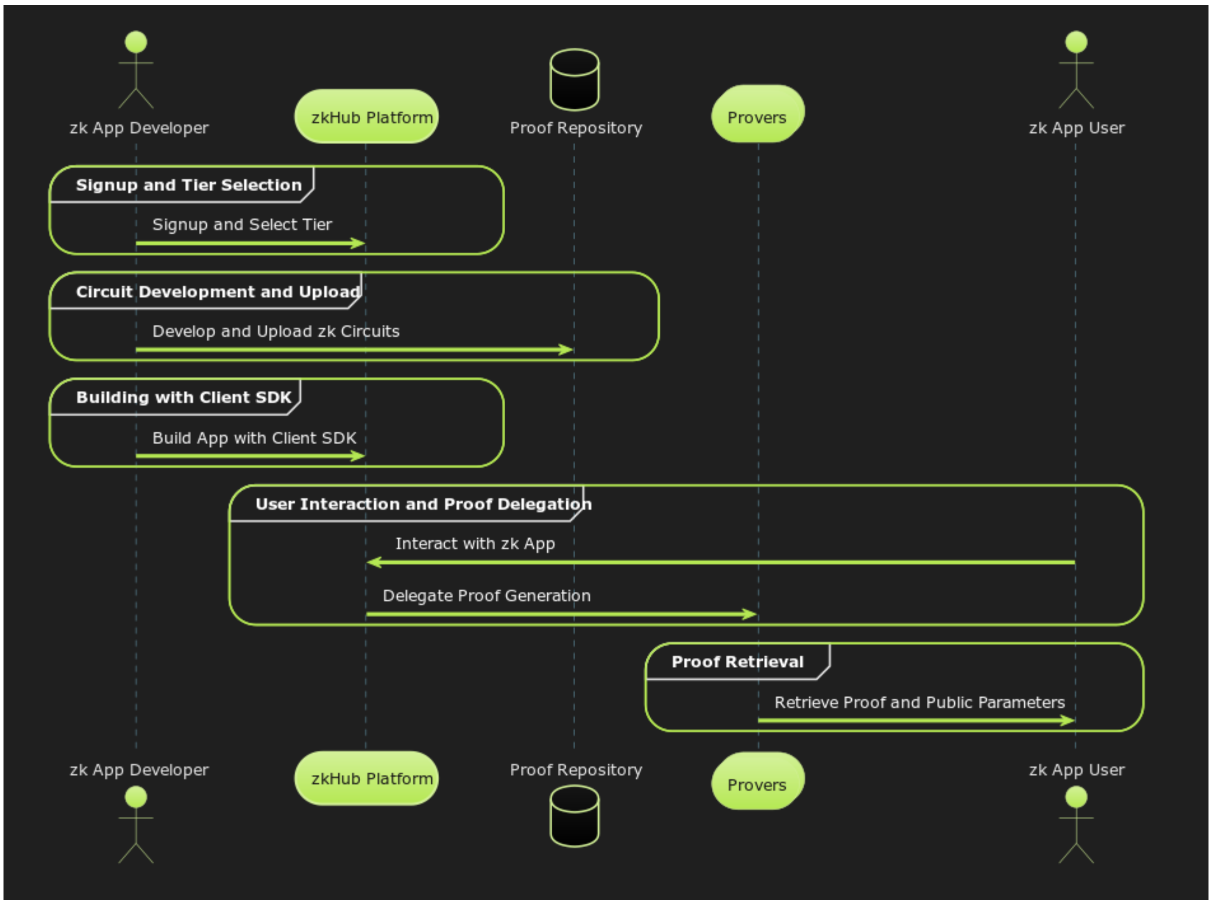
Product Design Document for zkHub

Our product comprises four essential components that combine to facilitate the generation of proofs at an unprecedented speed. These components include the Client SDK, Developer SDK, Node Runner Package, and Backend SDK.

User Journey Workflow



1. **Signup and Tier Selection**: zk App Developers start their journey by signing up onto the zkHub platform. They select their preferred tier and price structure for proof generation based on their circuit size, allowed latency, and Quality of Service (QoS). zkHub also offers a free usage tier as a trial for developers.

2. **Circuit Development and Upload**: Developers then design their zk circuits and upload the compiled circuits to the zkHub Proof Repository. The circuits are uploaded in an intermediate representation format like R1CS (Rank 1 Constraint System), which is a common format for representing zk circuits.

3. **Building with Client SDK**: Developers build their applications using the zkHub Client SDK. The SDK, written in WebAssembly (WASM), supports Multi-party Computation (MPC) based proof delegation. It efficiently creates an MPC-based secret shared witness, which is then sent to the zkHub proof generation platform.

4. **User Interaction and Proof Delegation**: Users of the zk app interact with the application and delegate the proof generation to the zkHub platform. This delegation allows for faster proof generation and supports massive circuits. There are two configurations:
    - one where the proof delegator (end user's device) needs to be online for some interactions with the zkHub platform.
    - another where everything happens completely on the zkHub platform with some initial preprocessing on the user's device.

5. **Proof Retrieval**: The user retrieves the generated proof along with the public parameters, without leaking any secret witness data. This proof and parameters can be stored on the zkHub platform for further reuse if needed. This completes the user's journey, providing them with a secure and efficient solution for generating and managing zk proofs.

SDKs/Libraries at various levels

Developer SDK

The Developer SDK assists developers in testing and deploying zk-circuits.
Current methods of testing are slow, and limited by the computational capacities of the developers. zkHub aims to enhance the developer experience by shifting the burden of testing from user machines to the cloud, resulting in accelerated proof testing and allowing developers to keep their machines idle during this process.

Additionally, the Developer SDK will incorporate user-friendly tools that are crucial for deploying these circuits to the zkHub network.
It will offer multiple tiers catering to both individuals and businesses, each with distinct variations in limits, proving times, and analytics.

On deployment of a proving circuit, the Developer SDK will store the circuit specifications in the **circuits repository.** Developers also get to choose privacy and speed requirements for their applications.

At launch, we have plans to provide support for circuits written in arkworks and Circom.

Circuits Repository

The Proof Circuit Repository is a vital component of the zkHub platform, storing zk-circuit specifications in a centralized location. When developers deploy a zk-circuit via the Developer SDK, the specifications, usually in R1CS format, are securely stored in the repository. This facilitates efficient retrieval during proof generation, adhering to developers' privacy and speed requirements.

Additionally, the repository promotes optimization by enabling circuit reuse, thereby enhancing the overall efficiency and scalability of the zkHub network.

Client Side SDK

By leveraging the Client SDK, developers can seamlessly integrate their proving circuits into their applications (including dapps, zkML apps, games, etc.) after publishing the circuit specifications in the proof repository.

When a proof is to be generated by the end user, the Client SDK pings the proof repository to fetch the details of the circuit.

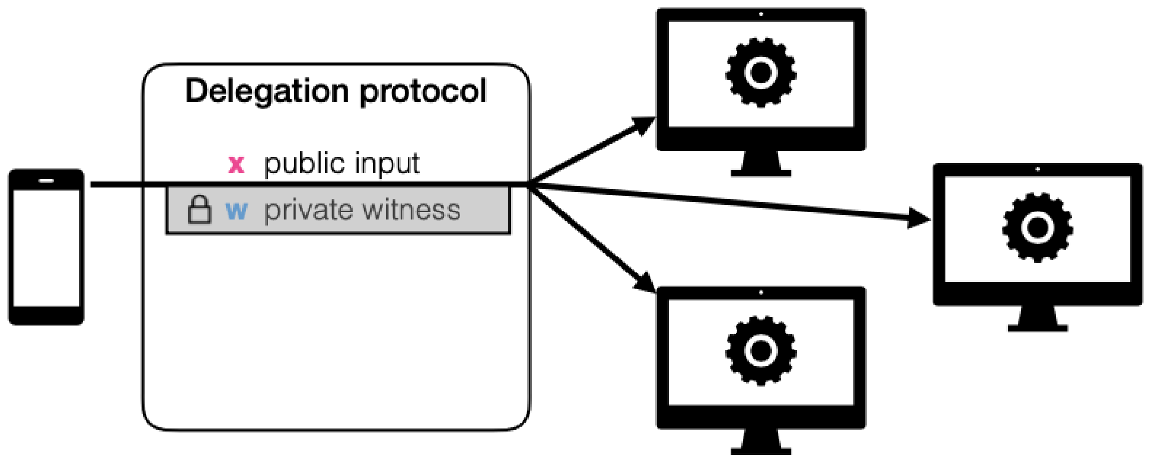
It then pings the zkHub network to fetch information about available machines for proof generation.

The next step is the initialisation of the MPC protocol.
MPC works on a representation of any computation in terms of addition and multiplication operations. This is because these two operations form the basis of arithmetic circuits, which are a common way to represent computations in the context of secure computation.

At zkHub, we use the SPDZ protocol for MPC for its security and performance. The SPDZ protocol, however, has some overheads that can affect its performance. These overheads primarily arise from the need for expensive public-key cryptography to generate randomness, which is required for computing multiplication operations.

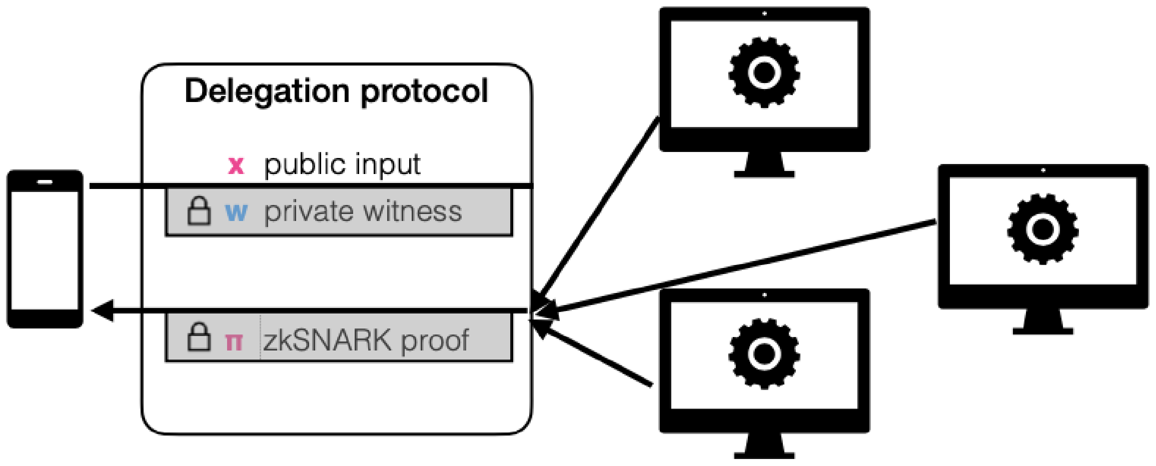
In the case of zkHub, this overhead can be significantly reduced because we can always assume that the client (delegator) is honest and can be trusted by all other parties(the nodes). Hence, the randomness can be generated on the client side without the need of expensive public-key cryptography leading to a more efficient and practical MPC protocol.

The randomness is a special triplet of numbers called Beaver Multiples. It is a triplet of 3 numbers which satisfy the equation (a*b = c).



After the correlated randomness is generated by the delegator, it is shared among the participating nodes. Each node receives a portion of the randomness, specifically, a share of each value in the Beaver triple (a, b, c). However, the actual values of a, b, and c remain unknown to each individual node. They only have access to their respective shares, not the complete values. This ensures the privacy of the computation while still enabling the secure execution of the protocol.

Once the proof is generated by the nodes, the Client SDK verifies the correctness before accepting it.

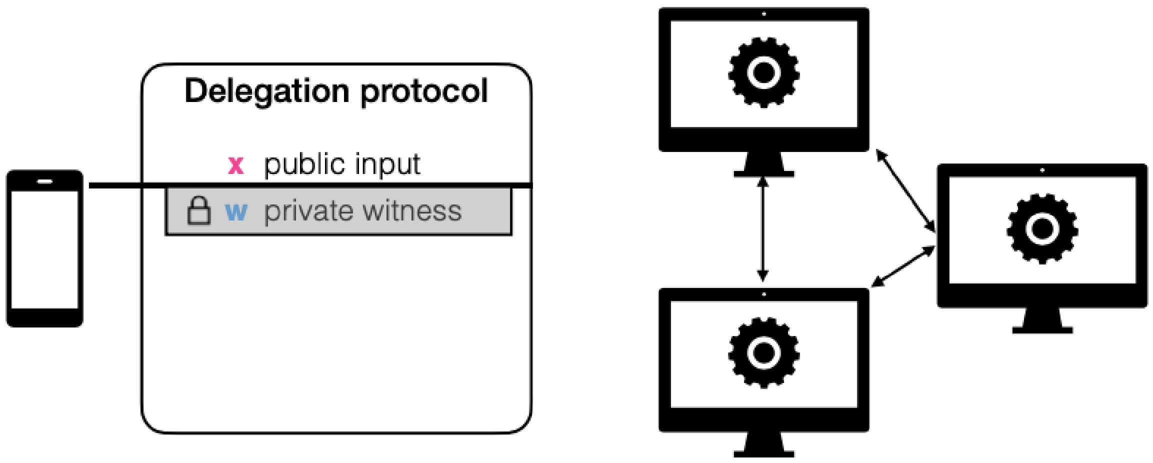


The zkHub Client SDK, written in WebAssembly (WASM). WASM is chosen for its high performance and portability, allowing the SDK to run in various environments, including web browsers and standalone virtual machines.

Node Runner Package

The Node Runner package is a runtime software installed on the hardware that is to be plugged into the zkHub network. The node runner package is responsible for the following -

- Running benchmarking on hardware to determine capabilities.
- Accepting requests from the client and establishing a connection with the other nodes to begin generating distributed proofs.
  There are multiple inexpensive checks performed during the generation of the proof to ensure that each node is generating the proof honestly.
- The proof generators running on the nodes are optimized for MPC



Healthy nodes receive a better rating. (Higher rated nodes generate proofs for organizations and enterprises)

Backend SDK

The Backend SDK manages the overall operations of the zkHub platform:

- Handles transaction processing, consensus mechanism, and interaction with the proof generation network.
- Provides APIs and tools to the Client SDK for efficient management and interaction of applications.
- Features for storing compiled circuits for reuse and providing analytics and alerts for system monitoring.

Optimizations

Crypto

- No heavyweight malicious security techniques
- Avoiding MPC for the witness independent part of the SNARK
- Multiplication triple generation at the delegator
- Novel security-efficiency tradeoffs

System

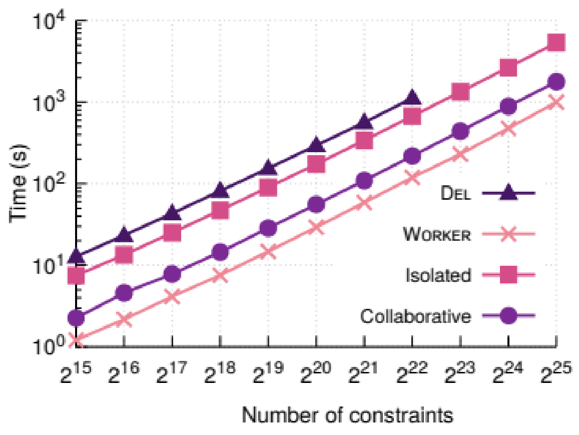
- Better parallelization for high-core machines
- Eager memory reclamation

Future plans

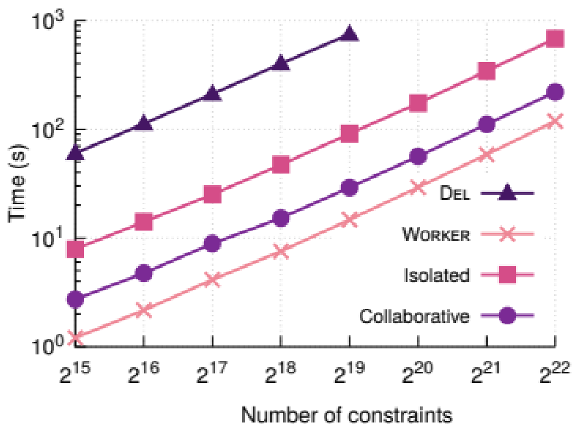
- Using the techniques of the DIZK paper to speed up each node with distributed computing techniques with Apache Spark
- Support STARK proof systems with the techniques suggested in the collaborative proofs paper by Alex Ozdemir et. al.

Benchmarks

When compared to local proving, using the Eos cryptographic collaborative proof primitive to delegate proof generation from a recent smartphone (a) reduces end-to-end latency by up to **30×**, (b) lowers the delegator's active computation time by up to **1500×**, and (c) enables proving up to **250×** larger instances. These numbers are similar in the case of a high-config consumer laptop too (32GB of RAM and 4 cores of an Intel Xeon E5-2686 CPU at 2.3GHz).



(b) LAPTOP



(c) MOBILE

Testing Tooling in the Platform (medium to long-term goal)

Formal Verification

This module will incorporate formal methods to verify the correctness of the zk-proofs. It will use mathematical logic to check whether the proof satisfies the properties of completeness, soundness, and zero-knowledge. The formal verification process will be automated to provide quick feedback to the developers.

Static Analysis

This component will provide tools for developers to test their zk-proofs. It will include unit testing tools to test individual components of the proof, as well as integration testing tools to test the proof as a whole. The testing tools will also provide coverage metrics to help developers identify untested parts of their proofs.

User Interface

The platform will provide a user-friendly interface for developers to interact with formal verification and testing modules. The interface will provide visual feedback on the verification and testing results, making it easy for developers to identify and fix issues.

Mechanism Design

In the future, we plan on exploring mechanism designs focusing on creating an efficient marketplace for zk-proof generation. Some ideas could be-

- Use a sealed-bid auction system where customers submit their statement and a budget for the proof, and provers submit their bids with the proof and a price.
- Prevent centralization of the market by using a rotating set of provers and a reputation system to prioritize bids from high-rated provers.
- Ensure fair compensation for provers through a smart contract system that releases payment once the proof is verified as correct and delivered to the customer.
- Balance efficiency and privacy, ensuring no single party has access to the complete data, and preventing other kinds of denial-of-service attacks.