# Applied ZKP Workshop #2

Shumo Chu

P0X LABS

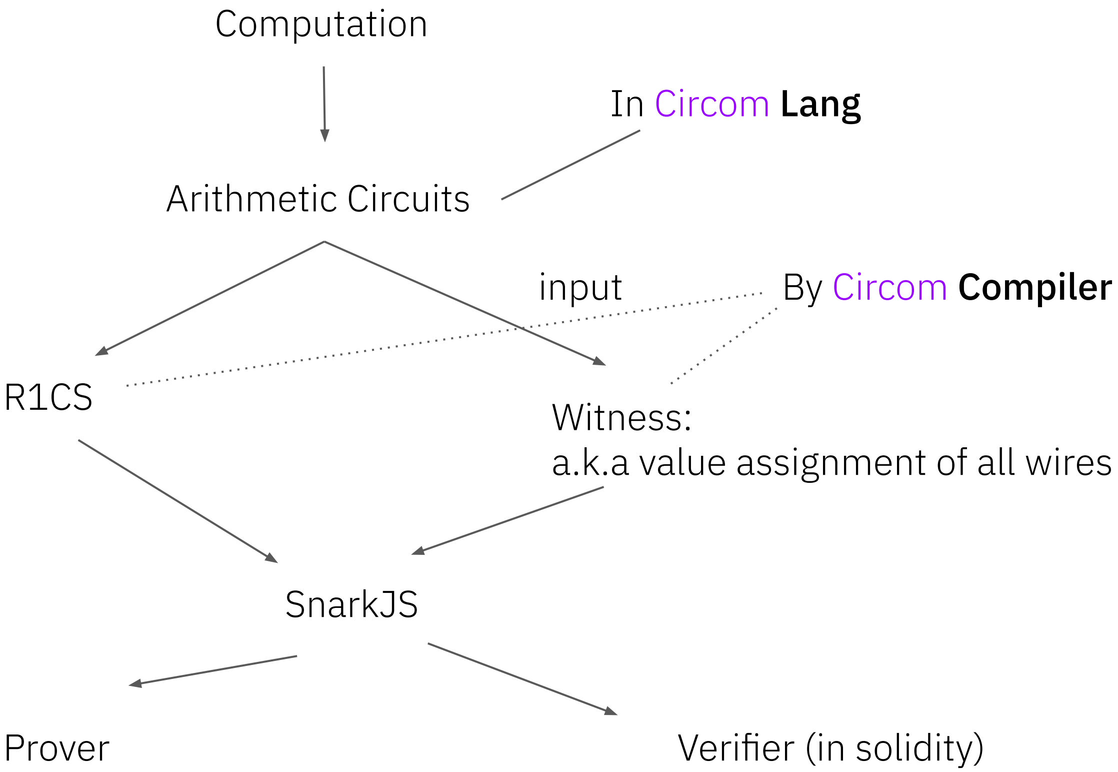
# Circom / SnarkJS

https://github.com/iden3/circom

https://github.com/iden3/snarkjs

Circom documentation and reference: https://docs.circom.io/

# Circom **Workflow**



Computation

In Circom **Lang**

Arithmetic Circuits

input

By Circom **Compiler**

R1CS

Witness:
a.k.a value assignment of all wires

SnarkJS

Prover

Verifier (in solidity)

# Basic Concepts

- **signal**: “big unsigned int” mod by p, where p is the size of FF of Jubjub base field
- **template**: circuit gadget, think as class in java
- **component**: instantiation of template, think as instance in java
- --> : witness computation
- ===: constraint creation
- ==>: witness computation + constraint creation

Constraint Checking (**Symbolic** Computation)

v.s.

Witness Generation (**Concrete** Computation)

# Example

```
pragma circom 2.1.0;

template Main(){

    signal input x;

    signal input y;

    signal z;

    signal output out;

    z <-- x * y;

    out <-- z * x;

    out === x * y;

}

component main {public [x]} = Main();
```

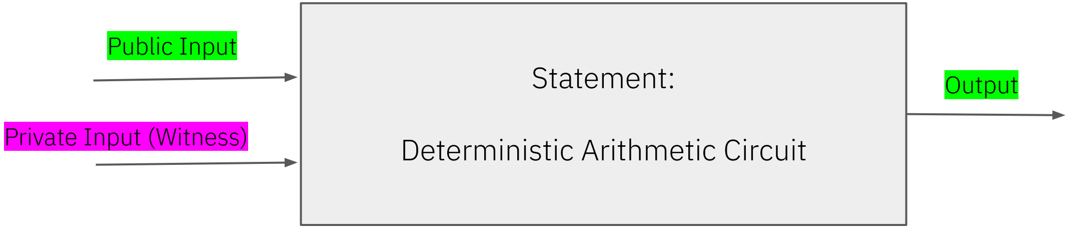
**What’s wrong with this circuit?**

# Circom Programming Demo

P0X LABS

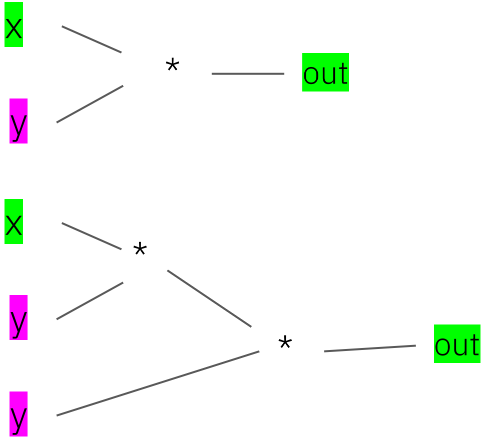
# Example

```
pragma circom 2.1.0;

template Main(){

    signal input x;

    signal input y;

    signal z;

    signal output out;

    z <-- x * y;

    out <-- z * x;

    out === x * y;

}

component main {public [x]} = Main();
```



**Constraints checking**

**Witness generation**



P0X LABS

It is **programmer’s** responsibility to make sure the circuit’s **constraint checking logic** and **witness computation** logic is the **same**

# Programming Safe Circuit

Why not just using ==> ?

- It is not always possible
- We want to build a circuit to check if the input is zero

```
template IsZero(){
    signal input in;
    signal input out;
}
```

# First Attempt

```
template IsZero(){
    signal input in;
    signal input out;
    out <== (in == 0);
}
```

# First Attempt

```
template IsZero(){
    signal input in;
    signal input out;
    out <== (in == 0);
}
```

Non Quadratic Constraint, not supported here

# Second Attempt

```
template IsZero(){
    signal input in;
    signal input out;
    out <-- (in == 0);
}
```

# Second Attempt

```
template IsZero(){
    signal input in;
    signal input out;
    out <-- (in == 0);
}
```

Even worse! No constraint at all

# Correct solution

```
template IsZero(){
    signal input in;
    signal input out;
    signal inv;
    inv <-- in!= 0 ? 1/in : 0;
    out <== - in * inv + 1;
    0 === in * out;
}
```

# Best Circuit Writing Practice

- https://github.com/iden3/circomlib/tree/master/circuits
- Do not reinventing the wheel (but do need to understand other's circuit's semantics)
- Write unit tests! (**will cover on the next Workshop**)
- Formal verification (**will covered by Verdise in Workshop #5**)

# Template Instantiation

# Define compile time constant for template

```
template Num2Bits(n) {

    signal input in;

    signal output out[n];

    var lc1=0;

    var e2=1;

    ….

    lc1 === in;

}
```

constant need to be decided at compile time

```
Component num2bit4 = Num2Bits(4);
```

# Circom Invariants

- Think of the circuit as a physical circuit
- Constraint generation cannot depend on unknown information at compile time

```
Template … {
    signal … x;
    signal … y;
           if (x < 6) {
          y < == …
    }

}
```

```
Template … {
    signal … x;
    signal … y[254];
    y[x] < == …

}
```