zkVerify Protocol:
A Modular & Decentralized
Proof Verification Network

Horizen Labs Research
research@horizenlabs.io
v0.1.0 - May 20, 2024

Abstract

We introduce a novel decentralized network tailored for zero-knowledge (ZK) proof verification, specifically designed to overcome cost and computational limitations in Web3. By modularizing the verification process, our zkVerify protocol significantly reduces costs and computational burdens associated with traditional Layer 1 blockchains. This shift not only facilitates broader experimentation with and adoption of diverse proving systems, but also ensures the scalability and efficiency of these systems. zkVerify's approach leverages a secure protocol to maintain the integrity and reliability of proofs across the network. Through this framework, we aim to expand the frontier of verifiable computing, making it more accessible and adaptable to a wide array of applications.

1 Introduction

Recent advancements in proving systems have integrated a range of diverse finite fields, state-of-the-art hash functions, elliptic curves, and polynomial commitment schemes. Despite these exciting developments, significant barriers hinder adoption. Many projects using these systems require their proofs to be verified, and in doing so, they encounter prohibitive verification costs and arduous DAO processes, ultimately stifling potential exploration and innovation.

We introduce zkVerify, a decentralized network for ZK proof verification, that substantially lowers the costs of verification and extends the range of possibilities within Web3 to use different cryptographic primitives and systems. As execution and data availability have been offloaded from monolithic blockchain stacks, we believe that proof verification and settlement need specialization. This system provides such specialization and, by doing so, enables the next wave of innovation for ZK cryptography within Web3 and beyond.

1.1 The State of Proof Verification

1.1.1 Proof Verification Market

As we transition into the age of verifiable computing, ZK proofs emerge as pivotal accelerators, enabling the secure and private verification of computations using arguments of computational integrity. This technological leap forward has already led to revolutionary breakthroughs in blockchain scalability, including zkRollups, zkVMs, zkApps, and other ZK-enabled solutions. Some research reports have projected ZK proofs to exceed **$10 billion in revenue** by the year 2030 [1].

In terms of the total number of proofs created by 2030, it has been estimated that **90 billion proofs will be generated** by Web3 applications alone [1]. When discussing proofs, most of the discussion has revolved around either proof generation and/or proof aggregation, and legitimately so, since the majority of improvements can be realized in those areas. However, the aspect of **proof verification has not received equal attention**. For those anticipated 90 billion proofs that will be generated, 90 billion proofs will also need to be verified.

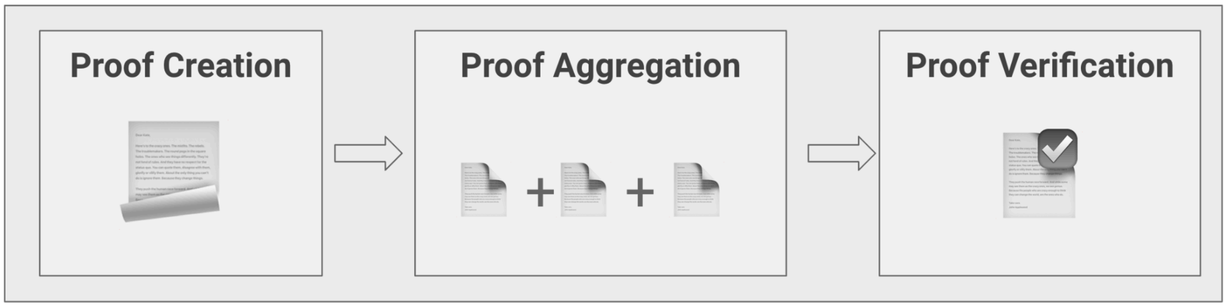


Figure 1: A simplified view of the lifecycle of a ZK proof, where proof generation and aggregation has received most of the focus. Proof verification has not been given equal emphasis.

1.1.2 Hampering Innovation

Ethereum Improvement Proposals (EIPs) are design documents that outline new features, standards, or processes for the Ethereum blockchain, serving as a primary mechanism for proposing changes to the network. Two EIPs (EIP-196 and EIP-197) significantly impacted the development of rollups and zkVMs (zero-knowledge virtual machines) by providing essential cryptographic **building blocks to perform ZK proof verification** on the Ethereum blockchain.

They introduced precompiled contracts for elliptic curve operations and pairings on the **alt_bn128 elliptic curve** (BN254 / Barreto-Naehrig curve), which is a popular curve for zk-SNARKs. Released to Mainnet as a part of the Byzantine hard fork on October 16, 2017, both EIPs directly influenced many rollups (and other clients) to **convert their proofs into compatible BN254 pairing-friendly proofs** before on-chain verification. This conversion process also included aggregation of proofs, which resulted in longer finality times.

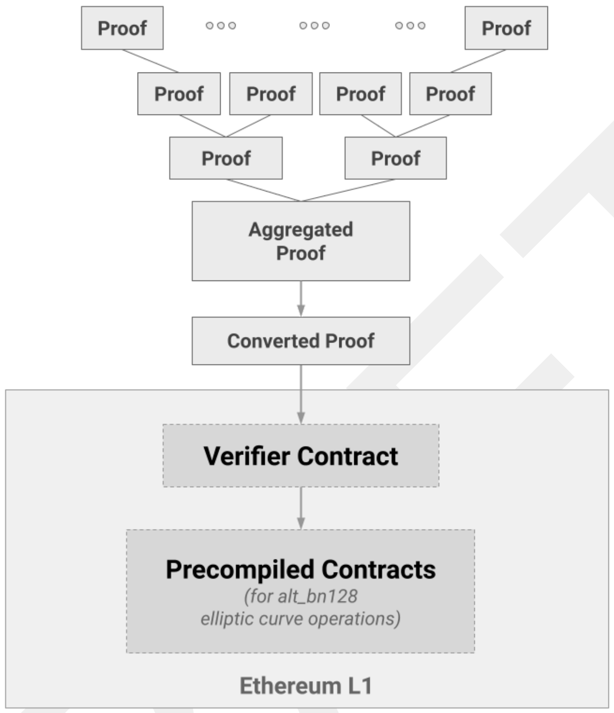


Figure 2: The process followed by zkRollups (and other clients) to aggregate and convert their proofs for on-chain verification, leveraging the available precompiled contracts.

The choice to **standardize around the BN254 curve**, while practical at the time of implementation, means that operations involving other elliptic curves are not directly supported and are prohibitively expensive to execute. A notable example is the **BLS12-381 elliptic curve**, which is increasingly preferred in modern cryptographic applications due to its higher security margins and efficiency in handling certain types of cryptographic proofs. EIP-2537 was created in February 2020, and it is unclear whether its implementation will be available soon. This lack of support restricts the variety of cryptographic techniques that can be efficiently employed on the platform, **limiting innovation as cryptographic standards evolve**.

In general, progressing EIPs forward can be challenging due to the rigorous process involved, its need for widespread consensus via the DAO, and the lack of effective prioritization amongst multiple competing priorities with different stakeholders. EIP-2537 is just one example that highlights how the slow EIP process can stifle innovation, especially in areas like ZK cryptography where rapid development is crucial. The bottlenecks in approving and implementing EIPs hinder the adoption of cutting-edge proving systems, limiting the potential advancements in scalability and privacy.

1.1.3 Prohibitive Cost

From a macro cost perspective, the proof verification market is estimated to incur $100+ million in security expenses alone for zkRollups in 2024, extending to $1.5 billion by 2028 when including ZK applications.

Proof Verification Market	**Projected Annual Expenses**
2024: zkRollups Only	$100+ million
2025-2027: zkRollups Only	$500+ million
2028+: Including zkApps, such as Private Voting, ID, Gaming.	$1.5+ billion

Figure 3: Projected annual security expenses for proof verification alone [2].

On a more granular level, the verification of a single ZK proof on Ethereum can range from $16 to $205 per proof, depending on the proof type in the current market. Beyond nominal fees today, the variability of future fees inhibits product adoption. Offloading proof verification from L1s, such as Ethereum, serves to both drastically lower nominal costs, but also to stabilize costs over time in a way that segregates fees from gas volatility.

Proof Type	**Gas Cost** (per proof)	**USD Cost** (per proof)
Fflonk	179,187	$16.13
Plonk	292,685	$26.34
Halo2	320,108	$28.81
Groth16	194,060	$29.39
STARK	2,273,943	$204.65

Figure 4: Cost of verifying a single proof, assuming 30 gwei and ETH at $3,000 [3].

Proof verification costs can contribute to being prohibitively difficult to perform private DeFi transactions, cast a private vote on a DAO, or validate proof of personhood. Identity applications, such as Worldcoin [4], verify ZK proofs to ensure the user's uniqueness without revealing personal data. Each proof verification can consume upwards of 200,000 to 300,000 gas units. Given that gas prices can fluctuate significantly, the cost of verifying a single proof can vary widely. In times of network congestion, gas prices have reached over 100 Gwei, which means that verifying a single proof could cost between \$20 to \$60 or even more, depending on the Ethereum price and network conditions. To alleviate costs, Worldcoin has recently announced a move to an L2 blockchain based on Optimism's OP stack. While this may mitigate the issue in the near term, it does not address the fact that billions of dollars of users' liquidity and assets still live on Ethereum and not its L2s.

1.2 Our Contribution

With zkVerify, we are committed to advancing innovation by supporting the continued rise of novel proving systems that utilize diverse finite fields, cutting-edge hash functions, elliptic curves, and polynomial commitment schemes. As newer and more efficient proving systems are being introduced, the zkVerify protocol will seamlessly **integrate new proof types into our proof verification network**.

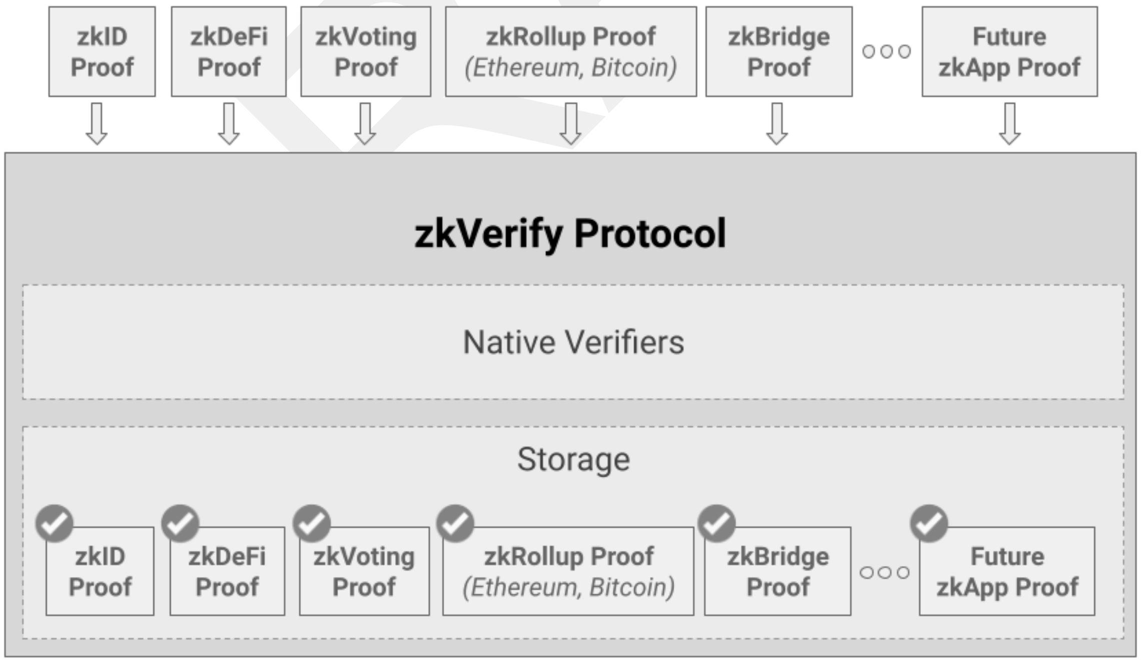


Figure 5: The versatility of zkVerify to accept proofs from any client, including zkRollups, zkBridges,

zkApps, and support for any new future zkApps.

As ZK proof technology becomes more accessible with both hardware and software improvements, we anticipate a **significant surge in demand for proof verification services**. This increase will stem from a variety of sectors seeking enhanced privacy and security measures, necessitating a robust and flexible infrastructure capable of supporting a wide array of cryptographic proofs without the limitations currently faced on L1 platforms.

zkVerify is designed to meet this burgeoning demand, ensuring scalability, security, and cost. Below are projections on cost savings using the zkVerify protocol.

Proof Type	**Gas Cost** (per proof)	**USD Cost** (per proof on Ethereum)	**Projected cost with zkVerify**
Fflonk	179,187	\$16.13	\$1.45
Plonk	292,685	\$26.34	\$2.37
Halo2	320,108	\$28.81	\$2.59
Groth16	194,060	\$29.39	\$2.65
STARK	2,273,943	\$204.65	\$18.42

Figure 6: Potential cost from leveraging zkVerify,
after applying a projected cost reduction of 91% [5].

The proof verification process stands out as the **next frontier for both cost reduction and innovation**. It unlocks the freedom to explore new paradigms, including new proving systems (such as efficient SNARKs over binary fields [6]), exploration into new hash functions, continued advancement in both Ethereum and Bitcoin rollups, and cost efficiency to zkApps. Just like the way L2s innovated Ethereum, our goal is to **supplement and partner with L1s to accelerate their cryptographic roadmaps** to bring even more breakthroughs to life.

2 zkVerify Architecture

2.1 Overview

zkVerify consumes the influx of proofs that come from various sources, such as proofs from zkID apps, zkDeFi apps, zkVoting apps, ZK games, ZK NFT apps, zkRollups, and ZK bridges. These proofs can be created from **different proving systems** with various formats, proof sizes, recursive composition, and circuit complexity. These could include proof types such as Groth16, Plonk, Fflonk, UltraPlonk, Halo2, STARK, and Boojum.

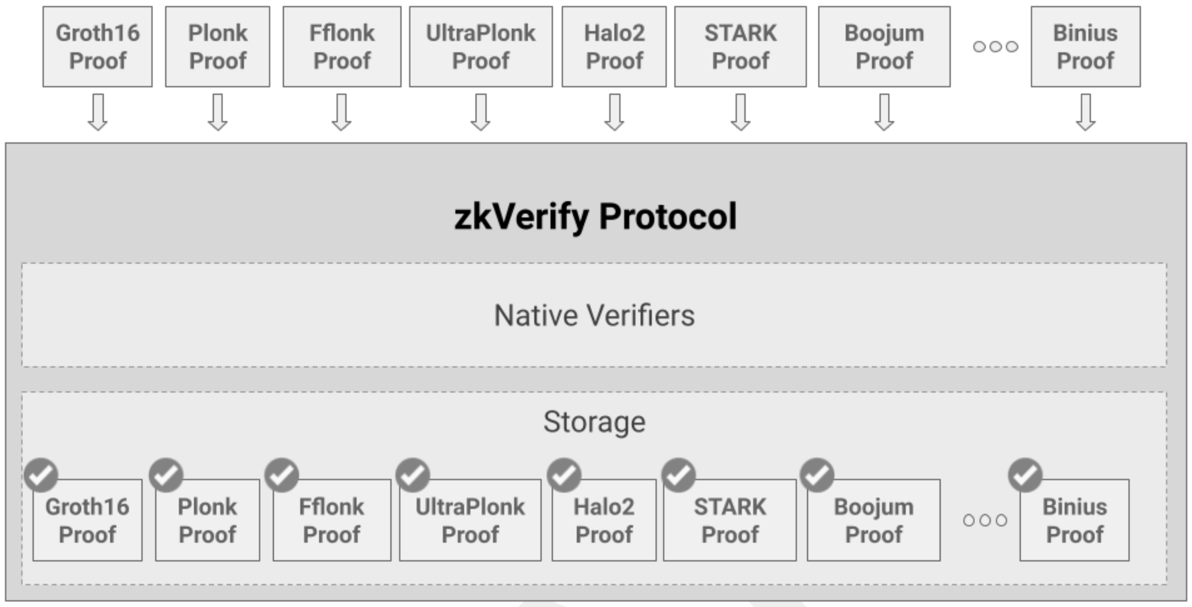


Figure 7: An alternative view of Figure 5, where zkApps, rollups, and bridges submit proofs that are created from various heterogeneous proving systems.

Developers have the freedom to choose the proving system tailored for their use case, adjusting for speed, security, proof size, cost, and any other considerations. Figure 5 shows the high-level workflow where:

- Proofs are accepted from any client, ranging from zkRollups, zkBridges, and zkApps. These proofs can originate from many heterogeneous proving systems.

- zkVerify verifies these proofs with native verifiers written in Rust, which is more efficient and more easily upgradable than Solidity. As new proving systems are introduced, new native verifiers can be conveniently added/upgraded without impacting clients.

- The verified proofs are then included in storage for public accessibility.

2.2 Core Components

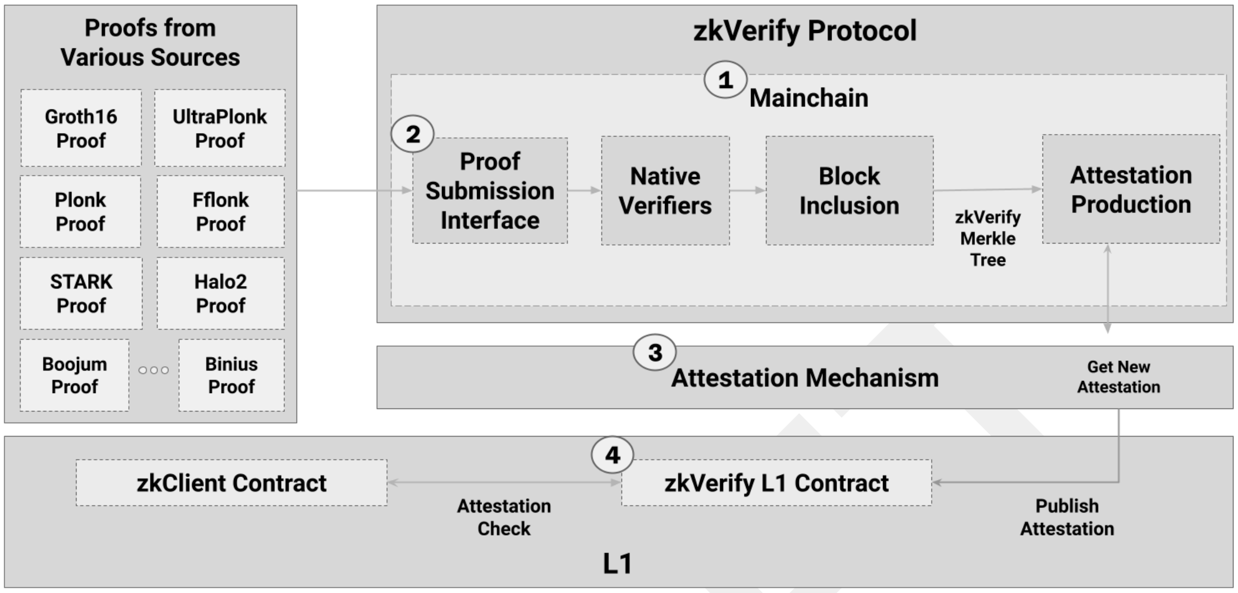


Figure 8: zkVerify architecture
and its high-level components.

From a high-level architectural perspective, the core components of zkVerify are:

1. **Mainchain:** This is an L1 Proof-of-Stake blockchain, whose main responsibilities are to receive, verify, and store validity proofs. It houses the verifier modules (such as the native Rust verifiers for Fflonk and Groth16), ensures that the proof is included into a block, and then creates the zkVerify Merkle tree.

2. **Proof Submission Interface:** This is the interface (for transactions and RPC calls) that receives heterogeneous proofs from various sources, such as zkApps, zkRollups, zkBridges, and other ZK clients.

3. **Attestation Mechanism:** This refers to the protocol that creates and publishes an attestation, which contains the Merkle root (of a heterogeneous proof tree), onto the zkVerify L1 contract once a given policy is met.

4. **zkVerify L1 Contract:** The core responsibility of this smart contract is to store new attestations, validate them, and provide capabilities for users to verify that their proof is part of the attestation.

2.3 High-Level Workflow

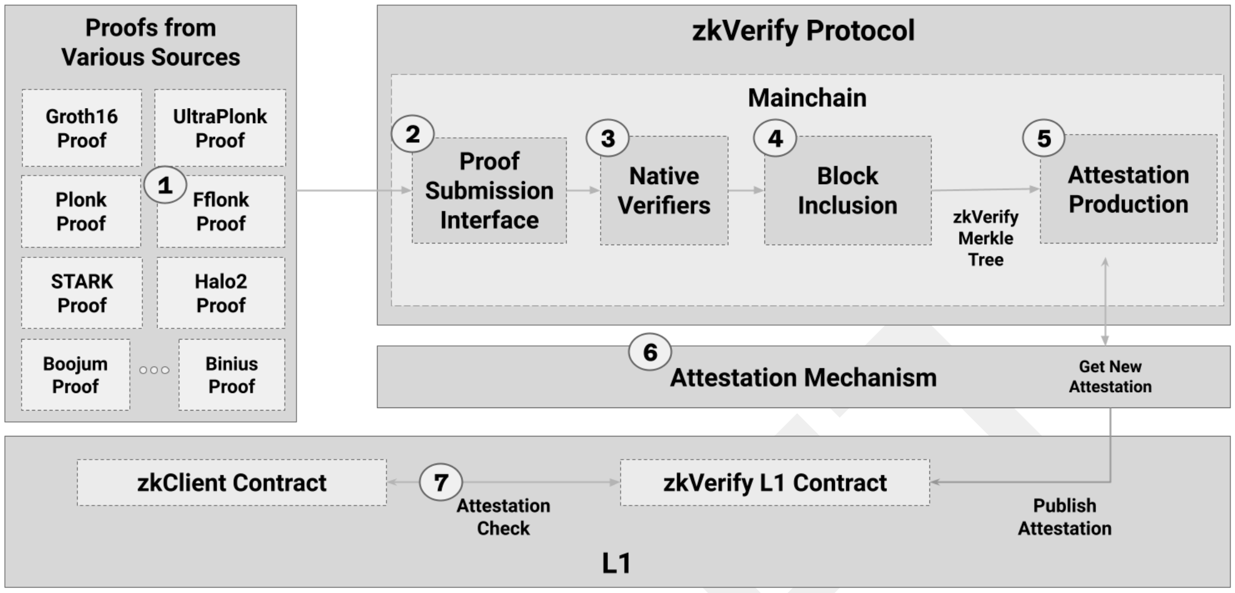


Figure 9: zkVerify high-level workflow.

The workflow of the zkVerify protocol is designed with foundational principles of efficiency, scalability, and upgradability. The steps in this workflow are as follows:

1. A zkApp wants to verify a proof. In this example, it is an Fflonk proof.

2. The Fflonk proof is submitted to the zkVerify protocol via the Proof Submission Interface, which has the following format:

Transactions

- **Signature**: `submitProof(proof, vk, public_inputs) -> Result<AttestationId, Error>`
  Pallet: `SettlementFflonkPallet`
  Type: Non-blocking
  Description: Submit the proof and verify it against the supplied vk and public inputs. Returns the id of the attestation in which the proof will be included, `Error` otherwise (please note that this will be done only upon inclusion of the proof in a mainchain block).
  Note: Each different proof verification pallet will expose this API.

RPC Commands

- **Signature**: `proofPath(attestation_id, proof_leaf) -> Result<merkle_path, Error>`
  Endpoint: `PoE (Proof of Existence)`
  Type: Non-blocking

Description: Get the Merkle path of the proof within the provided attestation, returning `Error` if the `attestation_id`'s `Attestation` event isn't published yet.

3. The Fflonk proof is then verified by a native Rust verifier.

4. After verification, the proof is then queued for block inclusion.

5. An attestation is produced based on a set of given policies which includes the attestation size and frequency of submission. The attestation data structure is a digitally-signed message that contains the
   - Merkle root of the zkVerify Merkle tree that contains proofs as leaves, and the
   - Attestation ID used for identification, synchronization, and security purposes.

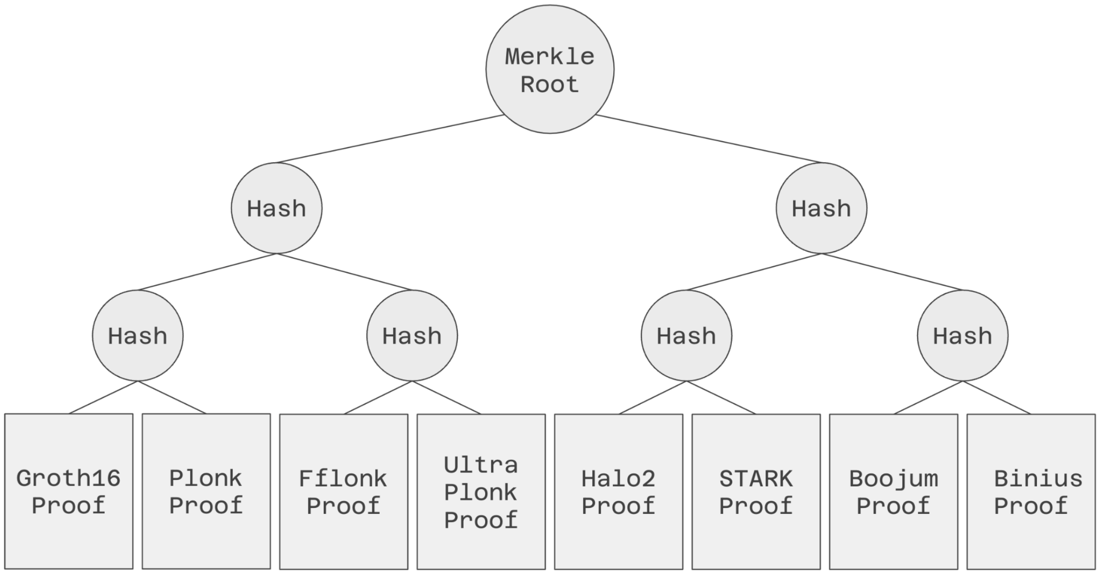


Figure 10: zkVerify Merkle tree, which shows various proof types at its leaves, serving as a form of natural aggregation across heterogeneous proving systems.

6. The attestation is then sent to a relayer that publishes the attestation to a zkVerify L1 contract. The policy leading to the publication of a new attestation is met when one of the following rules is satisfied:
   - Last attestation contains `N` proofs, where `N` is the attestation size.
   - Last published attestation is older than `T` seconds and there is at least one proof in the new tree.

zkVerify will provide various options on which destination chains to send an attestation. The users, when submitting a proof, will need to specify the destination chain. Note that:

- We can envision different attestation publication policies for each destination chain, depending on different characteristics. Ideally, we would like this policy to change dynamically depending on how the destination chain changes (e.g. we might employ an oracle) and/or how many users are interested in bridging on that chain, instead of it being fixed since genesis and thus hardly changeable.

- Users submitting proofs to zkVerify will pay the fee required to perform basic operations plus the fee required to compensate the relayer for posting the proof on the destination chain (different for each chain).

7. The zkApp's contract verifies that their proof has been attested by a published attestation via a Merkle proof of inclusion.

3 zkVerify Use Cases

3.1 Unlocking New Proving Systems

3.1.1 Building Blocks of ZK Proof Systems

The performance of a modern ZK proof system heavily relies on:

- Its method for polynomial interpolation, and
- Its underlying hash function.

Together, these two building blocks easily make up more than 70% of many modern proving systems, which is why significant amounts of research have been put into making both of them faster[1].

[1] The remaining 30% is determined by the specific circuit and its operations, leading to variations across different use cases. It's uncertain whether this portion can be easily optimized, if at all, whereas optimizing interpolation methods and hash functions presents a more straightforward target.

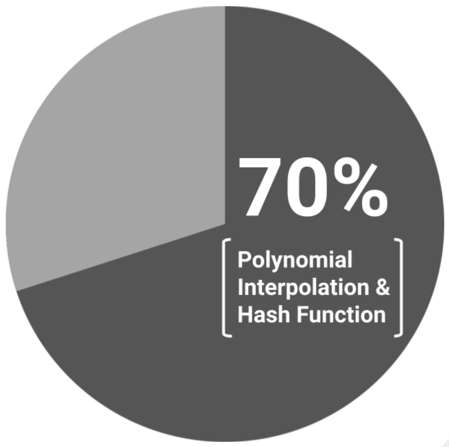


Figure 11: Chart illustrating how the performance of a modern proving system is impacted by two main building blocks: its polynomial interpolation method & its underlying hash function.

Polynomial interpolation for a proving system is usually implemented by applying methods such as the fast Fourier transform (FFT). **New methods for interpolation are being discovered at a fast pace**, such as ECFFT [7] and Circle STARKs [8].

Innovation in the field of hash functions is also moving swiftly. Latest developments include a focus on FRI-friendly fields. FRI (Fast Reed-Solomon Interactive Oracle Proof of Proximity) is a technique used by modern proving systems to use solely hashing, avoiding the complexities associated with elliptic curves. Some of these emerging hash functions for FRI-friendly fields include Poseidon2 [9], Monolith [10], and Vision Mark-32 [11].

For example, while most hash functions focus on prime fields, the recent Vision Mark-32 hash function relies on binary extension fields, allowing for an efficient recursive proving approach in some settings. In hardware implementations, binary extension fields are the more popular choice compared to prime fields, mainly because they do not require complex wiring for carry bits.

As building blocks to modern proving systems, both polynomial interpolation and hash functions are achieving remarkably rapid progress. The zkVerify protocol can **fast-track these innovations** as they are being integrated into newer and more efficient proving systems.

3.1.2 Enabling the Adoption of new Proving Systems

As the foundational building blocks of proving systems are seeing significant advancements, so are the proving systems and its underlying proofs themselves. We have now progressed beyond the standard two types of proofs: (elliptic-curve-based) SNARKs and (hash-based) STARKs.

Some noteworthy innovations include:

1. **Binius**: This is a novel proof system designed specifically for **operations over binary fields (fields consisting of zeros and ones)** rather than larger prime fields. Traditional systems like SNARKs and STARKs require working over prime fields to maintain efficiency (mainly related to interpolation approaches), which can introduce inefficiencies due to the large size of field elements. Binius bypasses this limitation by focusing on binary fields, which are naturally smaller and align well with standard computer architecture, thus allowing for proofs that are faster and less resource-intensive.

   Binius leverages the inherent properties of binary computation - such as simplicity in arithmetic operations like addition, which can be performed using XOR operations without carry operations. This results in higher efficiency in both space and time compared to proofs over larger finite fields. Finally, and this is one of the biggest advantages of Binius, it exploits the structure of binary extension fields in order to embed naturally small elements (i.e., elements which are known to be small) in equally small memory places. This property is sometimes referred to as *zero embedding overhead*, and it is not compatible with prime fields.

2. **Rinocchio** [12]: Almost all proving systems used in practice today work over finite fields. Rinocchio is a modern proving system that extends SNARKs to support **computations over rings (such as integers modulo 2^{64}) instead**. Traditional SNARK constructions often depend on specific groups of matching order with secure bilinear maps, constraining them to operations over prime fields. Rinocchio, by enabling verifications over rings, overcomes these limitations, providing more flexibility for verifiable computation over encrypted data and for proving computations that align with real-life computer architectures. It is, however, unclear whether the additional overhead of Galois ring operations is sufficiently small to allow these proofs to be competitive.

The zkVerify protocol provides a robust platform that significantly aids in the advancement and adoption of innovative proving methods like Binius and Circle STARKs. By integrating these proofs into our protocol, verifiers on our network can efficiently validate the proofs generated by these new technologies. As a result, the **barrier to entry is lowered** for using these advanced cryptographic proofs, making it more accessible for developers to implement and leverage these technologies in real-world applications. Our protocol serves as a catalyst for innovation, fostering a faster uptake and broader implementation of cutting-edge proving systems across diverse sectors.

3.2 Innovating Rollups

3.2.1 Bitcoin zkRollups

There has been a recent surge in companies proposing zkRollups on Bitcoin (Citrea, BoB, and Arch). All of these solutions encounter the same issue: due to Bitcoin Script language limitations, it is comparatively expensive to directly verify a zkProof on-chain. Instead, the verification happens off-chain. The proof is then inscribed on Bitcoin, where a party could challenge the validity of such a proof by playing a rather complex, expensive, and limited on-chain fraud proof protocol (see Citrea with BitVM [13]).

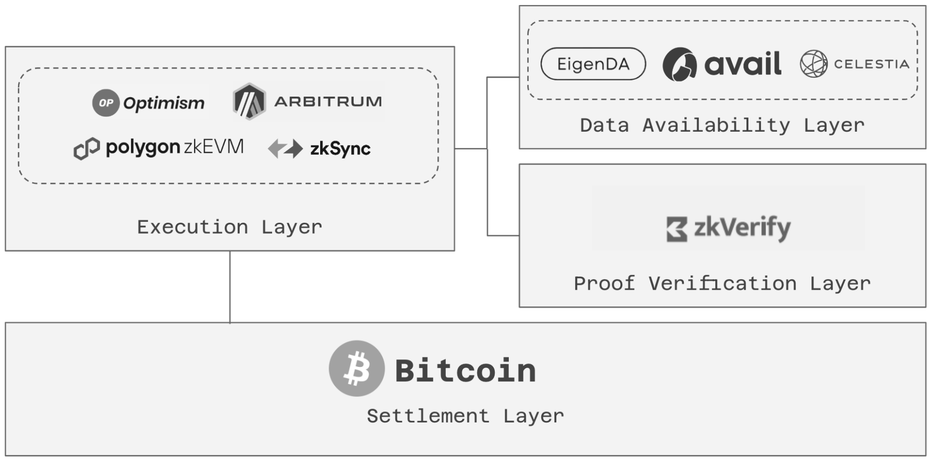


Figure 12: High-level architecture for Bitcoin zkRollups.

The zkVerify protocol is well-positioned to play a significant role in this context, acting as a decentralized verifier network for proofs coming from Bitcoin L2 (usually zk(E)VM proofs), thus providing additional guarantees over the optimistic assumptions.

3.2.2 Fast-Finality Optimistic Rollups

Optimistic rollups do not generate validity proofs, but instead post a transaction batch and its updated state root on-chain for public verification. A week-long window remains open to allow a party to challenge the validity of the batch by submitting a fraud proof. However, there is no guarantee that any party will in fact take on this responsibility. The fraud proof protocol must also be played on-chain, which can be expensive and difficult to implement. In many cases, its implementation is still pending, leaving a notable security vulnerability.

Optimistic rollups and their users stand to benefit from achieving faster finality and reducing reliance on fraud proofs. Certain companies, such as AltLayer, are already developing solutions based on this concept.

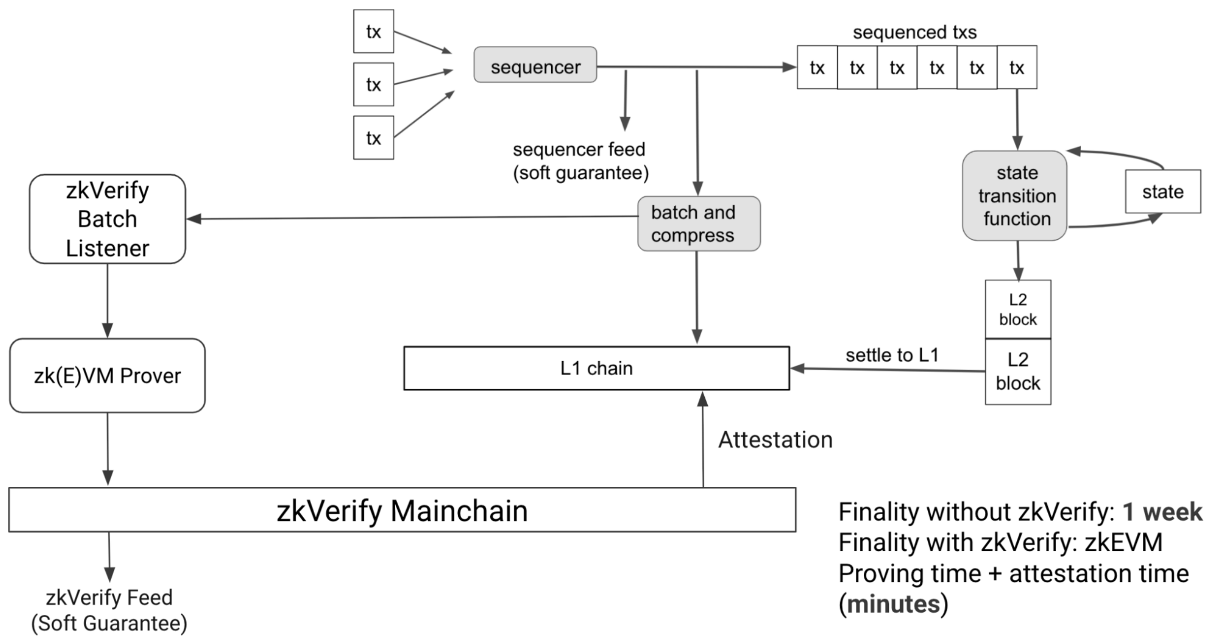

tx
tx
tx
sequencer
sequenced txs
tx tx tx tx tx tx
sequencer feed (soft guarantee)
state transition function
state
zkVerify Batch Listener
batch and compress
L2 block
L2 block
settle to L1
zk(E)VM Prover
L1 chain
Attestation
zkVerify Mainchain
Finality without zkVerify: **1 week**
Finality with zkVerify: zkEVM Proving time + attestation time (**minutes**)
zkVerify Feed (Soft Guarantee)

Figure 13: An example of integrating the zkVerify protocol with optimistic rollup.

In the same vein, the zkVerify protocol can intercept transaction batches as they are produced by rollup sequencers, create a validity proof of the batch itself, and finally submit it to zkVerify for verification. This validity proof can be:

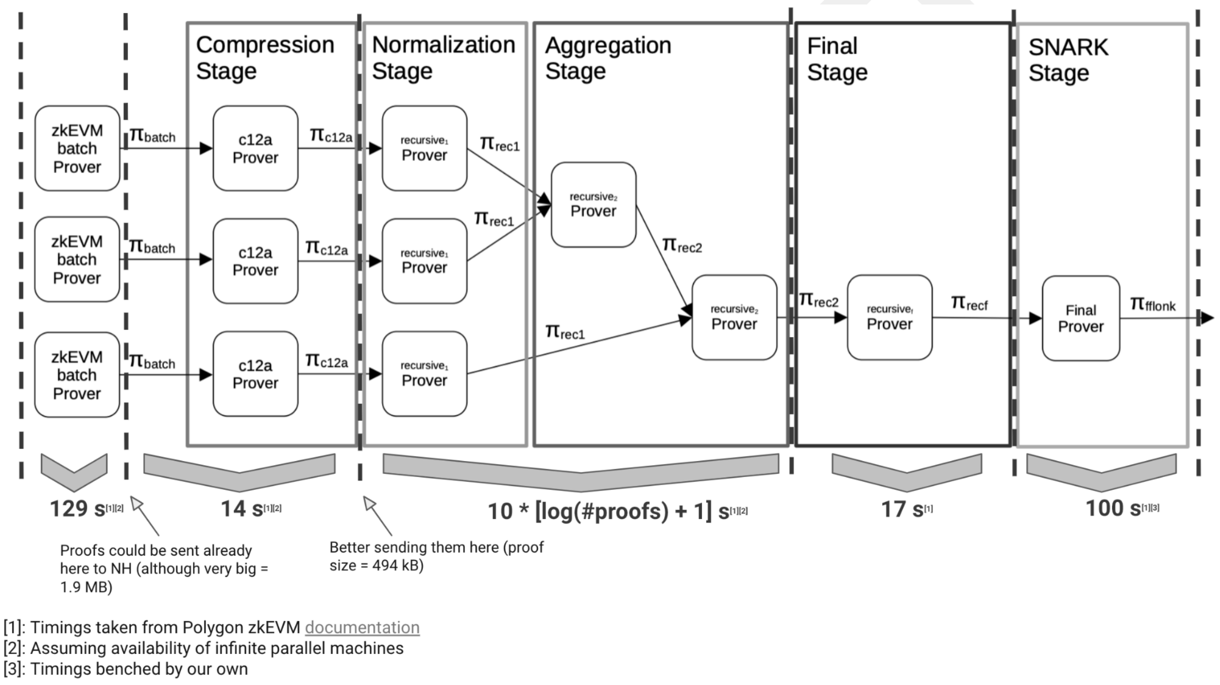
- A Merkle proof of correct state update.
- A proof of transactions validity and correct state updates, obtained by executing the transaction batch in a zk(E)VM.

Using this approach, finality decreases from one week to minutes. Many complementary approaches can also be built off the underlying premise of leveraging ZK proofs for optimistic rollups, including the idea of running the fraud proof protocol off-chain and producing a ZK proof of the correct state update once the faulty transaction has been found.

3.2.3 Skipping Aggregation & STARK-to-SNARK

zkRollups produce proofs of multiple transaction batches and aggregate all of them, via recursive proof composition, into a single one that gets posted on-chain. This is required in order to amortize the costs of the on-chain proof verification and data storage. This solution is effective, but it also has the following drawbacks:

- The time to finality increases dramatically with increasing recursion depth.
- Extra costs are sustained to account for the proving infrastructure.
- An expensive final transformation step (usually a STARK to SNARK conversion) is required to reduce the final on-chain costs.



Compression Stage
Normalization Stage
Aggregation Stage
Final Stage
SNARK Stage
zkEVM batch Prover
π_{batch}
c12a Prover
π_{c12a}
recursive₁ Prover
π_{rec1}
recursive₂ Prover
π_{rec2}
recursive_f Prover
π_{recf}
Final Prover
π_{fflonk}
129 s[1][2]
14 s[1][2]
10 * [log(#proofs) + 1] s[1][2]
17 s[1]
100 s[1][3]
Proofs could be sent already here to NH (although very big = 1.9 MB)
Better sending them here (proof size = 494 kB)
[1]: Timings taken from Polygon zkEVM documentation
[2]: Assuming availability of infinite parallel machines
[3]: Timings benched by our own

Figure 14: Polygon zkEVM proof aggregation architecture.

In order to overcome such disadvantages, zkRollups can capitalize on zkVerify's unique capabilities by:

- Directly verifying the proofs of the single transaction batches without the need to recursively aggregate them (up to some extent).
- Omitting the expensive final STARK-to-SNARK conversion.

Although cost savings and time-to-finality require careful balancing, the zkVerify protocol presents an improvement over traditional proof aggregation layers, providing users with greater freedom and authority to determine their optimal balance between cost, finality, and security.

3.3 Streamlining zkApps

3.3.1 Current Developer Landscape

Outside of the L2 scalability use cases, ZK proofs also play a pivotal role in **privacy preserving applications**. These applications include consumer apps (such as gaming and NFT's), identity, DeFi, voting, and more. They each have unique, purpose-built business logic, along with their own requirements for preserving digital privacy.

Just as several programming languages and frameworks exist for building traditional applications, the same is also true in the ZK applications landscape. zkApp developers may choose from a variety of tools, languages, and frameworks, each maintaining their own degrees of abstraction, complexity, generalization, and vendor lock-in. One developer may need fine grained control over their entire stack and reach for low-level libraries of ZK primitives, such as Arkworks. Another may already have some knowledge of ZK core concepts, and choose to encode their business logic into a custom circuit, leveraging a Domain-Specific Language (DSL), such as Circom or Noir. Yet another may not be concerned about optimizing proof size or generation time, and opt for a more general tool, such as a zkVM like RISC Zero. Going even further, one may choose a zkEVM or ZK-optimized EVM (such as zkSync or Manta), if they already have an existing Solidity-based application.

As we see, the ZK developer toolkit is only growing and maturing, creating more and more avenues for proof generation. As more proofs are being generated, those proofs will also need to be verified. zkVerify is well-positioned to meet this demand by means of a modular proof verification layer, integrating **easily into the software stack of any ZK application**.

3.3.2 Innovating on the Current Stack

A common pattern for end user ZK applications is for proofs to be generated client-side (either in the user's browser, local hardware, or even a secure cloud) and then submitted to a verifier smart contract. See simplified diagram below:

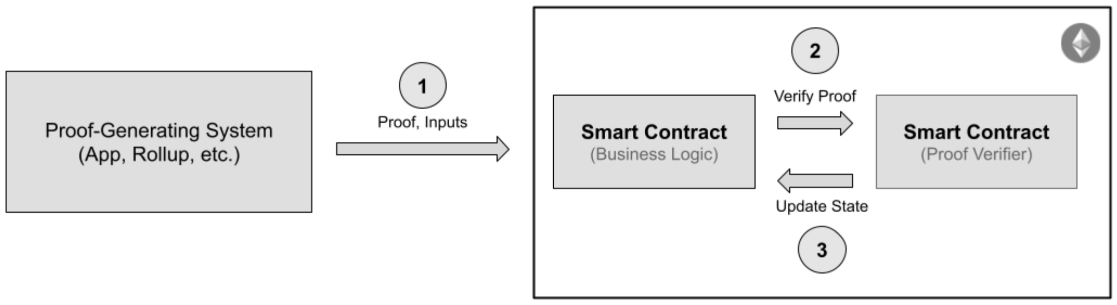


Figure 15: The general pattern of proof generation and verification for ZK applications.

This pattern has some advantages, namely the ability to co-locate application state transitions along with their associated proof verifications on-chain. The downside, though, is the increased cost associated with verifying a proof on Ethereum. Issues ranging from limited blockspace to optimization bottlenecks can cause applications to become cost prohibitive.

Similarly to how blockchain data storage solutions have taken shape recently (Data Availability Layers, Data Availability Committees (DACs)), there is a need to bring innovation and new solutions to the proof verification space. As a blockchain optimized for verifying proofs, zkVerify can solve the cost overhead issues faced by many applications today. By adding zkVerify to the application architecture, developers can optimize their proof verification cost, improving user experience, and allowing for production grade applications at scale.

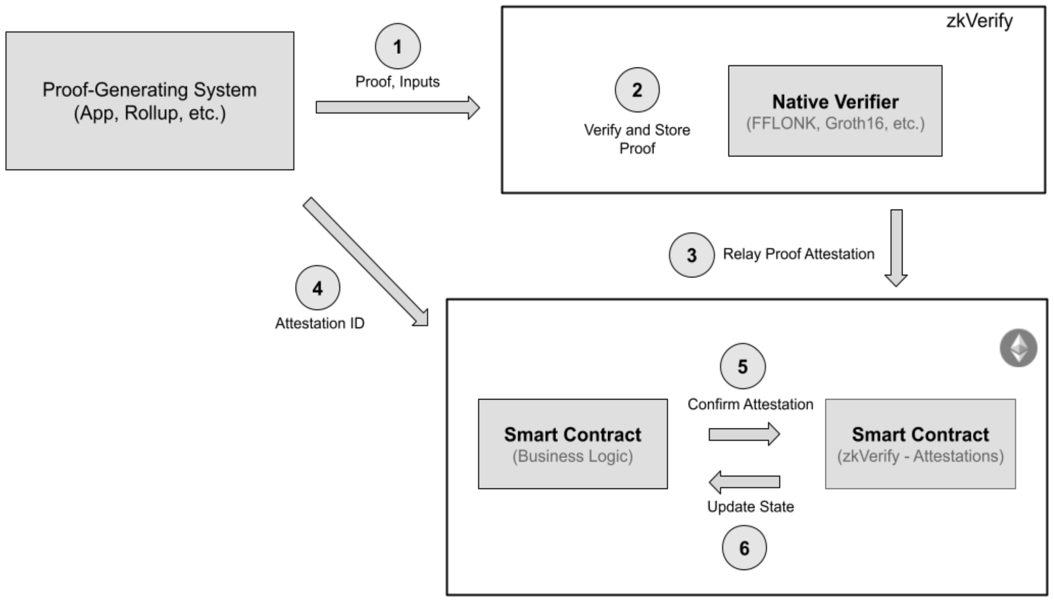


Figure 16: The workflow of a ZK application

with zkVerify for proof verification.

The figure above demonstrates one possible integration of zkVerify into a typical ZK application flow. We notice that the responsibility of verifying proofs has been delegated entirely to zkVerify, and only attestations (small hashes) are ever stored on the settlement chain.

Zk application developers have the opportunity to leverage different levels of finality in their application, which could even further improve the user experience. For example, instead of waiting for attestations to be written to the settlement chain, ZK applications can optimistically move forward on their application flow, while asynchronously awaiting future levels of confirmation.

4 Standardization for Proof Verification

The zkVerify protocol has the potential to **establish a standardized approach to proof verification** across various blockchain L1 platforms, bridging the gap between differing proof mechanisms and enhancing interoperability. By providing a unified framework for verifying zk-SNARKs, zk-STARKs, Binius, and any future proof innovations, our protocol ensures that all transaction batches, regardless of their source, adhere to a consistent verification process. This simplifies the process of assessing the validity of transactions, thereby fostering a more inclusive and streamlined ecosystem.

5 Conclusion

As ZK proof technology becomes increasingly accessible through advancements in both hardware and software, we anticipate a significant surge in demand for proof verification services. This demand will arise from diverse sectors seeking enhanced privacy and security measures, necessitating a robust infrastructure capable of supporting a wide array of cryptographic proof types. The zkVerify protocol is designed to meet this burgeoning demand, ensuring scalability, security, and better performance.

By supporting and efficiently combining new paradigms, including modern proving protocols like Binius and Circle STARKs, we are committed to driving the next frontier of innovation in ZK proof verification. In particular, this process represents a convincing next step towards further cost reduction and easier user interaction and adoption, expanding the frontier of verifiable computing and making it more accessible to blockchain and Web2 applications. Just as Layer 2 solutions have transformed Ethereum, our protocol aims to supplement and partner with Layer 1 platforms to accelerate their cryptographic roadmaps and bring even more breakthroughs to life.

6 References

[1] Protocol Labs. (2023). The Future of ZK Proofs. https://protocol.ai/protocol-labs-the-future-of-zk-proofs.pdf

[2] Internal Analysis, Horizen Labs, 2024. Projected annual security expenses from proof verification alone based on increased L2 market expansion and growing use of privacy-preserving zk applications.

[3] Internal Analysis, Horizen Labs, 2024. Cost of verifying a single proof based on on-chain data surveyed across various proof types.

[4] Worldcoin. (2023). Worldcoin Whitepaper. https://whitepaper.worldcoin.org/

[5] Internal Analysis, Horizen Labs, 2024. Cost savings projections derived from historical cost data, market trends in ZK proof verification costs, and underlying assumptions based on increased verification efficiency from optimized algorithms.

[6] B. E. Diamond and J. Posen, "Succinct arguments over towers of binary fields," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2023/1784.

[7] E. Ben-Sasson, D. Carmon, S. Kopparty, and D. Levit, "Elliptic curve fast fourier transform (ECFFT) part I: Fast polynomial algorithms over all finite fields," arXiv.org, https://arxiv.org/abs/2107.08473.

[8] U. Haböck, D. Levit, and S. Papini, "Circle Starks," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2024/278.

[9] L. Grassi, D. Khovratovich, and M. Schofnegger, "Poseidon2: A faster version of the poseidon hash function," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2023/323.

[10] L. Grassi et al., "Monolith: Circuit-friendly hash functions with new nonlinear layers for fast and constant-time implementations," IACR Cryptology ePrint Archive, https://eprint.iacr.org/2023/1025/.

[11] Ashur, M. Mahzoun, J. Posen, and D. Šijačić, "Vision mark-32: ZK-friendly hash function over binary tower fields," IACR Cryptology ePrint Archive,

https://eprint.iacr.org/2024/633.

[12] C. Ganesh, A. Nitulescu, and E. Soria-Vazquez, Rinocchio: Snarks for ring arithmetic, https://eprint.iacr.org/2021/322.pdf.

[13] R. Linus, “Compute Anything on bitcoin,” BitVM, https://bitvm.org/bitvm.pdf.