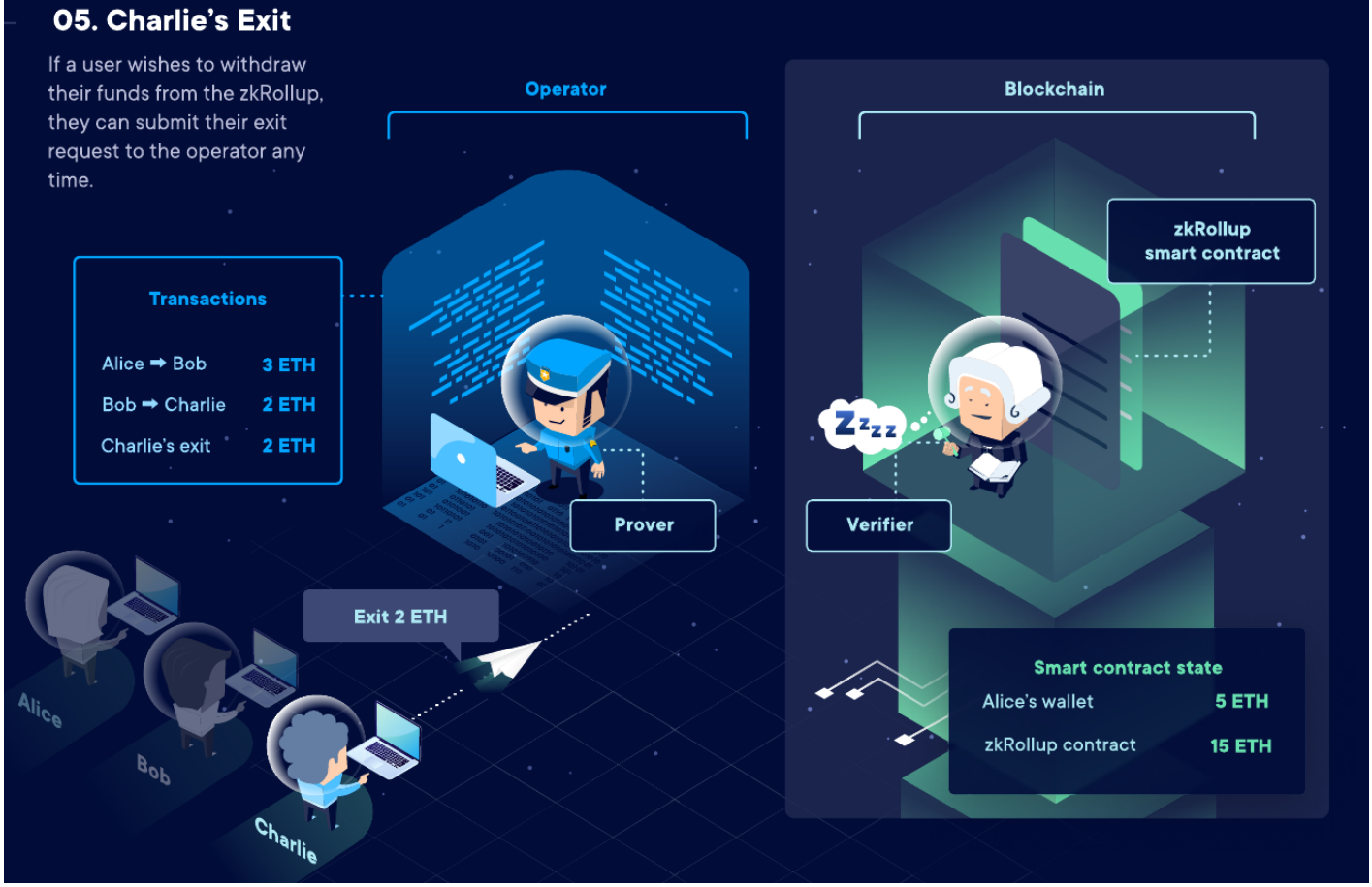
To enter the system, the user needs to transfer their funds to the zkRollup. The assets are sent to a smart contract.


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


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

05. Charlie's Exit

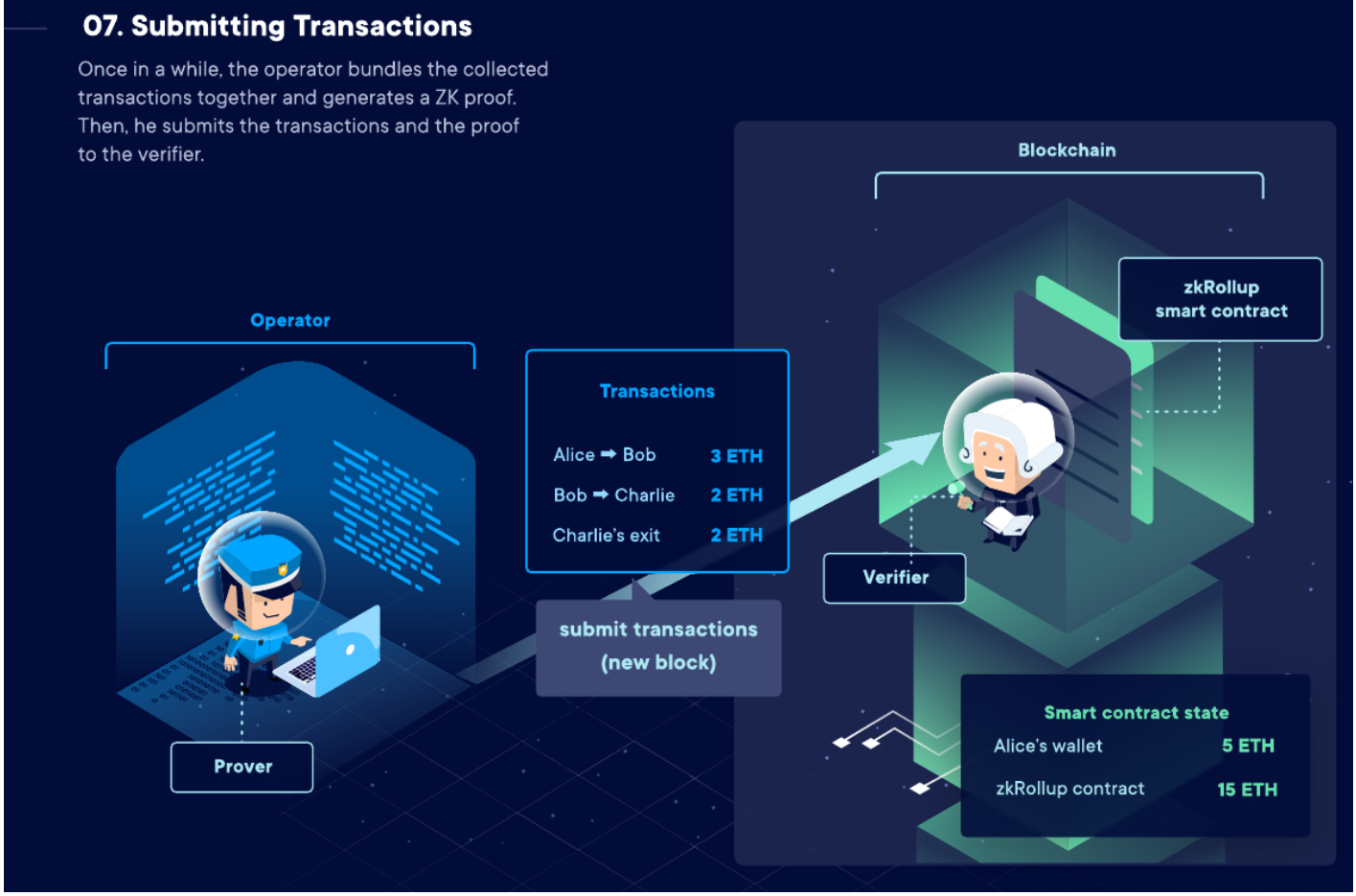
If a user wishes to withdraw their funds from the zkRollup, they can submit their exit request to the operator any time.

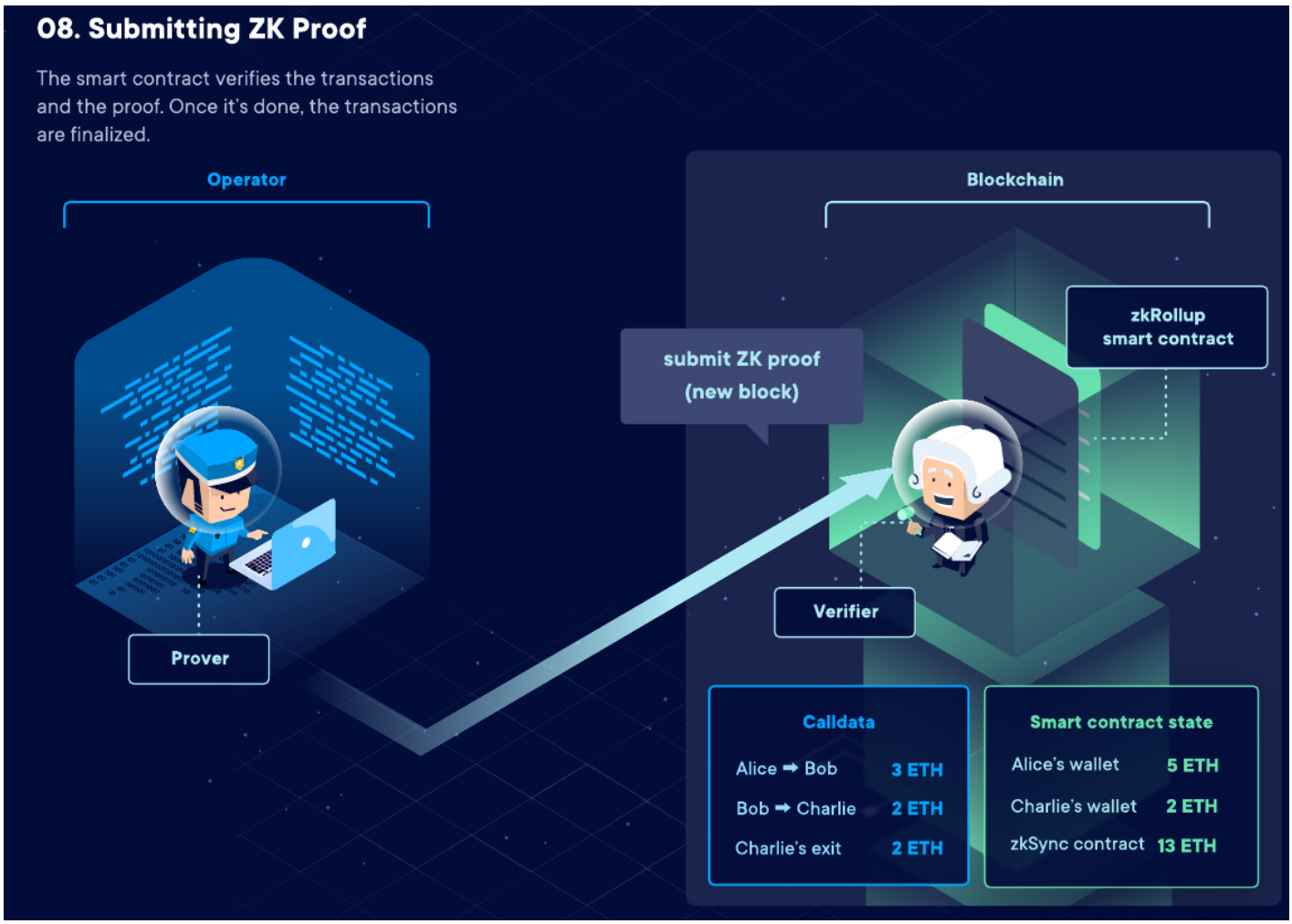


06. Collecting Transactions

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.





Comparison of the types

Property	Optimistic rollups	ZK rollups

Property	Optimistic rollups	ZK rollups
Fixed gas cost per batch	~40,000 (a lightweight transaction that mainly just changes the value of the state root)	~500,000 (verification of a ZK-SNARK is quite computationally intensive)
Withdrawal period	~1 week (withdrawals need to be delayed to give time for someone to publish a fraud proof and cancel the withdrawal if it is fraudulent)	Very fast (just wait for the next batch)
Complexity of technology	Low	High (ZK-SNARKs are very new and mathematically complex technology)
Generalizability	Easier (general-purpose EVM rollups are already close to mainnet)	Harder (ZK-SNARK proving general-purpose EVM execution is much harder than proving simple computations, though there are efforts (eg. Cairo) working to improve on this)
Per-transaction on-chain gas costs	Higher	Lower (if data in a transaction is only used to verify, and not to cause state changes, then this data can be left out, whereas in an optimistic rollup it would need to be published in case it needs to be checked in a fraud proof)
Off-chain computation costs	Lower (though there is more need for many full nodes to redo the computation)	Higher (ZK-SNARK proving especially for general-purpose computation can be expensive, potentially many thousands of times more expensive than running the computation directly)

Proofs

Optimistic rollups use fraud proofs: the rollup contract keeps track of its entire history of state roots and the hash of each batch.
If anyone discovers that one batch had an incorrect post-state root, they can publish a proof to chain, proving that the batch was computed incorrectly. The contract verifies the proof, and reverts that batch and all batches after it.

ZK rollups use validity proofs: every batch includes a cryptographic proof called a ZK-SNARK (eg. using the PLONK protocol), which proves that the post-state root is the correct result of executing the batch. No matter how large the computation, the proof can be very quickly verified on-chain.

Transaction Compression

How does compression work?

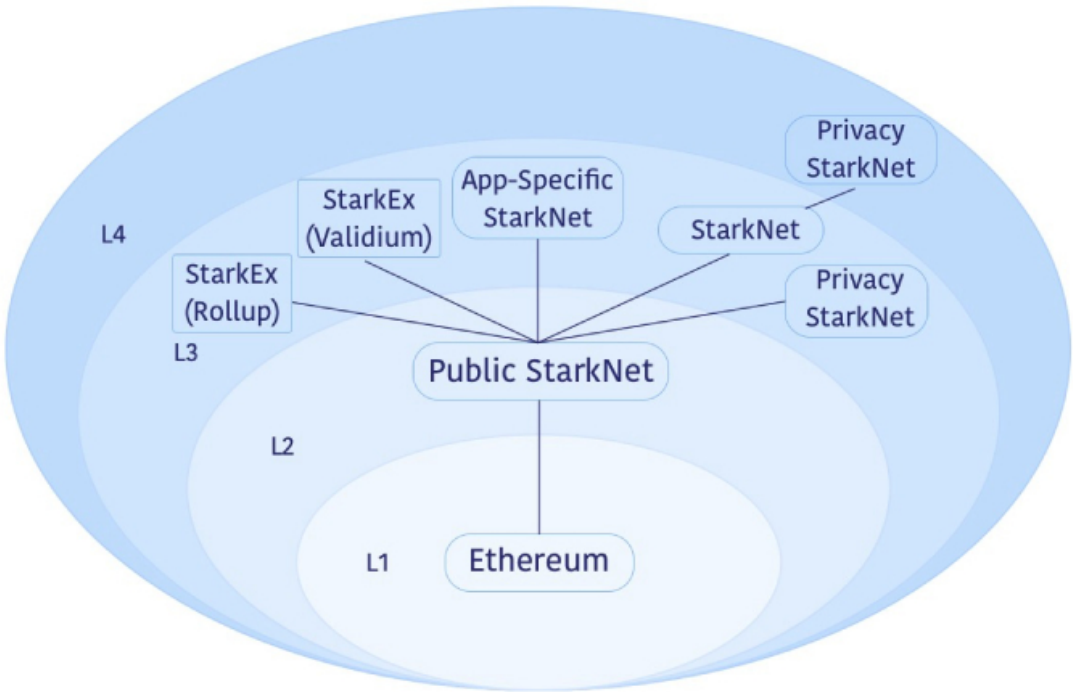
A simple Ethereum transaction (to send ETH) takes ~110 bytes. An ETH transfer on a rollup, however, takes only ~12 bytes:

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:

L3 and L4 ?

See Fractal scaling article



L2 Statistics

See L2 Beat

"Screenshot 2023-03-29 at 07.01.58.png" is not created yet. Click to create.

"Screenshot 2023-03-29 at 07.02.47.png" is not created yet. Click to create.

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.

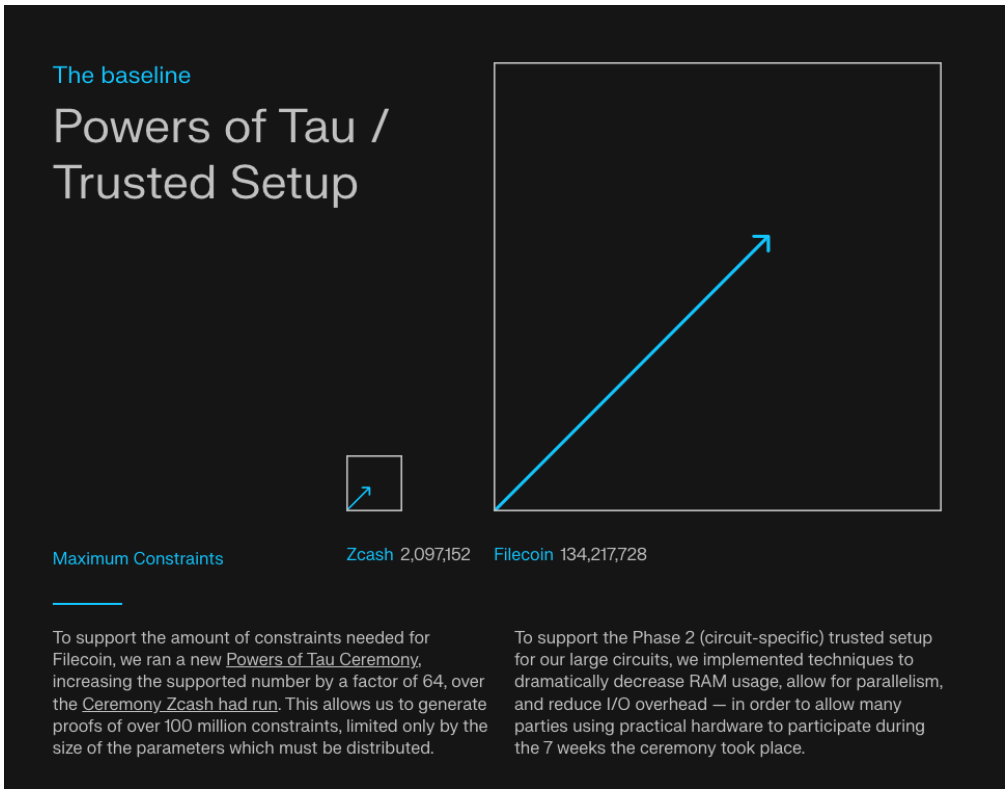
See zkSNARKS for the world

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.

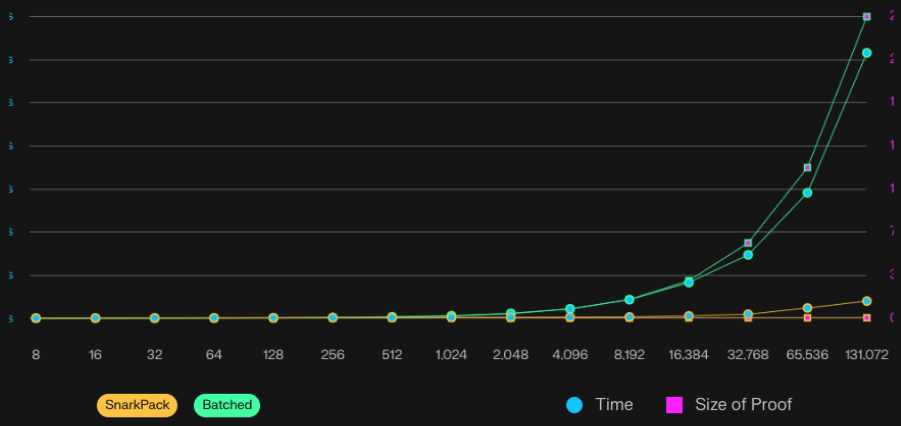
The result of an on-chain interaction in which the *prover* and *verifier* agree that data has been stored and maintained in an appropriate manner is a *proof.* As mentioned above, without a solution to make these proofs small and efficient, they would take up a tremendous amount of the network's bandwidth 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).



Pack 'em tighter

SnarkPack



SnarkPack Batched Time Size of Proof

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.

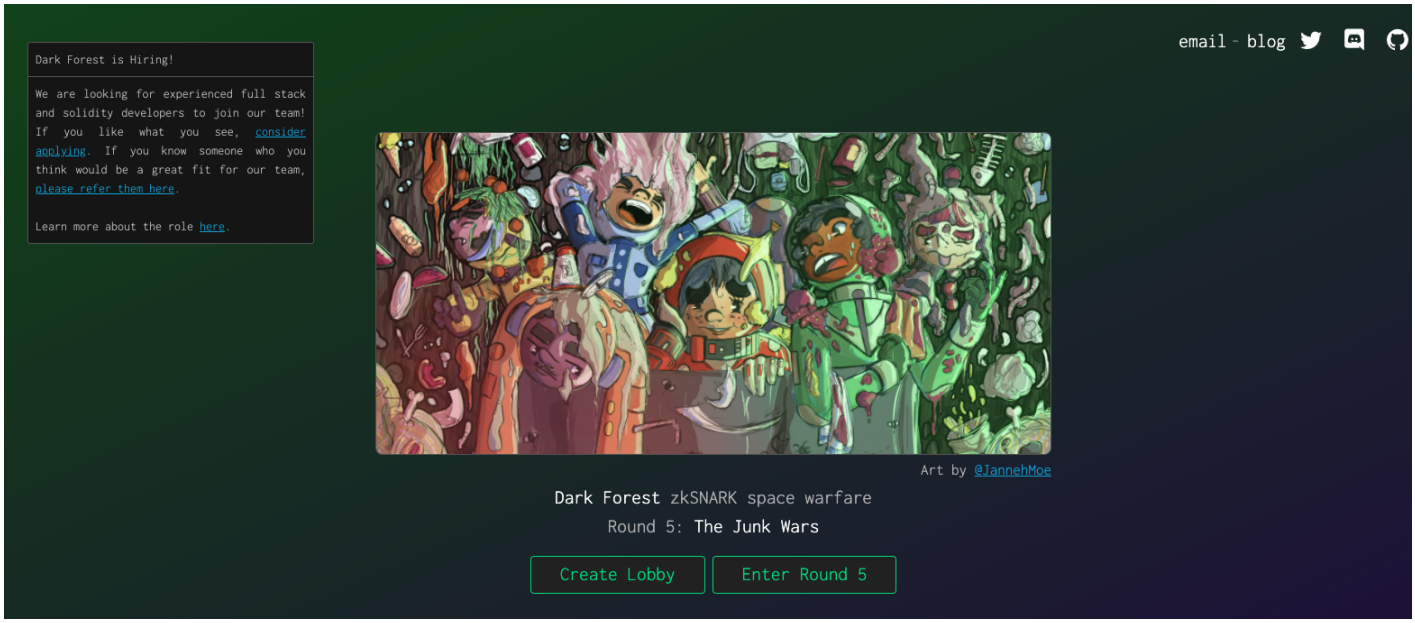
TORNADO CASH Verifier Contract

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."

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