# Groth16 zkSNARK Proof Verification on Freeton : Three Use Cases

OcamlPro - Fabrice Le Fessant et Thomas Sibut-Pinote

July 11, 2021

The present document is an official submission to the 18$^{th}$ contest[1] of the DevEx governance: *Groth16 zkSNARK Proof Verification Use Cases*. Although we accepted earlier to be part of the jury, we understand that by submitting this document we waive our right to vote in the present contest.

In this submission, we propose three different use cases for Groth16 zk-SNARK proofs on the FreeTON blockchain:

- Sudoky: Zksnarks are used to prove that the user has a solution to a given Sudoky problem, without providing the solutions to other users ;
- Euler Project: Zksnarks are used to prove that the user has found the solution to an Euler problem ;
- Pin-code reset of forgotten keys: Zksnarks are used to replace a pubkey when the user has lost his secret key

Finally, we finish this document by a section showing the contributions that we did for the Free TON community while working on this contest.

## 1 A toy example: Sudoku

Let's start with our first toy use case. Suppose you want to set up a Sudoku problem for your students to solve, so that upon completion they receive some token. The issue is that once any student has solved it, the solution sits on the blockchain for all to see. All the other students can cheat by copying it, instead of doing the work by themselves, which defeats the whole purpose. One other advantage is that we don't need to implement the whole logic of the computation on the smart contract. For some size of Sudoku, this is probably worthwhile.

## 2 A better example: Project Euler

The previous example is somewhat limited by the fact that sudokus are easily solvable problems, either by hand or with a computer; they are not a realistic use case of zk-snarks. Let's keep the idea of a decentralized classroom. the

[1] https://devex.gov.freeton.org/proposal?proposalAddress=0%3Ae6b65075478e7d412fdb0870452f30dfa8bf51272e28a3167abc5c5df6fd051d

famous website Project Euler[2] offers math and programming problems of various difficulty to its users. It uses captchas to prevent brute force attempts at guessing the answer to a problem. The answers to problems are stored statically on its servers and the answers are checked against them. Can we make Project Euler decentralized? Using the idea from Section **??**, we can prevent users from stealing others' solutions **and** from brute-forcing them, due to the cost of generating proofs.

# 3 Pin code reset for forgotten keys

In this use case, we use Zksnarks to allow users to replace their public keys in multisig wallets. The typical use is when a user has lost the associated secret key, and is unable to sign/confirm any new transaction. With our approach, the user first creates a recovery contract, where a passphrase is associated to his public key. If he needs to change his public key, he can use Zksnarks to associate a new public key in his recovery contract, and tell multisig wallets to safely replace his pubkey by the new one.

Compared to other approaches, this use can provide several improvements:

- The user does not disclose his passphrase. Most other approaches (based on hashes for example) give an opportunity for man-in-the-middle attacks ;
- As the user does not disclose his passphrase, the pubkey can be replaced several times ;
- The user does not need to create the new public key ahead of time. Instead, he can create it when he has lost the former one, decreasing the likelihood of losing the associated secret key also ;

## 3.1 Technical solution

This use case includes 3 different components:

**PinCode Client:** it is a C++ program linked to the Blueprint Zksnarks library. It can be used in two different ways:

- To create the initial circuit using the passphrase. The client should be called as:

  ```
  pincode-client prepare "my pass phrase"
  ```

  This will generate a file verifkey.hex that can be used as an argument when deploying the PubkeyRecovery smart contract, and a file provkey.hex that can be used to create new witnesses ;
- To create a witness that the passphrase is known when providing a new public key. Because the witness contains the new public key, an attacker cannot intercept the message and replace it. The client should be called as:

  ```
  pincode-client prove "my pass phrase" "PUBKEY_AS_HEX_NUMBER"
  ```

[2] https://projecteuler.net/

The command expects to find the file provkey.hex and generates a file proof.hex that should be submitted to the PubkeyRecovery smart contract together with the new public key.

**PubkeyRecovery smart contract:** This smart contract is a TIP-3 contract, using the initial public key to verify its address. This contract is very simple. It contains:

- A constructor to set the circuit that will be used to verify that the passphrase is known when it is used ;
- An external function SetNewPubkey(bytes proof, uint256 pubkey) to define the new public key, while providing a proof of knowledge of the passphrase ;
- An external function RecoverPubkey(address multisigaddr) to call a multisig contract to update the corresponding custodian. The multisig contract will verify that the address is correct before replacing the public key ;

**Modified Multisig Wallet:** This is a standard multisig wallet, modified to recognize addresses of PubkeyRecovery smart contracts. They provide two additionnal functions:

- An external function SetPubkeyRecoveryCode(TvmCell code) to define the code of the PubkeyRecovery smart contract. Because the code hash is known, anybody can call this function with the correct code.
- An external function RecoverPubkey(uint256 oldkey, uint256 newkey) that can only be called by a PubkeyRecovery smart contract.

# 4 Contributions to the Free TON community

In the course of participating to this contest, we were led to refine our in-house tools in order to more easily manipulate the Nil Foundation forks of the Ton Virtual Machine (TVM), the TON Solidity compiler and the TVM linker.

Our first contribution is a public Docker image containing the TONOS SE compiled with Zksnarks support. The image is published as `ocamlpro/nil-local-node` and is built from `https://github.com/NilFoundation/tonos-se`.

Running it is simple:

```
docker run -d --name local-node -e USER_AGREEMENT=yes -p80:80 ocamlpro/nil-local-node
```

Another way to use it is through our development tool `ft`, a Free TON Wallet designed for developers: `https://github.com/OCamlPro/freeton_wallet`

For this contest, a set of improvements has been contributed to `ft` to ease using it.

In particular:

- The interface has been improved to provide multiple levels of subcommands, making them easier to understand and to use ;

- A new argument `-image` is available with `ft switch create` to specify the Docker image to use. This new argument is specially useful for our Docker image for NilFoundation TONOS SE.

As a consequence, a Zksnark-ready sandbox (local network) can be installed by simply running:

```
ft switch create sandbox --image ocamlpro/nil-local-node
```

Now all commands from the ft documentation will work, in particular it will be easy to create accounts, deploy contracts, and call them as seen in https://ocamlpro.github.io/freeton_wallet/sphinx/use-cases.html.

The Docker image of ft has been updated to use NilFoundation tools (solc, tonos-cli, tvm_linker), it is the easiest way to use ft if you don't want to build it. See the installation instruction at (https://ocamlpro.github.io/freeton_wallet/sphinx/install.html#using-docker).