



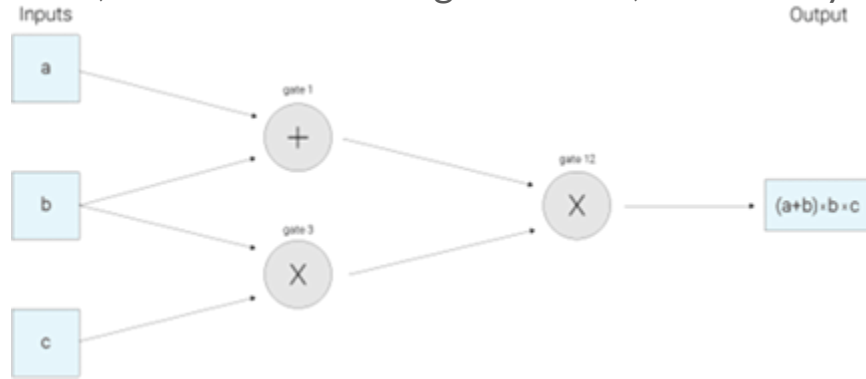
Slushie: a zero-knowledge proof-based token mixer in Substrate

Oleksandr Mykhailenko, Kyrylo Gorokhovskiy

Subject of the study

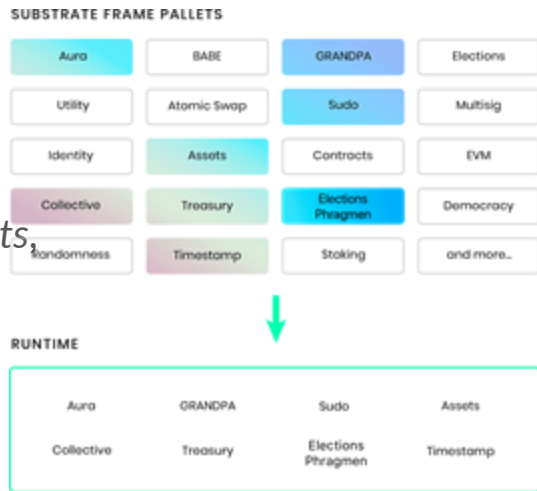
We research the possibility of implementing a working zero-knowledge proof-based cryptocurrency mixer for Substrate blockchains.

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 relay 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

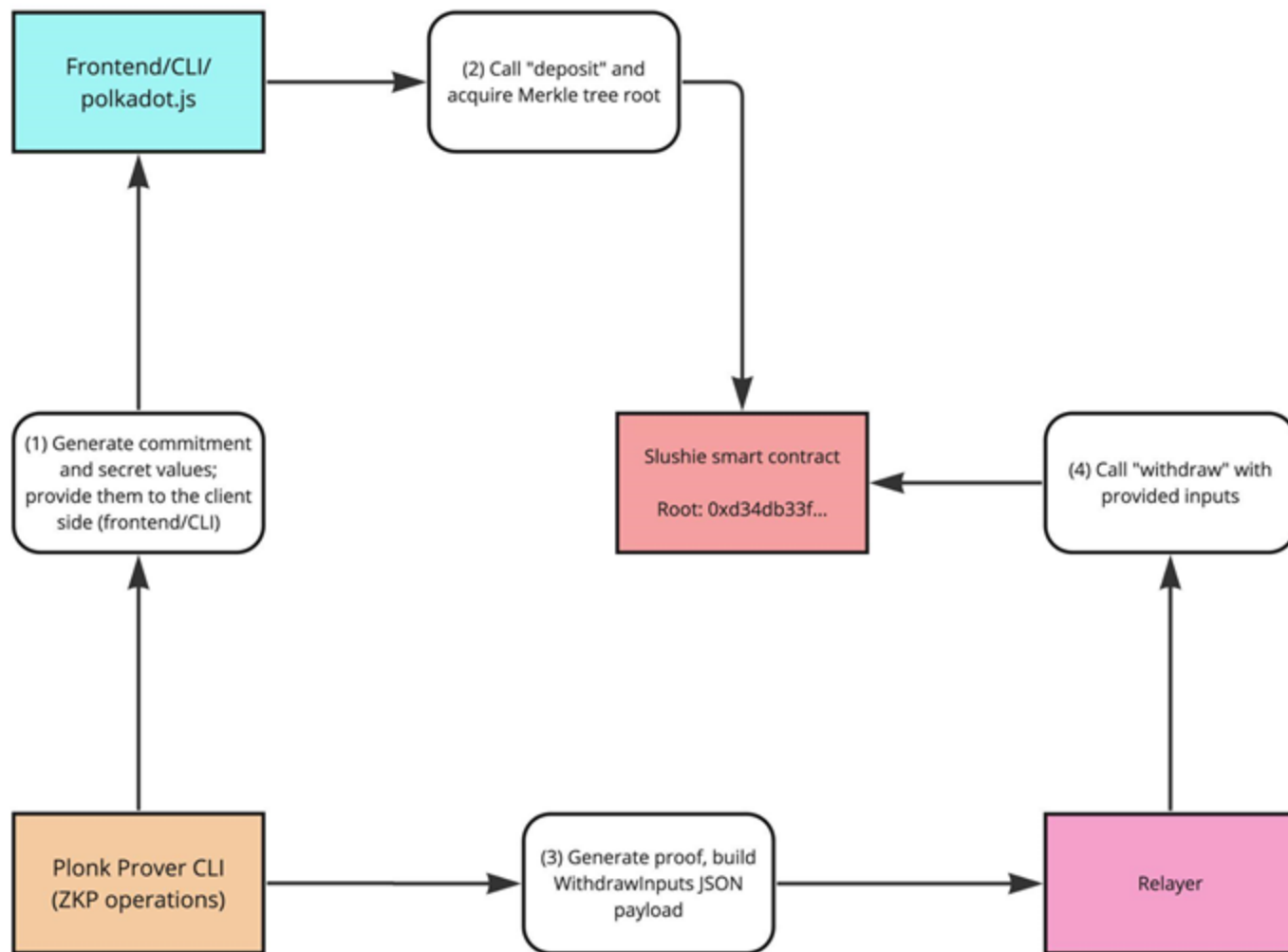
- Research the possibility to create a WebAssembly-based cryptocurrency mixer for Substrate-based blockchains using Plonk zkproof system;
- Improve client-side code integrity for producing and generating zero-knowledge proofs;
- Evaluate the ecosystem readiness for a production deployment of the mixer.



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 relay server to send **withdraw** transactions without any RPC provider tracing.





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 relay 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

upload & deploy code 1/2

deployment account
ALICE

transferrable 1.1529 slushie
SGreaseFS... →

json for either abi or .contract bundle

Constructors [1] ▲

new (depositSize: Balance)
create a new Slushie contract

Messages [3] ▲

▶ exec

deposit (commitment: PoseidonHash): Result<Result<[u8;32], SlushieError>, InkPrimitivesLongError> [x]
Deposit a fixed amount of tokens into mixer

▶ exec

withdraw (publicInputs: PublicInputs): Result<Result<Null, SlushieError>, InkPrimitivesLongError> [x]
Withdraw a fixed amount of tokens from the mixer

▶ read

getRootHash (): Result<[u8;32], InkPrimitivesLongError> [x]
Returns the merkle_tree root hash

165,917 WASM bytes

code bundle name
slushie

Next Deploy

upload & deploy code 2/2

deployment constructor
new (depositSize: Balance)

depositSize: Balance
42

Unit

max reftime allowed (m)
1199038364781

max proofs size allowed
11990383647911208550

0.200s execution time, 9.98% of block weight

Prev Deploy



Generating commitment & secret values

```
theirshadow@theirshadowmac plonk_prover_tool % cargo run -r --
generate-commitment
# ...
Successfully generated! Please save this values:
Nullifier: 3025989428
Randomness: 2767997642
Commitment:
3FBC57CFDEB37BA4D3578D10D38C97A5A6C801C4BAF46F75314DBF54582E4B09
Nullifier Hash:
BACA2103315821D8CCDA1BF7748A9D51648FCFADF7E1B03160E8C9FE33074A03
Root: 2E1D1E3EDD3311246FDBFAE853D6673A415B76BFA00B8FF448F12784A1DA1B60
You can use:
• commitment to call deposit contract method
• randomness, nullifier and nullifier hash to generate your Proof
• nullifier hash to call withdraw contract method
```

Depositing to Slushie

call a contract

contract to use
SLUSHIE

5H5U6@ww...

call from account
ALICE

transferrable 1.1529 MUNIT
5GnvaEFS...

message to send

deposit (commitment: PoseidonHash): Result<Result<[u8;32], SlushieError>, InkPrimitiveLangError>

commitment: PoseidonHash

0x3FBC57CFDEB37BA4D3578D10D38C97A5A6C801C4BAF46F75314DBF54582E4809

value

42Unit

max reftime allowed (m)

1199038364781

max proofsize allowed

11990383647911208550

0.200s execution time, 9.98% of block weight

read contract only, no execution

Execute



Generating the proof

```
theirshadow@theirshadowmac plonk_prover_tool % cargo run -r --  
generate-proof --pp ../public-parameters/pp-test --l 0 --root  
2E1D1E3EDD3311246FDBFAE853D6673A415B76BFA00B8FF448F12784A1DA1B60 --o  
../public-parameters/my-openings.json --k 3025989428 --r 2767997642 --a  
5CiPPseXPECbkjWCa6MnjNokrgYjMqmKndv2rSnekmSK2DjL --t  
5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY --f 14 --output-file  
prod-proof
```



Getting proof bytes (using node.js)

```
const proofFile = fs.readFileSync('plonk_prover_tool/prod-proof');  
  
// read proof bytes as hex  
const buffer = Buffer.from(proofFile, 'hex');  
console.log(buffer.toString('hex'));
```



Launching relay

```
theirshadow@theirshadowmac relay % npm run start
```

```
> slushie-relayer@1.0.0 start
```

```
> tsc && node dist/relay/index.js
```

```
set up api;
```

```
Listening on port 7777
```

Sending withdrawal request

```
theirshadow@theirshadowmac plonk_prover_tool % curl -XPOST -H 'Content-Type: application/json' "http://localhost:7777/withdraw" -d '{"nullifier_hash": "BACA210331582108CCDA1B7748A9051648FCFADF7E1803160E8C9F533074A03", "root": "2E1D1E3EDD3311246FDBFAE853D6673A415B768FA00B8FF448F12784A1DA1B60", "proof": "b5f578d7e7f6b5d154b5fdb48e5c31ba7cda15ce4d8ead17d8f89641ff04d4bbb3f61eea3662beee49eef2901c20e4aa76777dd0c3592817b9e2f7a048d54b0868ac28181b7d9837e2dc09260e40fd615b7085ce30ffe7eed60f191c43291d48b658eb53eafa7fcb67500090987deab795b10876495a8ebd699dec78f0831225b7cfff0306c601e0d38b4d47c79cfa468130e78206c9207de420744ad99b72c5f08c255aab6c3793a3e4a00c20905a9cab3faa0aef9889a80c348a06e21954d4b74b845a6081e77542bb27c69ffc83d9cdcaf62417ed41f79db1a179cbac58ce34e4e01f391277a675b9cbf45921e15499d84b816be5387f25acb3f2280935d31b8a9a71d48f20437c757119a2c82d46ba96d150799e58a2e009d75808d841b9639c7590d71894f75b6c75c485a5f3f4513f943ee66ae8cf274b68dfa1c8a4b3db7bbc49a6bf1351f629e8c9522cd27a463a46b5e3994e26ddc395c7093f956d6c28524e695a34c5526bcc674385e66fd797bcebc7b83d6c029a787e7e5877ae9d297d130a68599f6e29c57e720e0229908f4f1f027ca8919d80e40dd26de9131866bb76c98f8897053edab57073118e71d6032715a9947b3f9d76b62c93f7d68fbd3335d0d442184b9713575ceaa48afbc73fd10d41925cca9dd33f9b5befb93ed9c9ea2bb273e52c2d12f710e5a0a8b8d7a074a681fabcf5486e42614f8a8676caad615826735ba9eb829885387fa069eb82ac6f67341c32e8b33c93cc85ea8c2e86b008ddb103ded593fe57250463c913d785802b78f087829ecac7ebelfdc756e6ffa7be4f207331d08ca27301366f127fa3ac086e3ffe09d34928e9abde2b792f4f1b299b841343a4f8edbbc2af4902fdb8d9acafbb09a09be497aa c991dda6bec2e7912be1d748ae7fb4da8146df2dd7753fcccfa42a8663313cd0fa1610 deef4b859f7926ebe4196daf665bb9980b1464e4d66b492607a64965a438c62484c950d70447612144d77a1c4951725e270c5ddc67a29616ef1d07b28182a090f0b9a2285df0d98d5a35dc3f2f1e39c827c20df7a36918e8f0ddb1a15a751206509bf1306058eb0bb8e6 fe87219e423e042ccd4cccb7e9a7142ac7bdd53b29a778728c9ca1f7c263ee596040ff210781873ec8dc30b17ca7ac04bc2f0e2d1aa8f4f1219f477fa1342584a256d872ac947f71d0a9cb0ea94b276b08ec4f9247297075c799fdd5e6406a9e03ecca0b6e8a1c9e182e89c41f7b79da26fca43e0df4e3b7ef465582924ad43cef9687081f43d28abea21627457e8ad208a1adce146fbb28b2c2a838bee5f4987b12ca18e74dee941fa23686bda88df7b6e07db8ef9095b991d296fdead63e6949f13c1fd6aa00605ae0d758cbab35731d75cac58bb510e0eac4101090486558eb2d6d3200a8d7c2081e923bc0641e78d38db0e7a60ed8d43bd24a9927fb745c8ecd4e0e", "fee": 14, "recipient": "SC1PPseXPECbkjWCa6MnJNokrgYjMqmKndv2rSnekSK2DjL", "relayer": "5GrwvaEF5zXb26Fz9rcQpdwWS57CtERHpNehXCPcNoHGKutQY", "contract_address": "5SHU60hwAHO26LFBcbWUQ1bSYncavvKsh7aGChE2tCcBTyxE"}'
```

Events about withdrawal

▲ balances.Transfer
Transfer succeeded.



from: AccountId32
SLUSHIE

5H5U68ww...



to: AccountId32
FERDIE

5CiPPseXP...

amount: u128
41.9999999999986

Unit

▲ balances.Transfer
Transfer succeeded.



from: AccountId32
SLUSHIE

5H5U68ww...



to: AccountId32
ALICE

5GrwvaEF5...

amount: u128
0.0000000000014

Unit



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.