# Configuration Manual

MSc Research Project
MSc Cybersecurity

Noel Varghese Oommen
Student ID: 23210567

School of Computing
National College of Ireland

Supervisor: Joel Aleburu

**National College of Ireland**

**MSc Project Submission Sheet**

**School of Computing**

|  |  |  |  |
|---|---|---|---|
| **Student Name:** | Noel Varghese Oommen | | |
| **Student ID:** | 23210567 | | |
| **Programme:** | MSc Cybersecurity | **Year:** | 2025 |
| **Module:** | Practicum Part 2 | | |
| **Lecturer:** | Joel Aleburu | | |
| **Submission Due Date:** | 29/01/2025 | | |
| **Project Title:** | An Evaluation of Privacy Enhancing Technologies for Blockchain Based Voting | | |
| **Word Count:** | 1087 | **Page Count:** | 13 |

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** *Noel*

**Date:** 29/01/2025

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

|  |  |
|---|---|
| Attach a completed copy of this sheet to each project (including multiple copies) | □ |
| **Attach a Moodle submission receipt of the online project submission,** to each project (including multiple copies). | □ |
| **You must ensure that you retain a HARD COPY of the project**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. | □ |

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

| **Office Use Only** |  |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# Configuration Manual

Noel Varghese Oommen
Student ID: 23210567

## 1 Introduction

This configuration manual details the software and hardware specifications and versions used for setting up the experimental setup. Three separate experients were carried out for different scenarios. The sections below explains the step by step process for implementing each of them along with the hardware specifications, and the tools used for them.

## 2 Experimental Setup

This experiment was carried out on my personal laptop, having the following specifications:

- Laptop Model: HP Pavillion TPN-Q191
- Operating System: Windows 10  Home, Version 22H2, OS build: 19045.5131
- Processor: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz   2.90 GHz
- RAM: 16GB
- System Type: 64-bit operating system, x64-based processor

Some of the testing's were carried out in a virtual machine with the following system specifications:

- Hypervisor: Oracle Virtual Box Version 7.0.20 r163906 (Qt5.15.2)
- Operating System: Ubuntu 24.04.1 LTS (x64)
- RAM: 11GB
- Cores Allocated: 2
- Storage Allocated 90GB

## 3 Technologies and Software used for Implementation

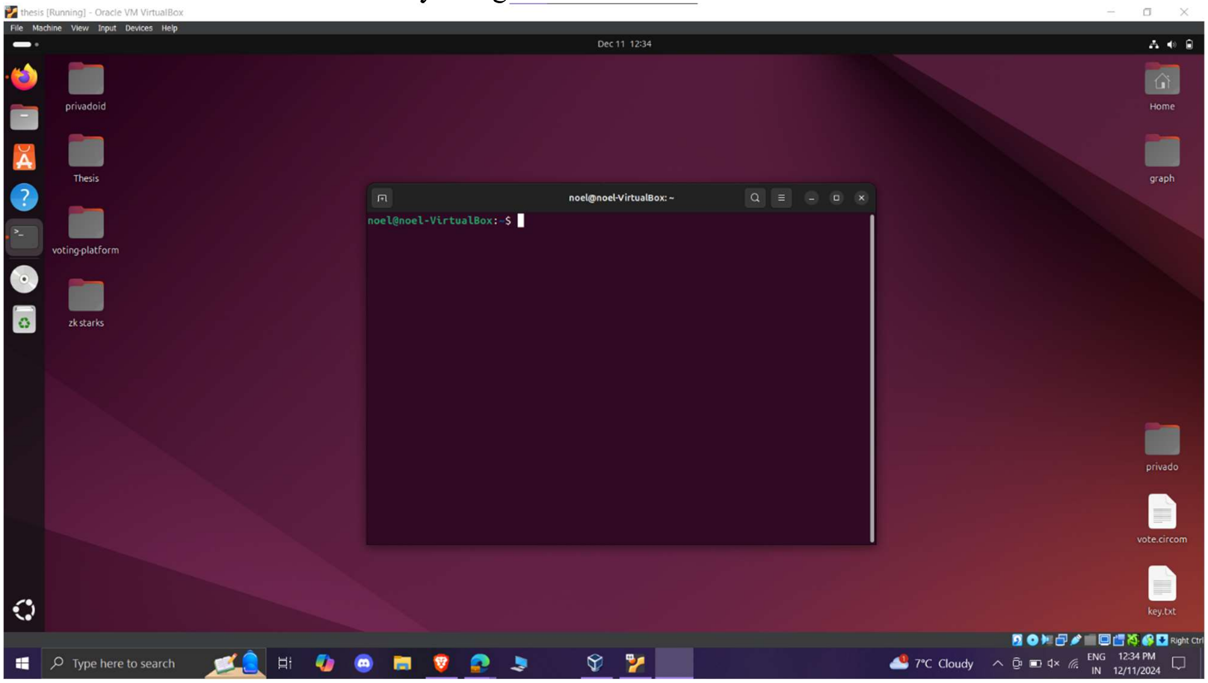
- **Virtual Box:** Oracle Virtual Box Version 7.0.20 r163906 (Qt5.15.2)
- **Remix IDE:** An online Ethereum IDE for compiling and deploying solidity code (remix.ethereum.org, n.d.).
- **MetaMask Wallet:** Installed as a browser plugin to interact with smart contracts.
- **Circom:** It is a compiler that can run circuits written in the Circom programing language and helps in generating the proof and verifying it (Iden3.io, 2025)
- **SnarkJS:** The proof generator
- **NodeJS:** It is a java script run time environment
- **PrivadoID Issuer:** Self Sovereign Identity credential issuer (Privado.id, 2024).
- **PrivadoID Wallet:** Self Sovereign Identity credential wallet (Privado.id, 2024).

## 4 Common Steps for all three implementations

Although the experiment is done as three separate parts, there are a few steps that are common for all of them. That are:

- Running Ubuntu Virtual Machine in the Windows 10 operating system:

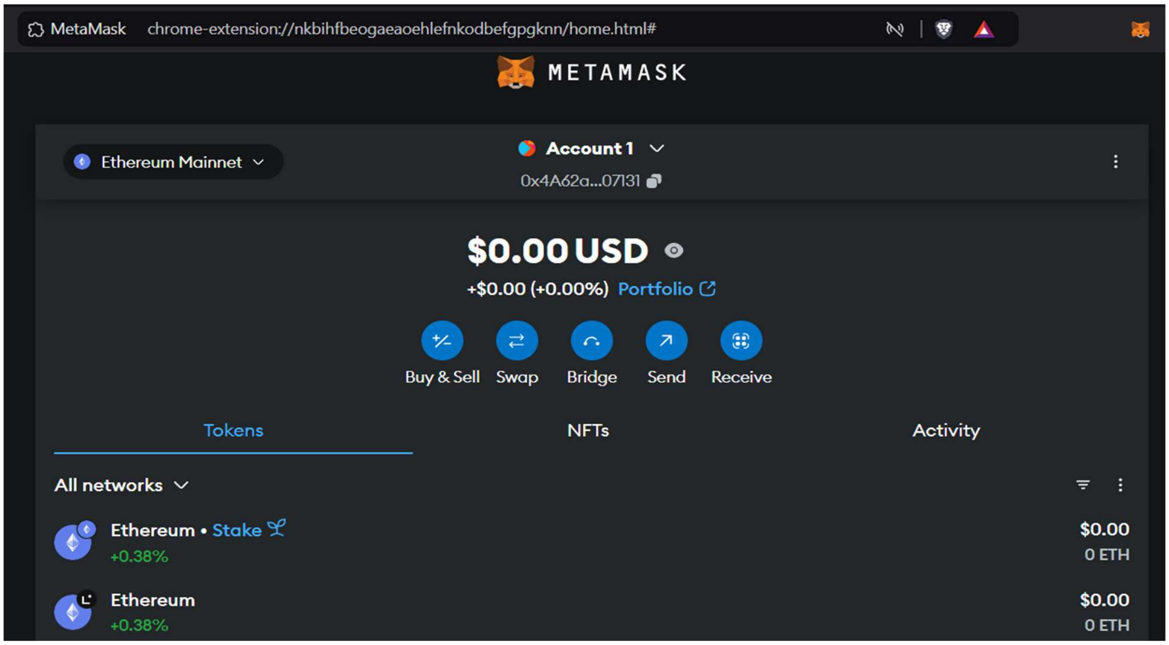
Ubuntu was installed because Circom, the tool used for creating the zero knowledge proofs is better optimized for the Linux operating system. Any flavour of Linux can be used instead. I chose Ubuntu due to my familiarity with the OS. Circom can also be executed on Windows by using Docker containers.



```
noel@noel-VirtualBox:~$ 
```

**Figure 1**

- Installing and creating a wallet on MetaMask:
  A wallet address was created on the Ethereum blockchain and the secret key of the wallet was securely stored.



MetaMask chrome-extension://nkbihfbeogaeaoehlefnkodbefgpgknn/home.html#

Ethereum Mainnet

Account 1

0x4A62a...07131

$0.00 USD

+$0.00 (+0.00%) Portfolio

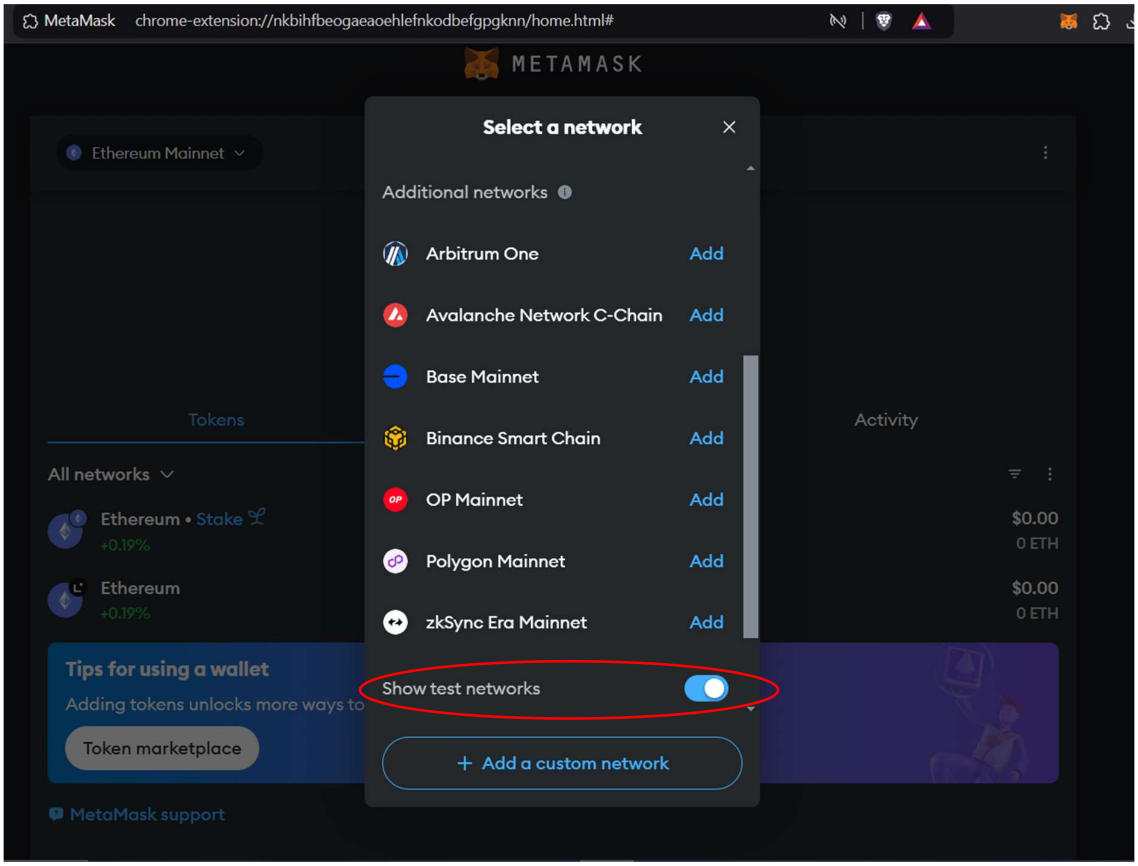
All networks

Ethereum • Stake +0.38% $0.00 0 ETH

Ethereum +0.38% $0.00 0 ETH

**Figure 2**

- Make sure to enable “Show test networks” in the network list



Show test networks ●

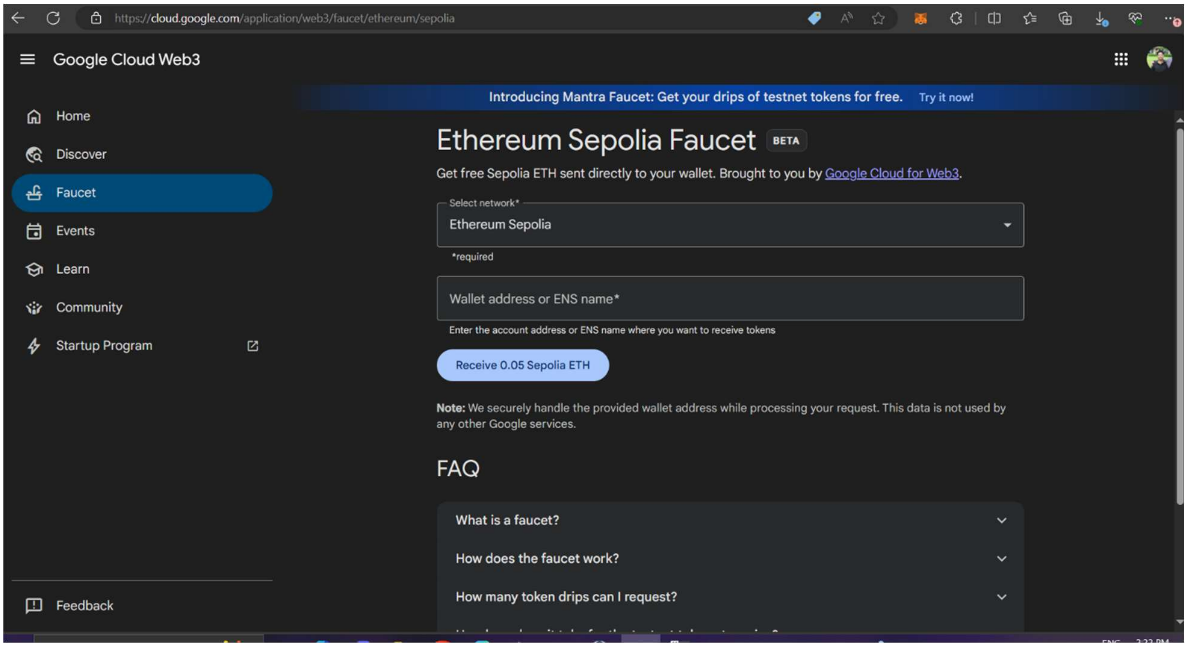
**Figure 3**

- Connect to Ethereum’s Sepolia test network
  The test networks are not listed by default. Click on Add custom network to add the our custom network and fill out the following network configuration details for sepolia:
  Network Name: Sepolia test network
  New RPC URL: https://sepolia.infura.io/v3/
  Chain ID: 11155111
  Currency Symbol: SepoliaETH
  Block Explorer URL: https://sepolia.etherscan.io

- Connect to Polygon’s Amoy test network:
  Fill in with following network configuration details to connect to the Amoy test network:
  Network Name: POLYGON AMOY TESTNET
  New RPC URL: https://rpc-amoy.polygon.technology/
  Chain ID: 80002
  Currency Symbol: POL
  Block Explorer URL: https://www.oklink.com/amoy

- Get test tokens from the Sepolia faucet

Once the network is connected, get the tokens from the Sepolia faucet. The tokens are needed for confirming transactions in the blockchain. Copy and paste the wallet address to receive 0.05 ETH tokens to the wallet. The faucet only provides 1 transaction every 24 hours.


https://cloud.google.com/application/web3/faucet/ethereum/sepolia
Google Cloud Web3
Home
Discover
Faucet
Events
Learn
Community
Startup Program
Feedback
Introducing Mantra Faucet: Get your drips of testnet tokens for free. Try it now!
BETA
Select network*
*required
Receive 0.05 Sepolia ETH

Ethereum Sepolia Faucet

Get free Sepolia ETH sent directly to your wallet. Brought to you by Google Cloud for Web3.

Select network*: Ethereum Sepolia

Wallet address or ENS name*

Enter the account address or ENS name where you want to receive tokens

**Note:** We securely handle the provided wallet address while processing your request. This data is not used by any other Google services.

FAQ

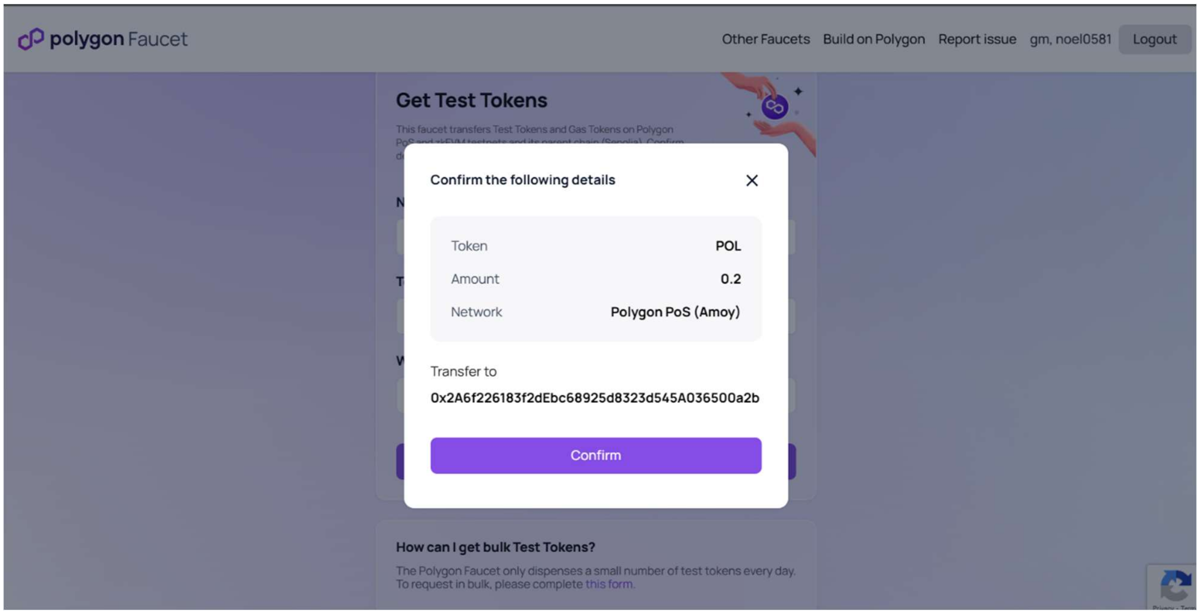
What is a faucet?

How does the faucet work?

How many token drips can I request?

**Figure 4**

- Get test tokens from the Polygon faucet:
  Exactly like the Sepolia faucet, copy and paste the wallet address to get 0.2 token. But as we are doing rigorous testing, we need more tokens. We can apply for bulk tokens sending a request to the polygon team. I received 100 test tokens from polygon support.



Get Test Tokens

This faucet transfers Test Tokens and Gas Tokens on Polygon PoS and zkEVM testnets and its parent chain (Sepolia). Confirm

Confirm the following details

Token POL

Amount 0.2

Network Polygon PoS (Amoy)

Transfer to

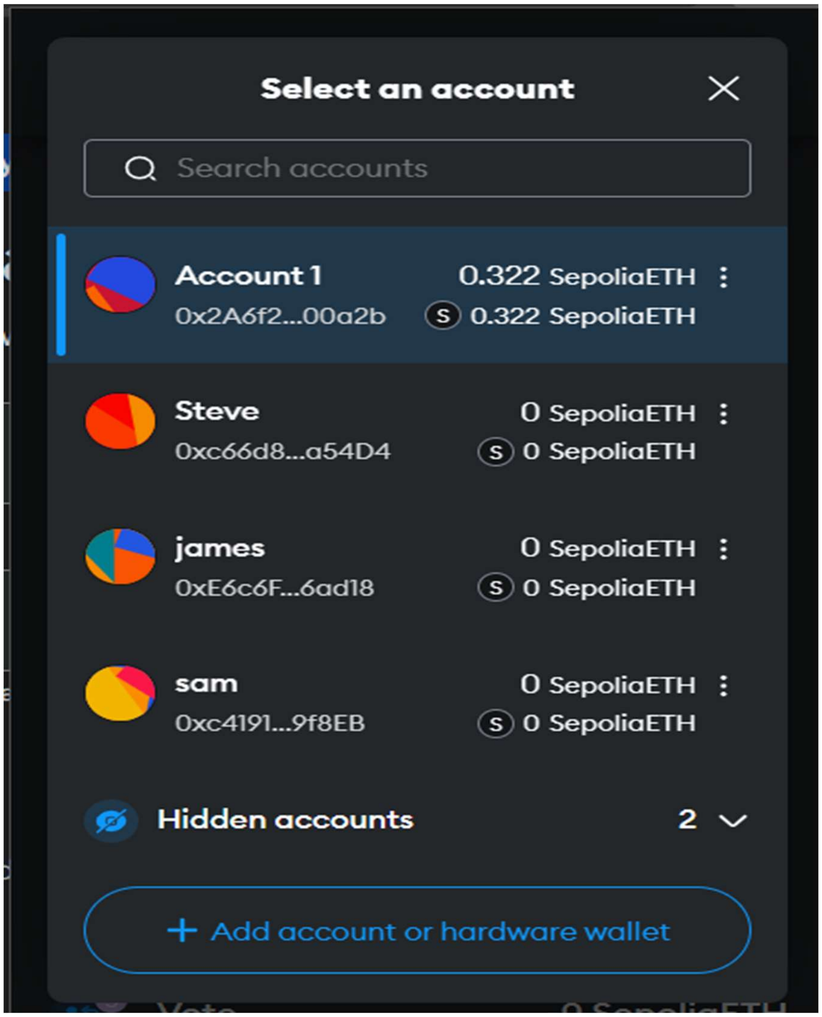
0x2A6f226183f2dEbc68925d8323d545A036500a2b

How can I get bulk Test Tokens?

The Polygon Faucet only dispenses a small number of test tokens every day. To request in bulk, please complete this form.

**Figure 5**

- Create candidate profiles in the wallet:
  Create multiple sample accounts in the MetaMask wallet to simulate voting. We will require multiple accounts as only a single vote is allowed from a wallet address.



Account 1 0.322 SepoliaETH
0x2A6f2...00a2b S 0.322 SepoliaETH

Steve 0 SepoliaETH
0xc66d8...a54D4 S 0 SepoliaETH

james 0 SepoliaETH
0xE6c6F...6ad18 S 0 SepoliaETH

sam 0 SepoliaETH
0xc4191...9f8EB S 0 SepoliaETH

Hidden accounts 2

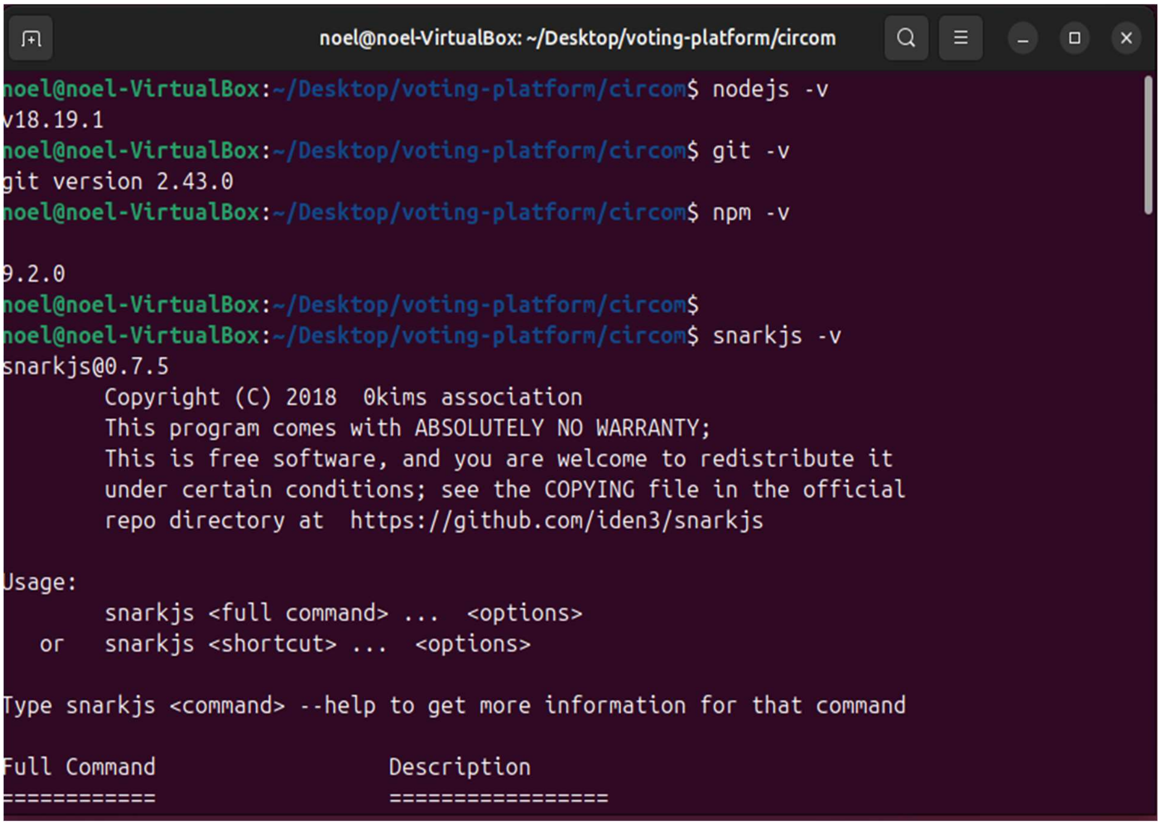
Figure 6

## 5 Proof and Verifier Generation

Install the prerequisites:

- **Node.js (v14 or later):** sudo apt install nodejs npm -y
- **Npm**: It comes with Node.js
- **Git**: sudo apt install git -y
- **SnarkJS**: A library to work with zk-SNARK proofs. npm install -g snarkjs

Following the above commands we can install all the prerequisites needed to run Circom. The versions of the respective software's installed are given below.

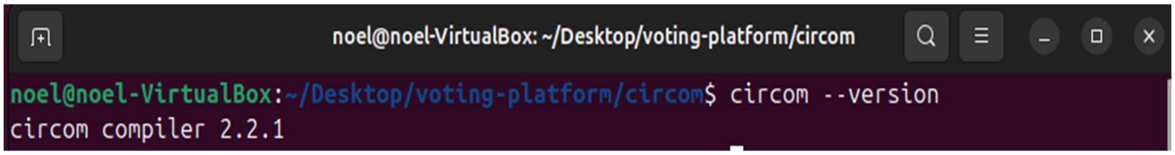


```
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$ nodejs -v
v18.19.1
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$ git -v
git version 2.43.0
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$ npm -v

9.2.0
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$ snarkjs -v
snarkjs@0.7.5
        Copyright (C) 2018  0kims association
        This program comes with ABSOLUTELY NO WARRANTY;
        This is free software, and you are welcome to redistribute it
        under certain conditions; see the COPYING file in the official
        repo directory at  https://github.com/iden3/snarkjs

Usage:
        snarkjs <full command> ...  <options>
   or   snarkjs <shortcut> ...  <options>

Type snarkjs <command> --help to get more information for that command

Full Command                  Description
============                  =================
```

**Figure 7**

Now that we have the prerequisites, install Circom:

- Clone the repository: git clone https://github.com/iden3/circom.git
- Move into the directory: cd circom
- Build the Circom binary: cargo build –release

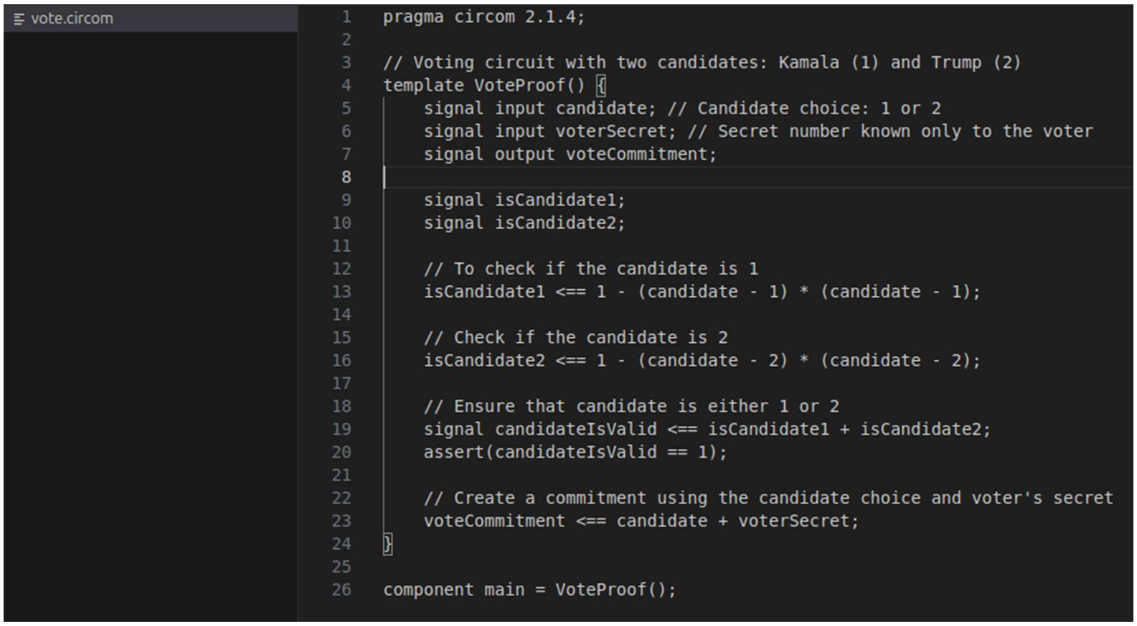


```
noel@noel-VirtualBox:~/Desktop/voting-platform/circom$ circom --version
circom compiler 2.2.1
```

**Figure 8**

Ensure that all the prerequisites are installed by checking the versions
Now that we have all the tools, we can write the circuit and create our proof

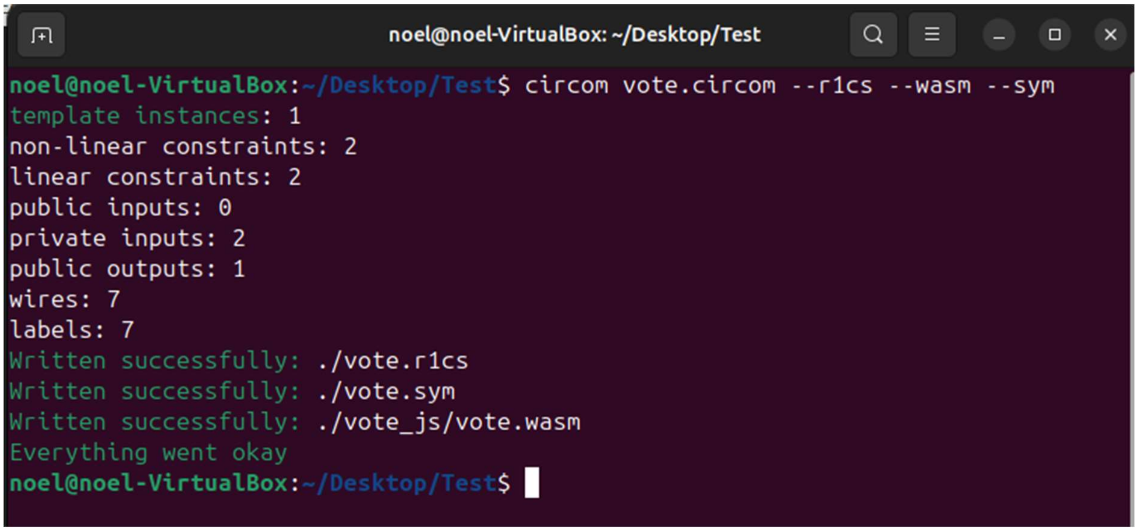
5.1 Write the circuit: Create a file called vote.circom with the circuit logic



```
pragma circom 2.1.4;

// Voting circuit with two candidates: Kamala (1) and Trump (2)
template VoteProof() {
    signal input candidate; // Candidate choice: 1 or 2
    signal input voterSecret; // Secret number known only to the voter
    signal output voteCommitment;

    signal isCandidate1;
    signal isCandidate2;

    // To check if the candidate is 1
    isCandidate1 <== 1 - (candidate - 1) * (candidate - 1);

    // Check if the candidate is 2
    isCandidate2 <== 1 - (candidate - 2) * (candidate - 2);

    // Ensure that candidate is either 1 or 2
    signal candidateIsValid <== isCandidate1 + isCandidate2;
    assert(candidateIsValid == 1);

    // Create a commitment using the candidate choice and voter's secret
    voteCommitment <== candidate + voterSecret;
}

component main = VoteProof();
```

**Figure 9**

5.2 Compile the circuit:

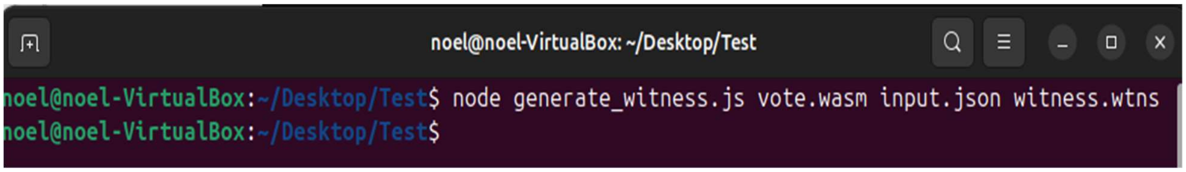


```
noel@noel-VirtualBox:~/Desktop/Test$ circom vote.circom --r1cs --wasm --sym
template instances: 1
non-linear constraints: 2
linear constraints: 2
public inputs: 0
private inputs: 2
public outputs: 1
wires: 7
labels: 7
Written successfully: ./vote.r1cs
Written successfully: ./vote.sym
Written successfully: ./vote_js/vote.wasm
Everything went okay
noel@noel-VirtualBox:~/Desktop/Test$
```

**Figure 10**

5.3 Create an input file called input.json specifying the choice of candidate

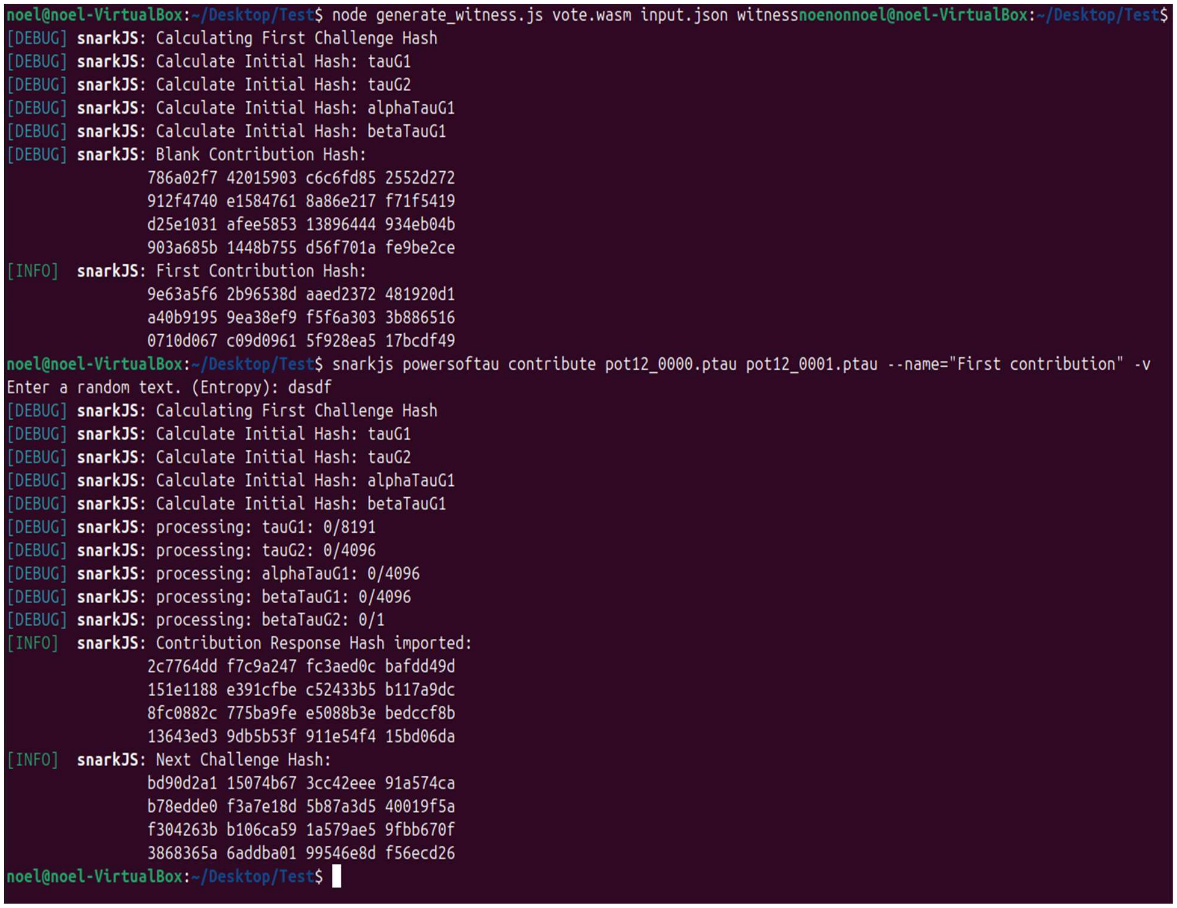
5.4 Generate a witness of the input:



```
noel@noel-VirtualBox:~/Desktop/Test$ node generate_witness.js vote.wasm input.json witness.wtns
noel@noel-VirtualBox:~/Desktop/Test$
```

**Figure 11**

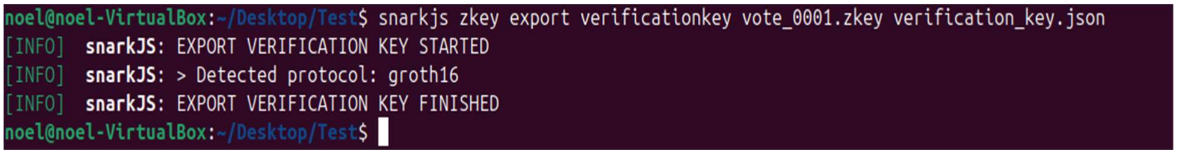
5.5 Now we start with the trusted setup process. In this we generate power of Tau twice. Powers of Tau is a cryptographic parameter generated to make the proof uncompromisable.



```
noel@noel-VirtualBox:~/Desktop/Test$ node generate_witness.js vote.wasm input.json witnessnoenonnoel@noel-VirtualBox:~/Desktop/Test$
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: Blank Contribution Hash:
                786a02f7 42015903 c6c6fd85 2552d272
                912f4740 e1584761 8a86e217 f71f5419
                d25e1031 afee5853 13896444 934eb04b
                903a685b 1448b755 d56f701a fe9be2ce
[INFO]  snarkJS: First Contribution Hash:
                9e63a5f6 2b96538d aaed2372 481920d1
                a40b9195 9ea38ef9 f5f6a303 3b886516
                0710d067 c09d0961 5f928ea5 17bcdf49
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs powersoftau contribute pot12_0000.ptau pot12_0001.ptau --name="First contribution" -v
Enter a random text. (Entropy): dasdf
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: processing: tauG1: 0/8191
[DEBUG] snarkJS: processing: tauG2: 0/4096
[DEBUG] snarkJS: processing: alphaTauG1: 0/4096
[DEBUG] snarkJS: processing: betaTauG1: 0/4096
[DEBUG] snarkJS: processing: betaTauG2: 0/1
[INFO]  snarkJS: Contribution Response Hash imported:
                2c7764dd f7c9a247 fc3aed0c bafdd49d
                151e1188 e391cfbe c52433b5 b117a9dc
                8fc0882c 775ba9fe e5088b3e bedccf8b
                13643ed3 9db5b53f 911e54f4 15bd06da
[INFO]  snarkJS: Next Challenge Hash:
                bd90d2a1 15074b67 3cc42eee 91a574ca
                b78edde0 f3a7e18d 5b87a3d5 40019f5a
                f304263b b106ca59 1a579ae5 9fbb670f
                3868365a 6addba01 99546e8d f56ecd26
noel@noel-VirtualBox:~/Desktop/Test$
```

**Figure 12**

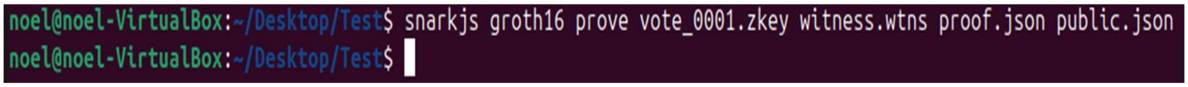
5.6 Export the verification key:



```
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs zkey export verificationkey vote_0001.zkey verification_key.json
[INFO]  snarkJS: EXPORT VERIFICATION KEY STARTED
[INFO]  snarkJS: > Detected protocol: groth16
[INFO]  snarkJS: EXPORT VERIFICATION KEY FINISHED
noel@noel-VirtualBox:~/Desktop/Test$
```

**Figure 13**

5.7 Generate Proof

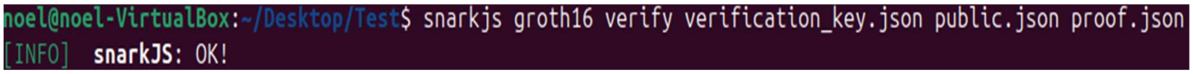


```
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs groth16 prove vote_0001.zkey witness.wtns proof.json public.json
noel@noel-VirtualBox:~/Desktop/Test$ 
```

**Figure 14**

The proof was succesfully generated and stored in the circom folder

5.8 Verify the proof: Now that we have the proof, we can verify it

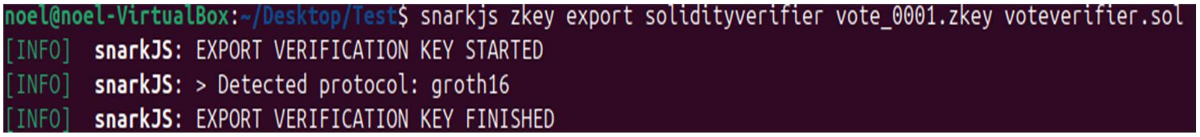


```
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs groth16 verify verification_key.json public.json proof.json
[INFO]  snarkJS: OK!
```

**Figure 15**

The OK means that the proof verifies that the choice of candidate is 1 i.e. Kamala

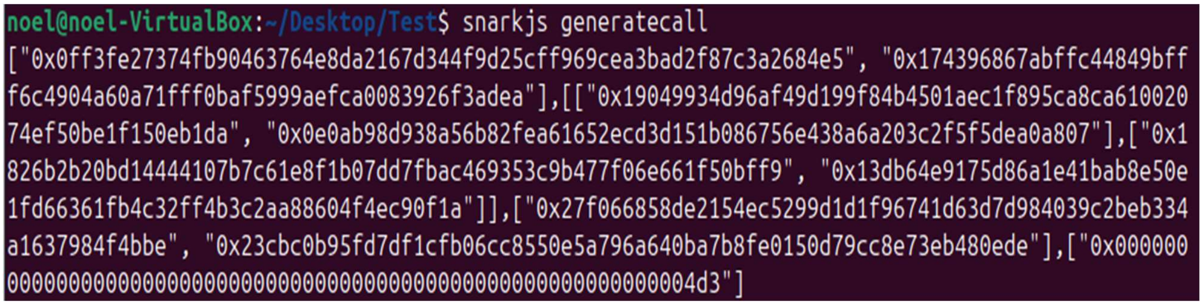
5.9 generate the solidity code for on chain verification



```
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs zkey export solidityverifier vote_0001.zkey voteverifier.sol
[INFO]  snarkJS: EXPORT VERIFICATION KEY STARTED
[INFO]  snarkJS: > Detected protocol: groth16
[INFO]  snarkJS: EXPORT VERIFICATION KEY FINISHED
```

**Figure 16**

5.10 Generate the proof text



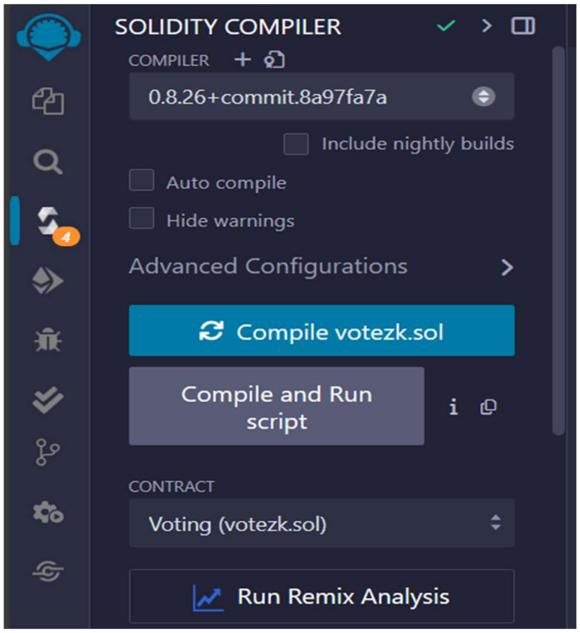
```
noel@noel-VirtualBox:~/Desktop/Test$ snarkjs generatecall
["0x0ff3fe27374fb90463764e8da2167d344f9d25cff969cea3bad2f87c3a2684e5", "0x174396867abffc44849bff
f6c4904a60a71fff0baf5999aefca0083926f3adea"],[["0x19049934d96af49d199f84b4501aec1f895ca8ca610020
74ef50be1f150eb1da", "0x0e0ab98d938a56b82fea61652ecd3d151b086756e438a6a203c2f5f5dea0a807"],["0x1
826b2b20bd14444107b7c61e8f1b07dd7fbac469353c9b477f06e661f50bff9", "0x13db64e9175d86a1e41bab8e50e
1fd66361fb4c32ff4b3c2aa88604f4ec90f1a"]],["0x27f066858de2154ec5299d1d1f96741d63d7d984039c2beb334
a1637984f4bbe", "0x23cbc0b95fd7df1cfb06cc8550e5a796a640ba7b8fe0150d79cc8e73eb480ede"],["0x000000
00000000000000000000000000000000000000000000000000000004d3"]
```

**Figure 17**

This proof and verifier codes can then be used for creating the zero-knowledge voting platform.

5.11 Remix smart contract deployment

To run the code, first compile the code, using the Remix compiler



SOLIDITY COMPILER

COMPILER

0.8.26+commit.8a97fa7a

□ Include nightly builds

□ Auto compile

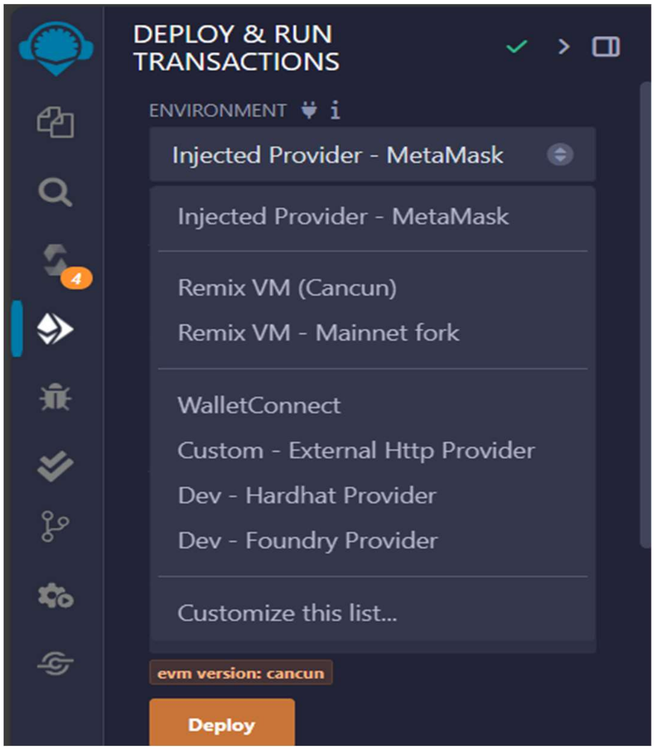
□ Hide warnings

CONTRACT

Voting (votezk.sol)

**Figure 18**

On the contract deployer, select “Injected Provider - MetaMask” to connect and deploy the code with your wallet address



DEPLOY & RUN TRANSACTIONS

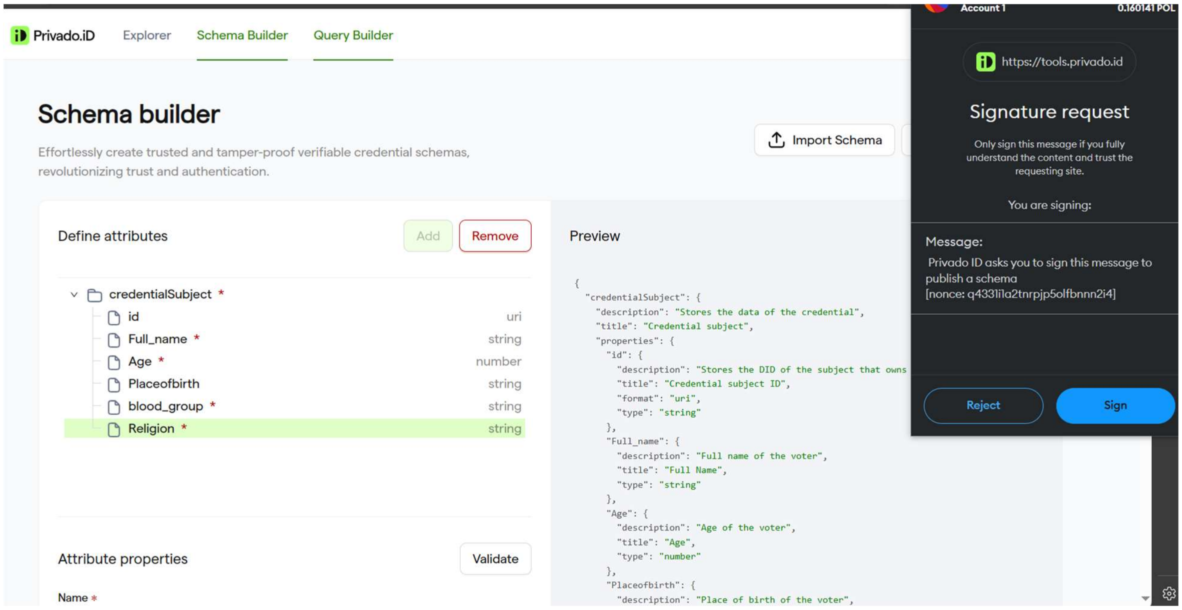
ENVIRONMENT

Injected Provider - MetaMask

**Figure 19**

# 6 Self-Sovereign Identity Implementation

## 6.1 Build the Schema, sign and deploy it



Schema builder

Effortlessly create trusted and tamper-proof verifiable credential schemas, revolutionizing trust and authentication.

Define attributes

| | |
|---|---|
| credentialSubject * | |
| id | uri |
| Full_name * | string |
| Age * | number |
| Placeofbirth | string |
| blood_group * | string |
| Religion * | string |

Attribute properties

Name *

Preview

```
{
  "credentialSubject": {
    "description": "Stores the data of the credential",
    "title": "Credential subject",
    "properties": {
      "id": {
        "description": "Stores the DID of the subject that owns
        "title": "Credential subject ID",
        "format": "uri",
        "type": "string"
      },
      "Full_name": {
        "description": "Full name of the voter",
        "title": "Full Name",
        "type": "string"
      },
      "Age": {
        "description": "Age of the voter",
        "title": "Age",
        "type": "number"
      },
      "Placeofbirth": {
        "description": "Place of birth of the voter",
```

https://tools.privado.id

Signature request

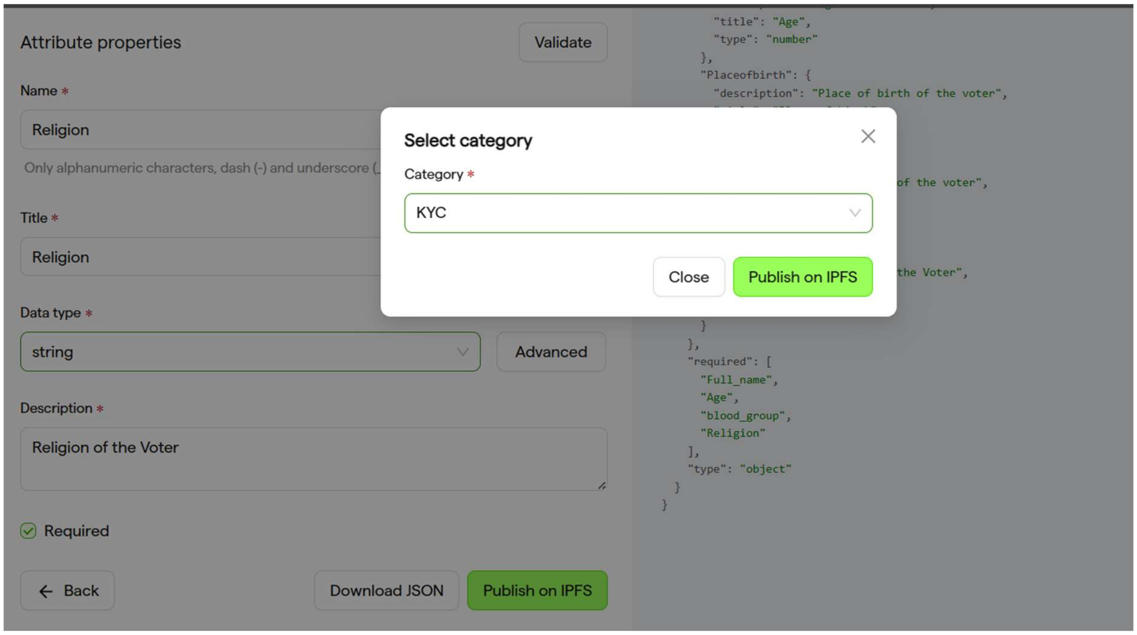
Only sign this message if you fully understand the content and trust the requesting site.

You are signing:

Message:
Privado ID asks you to sign this message to publish a schema
[nonce: q4331i1a2tnrpjp5olfbnnn2i4]

**Figure 20**

## 6.2 Choose the category as KYC



Attribute properties

Name *

Religion

Only alphanumeric characters, dash (-) and underscore (

Title *

Religion

Data type *

string

Description *

Religion of the Voter

☑ Required

Select category

Category *

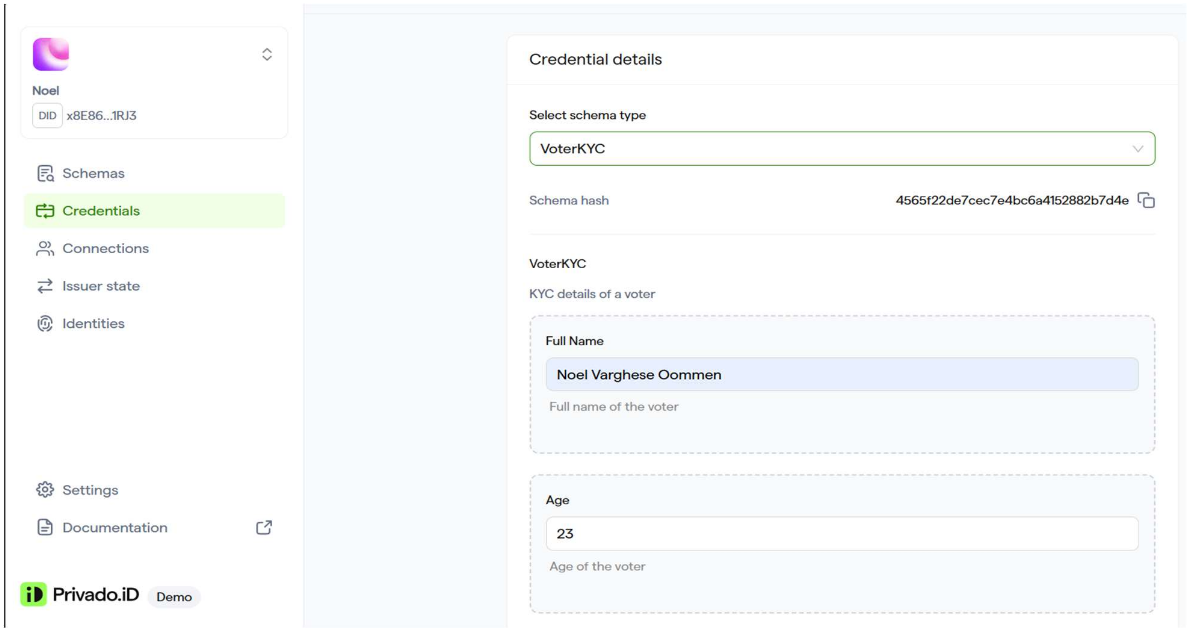
KYC

```
        "title": "Age",
        "type": "number"
      },
      "Placeofbirth": {
        "description": "Place of birth of the voter",
                              of the voter",
                              the Voter",
        }
      },
      "required": [
        "Full_name",
        "Age",
        "blood_group",
        "Religion"
      ],
      "type": "object"
    }
  }
```

**Figure 21**

6.3 Import the deployed schema to issue credentials with it

6.4 Fill up the details and issue the credentials



Credential details

Select schema type: VoterKYC

Schema hash: 4565f22de7cec7e4bc6a4152882b7d4e

VoterKYC

KYC details of a voter

Full Name: Noel Varghese Oommen

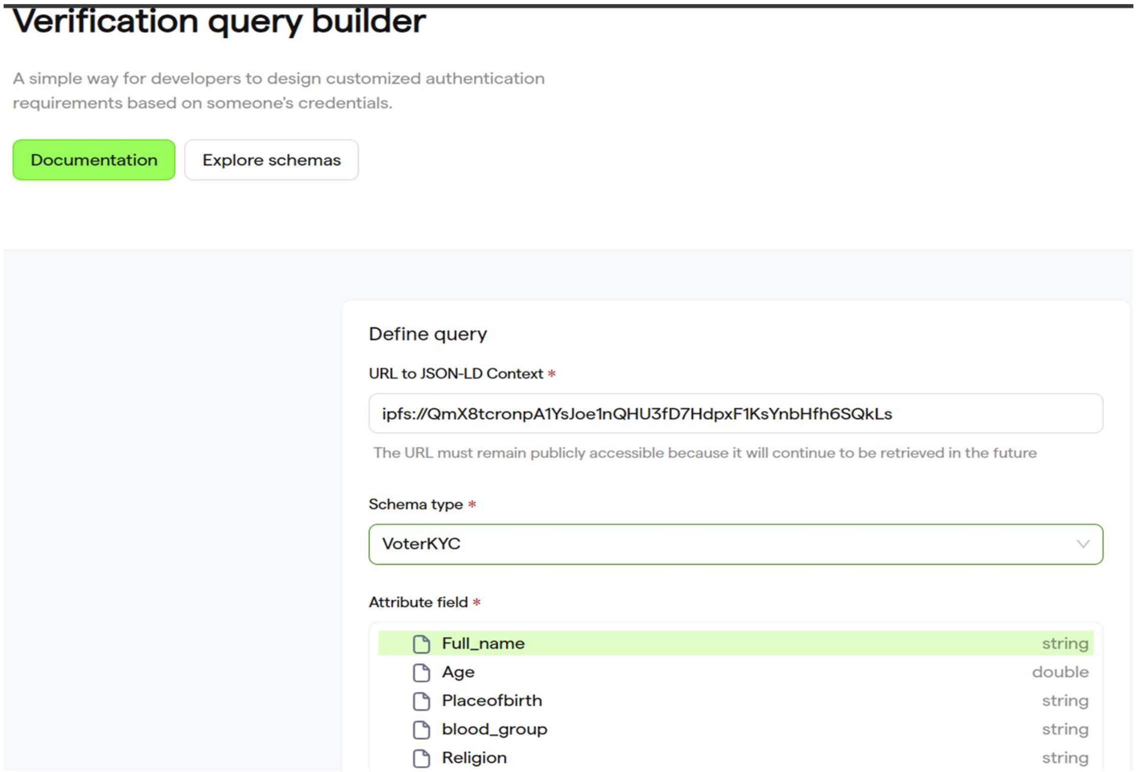
Full name of the voter

Age: 23

Age of the voter

**Figure 22**

6.5 Build the query to verify a voter credential


Verification query builder

A simple way for developers to design customized authentication requirements based on someone's credentials.

Define query

URL to JSON-LD Context *: ipfs://QmX8tcronpA1YsJoe1nQHU3fD7HdpxF1KsYnbHfh6SQkLs

The URL must remain publicly accessible because it will continue to be retrieved in the future

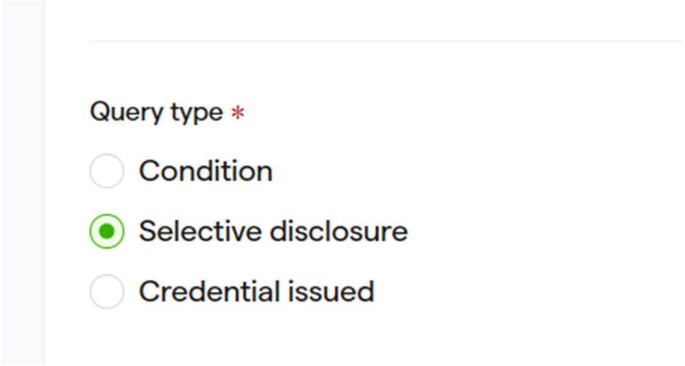
Schema type *: VoterKYC

Attribute field *

| | |
|---|---|
| Full_name | string |
| Age | double |
| Placeofbirth | string |
| blood_group | string |
| Religion | string |

**Figure 23**

6.6 On the query builder, select selective disclosure to only reveal required information of the candidate



Query type *

○ Condition

● Selective disclosure

○ Credential issued

**Figure 24**

# References

remix.ethereum.org. (n.d.). *Remix - Ethereum IDE*. [online] Available at: https://remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.26+commit.8a97fa7a.js.

Iden3.io. (2025). Circom / snarkjs - Iden3 Documentation. [online] Available at: https://docs.iden3.io/circom-snarkjs/ [Accessed 29 Jan. 2025].

Privado.id. (2024). Home | Privado ID. [online] Available at: https://www.privado.id/.