Intro

Introducing Placeholder, the proof system empowering =nil; Foundation products to perform secure interactions while being highly configurable to enable a variety of use cases. Key features include:

1. Nova over PLONK: Built-in folding scheme for Incrementally Verifiable Computations (IVC).
2. Circuit Density: Flexible gates.
3. Modularity: Configurable lookup argument, curve/field, proof size, and commitment scheme.
4. Verification: EVM specifics.
5. Recursion support.

Read on to explore the technical specification of the system and the reasons behind its design.

Why we've built our own proof system

Our roadmap and vision drive us to think of accommodating different applications on top of =nil; infrastructure – like Oracles, Bridges, Rollups, and other use cases we want to enable with zero-knowledge functionality while lowering the entry point for zk application developers.

One way to achieve that goal is to work with several proof systems, each suitable for one use case more than others. However, we took a different approach with Placeholder: designing a proof system with configuration possibilities so high that one wouldn't need different proof systems to build upon. In simple terms, if you build a zkBridge based on Placeholder, and market conditions change, requiring different security assumptions, you wouldn't need to change the proof system it's based on. Instead, you can achieve higher efficiency with Placeholder, regardless of the case.

This was the initial goal we kept in mind when designing the Placeholder architecture in 2021, in collaboration with Ethereum Foundation, Mina Foundation, and Solana Foundation. The same logic applies to all =nil; products. For example, creating our own configurable Crypto3 library under Placeholder is also not the easiest path, but in the long run, we believe having your own products at the core of your infrastructure is worth the effort.

What exactly is Placeholder?

Placeholder is a zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) based on PlonK-style arithmetization. Placeholder's strength lies in its modular design. Its internal components, such as commitment schemes, can be easily replaced and configured according to specific needs. Low-level Placeholder circuits can adapt to selected parameters, such as table size, gate degree, and lookup options. These qualities enable flexible configuration of Placeholder, providing trade-offs between circuit parameters, trust assumptions, and efficiency of proof generation. Due to this flexibility, Placeholder can accommodate particular cases while consistently achieving efficient results.

Background

Zero-knowledge proof systems are cryptographic protocols that enable a prover to demonstrate knowledge of a specific statement without revealing any information about the statement itself. SNARKs are a type of zero-knowledge proof that offers succinctness and non-interactivity. The term "succinct" refers to the property of generating very short proofs, which are much smaller than the original statement or computation. "Non-interactive" means that the prover can generate the proof without additional interaction with the verifier.

SNARKs have gained significant attention due to their potential applications in privacy-preserving technologies like decentralized systems. For example, SNARKs provide a way to prove the integrity of computations or programs without revealing the underlying data.

Circuit density: Flexible gates

Placeholder offers the flexibility to customize compilation strategies. Compilation refers to the process of transforming high-level code into a circuit representation. Customization in this area allows for optimizations such as circuit size reduction, improved gate count, or tailored compilation strategies based on the specific computational problem at hand.

To enable the implementation of customized compilation strategies, we use a Flexible gates technique. To facilitate this, our arithmetization procedure is backed by "flexible" circuits — they adapt their internal representation based on the number of available witness columns, lookup columns, and other proof system parameters. You are not 'locked into' any specific parameter set and can change them to make your app run better.

We possess a resultant circuit established through a variable base scalar multiplication employing n columns, 2 sha2-256 computations, each employing m columns, and 12 field additions, each employing k columns, with the condition that $k < m < n$.

The first picture shows the circuit density when utilizing the straightforward compiler approach. This approach results in a significant number of unoccupied cells within the table, consequently leading to extended durations for both the prover phase and increased verification expenses.

Alternatively, a more condensed arrangement of primitive circuits can be achieved, yielding an improved representation (depicted in the second picture). However, this advantageous layout comes at the cost of a larger overall gate count, which subsequently impacts the prover phase duration.

Lastly, the third picture employs the flexible gate technique to demonstrate a method by which we can eliminate any vacant cells while conserving the identical total gate count.

Verification: EVM Specifics

In addition, Placeholder is designed to produce the most dense circuits along with commitment scheme and proof size optimizations, which increases the efficiency of Ethereum Virtual Machine (EVM) verification by reducing the amount of logic necessary to verify a single proof (in comparison to other PLONK-based proof systems). This enables developers to incorporate zero-knowledge proof functionality directly into their EVM-based applications even if computations being proven are extremely large (e.g. billions of scalar multiplications over Pallas/Vesta curves) by smoothening the verification cost curve.

By providing the ability to replace or customize these components, along with an EVM-compatible Verifier, Placeholder empowers users to adapt the proof system to their specific requirements, optimizing performance, security, or efficiency based on the particular application domain or use case.

Modularity: Configurable Curve/Field, Lookup Argument and Commitment Scheme

In Placeholder, several components can be replaceable to enhance flexibility and accommodate specific requirements. Here are some key components that can be replaced or customized:

- Lookup Technique

Placeholder allows for the use of custom lookup techniques. Different lookup techniques can be employed based on the specific application or optimization requirements. The choice of lookup technique can impact the proof generation time and verification cost.

- Commitment Scheme

The commitment scheme used in Placeholder can be replaced or customized. Different commitment schemes, such as FRI or KZG commitments, can be utilized based on the desired security guarantees, proof size, or efficiency requirements.

- Proof Size Optimization Techniques

Proof size optimization is another aspect that can be customized in Placeholder. Different techniques can be employed to reduce the size of the generated proof, such as applying PoW for busting security, utilizing batching techniques, or employing specific representations. Customization in this area can lead to more efficient proof transmission and storage.

- Swappable Curve/Field.

Placeholder’s arithmetization provides a way to use curves/fields of different sizes to fit the underlying field size to a particular circuit.

Nova over PLONK: Built-in folding scheme for Incrementally Verifiable Computations (IVC)

We are currently working on increasing the flexibility of Placeholder and incorporating new features to enhance its capabilities.

One of these big features is Incrementally-verifiable computation (IVC). The overall proof generation time can be significantly reduced by efficiently handling repetitive computations. But existing IVC approaches may show less favorable results in terms of parallelization compared to many Placeholder provers and the following recursion. The collaborative approach of Placeholder and IVC may lead to innovative techniques and optimizations that address the challenges associated with parallelization and recursion while maximizing the benefits of IVC.

Security and trust

Placeholder can be initialized with different cryptographic primitives. As an option, Placeholder can use an FRI-based polynomial commitment scheme. This makes it possible not to use the trusted setup and to obtain a security justification in the RO model, assuming the cryptographically secure hash function is used. In other words, there is no need to trust the initial distribution of randomness nor to make non-standard cryptographic assumptions.

Alternatively, the Placeholder can be used with a KZG polynomial commitment scheme. Thus, we get a constant-size proof using the DLOG-based assumption and the universal updatable setup.

Why build on Placeholder

Utilizing Placeholder for diverse use cases offers several advantages due to the modularity of the system. First of all, it is reusability; Placeholder allows easy reapplication of various components. The system's inherent flexibility empowers developers to leverage existing functionality like Trusted Setup Variations, Prover Time vs. Verification Cost Trade Off methods, and proof size optimization techniques.

Thus, Placeholder can be adapted to:

- Facilitate secure and efficient interoperability between decentralized networks.
- Create secure and verifiable oracles.
- Implement Layer 2 scaling solutions.
- Prove the correctness of virtual state machines, enabling trustless and verifiable execution of complex computations.

Placeholder has been implemented successfully and is being utilized in real-world applications.

- Leveraging the Flexible Gates technique with Placeholder can significantly enhance the performance of result circuits generated by zkLLVM. zkLLVM compiles high-level programming languages into inputs for provable computation protocols. Optimized circuit designs enable faster execution and more efficient proof generation for computations performed within zkLLVM.
- The Proof Market is an application that utilizes Placeholder as one of its available proof systems. It functions as a marketplace where users can request zero-knowledge proofs for specific computations or statements. A network of specialized producers responds to these requests, providing users with the desired proofs. Having Placeholder implemented at Proof Market enables faster development of zk-powered applications because developers can get proofs without setting up costly infrastructure.

Performance

The time taken to generate a proof for a particular calculation is highly dependent on system parameters like arithmetization, lookup tables, and the chosen polynomial commitment scheme. The default FRI-based commitment scheme allows a quasi-linear running time and a polylog proof size. In this case, the prover works with size N polynomials. If gates of large degree (d) are utilized, the most costly operation for the prover will be to perform the FFT for polynomials of size $O(d\cdot N)$. Also, the use of the Reed-Solomon code with rate $\rho=1/2^R$ requires to perform the FFT for $O(2^{R}\cdot N)$-size polynomials. For clarity, these dependencies, taking into account 17 bits of grinding, are shown in the following images.

Conclusion

Summarizing what has been said, the Placeholder is an efficient and highly customizable proof system with broad applicability. Using interchangeable modern cryptographic mechanisms based on different cryptographic assumptions allows one to satisfy different security requirements depending on the context. Moreover, the ability to customize parameters allows for fine-tuned trade-offs between different performance aspects, including the prover's execution time and proof size. All this establishes Placeholder as a convenient tool for a wide range of tasks, for building proof compositions, and in separate independent applications.

- Placeholder research paper
- Implementation
- Discord
- Telegram
- Twitter