Category: “Brief proofs of critical computations in blockchain applications using NEAR as an example” inside the zk-Circuits Track

Introduction. We encourage community members to work with us in programming recursive proof systems for Near Protocol using Plonky2 and Rust frameworks. The goal is to design different aggregation and recursive circuits optimized for several practical cases using Near Protocol primitives and data. It is highly in demand on the market tradeoffs between size and proof generation speed, as well as on-chain verification.

Background. Zpoken has experience in programming recursive proof systems for distributed blockchain ledgers using the Plonky 2 protocol and Rust frameworks. This knowledge and experience is useful for popularizing this direction among the students during hackathons and other competitive forms of education. We offer a series of simple tasks to learn the basics of the Rust programming language, the Plonky 2 protocol and recursive proof systems. As the tasks progress, they become more complicated and supplemented with new scenarios, which, together with the competitive form of training, stimulates the cognitive functions of students.

Brief overview of the task. We offer simple tasks, which can be divided into three groups:

1. **Rust programming basics (optional).** The goal is to get acquainted with the principles of programming in the Rust language, in particular: the type system; memory management; syntax; object system; parallelism, etc. Students are offered to perform the simplest tasks related to sorting and searching for elements in arrays; recursive algorithms; combinatorics, etc. Upon successful completion, students will also implement the simplest algorithms of number theory: Euclid's algorithm; solution of the system of comparisons; modular arithmetic; basics of cryptography.
2. **Simple zkSNARKs in Rust.** The goal is to get familiar with the Rust framework when programming the simplest zkSNARKs: generating circuits, creating public settings for provers and verifiers; proof generation; proof verification. The task is to write the protocol of interaction between the verifier and the prover, describe how the data transfers between them. Students are proposed to solve the task of proving the computational integrity of hashing and digital signature algorithms. The competitive elements are the proof generation with the highest verification speed or with the smallest proof size.
3. **Aggregation and recursive zkSNARKs in Rust.** The goal is to deepen your understanding of zkSNARKs: aggregation of multiple proofs; recursive proofs; optimization by speed/size criteria. Students are asked to complete the task of aggregating multiple zkSNARKs (multiple hashes, multiple signatures; multiple hashes and signatures combined together), as well as the task of recursively calling proofs within other proofs. The complex tasks are to optimize proof generation and reach a high speed of verification and acceptable proof size at the same time.

Each group of tasks will be expanded by including crypto primitives from the NEAR protocol.

Named category structure. Our subsection contains a series of short presentations on the principles of building the NEAR blockchain project, the basics of programming in the Rust language, the Plonky 2 framework in the Rust language, as well as recursive zkSNARKs and aggregation schemes for solving NEAR scaling problems and building bridges with other large blockchain projects. Each presentation ends with a list of checklists and small assignments to complete on your own.
In addition, we offer several practice-oriented competition problems for groups of students on the most important problems related to the use of recursive zkSNARKs to prove critical computations in blockchain applications using the example of NEAR. These tasks are the continuation of presentation materials and tasks for self-control, they include more complex scenarios deployed on a larger number of input parameters and close to real processes using the example of NEAR.

Presentation plan:

1. Presentation based on the NEAR blockchain. The goal is to give a general idea of building a decentralized ledger with a brief description of the main architectural components: epochs; shards; producers and validators; generation and confirmations; used cryptography; the current state of the project and the prospects for its development. Students should have a basic understanding of how the system works and see the main problems in its further development – scaling the system and building bridges with other large blockchain projects;
2. Presentation on the Rust programming language. The goal is to give a general idea of the Rust programming language (type system; memory management; syntax; object system; concurrency, etc.), explain why this language is good for programming zkSnarks (memory ownership provides additional protection). Students should have a basic understanding of the basics of Rust programming, should be able to create small projects to solve simple problems;
3. Presentation on the Rust framework with Plonky 2. The goal is to give a brief description of the main elements of the framework: what crypto primitives are implemented in it and how they can be used; some examples of the simplest zkSnarks. Students should understand that the basic “building blocks” have already been created for designing complex proof systems to solve many of the practical problems from the first presentation;
4. Presentation on recursive zkSnarks schemes for NEAR. The goal is to provide a general introduction to building recursive proofs in Rust with Plonky 2; explain how to aggregate proofs and how this aggregation can be optimized; give some examples of aggregated proofs and show how they can be modified and optimized for speed/size. Students should be able to use these examples and be prepared to design their own recursive proofs for NEAR.

Rewards:

1. Team (or individual) who solved the practical task and submitted presentation first wins the prize of $10,000.