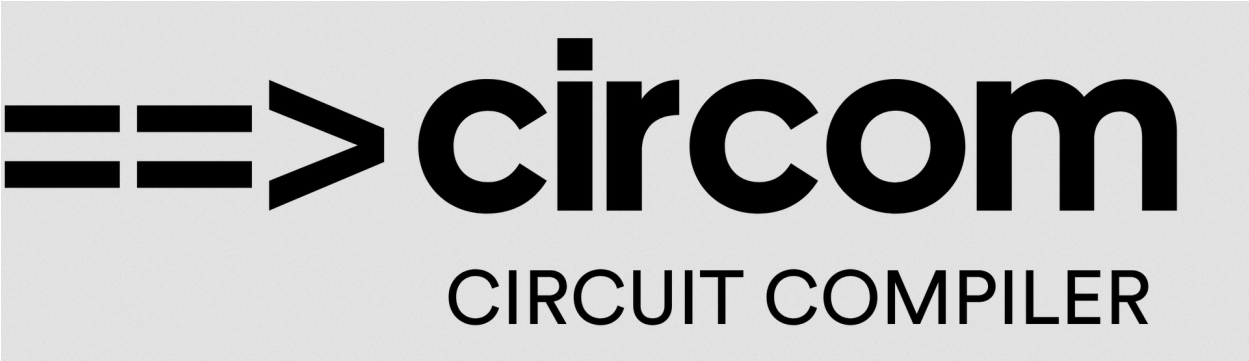
User privacy in DAOs

Zero-Knowledge Proofs libraries: Circom and SnarkJS implementation



https://github.com/iden3/circom

https://github.com/iden3/snarkjs

Ignas Apšega

byont

Fontys
University of Applied Sciences

Version Control

Version	Date of change	Change description
0.1	09-11-2022	Initial draft
0.2	15-12-2022	Added better graphics for circuit example Added more diagrams to explain Prover and Verifier relationship
Final Version	08-01-2023	Added new graphics, Conclusions, Advice

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. Zero-Knowledge Proofs (ZKP)	In cryptography, a zero-knowledge proof or zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that a given statement is true while the prover avoids

	conveying any additional information apart from the fact that the statement is indeed true. The essence of zero-knowledge proofs is that it is trivial to prove that one possesses knowledge of certain information by simply revealing it; the challenge is to prove such possession without revealing the information itself or any additional information. ("Zero-knowledge proof")
5. 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")
6. Computational Statement	In computer programming, a statement is a single line of code that performs a specific task. For example, the following line of programming code from the Perl programming language is an example of a statement: $a = 3; In this example statement, a variable ($a) is assigned the value of "3" that is stored as a string. This type of statement is known as an assignment statement because a value is being assigned to a variable. ("What is a Statement?")
9. JavaScript	Is a programming language that is one of the core technologies of the World Wide Web
10. Web Assembly	WebAssembly defines a portable binary-code format and a corresponding text format for executable programs as well as software interfaces for facilitating interactions between such programs and their host environment ("WebAssembly")
11. 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
12. 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.
13. 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”)
14. Binary	It describes **a numbering scheme in which there are only two possible values for each digit -- 0 or 1** -- and is the basis for all binary code used in computing systems. These systems use this code to understand operational instructions and user input and to present a relevant output to the user
15. Secure multi-party computation	It is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private
16. Telegram	It is Messenger which is a globally accessible freemium, cross-platform, encrypted, cloud-based, and centralized instant messaging service.

Table of Contents

1. Introduction

1.1 Purpose of the document

This document is part of the User Privacy in DAOs[1] research assignment. It will explain why Circom[2] and SnarkJS[3] libraries were chosen. It will also showcase the inside workings of how Circom and SnarkJS can be implemented as a Zero-Knowledge Proof[4] solution on the blockchain[5]. In addition, this document will give examples of possible solutions for the DAOs.

1.2 Context

The context of this document is within User Privacy in DAOs. It means that the presented technologies will be explained on how they can be used within this context.

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.

3. Findings

3.2 Arithmetic circuits - Circom

Introduction

Circom is a domain-specific programming language. It helps implement **zk-SNARKs** protocol by letting you define your problem in an arithmetic circuit. Which then can be used by SnarkJS to generate Prover and Verifier.

Circom was chosen because it was used in production-ready projects such as:

1. Tornado Cash - crypto mixer. A is a service that mixes potentially identifiable cryptocurrency funds with others to obscure the trail back to the fund's original source
2. Polygon Hermez - scalability solution for Ethereum. Uses zk-SNARKs.
3. Dark Forest - Zero-Knowledge proofs game on the blockchain.

Another reason it was chosen is because it had pretty good documentation compared to other libraries. For comparison between ZKP libraries check - **Zero-Knowledge Proofs Protocols and libraries** document.

Signals of a circuit

To start with Circom, there is a need to understand how it works. To give more context:

zk-SNARKs allows proving **computational statements[6]**, but they can not be applied to the computational problem directly, the statement(problem) first needs to be converted into the right form - an **arithmetic circuit**, and this is where **Circom** comes in.

An arithmetic circuit consists of

- **Input signals -** that can be either private or public. Private inputs are inputs that the user enter wants to keep private e.g age. Public inputs are publicly known inputs, for example, the public age limit is 18.
- **Gate -** it takes signals. It is either a multiplication or an addition gate.
- **Intermediate signals -** output signals of the gate. They are generated after private and public signals are supplied.
- **Output signal -** is the last output signal of the circuit.

To understand this better a diagram was drawn the on the next page.

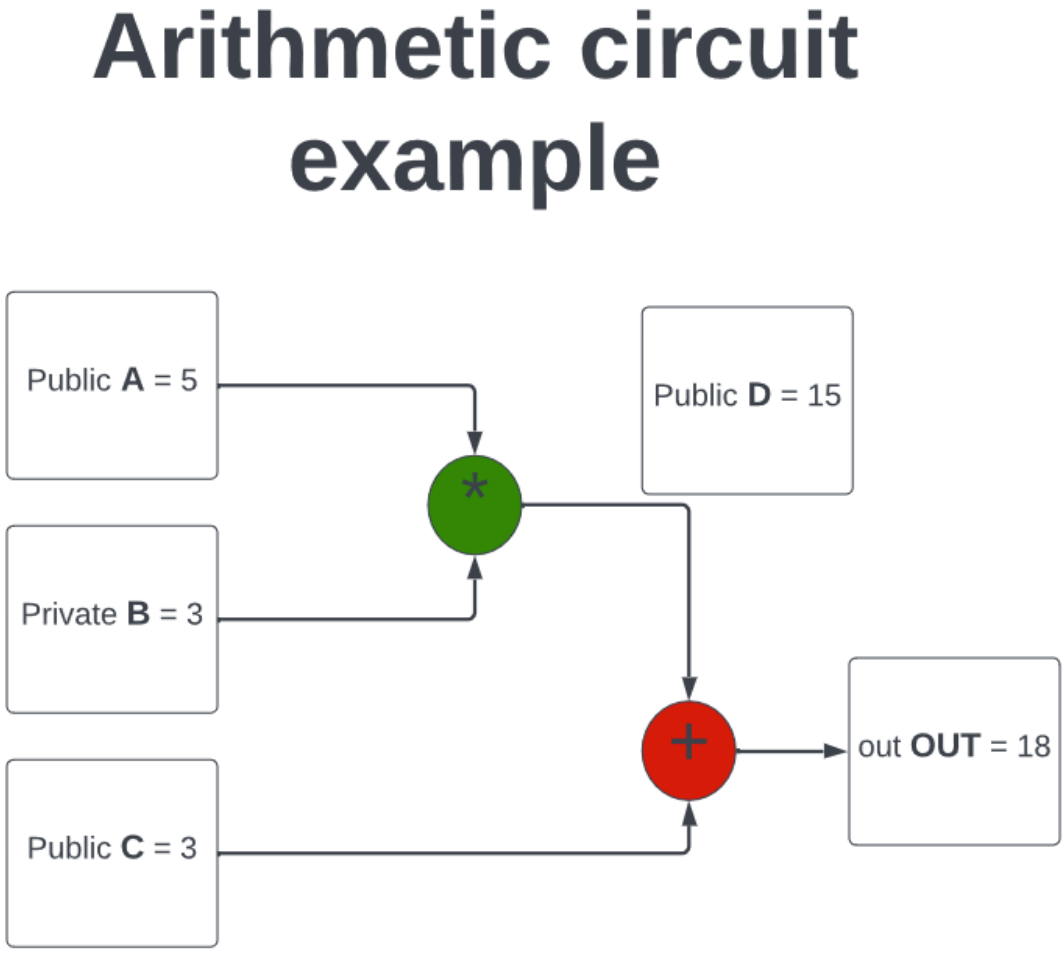


Figure 2. Arithmetic circuit example.

- **Public input signal -** A = 5.
- **Private input signal** - B = 3.
- **Gates -** *, +
- **Intermediate signals -** D = 15(depends on private input B).
- **Output signal (public) -** out C = 18.

On the next page, this Arithmetic circuit will be shown in Circom code.

```
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 Private value.
   signal output out; //Output value of the circuit.

   // 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 most basic circuit for Zero-Knowledge Proofs. In more complex circuits it is way harder to reverse engineer the circuit and its private inputs.

To visualize the circuit in a sense of what is seen by the prover and verifier following diagrams were made:

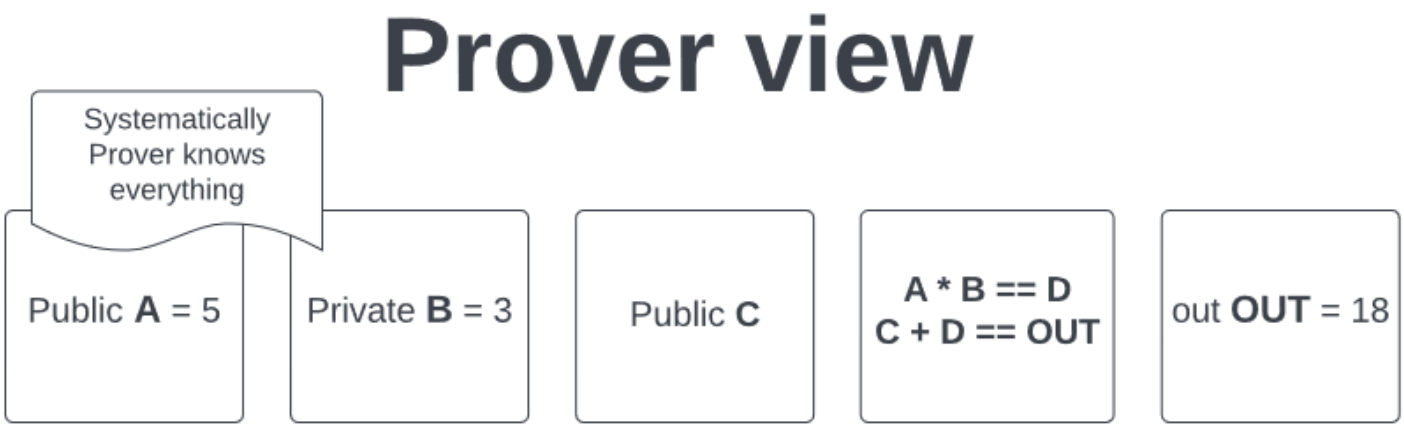


Figure 3. Prover view of the circuit

1

In the view of Prover. He knows everything which is:

- **Public input A** - can be set/given by Application owners.
- **Private input B** - entered/given by Prover/Himself.
- **Public input C** - can be set/given by Application owners.
- **Circuit logic** - It is known if it is open source. (**A * B == C, C +D == OUT**)
- **Output Out** - It is part of the circuit logic.

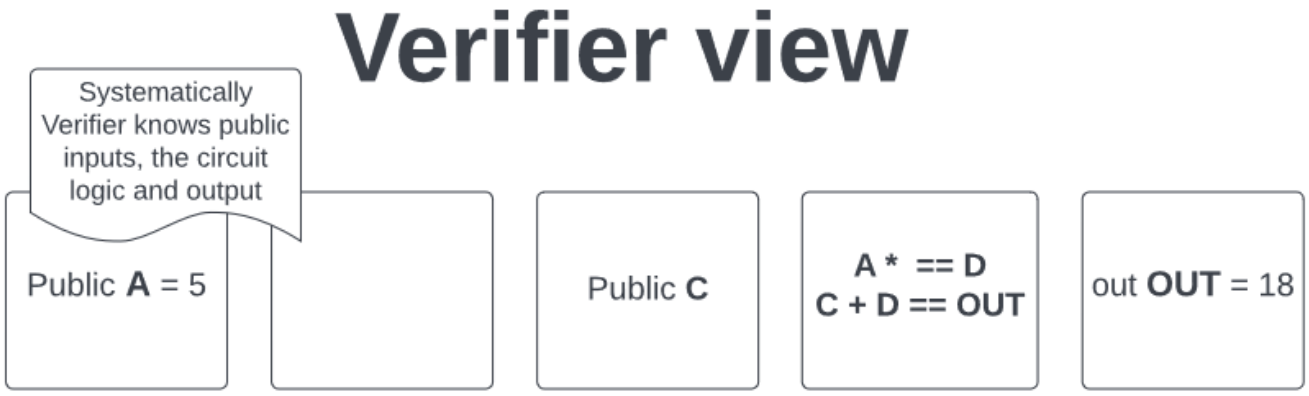


Figure 4. Prover view of the circuit

In the view of the Verifier. He knows only these parts of the circuit:

- **Public input A** - can be set/given by Application owners.
- **Public input C** - can be set/given by Application owners.
- **Circuit logic** - It is known if it is open source. (**A * B == C, C +D == OUT**)
- **Output Out** - It is part of the circuit logic.

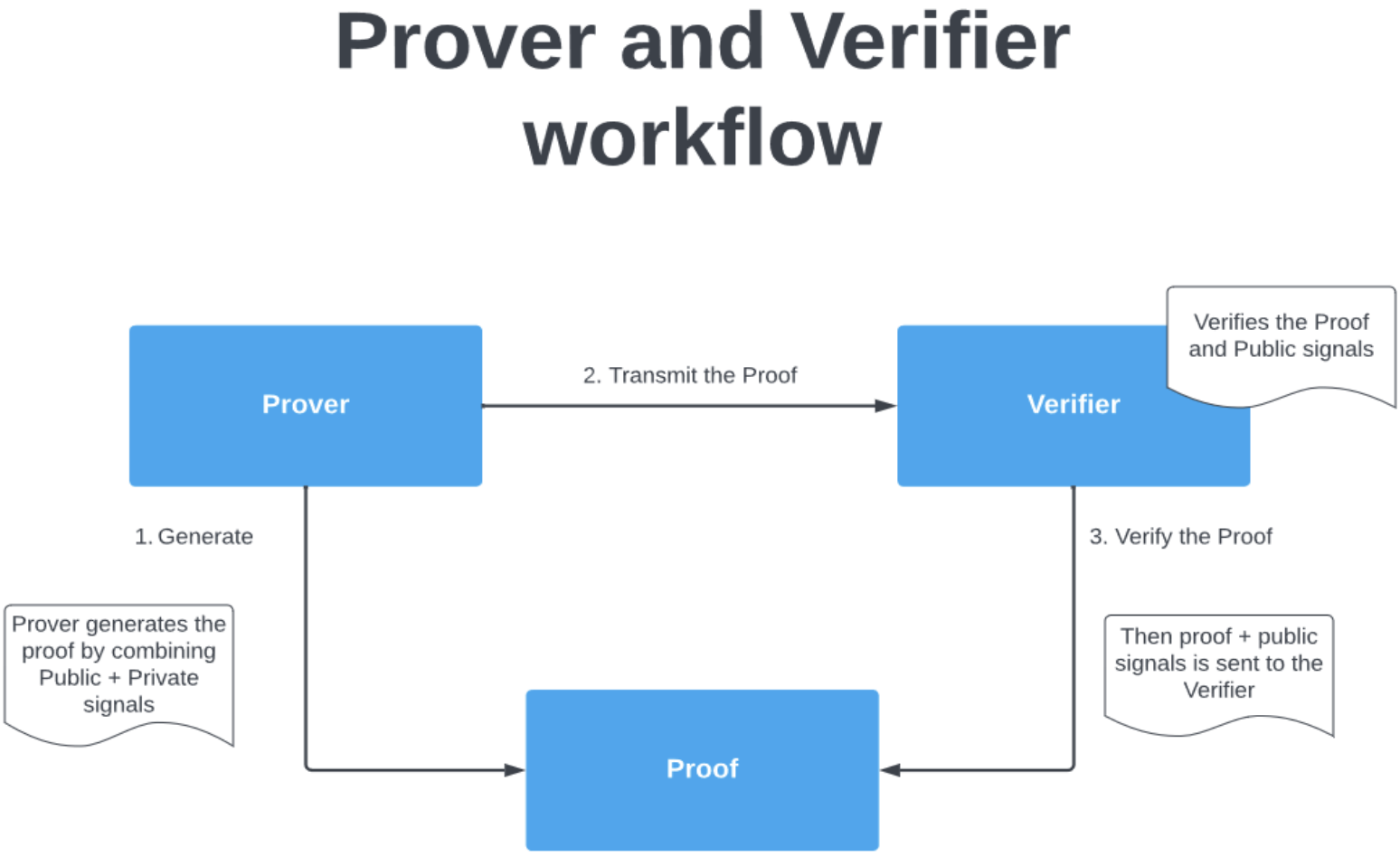


Figure 4. Prover and Verifier workflow

Proof of Concept Circuits

To start with building an appropriate circuit. We have to remember the most common privacy issues in DAOs:

- **Off-chain coercion and voting discrimination.** People might be found in real life and forced to act on other people's incentives. Also, transparent voting can lead to discrimination.
- **The limited talent for hiring** because not everyone like transparent salary flows.
- **Discourage to innovate and experiment** with their developers. They are scared to take risks and innovate because smart contracts are fully transparent, and if they are buggy or hacked, the blame might go to the developers.

Due to time constraints, I chose to tackle the voting privacy problem.

Yes Vote Circuit

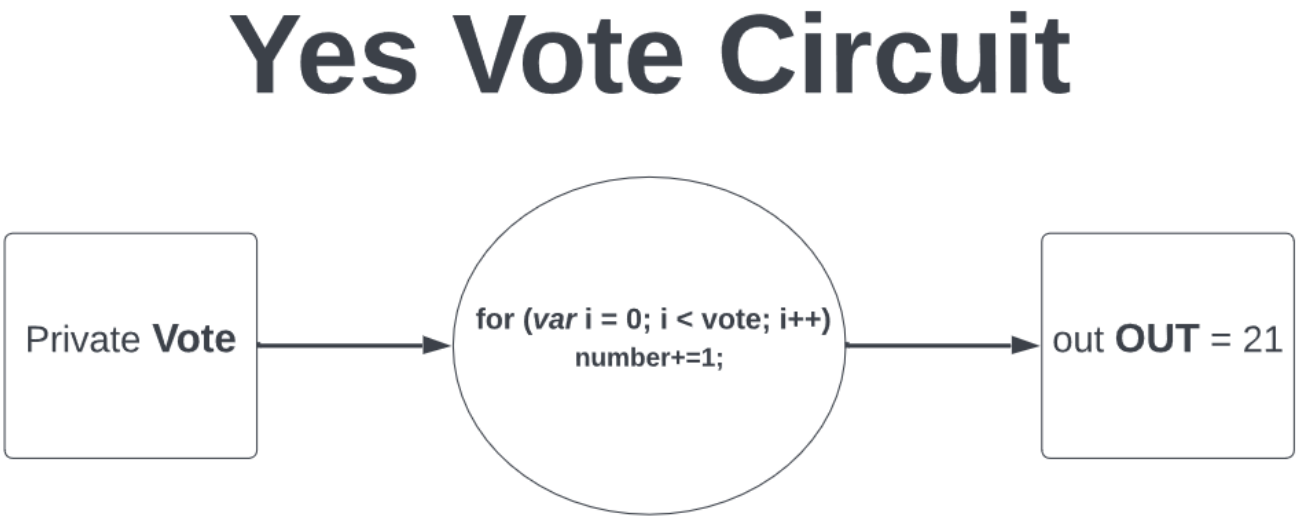


Figure 5. Yes Vote Circuit

In short, this circuit takes the binary representation of the Yes string and calculates the number of bits it has and then compares if it is **21** (1111001 1100101 1110011)

```
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();
```

The idea of this Circuit is to check whenever it is viable to verify strings. It proves that it is possible to do so.

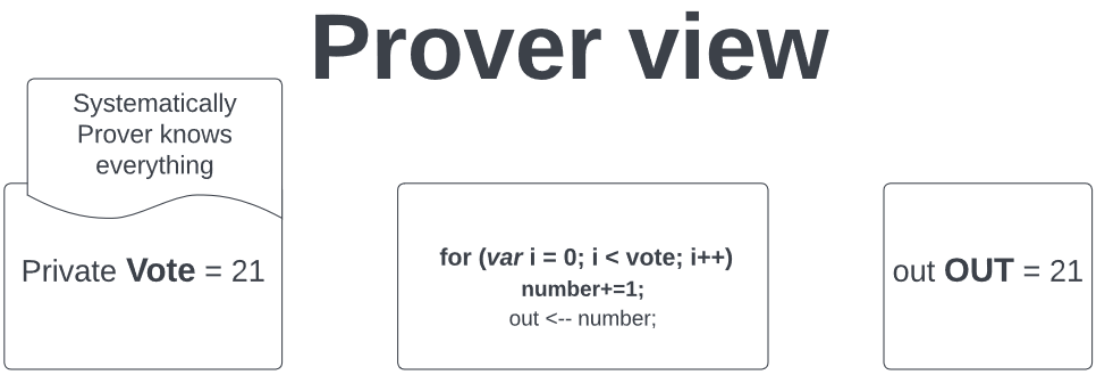


Figure 6. Prover view of the circuit

Once again, Prover knows everything about circuit.

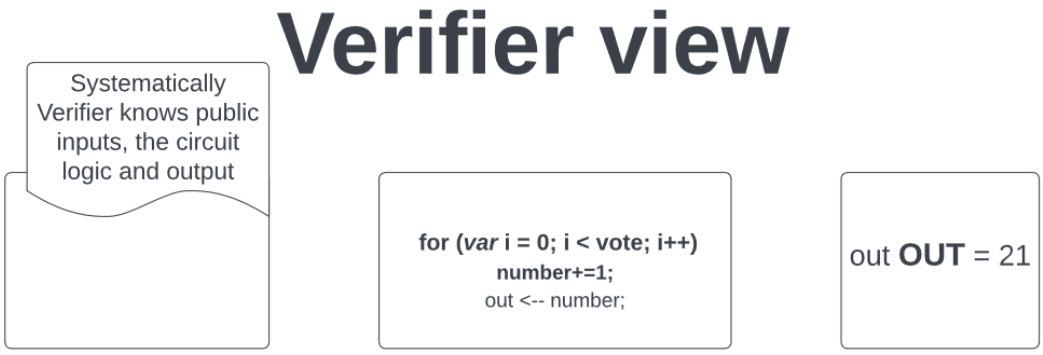


Figure 7. Verifier view of the circuit

Verifier only knows the logic of the circuit.

No Vote Circuit

The circuit which checks No string amount of bits works exactly the same, just that No string consists of 14 bits (1001110 1101111).

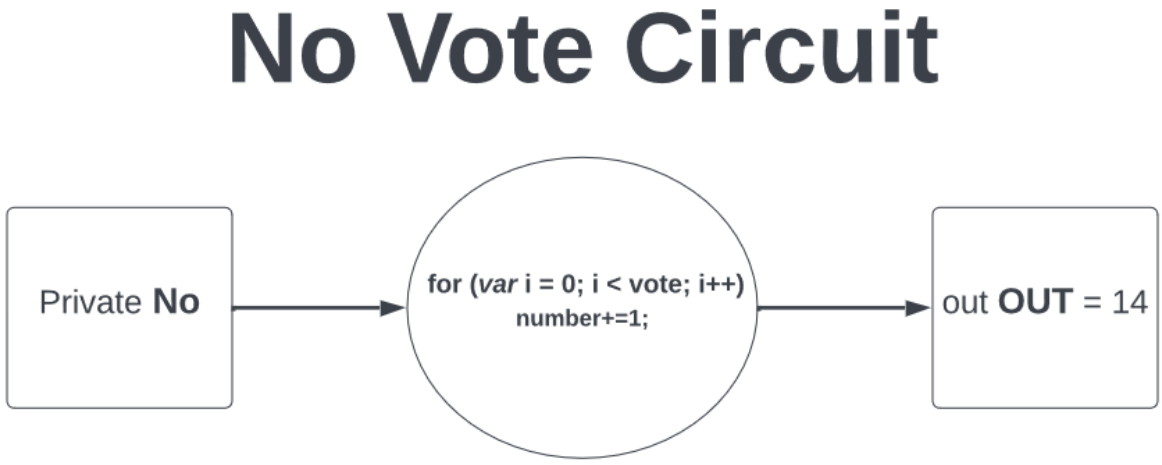


Figure 8. No Vote Circuit

```
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();
```

Finally, the views or the information Prover/Verifier knows about the circuit is exactly the same it was for Vote Yes Circuit.

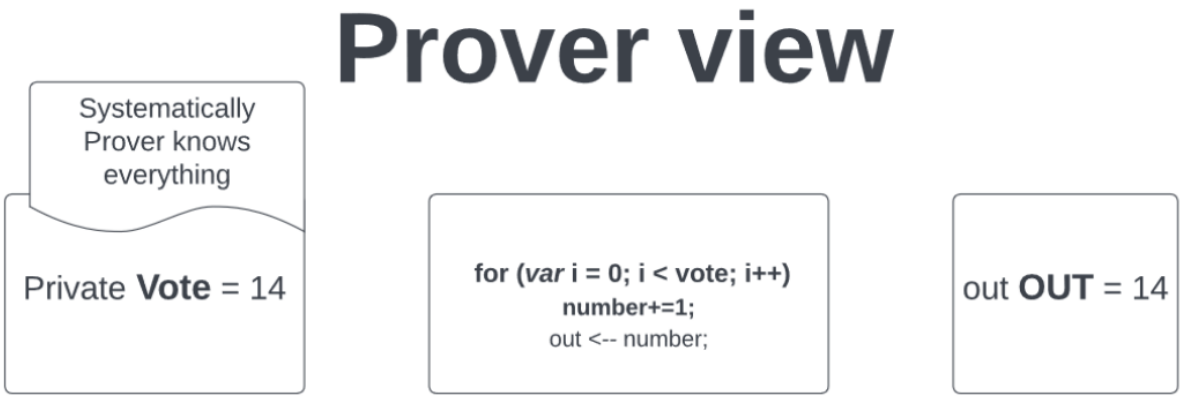


Figure 9. Prover view of the circuit

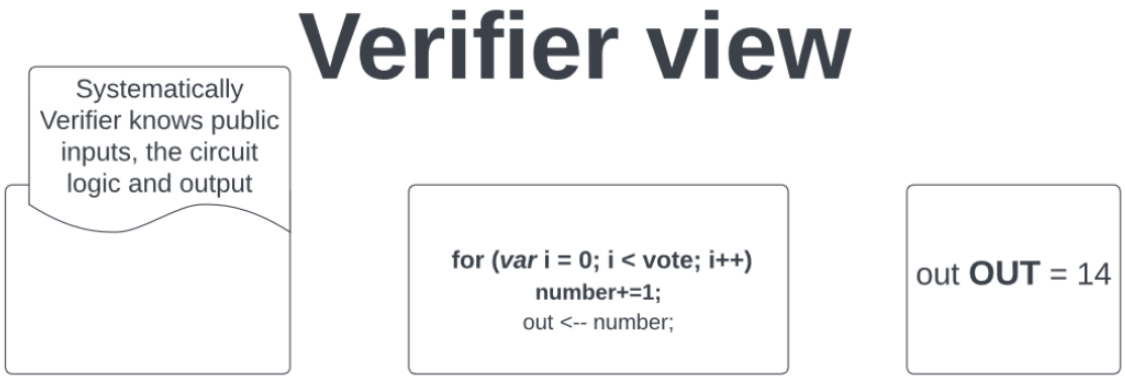


Figure 10. Verifier view of the circuit

Verifier only knows the logic of the circuit.

Summary Circom

Circom is used to translate your problem into an arithmetic circuit which then is used by SnarkJS to generate the needed files for Prover and Verifier. Circom in a mix with SnarkJS lets Prover proves information without revealing it.

3.3 SnarkJS

Introduction

It is **a JavaScript[9] and Pure Web Assembly[10] implementation of zkSNARK**. It can use either the Groth16[11] Protocol or PLONK Protocol[12].

SnarkJS in combination with **Circom** can encrypt your inputs in the smart contracts[13] which are stored on the blockchain and use a zero-knowledge proof to prove that what's encrypted is correct.

Trusted setup

To give some more context on the use of SnarkJS we have to understand some working in parts of **zk-SNARKs** protocol.

zk-SNARKs protocol consists of

Public protocol parameters/Public key - a combination of Proving key and Verifying key

Private protocol parameters/Private key - so called ‘toxic waste’ which lets you generate fake proofs.

In the process of generating protocol public parameters, it also generates private parameters as an intermediary step.

Someone has to generate these keys/parameters and here comes a trusted setup.

MPC - Multi-party computation

To do the trusted setup securely, MPC[15] has to be done. MPC allows multiple parties to make calculations using their combined data, without revealing their individual input.

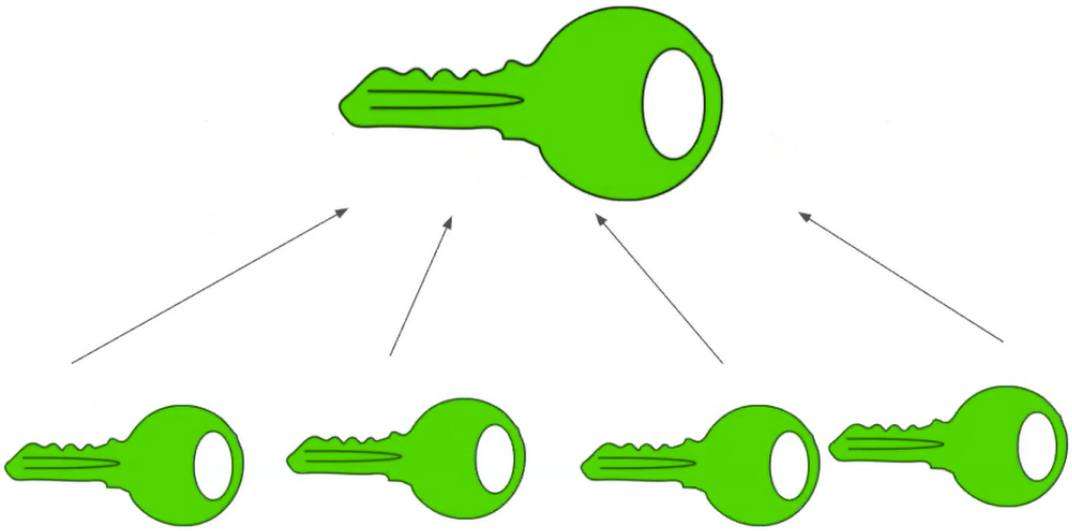


Figure 11. Visualization of zk-Snars Public key generation via MPC

With the help of MPC, zk-SNARKs Public and Private key/parameters are made out of separate ‘shards’. This brings more security to the protocol because instead of one party, multiple parties are generating one Public key and one Private key.

It means that for someone to generate a Private protocol key for creating fake proofs - They have to know all the shards of the Private protocol key and if at least one person was dependable during the MPC stage and delete his private key shard (toxic waste). It is impossible for the other parties in the trusted setup to generate the Private protocol key on their own.

Trusted setup - Phase 1

Trusted setup has 2 Phases. Both phases are for performance and optimization reasons. Starting with Phase 1 is the process of generating the Public Protocol Parameters (Public Key) and Private Protocol Parameters (Private key, toxic waste) no matter the circuit code logic. The baseline. (if a bunch of people - MPC did it once, then everybody can use that from then on)

Trusted setup - Phase 2

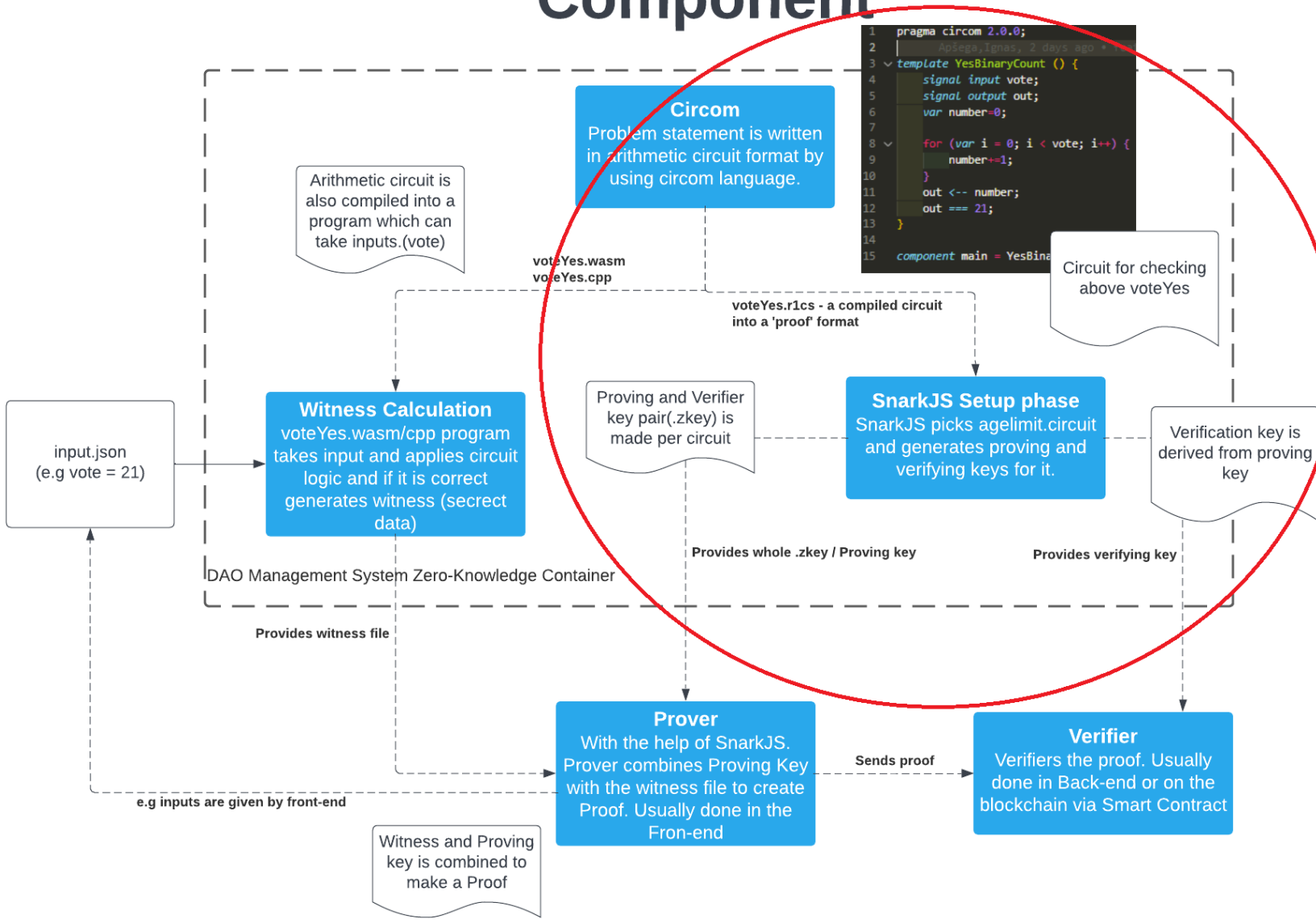
Computational less intensive - this phase is circuit-specific. Once you made changes to your circuit you have to re-do this phase. This phase is applicable only for the Groth16 implementation of Zero-Knowledge.

However, if you made changes in your circuit and your Phase 1 was made in PLONK implementation of Zero-Knowledge. You do not have to do this phase.

GROTH16 does phase 1/phase 2.
PLONK does only phase 1.

On the next page, it will be shown where the Trusted setup exactly happens in Circom/SnarkJS ecosystem.

Circom & SnarkJS flow in Zero-Knowledge Component



Summary SnarkJS

SnarkJS helps to do the Trusted setup phases where it generates:

Public protocol parameters/Public key - a combination of Proving key and Verifying key.
Private protocol parameters/Private key - so called 'toxic waste' which lets you to generate fake proofs.

Prover - gets the **Proving key**. Supplies his input (private data) to the **.wasm program** (shown in the diagram above) which follows circuit logic and if the input meets circuit constrains, the **.wasm program** generates **witness** (correct private signals and calculated intermediary signals). Finally, Prover combines this witness with the **Proving key** and sends it to the **Verifier** to **Verify**.

Conclusions

Circom and SnarkJS are useful libraries of implementing the zk-Snarks architecture of the Zero-Knowledge Proofs. It abstracts the mathematics needed for writing a problem statement with Circom - low level arithmetic circuit language. In addition, SnarkJS helps with the Trusted setup phase to generate the Proving/Verifying keys needed to identify the circuit and bring more security to the architecture.

However, knowing that Zero-Knowledge Proofs are an emerging technology. A lot of documentation on Circom and SnarkJS is not existent. It is not very beginner friendly and a person will need to dig around the internet to figure out some concepts and question themselves. Nonetheless, these libraries had the best documentation in comparison to others and it also has a very active Telegram chat group where questions are asked daily.

All in all, these are great tools for implementing zk-Snarks. These libraries already have been used in released projects and are time-tested solutions.

Currently, the example, vote yes/no circuits and SnarkJS trusted setup files were implemented in one of Byonts web3 application templates. Letting to bootstrap applications based on Zero-Knowledge more quickly.

2

Advise and applicability for User Privacy in DAOs

Vote Yes and **Vote No** could be used to bring more privacy in the DAOs voting function. A vote would be hidden via **Prover - Verifier** interaction. However, the DAOs could still collect the votes without revealing them to the public.

Another use case could be messaging. DAOs conversations usually happens via centralized applications but it should be possible to implement a Zero-Knowledge Proof system via Circom\SnarkJS where the messages are decentralized and stored on the blockchain encrypted via ZKP.

Finally, if decided to make a whole DAO Management tooling app. **Anonymous login** could be another solution to bring more privacy to the DAOs. A solution/library already exists for that called - **Semaphore protocol** - Using zero knowledge, Ethereum users can prove their membership in a group. Moreover, this protocol uses Circom to build its problem statement circuits(prove membership) and Smart Contracts to verify them.

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.

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.

"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.