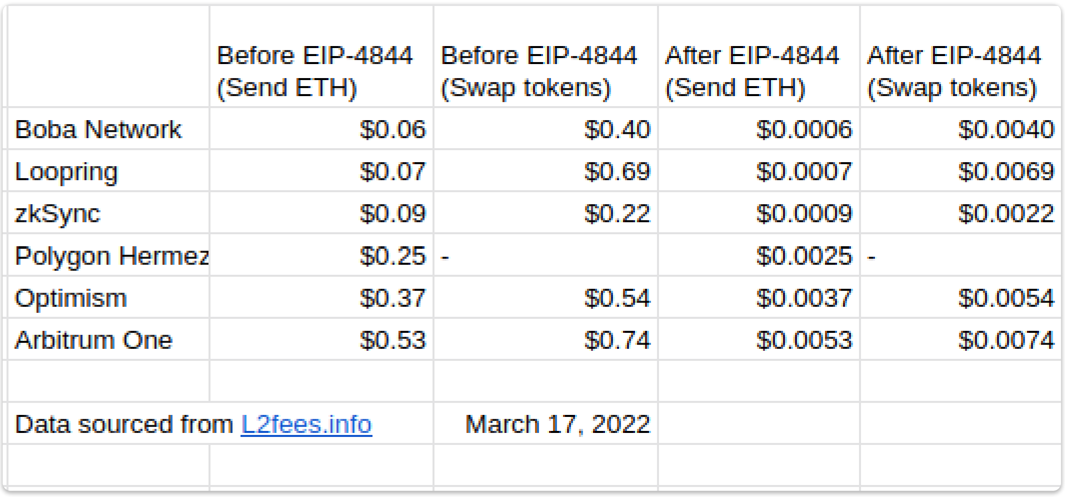
Lesson 15

News

- EIP4844- Shard blob transactions

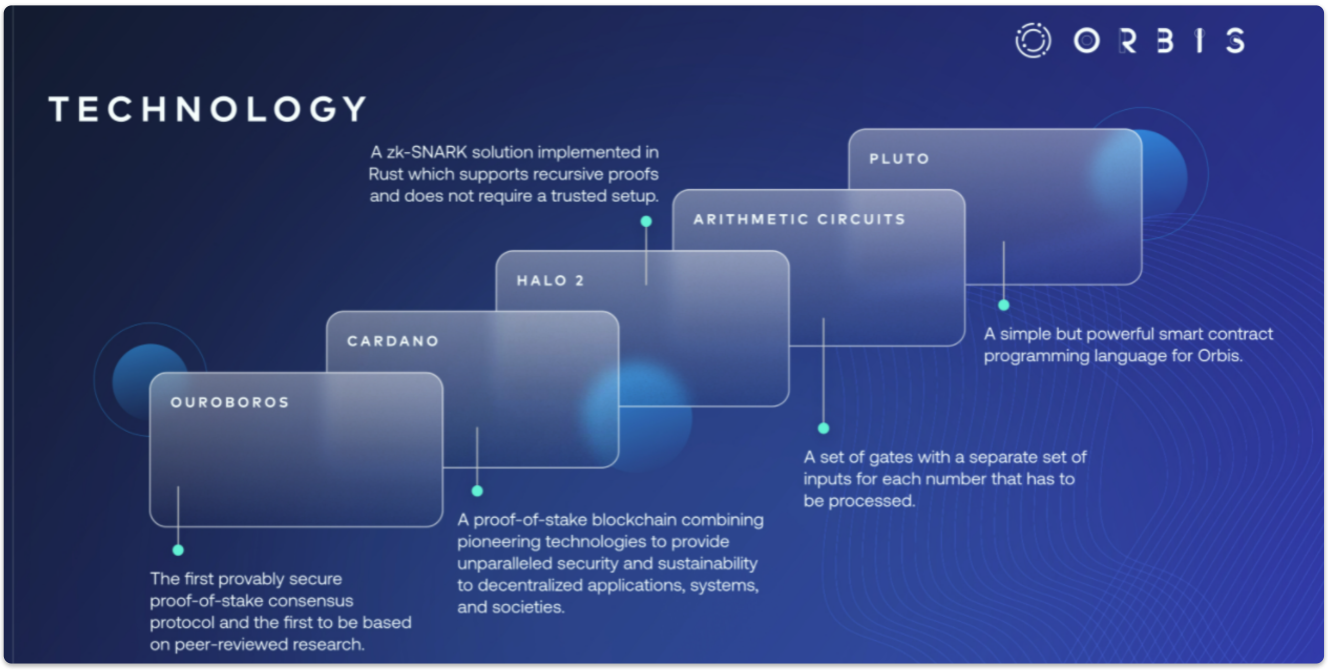
https://twitter.com/pseudotheos/status/1504457560396468231



	Before EIP-4844 (Send ETH)	Before EIP-4844 (Swap tokens)	After EIP-4844 (Send ETH)	After EIP-4844 (Swap tokens)
Boba Network	$0.06	$0.40	$0.0006	$0.0040
Loopring	$0.07	$0.69	$0.0007	$0.0069
zkSync	$0.09	$0.22	$0.0009	$0.0022
Polygon Hermez	$0.25	-	$0.0025	-
Optimism	$0.37	$0.54	$0.0037	$0.0054
Arbitrum One	$0.53	$0.74	$0.0053	$0.0074
Data sourced from L2fees.info		March 17, 2022		

- zkRollups on Cardano

https://docsend.com/view/x3xzywf8pxnixh7c



A zk-SNARK solution implemented in Rust which supports recursive proofs and does not require a trusted setup.

A simple but powerful smart contract programming language for Orbis.

A set of gates with a separate set of inputs for each number that has to be processed.

A proof-of-stake blockchain combining pioneering technologies to provide unparalleled security and sustainability to decentralized applications, systems, and societies.

The first provably secure proof-of-stake consensus protocol and the first to be based on peer-reviewed research.

- Starknetjs
  https://www.starknetjs.com/

Sigma Protocols

See article

A Sigma protocol follows these 3 steps:

1. **Commitment**: The prover generates a random number, creates a commitment to that randomness and sends the commitment to the verifier.
2. **Challenge**: After getting the commitment, the verifier generates a random number as a challenge and sends it to the prover. It is important that the verifier does not send the challenge before getting the commitment or else the prover can cheat.
3. **Response**: The prover takes the challenge and creates a response using the random number chosen in step 1, the challenge and the witness. The prover will then send the response to the verifier who will do some computation and will or will not be convinced of the knowledge of the witness.

EXAMPLE

To prove knowledge of x in $g^x = y$ without revealing x,

1. The prover generates a random number r, creates a commitment $t = g^r$ and sends t to the verifier.
2. The verifier stores t, generates a random challenge c and sends it to the prover.
3. The prover on receiving the challenge creates a response $s = r + x.c$. The prover sends this response to the verifier.

The process can be made non interactive with the Fiat Shamir heuristic as follows

1. The prover generates a random number r and creates a commitment $t = g^r$. The prover hashes g, t and y to get challenge $c : c = Hash(g, y, t)$.
2. The prover creates a response to the challenge as $s = r + c.x$. The prover sends tuple (t, s) to the verifier.

The verifier now generates the same challenge c as $Hash(g, y, t)$ and again checks if g^s equals $y^c.t$.

Sigma protocols are efficient at proving algebraic statements such as the discrete log problem, but for more general statements we use SNARKS / STARKS

Fermat's little theorem (for multiplicative inverses)

Fermat's little theorem states that if p is a prime number, then for any integer a, the number $a^p - a$ is an integer multiple of p.
In the notation of modular arithmetic), this is expressed as
$a^p \equiv a$ mod p

If a is not divisible by p, Fermat's little theorem is equivalent to the statement that
$a^{p-1} - 1$ is an integer multiple of p , that is
$a^{p-1} \equiv 1$ mod p
From this we get

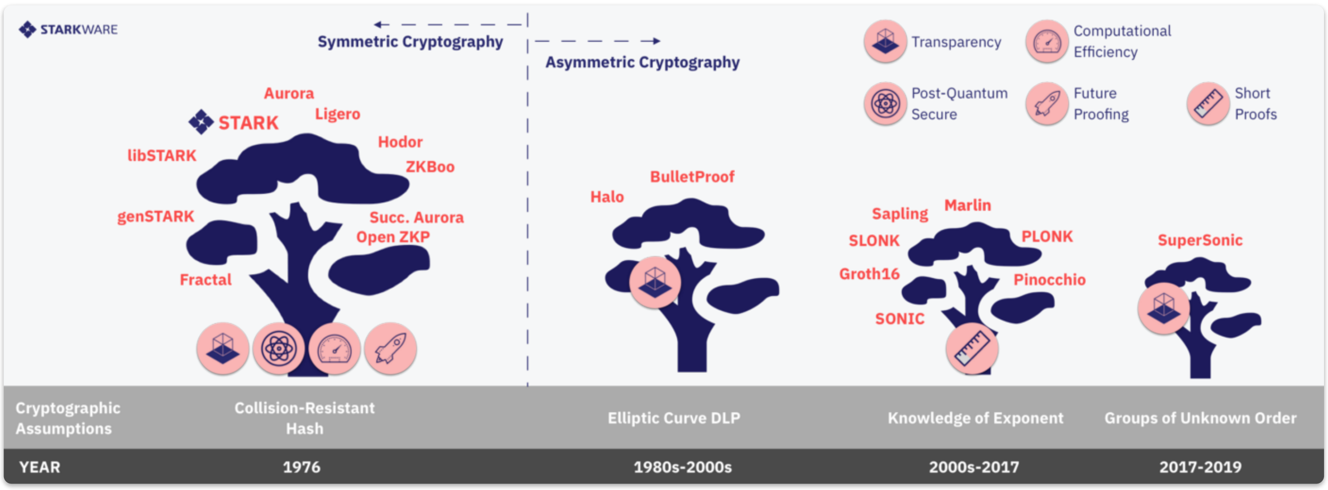
$a^{-1} \equiv a^{p-2}$ mod p
Let p = 7 and a = 2. We can compute the inverse of a as:
$a^{p-2} = 2^5 = 32 \equiv 4$ mod 7.

This is easy to verify: 2 x 4 ≡ 1 mod 7.

Overview of ZKP systems

From Cambrian Explosion article from Starkware

We can divide ZKP systems according to their underlying cryptographic assumptions, and catagorise into symmetric and asymmetric assumptions



This leads to the following consequences

1. Computational Efficiency
   Asymmetric primitives require that we arithmetize over large algebraic domains: large prime fields and large elliptic curves
   Symmetric systems can arithmetize over any field, leading to greater efficiency
2. Post-Quantum Security
   Only symmetric systems are post-quantum secure
3. Maturity of some assumptions
   1. Groups of Unknown Order and Knowledge of Exponent assumptions are less mature
4. Proof Size
   SNARK proofs are orders of magnitude smaller than STARKS, typically under 200 bytes, at 128 bits of security.
   Improvements in terms of trusted setups have increased the proof size, for example in PLONK.

Low degree compliance

There are 2 ways to acheive the 'hiding' that we need for the zero knowledge aspect of verification

1. Homomorphic hiding
   As used by libSNARK, Zokrates, ZCash etc.
   What is given to the prover is a sequence of encryptions of powers of x_0 (i.e., encryptions of $x_0^1, x_0^2, \dots x_0^1000$) so that the prover can evaluate any degree-1000 polynomial, but only polynomials of degree at most 1,000. Roughly speaking, the system is secure since the prover does not know what x_0 is, and thisx_0 is randomly (pre-)selected, so that if the prover tries to cheat then with very high probability they will be exposed.

2. Commitment Schemes
   This approach requires the prover to commit to the set of low-degree polynomials y sending some cryptographically crafted commitment message to the verifier.
   This is the method used by most STARKs like libSTARK and succinct Aurora, as well as by succinct proof systems like ZKBoo, Ligero, Aurora and Fractal.

Arithmetisation Schemes

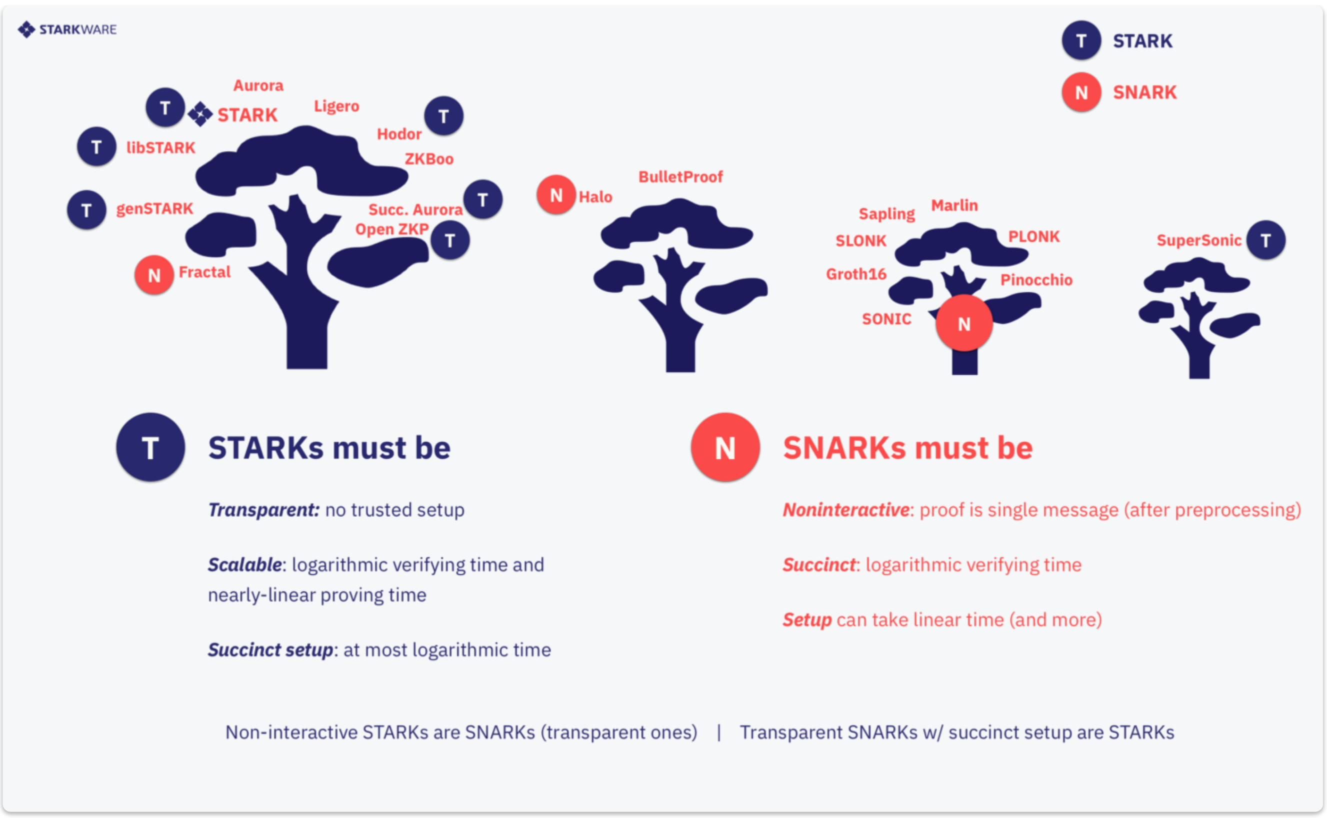
Most systems reduce computational problems to arithmetic circuits which are then converted to a set of constraints,typically, R1CS constraints.
This approach allows for circuit-specific optimizations but requires the verifier, or some entity trusted by it, to perform a computation that is as large as the computation (circuit) being verified.
Systems such as libSTARK, and succinct Aurora, must use a succinct representation of computation, one that is akin to a general computer program and which has a description that is exponentially smaller than the computation being verified.
The two existing methods for achieving this
(i) Algebraic Intermediate Representations (AIRs) used by libSTARK, genSTARK and StarkWare's systems, and
(ii) succinct R1CS of succinct-Aurora, are best described as arithmetizations of general computer programs.
These succinct representations are powerful enough to capture the complexity class of nondeterministic exponential time (NEXP), which is exponentially more expressive and powerful than the class of nondeterministic polynomial time (NP) described by circuits.

The type of assumption we follow will also dicate the domain of numbers we use, for example whether we use billeanear pairings , or cycles of curves.

Improvements to ZK Systems

Typically these have occured in 3 ways

1. Fundamentally different approaches
   Such as SNARK vs STARK



STARKs must be

Transparent: no trusted setup

Scalable: logarithmic verifying time and nearly-linear proving time

Succinct setup: at most logarithmic time

SNARKs must be

Noninteractive: proof is single message (after preprocessing)

Succinct: logarithmic verifying time

Setup can take linear time (and more)

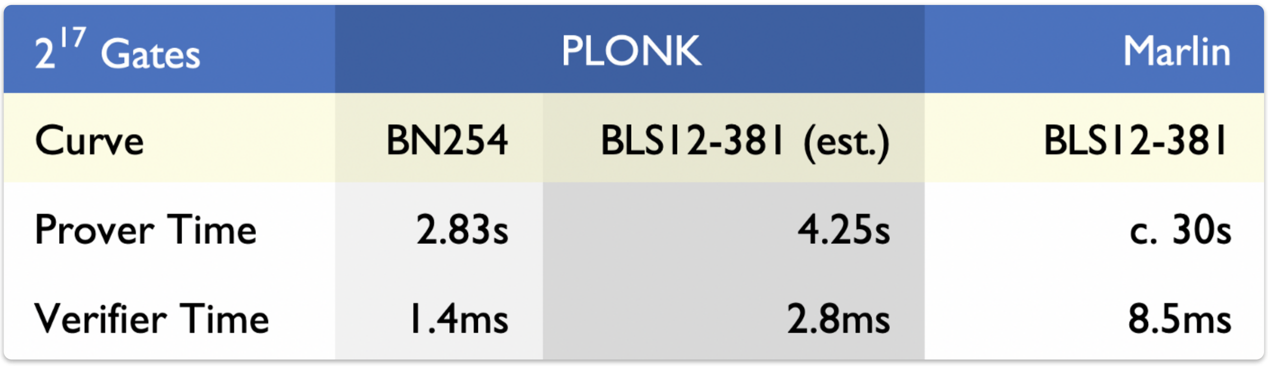
Non-interactive STARKs are SNARKs (transparent ones) | Transparent SNARKs w/ succinct setup are STARKs

RECURSIVE SNARKS

Initial attempts required paired curves to be found where as the recursion progressed the order of the curve became smaller, which meant less security.
A better approach was to use a cycle of curves which allows infinite recursion, however finding such cycles is not easy.
Examples are the pasta curves - Pallas and Vesta

2. Optimisation of existing schemes
   For example

- Amortisation of inner product
- PLONK versus SONIC

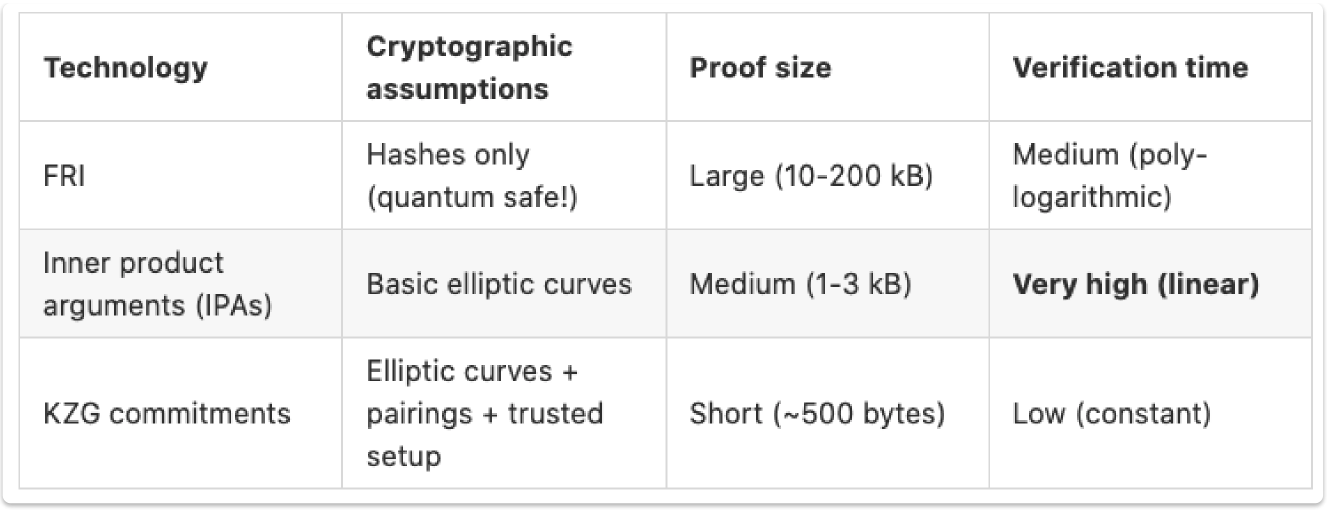


2^{17} Gates	PLONK		Marlin
Curve	BN254	BLS12-381 (est.)	BLS12-381
Prover Time	2.83s	4.25s	c. 30s
Verifier Time	1.4ms	2.8ms	8.5ms

- Incremental Verification

Article from Vitalik

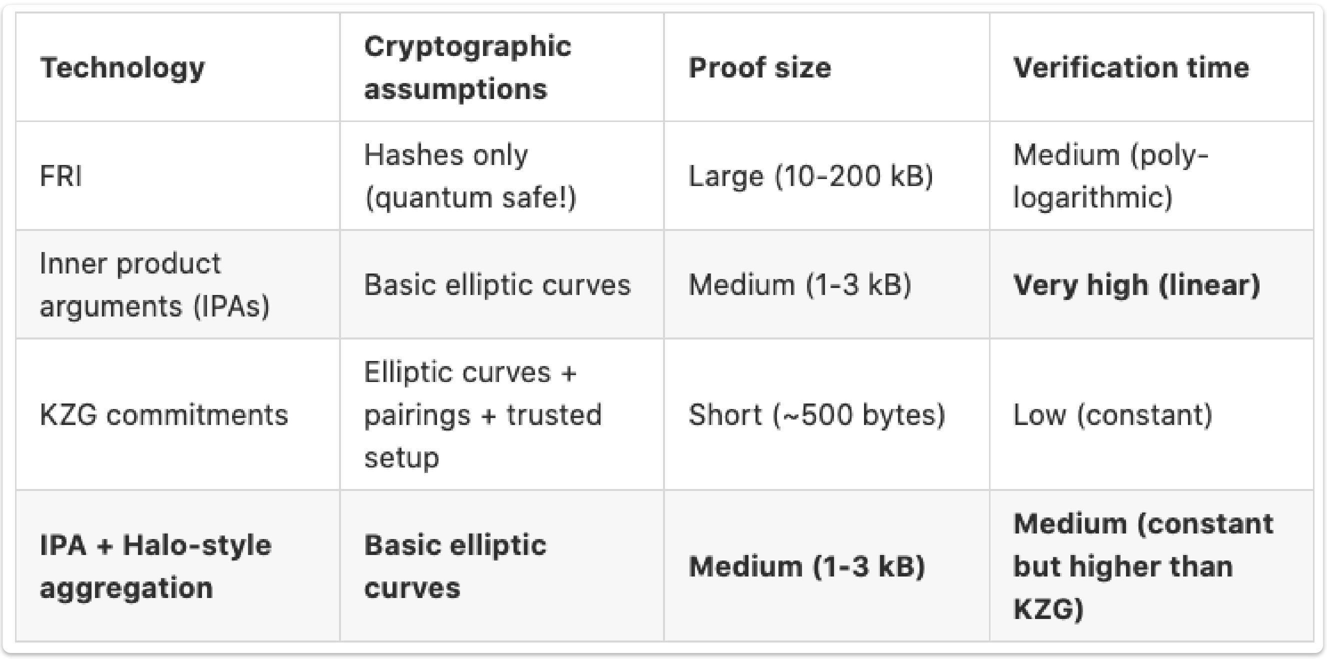
The problem is with verification times for IPAs



Technology	Cryptographic assumptions	Proof size	Verification time
FRI	Hashes only (quantum safe!)	Large (10-200 kB)	Medium (poly-logarithmic)
Inner product arguments (IPAs)	Basic elliptic curves	Medium (1-3 kB)	**Very high (linear)**
KZG commitments	Elliptic curves + pairings + trusted setup	Short (~500 bytes)	Low (constant)

instead of verifying a size-n computation with a proof that takes still takes $O(n)$ time to verify, break that computation up into smaller size-k steps, make n/k proofs for each step, and merge them together so the verifier's work goes down to a little more than $O(k)$. These techniques allow us to do incremental verification :
If new things keep being introduced that need to be proven, you can just keep taking the existing proof, mixing it in with a proof of the new statement, and getting a proof of the new combined statement out.

This is the approach taken by Halo



Technology	Cryptographic assumptions	Proof size	Verification time
FRI	Hashes only (quantum safe!)	Large (10-200 kB)	Medium (poly-logarithmic)
Inner product arguments (IPAs)	Basic elliptic curves	Medium (1-3 kB)	**Very high (linear)**
KZG commitments	Elliptic curves + pairings + trusted setup	Short (~500 bytes)	Low (constant)
IPA + Halo-style aggregation	**Basic elliptic curves**	**Medium (1-3 kB)**	**Medium (constant but higher than KZG)**

3. General Cryptographic improvements

MIMC - MINIMAL MULTIPLICATIVE COMPLEXITY

- Optimising for the large fields used in proofs.
  It addresses the question
  How could a construction for a secure block cipher or a secure cryptographic hash functions look like that minimizes the number of field multiplications?

STARKAD AND POSEIDON - NEW HASH FUNCTIONS FOR ZERO KNOWLEDGE PROOF SYSTEMS

Often the most expensive part of a ZKP is proving the knowledge of a pre image under a certain cryptographic hash function, which is expressed as a circuit over a large prime field. A zero-knowledge proof of coin ownership in the Zcash cryptocurrency is a notable example, where the inadequacy of SHA-256 hash function for such a circuit caused a huge computational penalty. Poseidon (https://eprint.iacr.org/2019/458.pdf) uses 8 times fewer constraints per message bit than a Pedersen hash.

Cost of hash verification
From https://ethresear.ch/t/using-gkr-inside-a-snark-to-reduce-the-cost-of-hash-verification-down-to-3-constraints/7550

Large arithmetic circuits C (*e.g.* for rollup root hash updates) have their costs mostly driven by the following primitives:

- Hashing (*e.g.* Merkle proofs, Fiat-Shamir needs)
- Binary operations (*e.g.* RSA, rangeproofs)
- Elliptic curve group operations (*e.g.* signature verification)
- Pairings (*e.g.* BLS, recursive SNARKs)
- RSA group operations (*e.g.* RSA accumulators)

Post Quantum Cryptography

'Sufficiently' powerful quantum computers could weaken

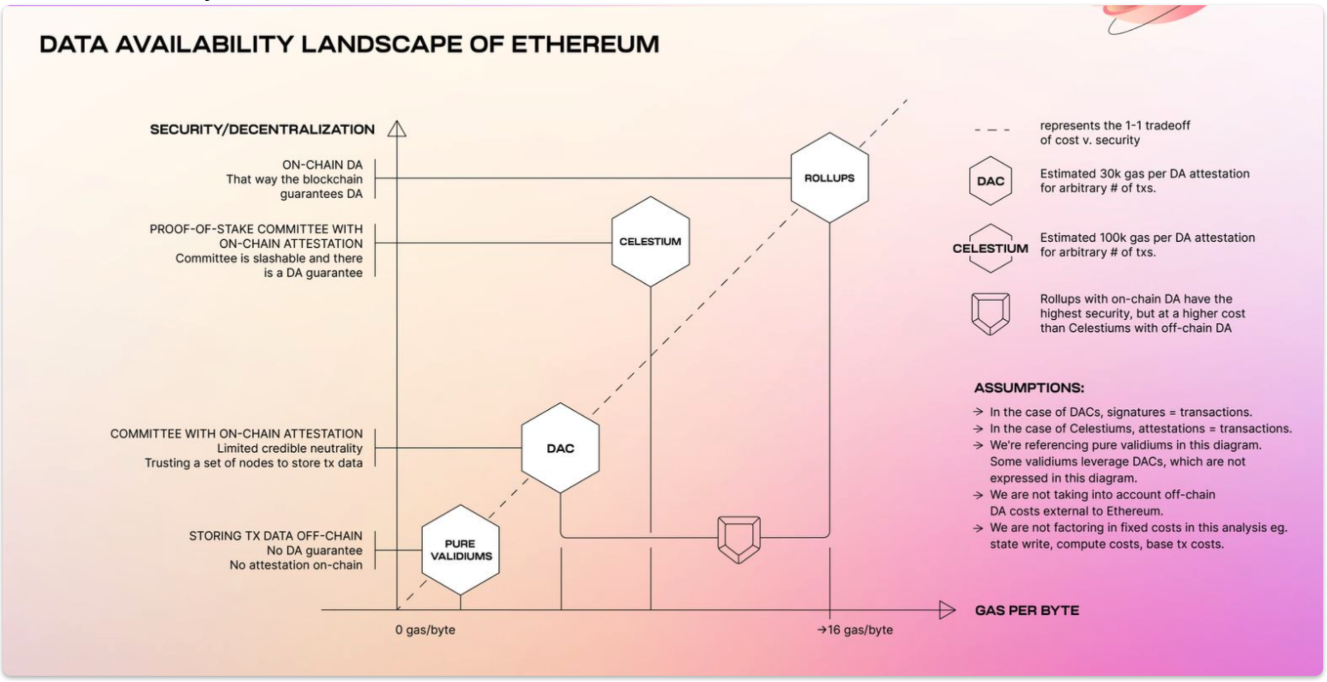
- Integer factorization problem
- Discrete logarithm problem
- EC discrete logarithm problem

Lattice-based constructions are currently important candidates for post-quantum cryptography. Unlike more widely used and known public-key schemes such as the RSA, Diffie-Hellman or elliptic-curve cryptosystems which are easily attacked by a quantum computer, some lattice-based constructions appear to be resistant to attack by both classical and quantum computers. (LWE - learning with errors is a computational problem that is thought to be hard to solve and thus can form the basis of a cryptosystem)

Lattice-based Zero-knowledge SNARGs for Arithmetic Circuits

Lattice-based Zero-Knowledge Proofs: New Techniques for Shorter and Faster Constructions and Applications

Data Availability


DATA AVAILABILITY LANDSCAPE OF ETHEREUM
SECURITY/DECENTRALIZATION
ON-CHAIN DA
That way the blockchain guarantees DA
PROOF-OF-STAKE COMMITTEE WITH ON-CHAIN ATTESTATION
Committee is slashable and there is a DA guarantee
COMMITTEE WITH ON-CHAIN ATTESTATION
Limited credible neutrality
Trusting a set of nodes to store tx data
STORING TX DATA OFF-CHAIN
No DA guarantee
No attestation on-chain
ROLLUPS
CELESTIUM
DAC
PURE VALIDIUMS
0 gas/byte
→16 gas/byte
GAS PER BYTE
DAC
CELESTIUM

represents the 1-1 tradeoff of cost v. security

Estimated 30k gas per DA attestation for arbitrary # of txs.

Estimated 100k gas per DA attestation for arbitrary # of txs.

Rollups with on-chain DA have the highest security, but at a higher cost than Celestiums with off-chain DA

ASSUMPTIONS:

→ In the case of DACs, signatures = transactions.
→ In the case of Celestiums, attestations = transactions.
→ We're referencing pure validiums in this diagram. Some validiums leverage DACs, which are not expressed in this diagram.
→ We are not taking into account off-chain DA costs external to Ethereum.
→ We are not factoring in fixed costs in this analysis eg. state write, compute costs, base tx costs.

https://pbs.twimg.com/media/FODoLYoVEAABhgd?format=png&name=small

More about rollups : https://twitter.com/pseudotheos/status/1504457560396468231

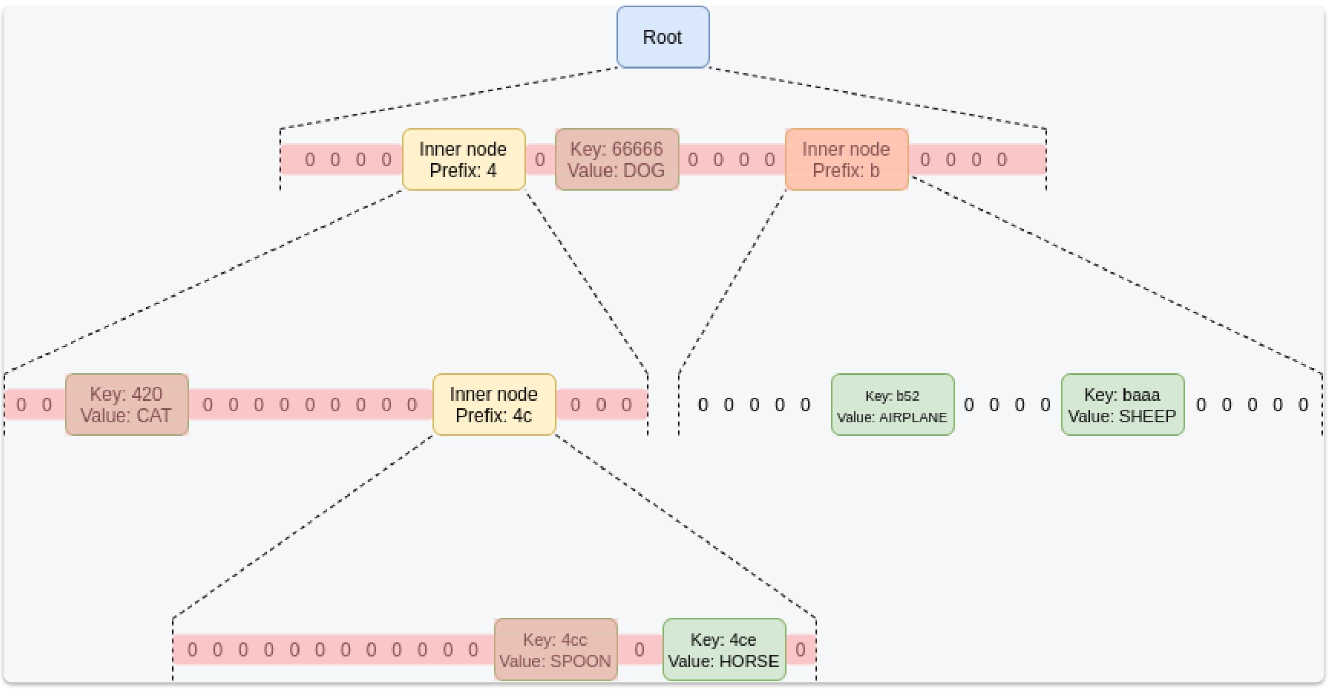
Zk rollups also differ from optimistic rollups in the case where we have a big computation in a transaction , maybe readingmany state variables, but with a single state transition, if we can create a proof of that computation, we have a big scaling advantage, but for this we need systems that can do general proof

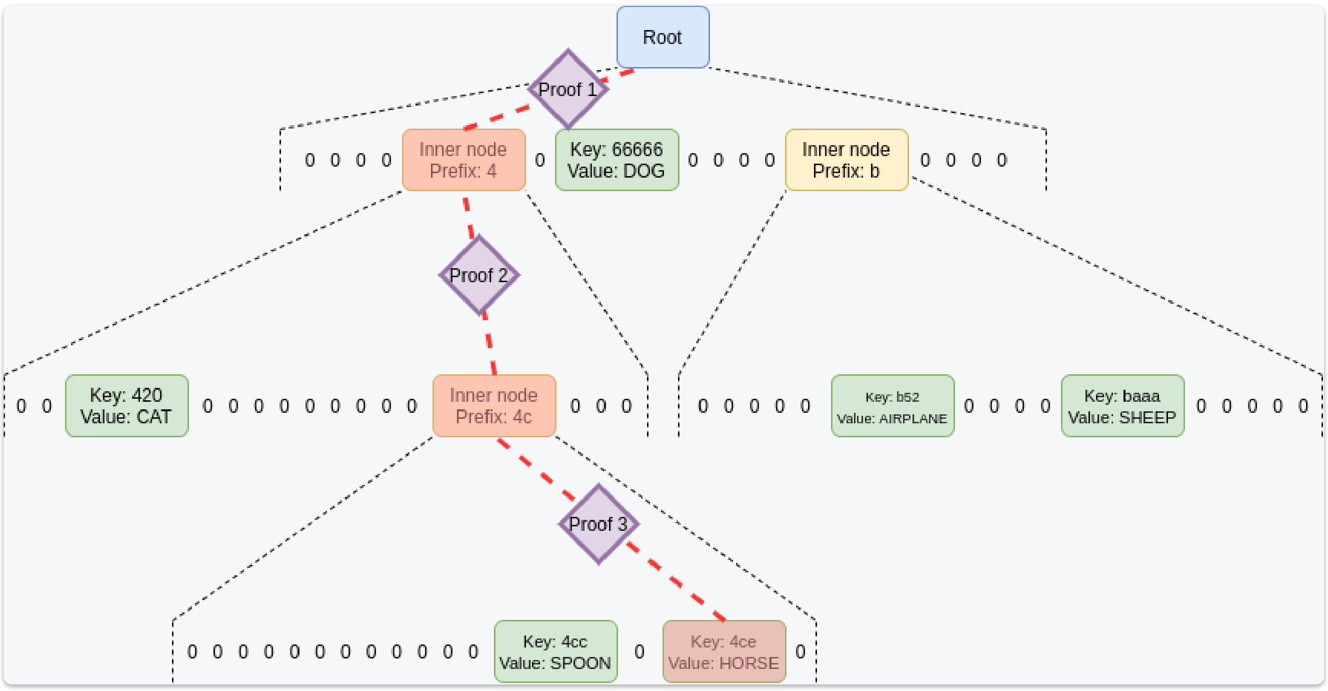
Verkle trees

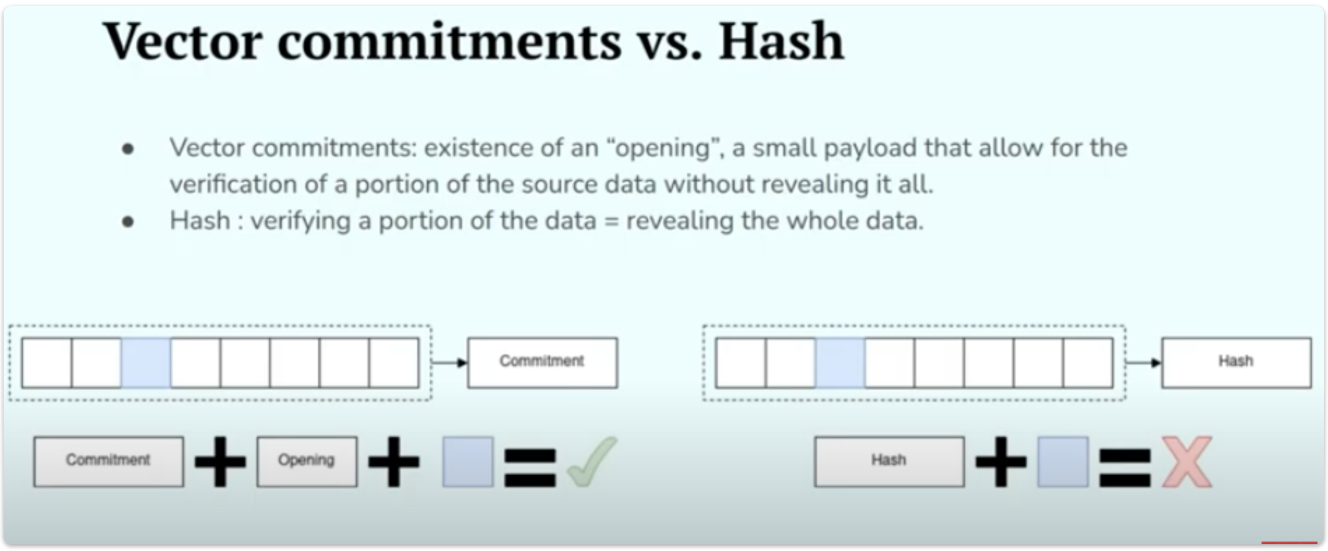
See article
and Ethereum Cat Herders Videos

Like merkle trees, you can put a large amount of data into a Verkle tree, and make a short proof ("witness") of any single piece, or set of pieces, of that data that can be verified by someone who only has the root of the tree.
Verkle trees provide, however, is that they are much more efficient in proof size. If a tree contains a billion pieces of data, making a proof in a traditional binary Merkle tree would require about 1 kilobyte, but in a Verkle tree the proof would be less than 150 bytes.
Verkle trees replace hash commitments with vector commitments or better still a polynomial commitment.
Polynomial commitments give us more flexibility that lets us improve efficiency, and the simplest and most efficient vector commitments available are polynomial commitments.

The number of nodes needed in a merkle proof is much greater than in a verkle proof



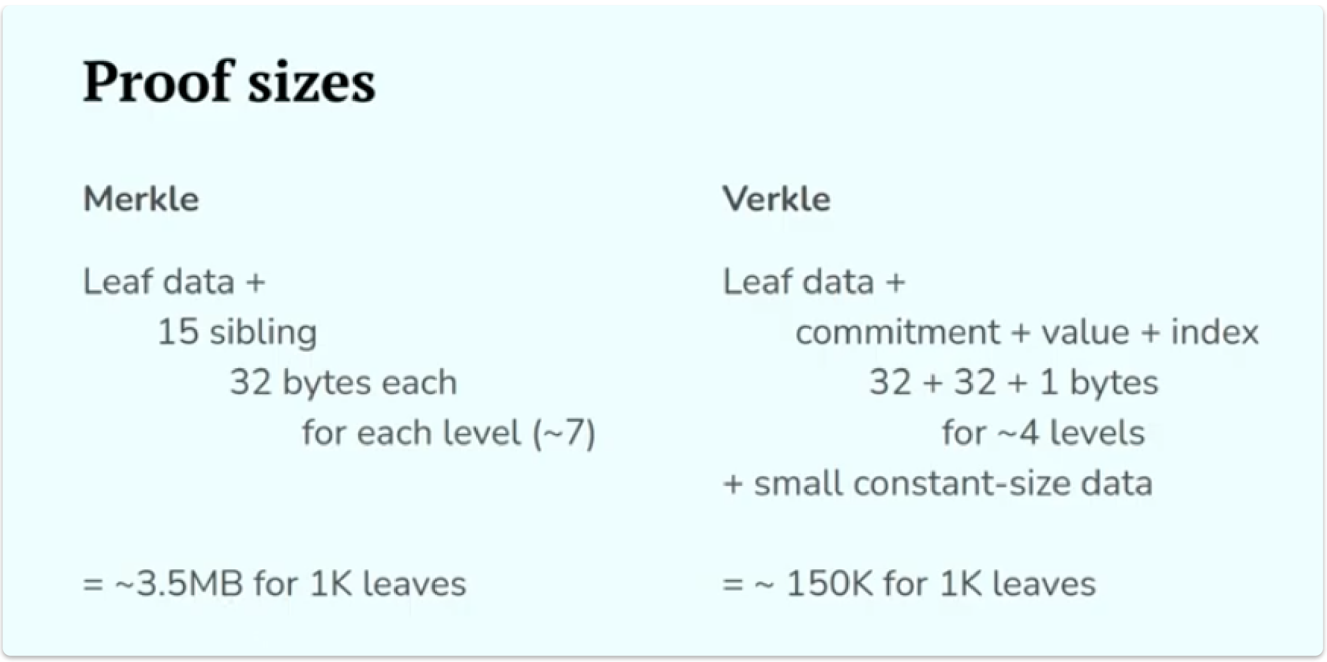




Vector commitments vs. Hash

- Vector commitments: existence of an "opening", a small payload that allow for the verification of a portion of the source data without revealing it all.
- Hash : verifying a portion of the data = revealing the whole data.

Commitment
Commitment
Opening
Hash
Hash



Proof sizes

Merkle

Leaf data +
15 sibling
32 bytes each
for each level (~7)

= ~3.5MB for 1K leaves

Verkle

Leaf data +
commitment + value + index
32 + 32 + 1 bytes
for ~4 levels
+ small constant-size data

= ~ 150K for 1K leaves

Other Research Areas

A useful resource is from Zkproof
Example papers from their latest workshops
Leo language
Sigma protocols
Framework for SNARKY ceremonies
Groth16 Aggregation
Security Analysis of MPC
Messaging on Ethereum
Benchmarking ZKP Systems