User privacy in DAOs

Zero-Knowledge Proofs libraries: Integration and Management in Byonts WEB3 template

3620557

Ignas Apšega

byont

Fontys
University of Applied Sciences

Version Control

Version	Date of change	Change description
0.1	10-11-2022	Initial draft
0.2	21-12-2022	Added Back-end explanation
Final Version	08-01-2023	Added Front-end explanation, Conclusions

Glossary

Term	Explanation
1. Decentralized Autonomous Organisation (DAO)	DAO, or Decentralised Autonomous Organisation, is an organization without any hierarchy among its members. A DAO also has a set of "rules" on how to manage its treasury (DAO's money) and how to handle votings, which are essential for making decisions, and a DAO usually has its own Token (Cryptocurrency, in this case). All these aspects are described inside a DAO smart contract.
2. Circom	It is a compiler written in Rust for compiling circuits written in the circom language. The compiler outputs the representation of the circuit as constraints and everything needed to compute different ZK proofs. ("Circom 2 Documentation")
3. SnarkJS	This is a **JavaScript and Pure Web Assembly implementation of zkSNARK and PLONK schemes.** It uses the Groth16 Protocol (3 point only and 3 pairings) and PLONK. ("iden3/snarkjs: zkSNARK implementation in JavaScript & WASM")
4. WEB3	**Web3** is an idea for a new iteration of the World Wide Web which incorporates concepts such as decentralization, blockchain technologies, and token-based economics

5. Trusted setup	A trusted setup is a procedure that involves more than one party. Its aim is to produce the standard parameters that a proof system or similar cryptographic protocols rely on.
6. GROTH16	Groth16 is **a zero-knowledge proof protocol** proposed by Daniel Jens Groth in the paper "On the Size of Pairing-based Non-interactive Arguments" published in 2016. Please note that the name of the general algorithm is composed of the first letter of the author's surname and the year. (Jiang)
7. PLONK	PlonK is the acronym **forPermutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge**. PlonK ia an implementation of the Universal Zero-Knowledge proof algorithm. Universal means that the trusted setup only needs to be initiated once. ("ZKP— PlonK Algorithm Introduction")
8. Blockchain	Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. ("What is Blockchain Technology? - IBM Blockchain")
9. Smart Contract	A smart contract is a computer program or a transaction protocol that is intended to automatically execute, control or document legally-relevant events and actions according to the terms of a contract or an agreement. ("Smart contract")

Table of Contents

User privacy in DAOs **0**

Zero-Knowledge Proofs libraries:
Integration and Management in Byonts WEB3 template 0

Version Control **1**

Glossary **2**

Table of Contents **4**

1. Introduction **5**

1.1 Purpose of the document 5

1.2 Context 5

2.1 Research Questions 5

2.2 Research strategy 5

2.3 Research methods 6

2.1 Back-end 7

2.2 Front-end 10

Verifier Smart Contract 12

Conclusions 12

To start with, it was hard to implement SnarkJS module into Byonts WEB3 template. Because the template is made in NextJS as a Front-end framework and its main language is Typescript. I did not have a background in any of those, thus it took longer than I thought to implement SnarkJS. In addition, I had to use the not-so-familiar library for connecting to the blockchain which took some time also to manage. 12

Nonetheless, I managed to implement Circom and SnarkJS and even implemented scripts for easier application management. 12

References 13

Appendix 16

1. 16

2. 17

1. Introduction

1.1 Purpose of the document

The purpose of this document is to describe the integration of the Circom[2] and SnarkJS[3] libraries into a WEB3 template as part of the User Privacy in DAOs project. It will also cover the management aspect of the application, including scripts for trusted setup.

1.2 Context

The context of this document is within the Implementation and management of Zero-Knowledge solutions to Byonts WEB3 template.

2. Research

2.1 Research Questions

This document was made to help understand what is Circom and SnarkJS. This research is part of the following sub-questions, which were also researched in the previous documents:

1. **What tools are there for improving privacy for DAOs?**
2. **Could the solutions found be implemented in the current projects of Byont?**

2.2 Research strategy

For this research mainly the strategy **Workshop** was chosen. It is done to explore opportunities. Prototyping, sketching, and co-creation activities are all ways to gain insights into what is possible and how things could work.

2.3 Research methods

Workshop research strategy has multiple research methods. But not all of them were used to conduct this research.

Prototyping was used to develop, evaluate, and design a solution to learn whether the libraries work and discover the technical limitations or possibilities.

2. Implementation

2.1 Back-end

To start with the implementation, particular scripts were written to help with the Trusted setup phase. Check Appendix[1] for code.

- **Execute_groth16_circuit_example.sh** - it builds needed files(proving, verification keys, verifying smart contract and etc) for example circuit. Implements Groth16[6] protocol.
- **Execute_plonk_circuit_example.sh** - same as the one above, just in PLONK.
- **Execute_vote_no_circuit.sh** - it builds needed files(proving, verification keys, verifying smart contract and etc) for vote no circuit. Implements PLONK protocol.
- **Execute_vote_yes_circuit.sh** - it builds needed files(proving, verification keys, verifying smart contract and etc) for vote no circuit. Implements PLONK protocol.

On the next page. An explanation of the circuits will be displayed.

```
pragma circom 2.0.0;
/*This circuit template checks that c is the multiplication of a and b.*/
template Example () {
   // Declaration of signals.
   signal input a; //Public value.
   signal input b; //Private value.
   signal input c; //Public value.
   signal d; //Intermediate signal.
   signal output out; //Output of the signal.
   // The logic or a 'constraint' which a private signal has to satisfy.
   d <== a * b;
   out <== c + d;
   out === 18;
}
component main { public [ a, c ] } = Example();
```

This is the basic example circuit which was implemented as an example and starting point for making circuits. It serves as a boilerplate.

There are Vote Yes and Vote No arithmetic circuits designed as Proof of Concept that it is possible to verify strings(binary) data.

```
pragma circom 2.0.0;

template YesBinaryCount () {
   signal input vote; // Private value. Number of bits a Yes string has
   signal output out; // Final value.
   var number=0; // A counter

   // Looping through number of bits a vote string has
   for (var i = 0; i < vote; i++) {
      number+=1;
}

   // The logic or a ‘constraint’ which a private signal has to satisfy.
   out <-- number; // Assigning out to number
   out === 21; // Comparing out to be 21
}
component main = YesBinaryCount();

pragma circom 2.0.0;

template NoBinaryCount () {
   signal input vote; // Private value. Number of bits a Yes string has
   signal output out; // Final value.
   var number=0; // A counter

   // Looping through number of bits a vote string has
   for (var i = 0; i < vote; i++) {
      number+=1;
   }

   // The logic or a ‘constraint’ which a private signal has to satisfy.
   out <-- number; // Assigning out to number
   out === 14; // Comparing out to be 14
}
component main = NoBinaryCount();
```

These circuits were presented more deeply in - Zero-Knowledge Proofs libraries: Circom and SnarkJS implementation documentation.
It is worth mentioning that the script generates Proving and Verifying keys and that the Verifying key can be transformed into Verifying Smart Contract:

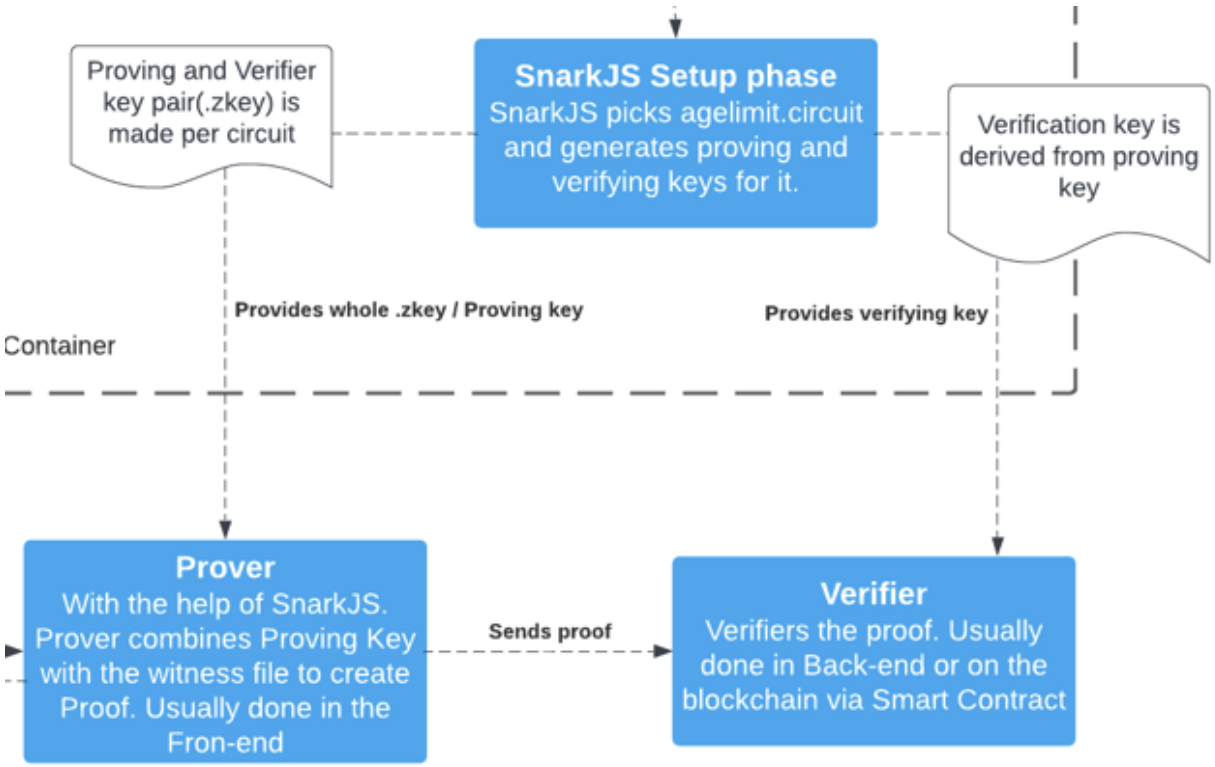

Figure 1. Prover/Verifier keys

2.2 Front-end

To continue, Front-end is using SnarkJS Javascript module to generate proofs in the Front-end:

```
const snarkjs = require('snarkjs');
const makePlonkProof = async (_proofInput, _wasm, _zkey) => {
  const { proof, publicSignals } = await snarkjs.plonk.fullProve(
    _proofInput,
    _wasm,
    _zkey
 );
return { proof, publicSignals };
};
```

This is a function to generate a proof. It takes:

- **input** - the private data, the prover is trying to prove
- **.wasm file** - It is a program that is generated by Trusted Setup Phase Script. It basically consists of circuit logic, takes input, and generates the witness(proof)
- **.zkey file** - It is Protocols public key generated by the Trusted Setup Phase Script. It is used to identify the circuit.
- **Uses PLONK** type to generate the **proof**(private and intermediary signals) and also encrypts public signals of the circuit

Refer to Appendix[2] to check the diagram for the explanation of file generation by the SnarkJS Trust Setup phase.

On the next page will be shown what proof and public signals look like:

1

In short **Proof is Private and Intermediary signals translated into points on the Elliptic curve.** It is a method of cryptography used for Prover and Verifier interaction, where the Verifier checks if the points were indeed on the Elliptic Curve.

Public signals are just public signals known to everyone from the circom circuit. In this example it is 21. Because Yes Vote circuit has OUT variable equal to 21 (amount of bits yes string has). Out signal is always public.

Proof:

{ "A": ["4826792331990762558065391795169209153882394122320176475715788171308728134685",
"9772921378318043433311942892932518576565660946734354545231153704645415636640", "1"], "B": [
"3611637822673880556448194735464817735364437149301721404418176406466415375034",
"8774476075755647184173754467255018930814681852866449021499722955399406298360", "1"], "C": [
"8301974023364737484742157471178466369996489563598000366126192247266839360408",
"1058558589259094971145479743392691206475579670099530673950296776965248805909 6", "1"], "Z": [
"1834728346397833155484568616402660736775128285793650869116845477108082752061 5",
"1948378974737310494747577589821874612620235284220434856598888221473416788256 9", "1"], "T1": [
"4658711495178597277684428477890250674837295803752352679243853242265925487832",
"1029702216190512611030571503632864792933993333973292143213437448894410115867 7", "1"], "T2": [
"9481250507948364647813309176914491327347671933506197976284765115690861561 56",
"1593221678657930252208037324015734105377082067073558716545164129107551848860 4", "1"], "T3": [
"7711710461862539077932822298433476041207749651406993342833822052907231666857",
"5749100902340722926701483305654272715643124036606583095238634283038354186198", "1"], "eval_a":
"1608139029332869631105194562825592226753125792219664293445826230072233350809", "eval_b":
"4022036583541191881055975133207972084255206181059235208615254481836057657593", "eval_c":
"1667134771309996741742562331354086591310505389377296374311982300653028015544 9", "eval_s1":
"5465777687847863775269559126646868179897158613732302041036169621219135114 63", "eval_s2":
"1914807806603471444299785208827912841325043426944163502101344878316575817546 9", "eval_zw":
"1145834834765431111317286044456133121343867861334618545239766156391577900503 4", "eval_r":
"2067760535835431069278212080394654647677928153873773048252941024939723918203 2", "Wxi": [
"3964702850067139952191167947237617082925665336628415212838105002678028151343",
"1798149153064624948839046621195773890218106986625983912200017816428864749208 0", "1"], "Wxiw": [
"1317983308719122121541700382870654593192271044377491772210690590011232160928 5",
"9541707540489057803908026799327633109511993298468472255336949792629532688059", "1"], "protocol": "plonk", "curve": "bn128" }

Signals:

["21"]

To continue, Verifier Smart Contracts were generated from Verifying key. The Smart Contracts[8] were uploaded to the Polygon blockchain[9] Mumbai Testnet. To add, generated proof and publicSignals have to be encrypted. It can be decrypted only by Verifier

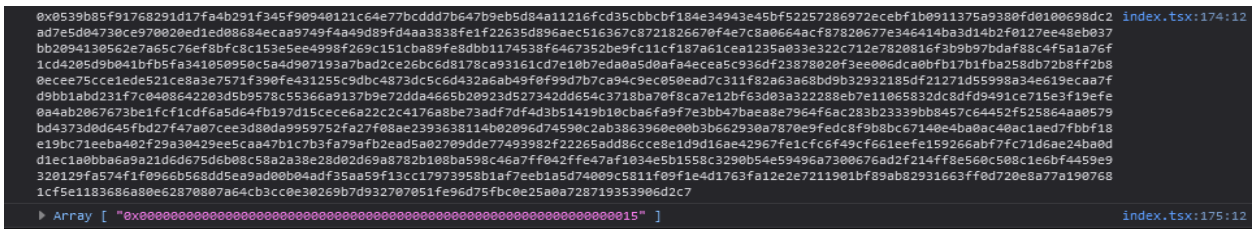

Figure 2. Encrypted Proof and Public Signals

Verifier Smart Contract

The Verifier Smart Contract takes the above-showcased proof and public signals. To do so, Front-end has to connect to the blockchain and call this function. If the proof and public signals both are correct the transaction on the blockchain passes, if not - the transaction fails. That is the normal behavior of generated Verifier Smart Contract.

The header of the function.

```
    function verifyProof(bytes memory proof, uint[] memory pubSignals)
public view returns (bool) {
```

The whole code of the function is too long to post here (~around 500 lines) but what it does is take the inputs and derives the Elliptic points from it. Then it compares these points on the curve with the one it already has and the transaction fails or passes depending on the comparison of the points.

Conclusions

To start with, it was hard to implement SnarkJS module into Byonts WEB3 template. Because the template is made in NextJS as a Front-end framework and its main language is Typescript. I did not have a background in any of those, thus it took longer than I thought to implement SnarkJS. In addition, I had to use the not-so-familiar library for connecting to the blockchain which took some time also to manage.

Nonetheless, I managed to implement Circom and SnarkJS and even implemented scripts for easier application management.

References

"Bulletproofs In Crypto – An introduction to a Non-Interactive ZK Proof." *Panther Protocol Blog*, 16 September 2022, https://blog.pantherprotocol.io/bulletproofs-in-crypto-an-introduction-to-a-non-interactive-zk-proof/. Accessed 14 November 2022.

"Bulletproofs | Stanford Applied Crypto Group." *Applied Cryptography Group*, https://crypto.stanford.edu/bulletproofs/. Accessed 25 October 2022.

"The C4 model for visualising software architecture." *The C4 model for visualising software architecture*, https://c4model.com/. Accessed 31 October 2022.

"Circom 2 Documentation." *Circom 2 Documentation*, https://docs.circom.io/. Accessed 25 October 2022.

"dalek-cryptography/bulletproofs: A pure-Rust implementation of Bulletproofs using Ristretto." *GitHub*, https://github.com/dalek-cryptography/bulletproofs. Accessed 25 October 2022.

"Domain-specific language." *Wikipedia*, https://en.wikipedia.org/wiki/Domain-specific_language. Accessed 25 October 2022.

"iden3/circom: zkSnark circuit compiler." *GitHub*, https://github.com/iden3/circom. Accessed 25 October 2022.

"iden3/snarkjs: zkSNARK implementation in JavaScript & WASM." *GitHub*, https://github.com/iden3/snarkjs. Accessed 10 November 2022.

Jiang, Xin. "How to Generate a Groth16 Proof for Forgery." *Medium*, 21 November 2019, https://medium.com/ppio/how-to-generate-a-groth16-proof-for-forgery-9f857b0dca

fd. Accessed 9 January 2023.

Ouaddah, Aafaf. “Arithmetic Circuit.” *ScienceDirect*, 2019, https://www.sciencedirect.com/topics/engineering/arithmetic-circuit. Accessed 25 10 2022.

“Scaling | ethereum.org.” *Ethereum.org*, https://ethereum.org/en/developers/docs/scaling/. Accessed 30 October 2022.

“Smart contract.” *Wikipedia*, https://en.wikipedia.org/wiki/Smart_contract. Accessed 11 November 2022.

“WebAssembly.” *Wikipedia*, https://en.wikipedia.org/wiki/WebAssembly. Accessed 10 November 2022.

“What are zk-STARKs?” *Bit2Me Academy*, 26 August 2022, https://academy.bit2me.com/en/que-son-las-zk-stark/. Accessed 10 November 2022.

“What Is a Layer 2?” *Chainlink Blog*, 19 July 2022, https://blog.chain.link/what-is-a-layer-2/. Accessed 30 October 2022.

“What is a Statement?” *Computer Hope*, 5 February 2021, https://www.computerhope.com/jargon/s/statemen.htm. Accessed 15 December 2022.

“What is Blockchain Technology? - IBM Blockchain.” *IBM*, https://www.ibm.com/topics/what-is-blockchain. Accessed 11 November 2022.

“What Is Layer 1 in Blockchain?” *Binance Academy*, 22 February 2022, https://academy.binance.com/en/articles/what-is-layer-1-in-blockchain. Accessed 30 October 2022.

“Zero-knowledge proof.” *Wikipedia*, https://en.wikipedia.org/wiki/Zero-knowledge_proof. Accessed 10 November 2022.

"ZKP— PlonK Algorithm Introduction." *https://trapdoortech.medium.com/zkp-plonk-algorithm-introduction-834556a32a*, 6 January 2023, https://trapdoortech.medium.com/zkp-plonk-algorithm-introduction-834556a32a. Accessed 9 January 2023.

"zk-STARKs vs zk-SNARKs: Differences in zero-knowledge technologies." *Panther Protocol Blog*, 2 September 2022, https://blog.pantherprotocol.io/zk-snarks-vs-zk-starks-differences-in-zero-knowledge-technologies/#what-are-zk-starks. Accessed 14 November 2022.

Appendix

1.

```
Apšega,Ignas, 3 days ago | 1 author (Apšega,Ignas)
# Entering to the right folder
cd ..
cd plonk/example

# Compile the example
circom example.circom --r1cs --wasm --sym

# Start a new powers of tau ceremony
snarkjs powersoftau new bn128 12 pot12_0000.ptau -v

# Contribute to the ceremony
snarkjs powersoftau contribute pot12_0000.ptau pot12_0001.ptau --name="First contribution" -v

# Provide a second contribution
snarkjs powersoftau contribute pot12_0001.ptau pot12_0002.ptau --name="Second contribution" -v -e="some random text"

# Provide a third contribution using third party software
snarkjs powersoftau export challenge pot12_0002.ptau challenge_0003
snarkjs powersoftau challenge contribute bn128 challenge_0003 response_0003 -e="some random text"
snarkjs powersoftau import response pot12_0002.ptau response_0003 pot12_0003.ptau -n="Third contribution name"

# Apply a random beacon
# We need to apply a random beacon in order to finalise phase 1 of the trusted setup
snarkjs powersoftau beacon pot12_0003.ptau pot12_beacon.ptau 0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f 10 -n="Final Beacon"

# SETUP PHASE 2 - Plonk specific
# Prepare phase 2
snarkjs powersoftau prepare phase2 pot12_beacon.ptau pot12_final.ptau -v

# Plonk setup
snarkjs plonk setup example.r1cs pot12_final.ptau example_final.zkey

# Export the verification key
snarkjs zkey export verificationkey example_final.zkey verification_key.json

# Turn the verifier into a smart contract
snarkjs zkey export solidityverifier example_final.zkey ExampleVerifier.sol      Apšega,Ignas, 3 days ago • feat: initial commit …
```

2.

Circom & SnarkJS flow in Zero-Knowledge Component