# Slushie: a zero-knowledge proof-based token mixer in Substrate

Oleksandr Mykhailenko, Kyrylo Gorokhovskyi

# Subject of the study

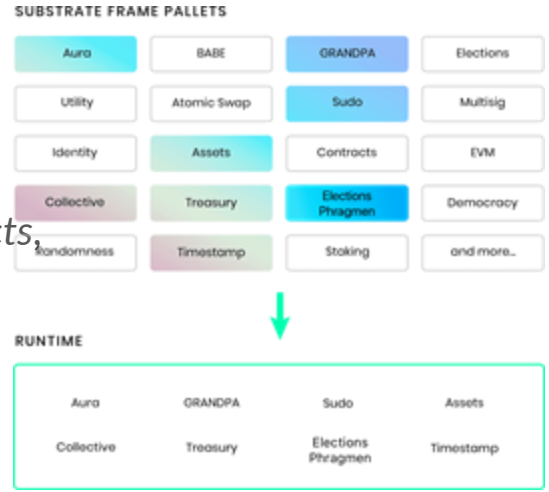
We research the possibility of implementing a working zero-knowledge proof-based cryptocurrency mixer for Substrate blockchains.

This includes creating a smart contract, a circuit for the quadratic arithmetic program to be used in the zk-SNARK, a CLI for mimicking front-end, and a relayer server.

# Relevance of the study

Substrate is a popular framework for creating blockchains from modular components. One of its components is *pallet-contracts*, a module for executing Wasm-based smart contracts.

Some decentralized networks, like Astar, already include *pallet-contracts* in their runtime, meaning anyone can deploy and use a smart contract.



# Relevance of the study

Apart from potentially providing users with more privacy by using Slushie for token transfers, we explore the usage of the Plonk ZK proof system and its capabilities to produce verification functions efficient enough for usage in a smart contract. To add to that, we circumvent the risk of errors in client libraries when producing/verifying proofs by reusing exactly the same proving/verifying code for both the CLI and the smart contract.

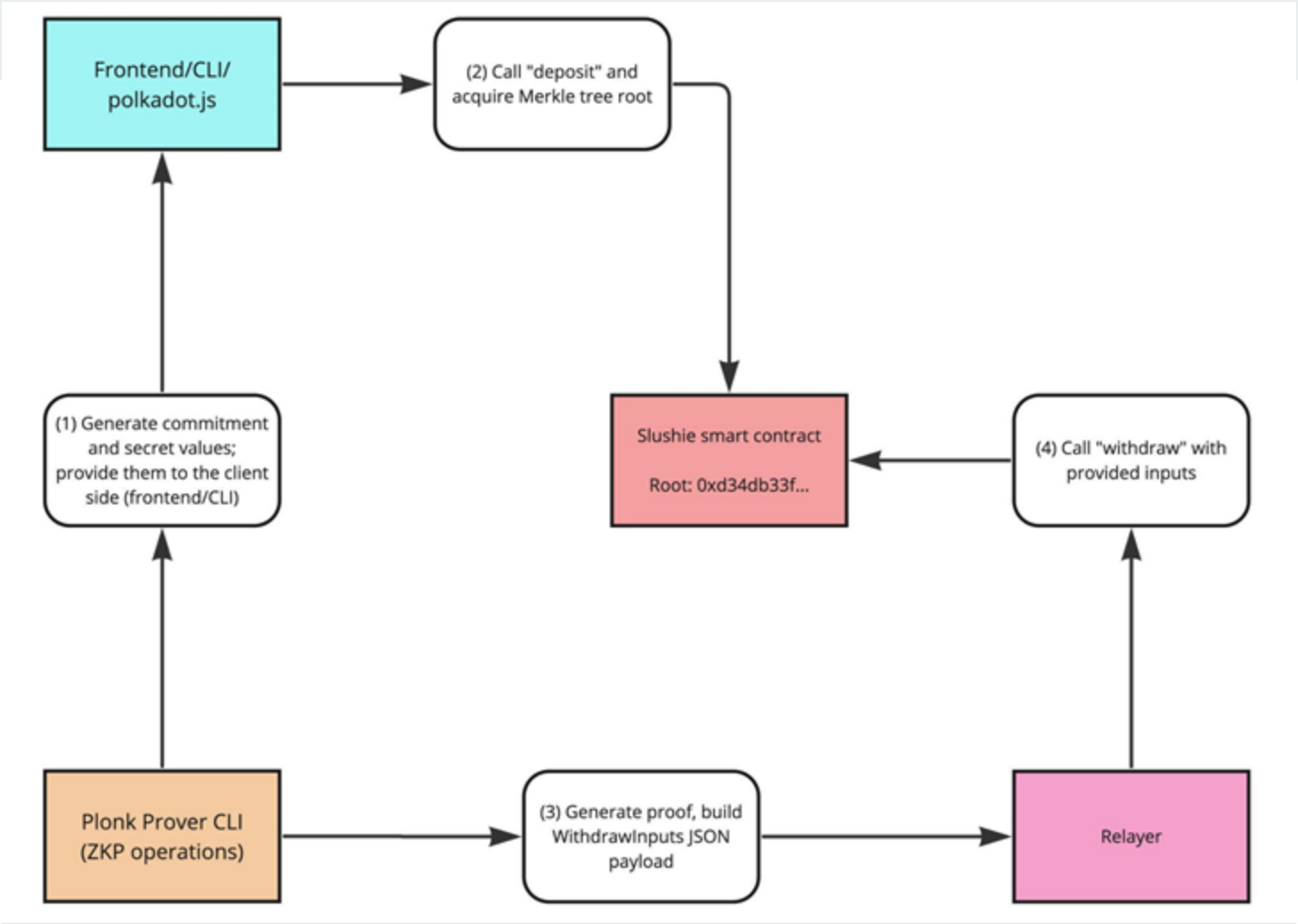
# Main goals

- Research the possibility to create a WebAssembly-based cryptocurrency mixer for Substrate-based blockchains using Plonk zkproof system;
- Improve client-side code integrity for producing and generating zero-knowledge proofs;
- Evaluate the ecosystem readiness for a production deployment of the mixer.

# Architecture

App architecture consists of four main parts:

- **Slushie** smart contract
- A library for operations with zero-knowledge proofs, called **plonk_prover**;
- CLI that exposes main functions of **plonk_prover**, called **plonk_prover_tool**;
- A relayer server to send **withdraw** transactions without any RPC provider tracing.


Frontend/CLI/
polkadot.js
(2) Call "deposit" and
acquire Merkle tree root
(1) Generate commitment
and secret values;
provide them to the client
side (frontend/CLI)
Slushie smart contract
Root: 0xd34db33f...
(4) Call "withdraw" with
provided inputs
Plonk Prover CLI
(ZKP operations)
(3) Generate proof, build
WithdrawInputs JSON
payload
Relayer

# Implementation

Technical stack:

- Rust was used to create the zk-SNARK circuit, utilities for operations on zk-proofs, and the smart contract (using the **ink!** e-DSL);

- Typescript was used to build the relayer server;

- An implementation of Plonk zk-proof system by Dusk Network B.V. was used for construction and verification of zero-knowledge proofs, as well as Poseidon (Grassi et al.) as the hash function, although Slushie also supports Blake2 hashing.

# The workflow

The workflow with Slushie consists of the following parts:

1. Start a node with *pallet-contracts* included
2. Deploy Slushie with some custom deposit amount
3. Generate commitment, secret values, acquire the root
4. Call **deposit** method
5. Generate proof with desired parameters
6. Launch the relayer
7. Call **withdraw** method using the relayer, acquire funds on a target account

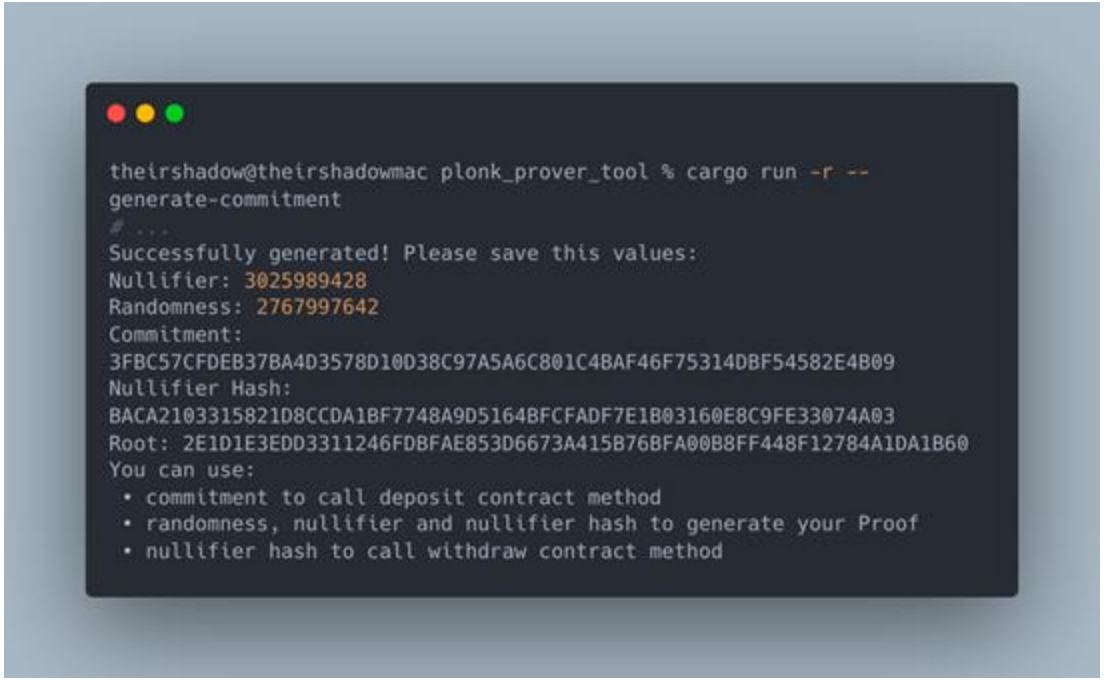
# Starting the node

```
// code omitted
impl pallet_contracts::Config for Runtime {
    type Time = Timestamp;
    type Randomness = RandomnessCollectiveFlip;
    type Currency = Balances;
  // code omitted
}
// code omitted
```

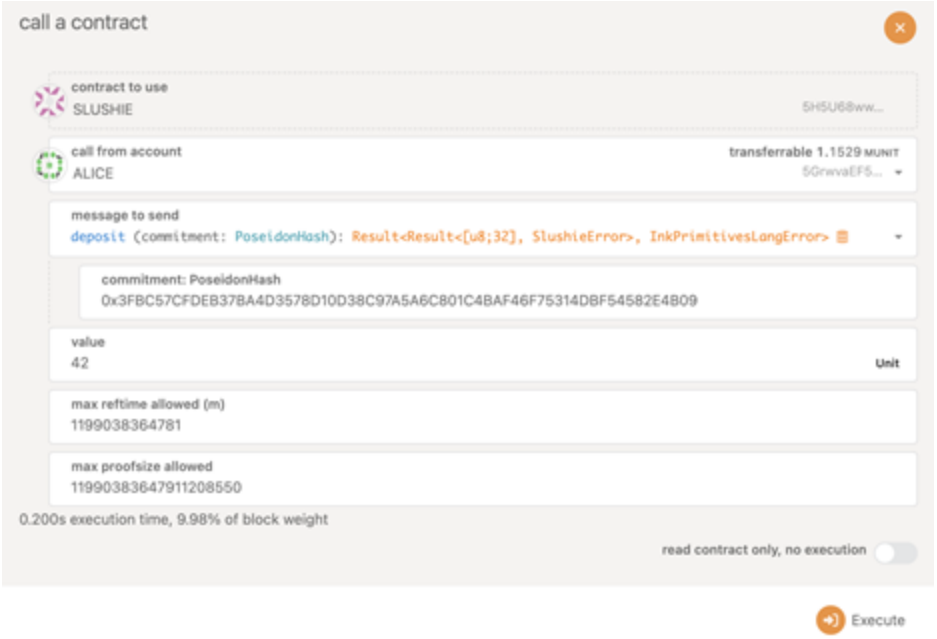
```
theirshadow@theirshadowmac substrate-contracts-node %
./target/release/substrate-contracts-node --dev

2023-05-11 18:48:04.251  INFO main sc_cli::runner: Substrate Contracts
Node
2023-05-11 18:48:04.252  INFO main sc_cli::runner: ✌️   version 0.24.0-
fe448511262
2023-05-11 18:48:04.252  INFO main sc_cli::runner: ❤️   by Parity
Technologies <admin@parity.io>, 2021-2023
```

# Deploying Slushie


# Generating commitment & secret values



# Depositing to Slushie



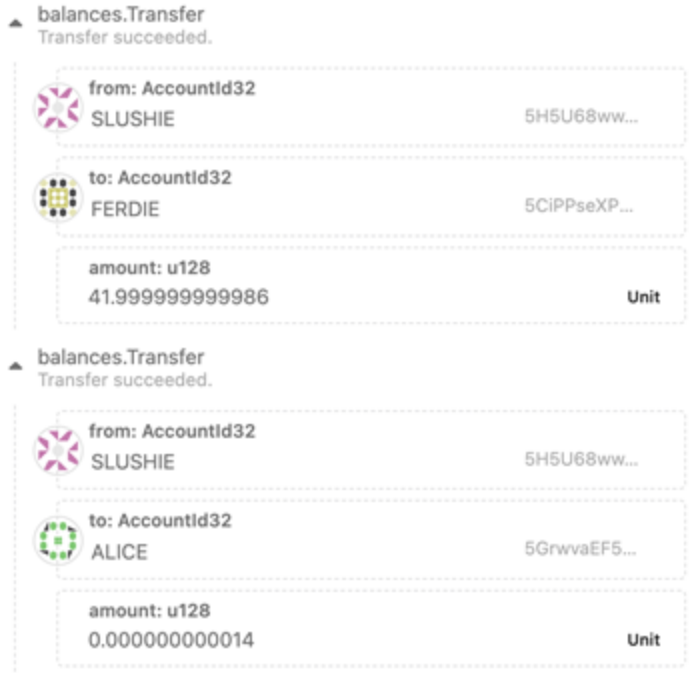
# Generating the proof

# Getting proof bytes (using node.js)

# Launching relayer

# Sending withdrawal request

# Events about withdrawal



# Conclusions

We succeeded to build a cryptocurrency mixer to be used in Substrate-based blockchains, which will allow the users to increase their privacy when transferring cryptocurrency. Moreover, we increased the integrity of the client-side code by reusing the same library (**plonk_prover**) within the CLI using native target compilation & within the smart contract using WebAssembly compilation. To add to that, we identified considerable issues with Substrate itself, like absence of important RPC methods in nodes, as well as bugs in Rust-based libraries for interacting with smart contracts.