**Circom & SnarkJS**

[[](https://medium.com/@ChiHaoLu?source=post_page-----728e4314e057--------------------------------)](https://medium.com/@ChiHaoLu?source=post_page-----728e4314e057--------------------------------)

[[SWF Lab](https://medium.com/swf-lab?source=post_page-----728e4314e057--------------------------------)](https://medium.com/swf-lab?source=post_page-----728e4314e057--------------------------------)

[ChiHaoLu](https://medium.com/@ChiHaoLu?source=post_page-----728e4314e057--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fe08b32f9faf8&operation=register&redirect=https%3A%2F%2Fmedium.com%2Fswf-lab%2Fcircom-snarkjs-728e4314e057&user=ChiHaoLu&userId=e08b32f9faf8&source=post_page-e08b32f9faf8----728e4314e057---------------------post_header-----------)

Published in

[SWF Lab](https://medium.com/swf-lab?source=post_page-----728e4314e057--------------------------------)

·

18 min read

·

Oct 3, 2022

125

1

本文介紹兩個撰寫零知識證明程式的工具：circom 和 SnarkJS

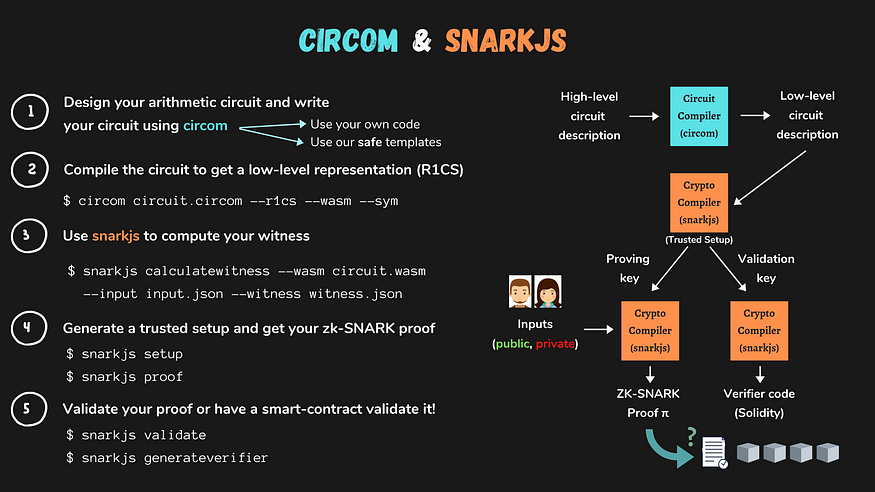
***Author:***[***ChiHaoLu***](https://chihaolu.me/)*(*[*chihaolu.eth*](https://chihaolu.eth.xyz/)*)*  
*Final Updated: 2022/10/4*

**Table of Contents**

* Intro
* Circom
* Exercise
* Reference

**Intro**

circom 是一個用 Rust 寫成的編譯器，編譯之後會 output 所有建置 ZKP 所需的限制和所需。snarkjs 是一個 npm package，可以用來產生與驗證 circom 產出的 artifacts（ZKP）。



Circom Compiler 和它整個生態系的工具可以幫助人們將原本撰寫零知識證明程式時，需要處理多項式與迴路（Circuits）的整個過程包起來，我們就只需要處理程式邏輯就好。還有其他用來撰寫零知識證明系統的程式語言，例如 Cairo、Zinc、ZoKrates（以上包含 SNARK / STARK 證明系統）。

**Download Packages**

Circom 的編譯器是使用 Rust 撰寫的，因此需要確認自己的電腦裡面有 Rust，MacOS & Linux 可以在終端機直接使用以下指令：

$ curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh

Circom 也支援使用 npm 或 yarn 等套件管理器下載：

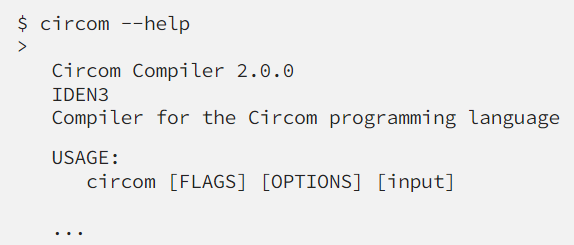
$ npm install -g circom  
$ npm install -g snarkjs

另外也可以使用 Cargo 來下載 Circom，細節可見[此處](https://docs.circom.io/getting-started/installation/#installing-circom)

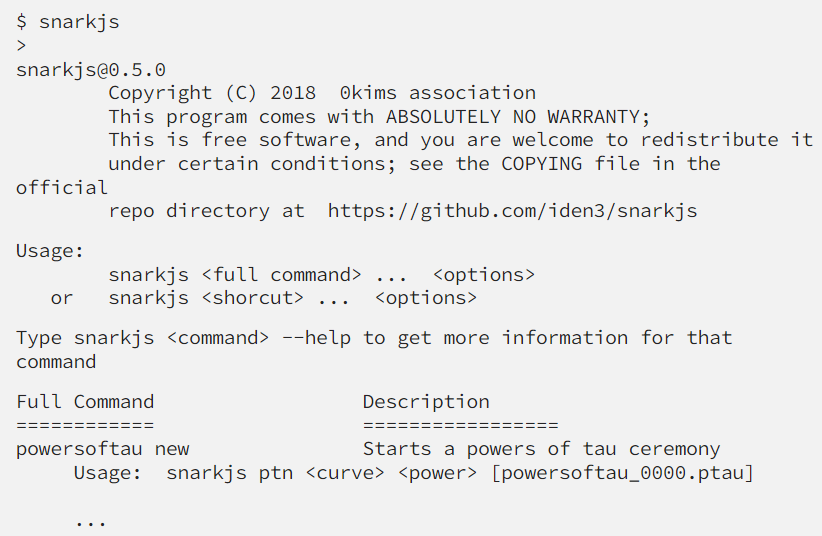
$ git clone https://github.com/iden3/circom.git  
$ cd circom  
$ cargo build --release  
$ cargo install --path circom

請確認自己的電腦裡面同時有 Circom 以及 SnarkJS 的存在，可以用以下指令確定自己有下載成功：

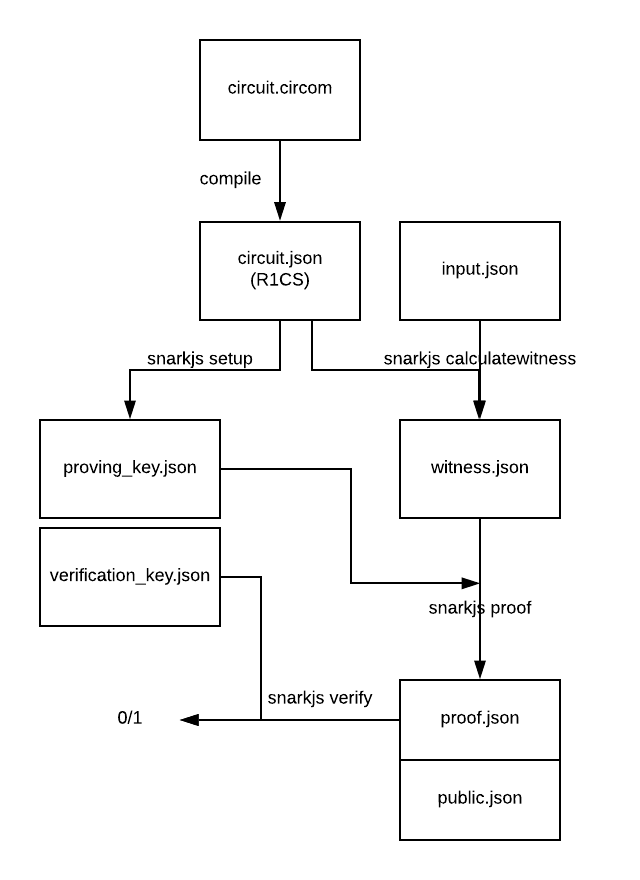
檢查 Circom



檢查 SnarkJS



**Overview**



1. Build Circuit *(Algebraic Circuit, R1CS)*:

* circom xxx.circom → circuit.json

2. Setup *(circuit + r to setup system)*:

* snarkjs setup → proving\_key.json, verification\_key.json

3. Calculate Witness *(Proof System)*: ← input.json

* snarkjs calculatewitness → witness.json

4. Generate proof *(Proof System)*:

* snarkjs proof → proof.json

**Command Steps**

如果我們有一個 Circom Program 的檔案名為 circuit.circom，撰寫完程式之後可以依序使用以下指令。

Build Circuit

* $ circom circuit.circom --r1cs --wasm --sym --c -o circuit.json
* 這一步我們會將程式邏輯（Circom Program）轉為多項式再轉為迴路
* -o circuit.json 是一個 optional 的指令，可以指定輸出的檔名要叫做什麼

Generate Input File

* $ node generate\_circuit\_input.js
* 這邊也可以使用其他方法，不一定要使用 js 產 json

Generate proving key

* $ snarkjs plonk setup circuit.r1cs pot12\_final.ptau circuit\_final.zkey
* 得到迴路之後可以透過 snarkjs 進行可信設定（trusted setup），trusted setup 會利用迴路還有隨機數產生兩把 Key
* 產出一個 .zkey 檔案，裡面已經包含了 proving 和 verification keys 兩種 key，而且是已經經過所有 phase 2 contributions 的結果
* 證明鑰匙（Proving Key），用於產生 ZKP
* 驗證鑰匙（Verification Key），用於驗證其他 Program 產生的 ZKP
* 需要注意 plonk 也可以使用其他演算法例如 groth16，在打指令的時候不要打錯了

Export verification key

* $ snarkjs zkey export verificationkey circuit\_final.zkey verification\_key.json
* 這一步會特別產生驗證鑰匙（Verification Key）

Calculate Witness

* $ snarkjs wtns calculate circuit.wasm input.json witness.wtns
* Witness 為隱私資訊，是不想漏漏的資料
* 也有與 witness 相對應的公開資料，可視系統設計而定
* 這一步會需要輸入值，以 input.json 的形式輸入
* 只要 witness 已經被計算，且 trusted setup 已經被執行完成，我們就可以依據 circuit 和 witness 來產生 zk-proof

Generate proof and public\_info

* $ snarkjs plonk prove circuit\_final.zkey witness.wtns proof.json public.json
* 這個指令會產生 ZKP， 需要 Proving Key（circuit\_final.zkey）、witness（witness.wtns）、Public Information（public.json） 三者
* 會產出這兩個結果：proof.json: 包含 zk-proof、public.json: 包含 public 的 input 和 output

Verify proof

* $ snarkjs plonk verify verification\_key.json public.json proof.json
* 驗證需要 Verification Key（verification\_key.json）、witness（proof.json）、Public Information（public.json） 三者
* 如果 Proof 是合法的，我們可以看見 command 會輸出結果 OK
* 一個合法的結果不只是我們知道符合電路的 signal 組合，也證明了 public 的 input 和 output 跟 public.json 檔案中的定義相符。

Generate verifier contract

* $ snarkjs zkey export solidityverifier circuit\_final.zkey verifier.sol
* 產生一個驗證合約，只要「產生證明者」提交他的 proof 到合約中的驗證函式，合約就可以判斷此 ZKP 是否合法
* 我們也可以繼承這個合約並且加上自己想要的系統設計，成為一個可以驗證零知識證明的 Dapps！
* 我們可以看見輸出的程式裡面包含兩個合約，Pairing 和 Verifier，我們只需要部署 Verifier 即可。

Generate calldata

* $ snarkjs zkey export soliditycalldata public.json proof.json
* Verifier Contract 裡面會有一個 view function 叫做 verifyProof，我們可以提交 proof 給這個函式並且得到回傳值，如果 proof 合法則為 TRUE，反之則為 FALSE。
* 我們可以藉由以上這個指令自動產出一個 calldata，直接複製就可以作為輸入函式的參數。

**Ceremony**

如果還沒準備過 Power of Tau 可以到 github 下載，或者透過以下指令下載：

基本上 trusted setup 實際上有兩個步驟：

1. The powers of tau, which is independent of the circuit.

* $ snarkjs powersoftau new bn128 12 pot12\_0000.ptau -v
* $ snarkjs powersoftau contribute pot12\_0000.ptau pot12\_0001.ptau --name="First contribution" -v

2. The phase 2, which depends on the circuit.

* $ snarkjs powersoftau prepare phase2 pot12\_0001.ptau pot12\_final.ptau -v

**Circom**

**Official Links**

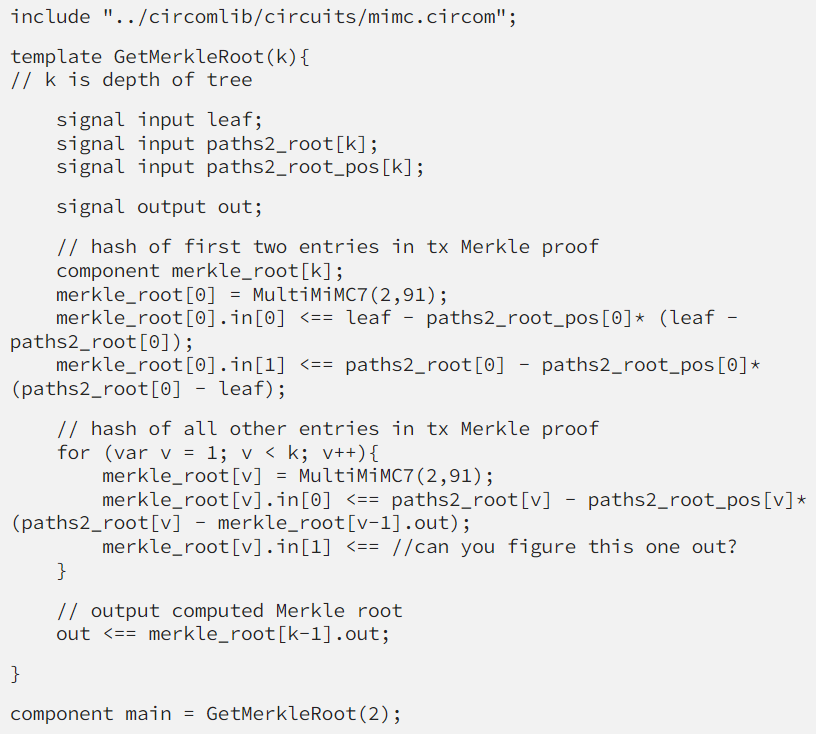
**Circom**

* [iden3/circom: zkSnark circuit compiler](https://github.com/iden3/circom)
* [Circom 2 Official Documentation](https://docs.circom.io/)
* [Syntax Documantation](https://docs.circom.io/circom-language/signals/)
* [Circom verifier.sol](https://docs.circom.io/getting-started/proving-circuits/#verifying-a-proof)
* [zkREPL an online playground for zk circuits](https://zkrepl.dev/)

**SnarkJS**

* [iden3/snarkjs](https://github.com/iden3/snarkjs)
* [SnarkJs Official Documentation](https://iden3-docs.readthedocs.io/en/latest/iden3_repos/snarkjs/README.html)

**Example**



* include: include modules
* template: function declaration（會在電路）
* var: variable declaration（不會在電路）
* signal: 代表此變數或陣列要轉換成電路中，有以下屬性：private: 代表隱私資訊，如果沒有指定就會是公開資訊；input: input；output: output
* component: 使用情境是在於承接函數、對函數內的 signal 變數作操作，可以想作是一個物件，而 signal 變數則為 class 的公開成員變數。
* main: main

At compilation time, the content of a signal is always considered unknown.

**Syntax**

Reserved Keywords: assert, Binary, component, do, else, FieldElement, for, function, if, include, input, log, output, public, return, signal, template, var, while.

**Assign**

* <== ==> : 給予 signal 變數值，並且建立一個迴路的約束。
* <-- --> : 給予 signal 變數值。
* ===: 建立一個迴路的約束。
* 因此 <== : <-- + ===

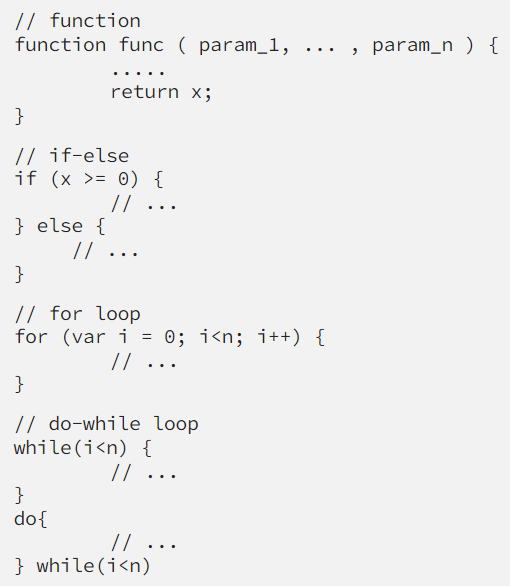
**Basic Operators**

* Boolean operator: &&, ||, !
* Rational operator: <, >, <=, >=, ==, !=
* Arithmetic: +, -, \*, /, \*\*, \, %
* Bitwise: &, |, ~, ^, <<, >>
* Operator with assignment: += , -= , \*= , /= , \*\*=, \=, %= , ^= , |= , &= , >>=, <<=

-= 有時候會報錯我還不確定原因。

**Function, Control flow, Debugging**

基本的流程控制還有函式操作 Circom 都有支援（望向隔壁棚的 Cairo…），不過需要注意的是 function 只可以用來算數值運算或陣列值等敘述，不可以在裡面有電路限制。



此外也有一些限制可以幫助開發，例如 assert(<boolean\_statement>) 以及 log(<print\_statement>)，在 calculate witness 時會產出 log() 的結果，如果是要印出跟 signal 有關的訊息會發現 compiler 的時候是不會知道值是多少的，因為 signal 必須等到有 input 的 witness 階段才知道值。

**Circuit Hint**

一開始在撰寫電路的時候不太清楚怎麼樣的限制或者變數要放在電路裡面，例如 assert(a == b) vs. a === b 或者 signal input a vs. var a 這些差別。

將敘述放在電路與否其實與 prover 和 verifier 的角度有非常大的關係，例如 assert 只是執行時的檢查條件（讓 prover 知道），電路裡並沒有真的限制式。真的違反這個限制式時，例如 witness a!=b 時，Verifier 是檢查不出來的。 但 === 就是電路裡真的有限制式，本身也有 alert 的效果，如果 witness 裡面沒有正確的限制， Verifier 是會知道的。

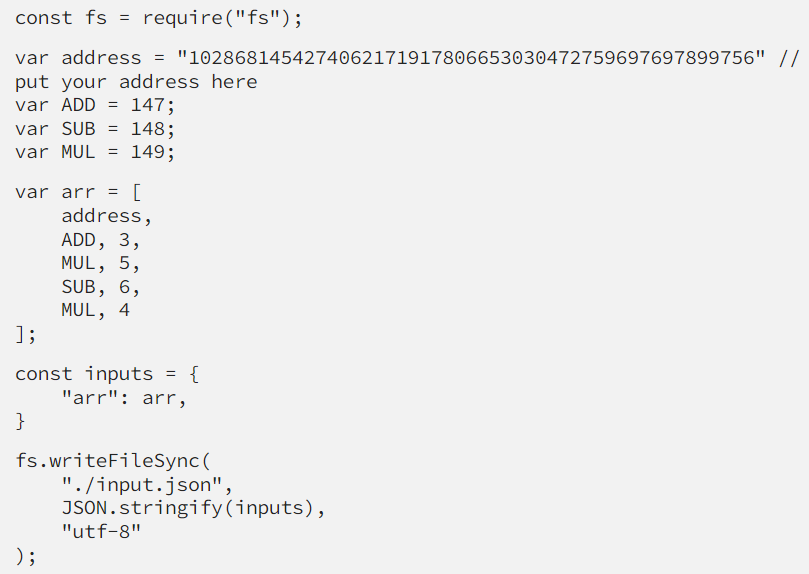
一般使用時機是，假設你的電路支援 10 層的 merkle tree ，你可能會 input 一個變數叫層數 n 。迴路不用真的檢查 n < 10 ，但是有 assert 檢查才不會讓 prover witness 一個不合格的 n ，提早把問題反映出來

而 signal input a vs. var a 的例子更好理解，signal 通常是 proving time 才知道的數字，而 var 是 compile time 就會知道的數字。所以在迴圈中的 loop counter 或者一些用來做運算的 temp，這些不需要等到 proving time 才知道的變數，就可以直接用 var 就好。

最後老師還提醒電路的限制條件要很小心。錯誤的條件可以讓壞人製造不合理的 proof 。例如 tornado cash 有些條件沒檢查好，壞人就可以憑空領錢。在每一個 <== 和 === 都要很小心問：為什麼我需要這條檢查，如果沒這個檢查，壞人可以怎樣偷錢。

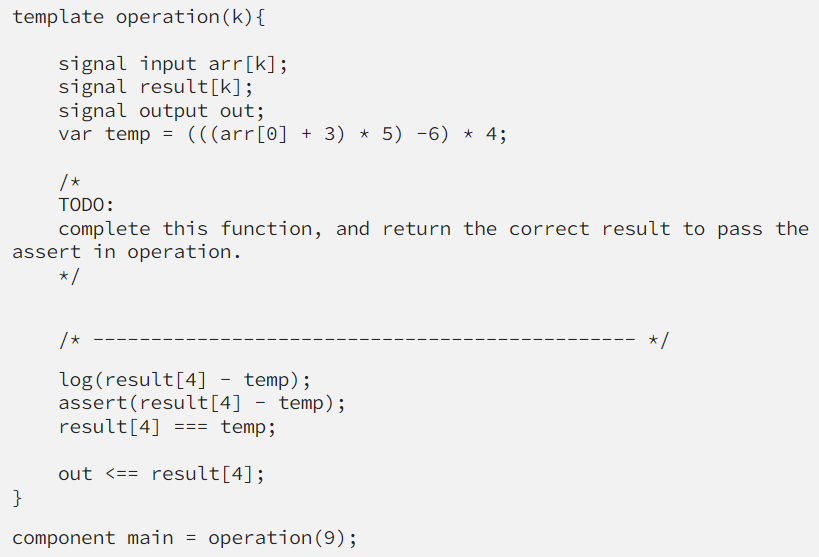
**Exercise**

大家可以來練習看看我自己想的一個小小練習題！



這邊要特別注意，js 的大小和 circom 的變數大小是不同的，因此當我們在把地址轉為 uint160 放進去之後，記得要使用字串，而不是直接使用整數。

接下來我們把以下 circom program 裡面的 TODO 完成！



做法其實有不少種，大家可以自己思考一下！以下是我自己提供的一個解法！

result[0] <-- arr[0];  
result[1] <-- result[0] + arr[2];  
result[2] <-- result[1] \* arr[4];  
result[3] <-- result[2] - arr[6];  
result[4] <-- result[3] \* arr[8];

在 Verifier.sol 中，我部署的合約中 function verifyProof(bytes memory proof, uint[] memory pubSignals) public view returns (bool) 裡面有包含著以下檢查程式碼：

address prover = address(uint160(pubSignals[1]));  
require(msg.sender == prover);

合約地址：0x4b67DB7d46A4fde775C0C837E33Ae1eeF2A3fB20

所以在製作這個題目的 ZKP 時有幾個點需要注意：

1. [**input.json**](https://github.com/ChiHaoLu/circom-practice/blob/main/generate_circuit_input.js) 必須要包含自己的地址：uint256(uint160("yout\_address"));
2. 必須要使用題目提供的 [**zkey**](https://github.com/ChiHaoLu/circom-practice/blob/main/circuit_final.zkey) 來製作 proof！

**Closing**

感謝 C.C. Liang, Vivian, KiMi 老師們的細心指導！

**Reference**

* [0xPARC circom ECDSA circuit](https://github.com/0xPARC/circom-ecdsa)
* [privacy-scaling-explorations/zk-kit](https://github.com/privacy-scaling-explorations/zk-kit)
* [therealyingtong/roll\_up\_circom\_tutorial](https://github.com/therealyingtong/roll_up_circom_tutorial)
* [動手實做零知識 — circom](https://medium.com/cryptocow/%E5%8B%95%E6%89%8B%E5%AF%A6%E5%81%9A%E9%9B%B6%E7%9F%A5%E8%AD%98-circom-d7ac1fa8bbd3)
* [Implementing Zero Knowledge Lottery’s Circom circuits PART 1 / 2](https://medium.com/@killari/implementing-zero-knowledge-lotterys-circom-circuits-part-1-2-16910b3732a2)
* [KimiWu123/Sample](https://github.com/KimiWu123/Samples)
* [fluree/example-zero-knowledge](https://github.com/fluree/example-zero-knowledge)
* [Zero-Knowledge Soul-Bound-Token (ZK SBT)](https://github.com/enricobottazzi/ZK-SBT)

<https://medium.com/swf-lab/circom-snarkjs-728e4314e057>