Reference

Wednesday, June 26, 2024 10:06 AM

1. https://www.rareskills.io/zk-book
2. https://www.rareskills.io/post/arithmetic-circuit
3. https://www.rareskills.io/post/circom-tutorial
4. https://www.rareskills.io/post/underconstrained-circom

Groth16 - Arithmetic circuit, R1CS, and basic Circom

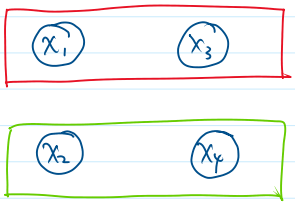
Monday, June 3, 2024 11:10 AM

https://x.com/ret2basic

https://github.com/ret2basic/ZK-by-Hand

Problem statement

Given vertices (x_1, x_2, x_3, x_4). prove the graph is bipartite.



The solution to this problem is a "witness vector":

$a = [1, x_1, x_2, x_3, x_4]$

↑ Constant term (public input)

how to "2-color" this graph (private input)

- Verifier has access to public inputs
- Prover has access to both public and private inputs
- Intermediate signals can exist but we don't need them for this specific problem.

Arithmetic circuit

1. Each vertex should have color 1 or 2

$$(x_1 - 1)(x_1 - 2) = 0 \rightarrow x_1^2 - 3x_1 + 2 = 0$$
$$(x_2 - 1)(x_2 - 2) = 0 \rightarrow x_2^2 - 3x_2 + 2 = 0$$
$$(x_3 - 1)(x_3 - 2) = 0 \rightarrow x_3^2 - 3x_3 + 2 = 0$$
$$(x_4 - 1)(x_4 - 2) = 0 \rightarrow x_4^2 - 3x_4 + 2 = 0$$

2. Vertices from different "groups" should have different colors:

$$x_1 x_2 - 2 = 0 \quad (1)$$
$$x_1 x_4 - 2 = 0$$
$$x_2 x_3 - 2 = 0 \quad (2)$$

($x_1 x_3 - 2 = 0$ is implied by (1) and (2) implicitly)

Comment:
We assume that verifying a problem is always easier than solving the problem. Otherwise, P = NP.

Q: Is it true that "all NP problems can be represented by arithmetic circuit"?

A:

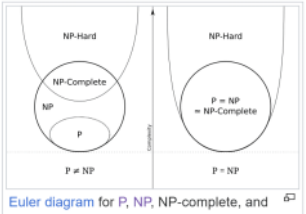
1. Cook–Levin theorem proved that SAT (boolean satisfiability problem) is NP-complete.
2. By definition of NP-completeness, every NP problem can be reduced to NP-complete problem, therefore can be reduced to SAT.
3. Boolean circuit can be transformed into arithmetic circuit (x OR y => x * y, x AND y => x + y - x * y)

(A boolean circuit in game: Analyzing the Blocky Logic Puzzle - Pwn Adventure 3)

- Thus every NP problem should have arithmetic circuit representation.

Resource: P / NP / NP-complete / NP-hard
https://stackoverflow.com/questions/210829/what-is-an-np-complete-in-computer-science

A nice diagram from Wikipedia:
https://en.wikipedia.org/wiki/NP-hardness



Euler diagram for P, NP, NP-complete, and NP-hard set of problems. The left side is valid under the assumption that P≠NP, while the right side is valid under the assumption that P=NP (except that the empty language and its complement are never NP-complete)

R1CS

Turn arithmetic circuits into system of quadratic equations (constraints).

1. Each vertex should have color 1 or 2

$$x_1 x_1 = 3x_1 - 2$$
$$x_2 x_2 = 3x_2 - 2$$
$$x_3 x_3 = 3x_3 - 2$$
$$x_4 x_4 = 3x_4 - 2$$

2. Vertices from different "groups" should have different colors:

$x_1 x_2 = z$

$x_1 x_4 = z$

$x_2 x_3 = z$

Turn system of equations into matrix form

What is Hadamard product?

From Wikipedia: https://en.wikipedia.org/wiki/Hadamard_product_(matrices)

Definition [edit]

For two matrices A and B of the same dimension $m \times n$, the Hadamard product $A \odot B$ (sometimes $A \circ B$[4][5][6]) is a matrix of the same dimension as the operands, with elements given by[3]

$$(A \odot B)_{ij} = (A)_{ij}(B)_{ij}.$$

For matrices of different dimensions ($m \times n$ and $p \times q$, where $m \neq p$ or $n \neq q$), the Hadamard product is undefined.

For example, the Hadamard product for two arbitrary 2 × 3 matrices is:

$$\begin{bmatrix} 2 & 3 & 1 \\ 0 & 8 & -2 \end{bmatrix} \circ \begin{bmatrix} 3 & 1 & 4 \\ 7 & 9 & 5 \end{bmatrix} = \begin{bmatrix} 2 \times 3 & 3 \times 1 & 1 \times 4 \\ 0 \times 7 & 8 \times 9 & -2 \times 5 \end{bmatrix} = \begin{bmatrix} 6 & 3 & 4 \\ 0 & 72 & -10 \end{bmatrix}$$

Recall that witness vector $a = [1, x_1, x_2, x_3, x_4]$

We turn systems of equation into $L \cdot a \circ R \cdot a = O \cdot a$

Hadamard product

Where L, R, O are 3 matrices with same dimension:

- # rows == # constraints
- # columns == dimension of witness vector

Begin transformation:

Constraint 1: $x_1 x_1 = 3x_1 - 2$

$\Rightarrow (x_1) \cdot (x_1) = 3x_1 - 2$

L R O

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ \text{don't know yet} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \circ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ \text{don't know yet} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2 & 3 & 0 & 0 & 0 \\ \text{don't know yet} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$L \cdot a \circ R \cdot a = O \cdot a$

Check if this transformation is done correctly:

$$[0\ 1\ 0\ 0\ 0] \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \circ [0\ 1\ 0\ 0\ 0] \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = [-2\ 3\ 0\ 0\ 0] \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$[x_1] \circ [x_1] = [-2 + 3x_1]$$

$$[x_1 \cdot x_1] = [3x_1 - 2]$$

Repeat this process for each constraint in the system of equation, or equivalently, for each row in the matrices.
In the end this is what we get:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot a \circ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \cdot a = \begin{bmatrix} -2 & 3 & 0 & 0 & 0 \\ -2 & 0 & 3 & 0 & 0 \\ -2 & 0 & 0 & 3 & 0 \\ -2 & 0 & 0 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot a$$

Implementing R1CS in Python

My implementation: https://github.com/ret2basic/Groth16/blob/main/groth16.py

groth16.py

```python
import numpy as np
import galois
from functools import reduce
from py_ecc.bn128 import G1, G2, multiply, add, curve_order, Z1, pairing, neg, final_exponentiate, FQ12

# curve_order = 1151
GF = galois.GF(curve_order) # we work with bn128/bn254 curve

```

galois library handles modular arithmetic over scalar field:

```
>>> import galois
>>> curve_order = 17
>>> GF = galois.GF(curve_order)
>>> operand1 = GF(6)
>>> operand2 = GF(14)
>>> operand1 + operand2
GF(3, order=17)
>>> int(_)
3
>>> operand1 * operand2
GF(16, order=17)
>>> int(_)
16
>>>
```

_ = return value from previous computation

operand1/operand2 is equivalent to operand1 * pow(operand2, -1, curve_order).

```
>>> operand1 - operand2
GF(9, order=17)
>>> int(_)
9
>>> operand1 / operand2
GF(15, order=17)
>>> int(_)
15
>>>
```

galois can also handle matrices and polynomials:

```
ret2basic@Pwnielsland: ~ 80x24
>>> import numpy as np
>>> np.array([[1,0,0],[0,1,0],[0,0,1]])
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
>>> GF(_)
GF([[1, 0, 0],
    [0, 1, 0],
    [0, 0, 1]], order=17)
>>> A = _
>>> B = GF(np.array([[0,2,0],[4,0,0],[0,0,8]]))
>>> A + B
GF([[1, 2, 0],
    [4, 1, 0],
    [0, 0, 9]], order=17)
>>> A * B
GF([[0, 0, 0],
    [0, 0, 0],
    [0, 0, 8]], order=17)
>>>
```

```
ret2basic@Pwnielsland: ~ 80x24
>>> poly1 = galois.Poly([1, 2, 3, 4], field=GF)
>>> poly1
Poly(x^3 + 2x^2 + 3x + 4, GF(17))
>>> poly2 = galois.Poly([5, 2, 3], field=GF)
>>> poly2
Poly(5x^2 + 2x + 3, GF(17))
>>> poly1 + poly2
Poly(x^3 + 7x^2 + 5x + 7, GF(17))
>>> poly1 * poly2
Poly(5x^5 + 12x^4 + 5x^3 + 15x^2 + 12, GF(17))
>>>
```

Back to groth16:

```
# R1CS matrices

L = np.array([
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 1],
    [0, 1, 0, 0, 0],
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
])

R = np.array([
    [0, 1, 0, 0, 0],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 1, 0],
    [0, 0, 0, 0, 1],
    [0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1],
    [0, 0, 0, 1, 0],
])
```

```
O = np.array([
    [curve_order-2, 3, 0, 0, 0],
    [curve_order-2, 0, 3, 0, 0],
    [curve_order-2, 0, 0, 3, 0],
    [curve_order-2, 0, 0, 0, 3],
    [2, 0, 0, 0, 0],
    [2, 0, 0, 0, 0],
    [2, 0, 0, 0, 0],
])

L_galois = GF(L)
R_galois = GF(R)
O_galois = GF(O)
```

Further improvement: write a function to handle negative numbers automatically.

Negative numbers such as -2 must be converted to curve_order - 2.

Prover computes witness:

```
# In reality this witness is prover's secret, only prover knows it
x1 = GF(1)
x2 = GF(2)
x3 = GF(1)
x4 = GF(2)
# a is the witness
a = GF(np.array([1, x1, x2, x3, x4]))

assert all(np.equal(np.matmul(L_galois, a) * np.matmul(R_galois, a), np.matmul(O_galois, a))), "not equal"
```

Separate public inputs and private inputs (for future use):

```
# witness = [1, x1, x2, x3, x4]
# Only the first entry [1] is public input
# [x1, x2, x3, x4] are private inputs that only the prover knows
l = 0
public_inputs = a[:l+1]
private_inputs = a[l+1:]
```

From circom doc:

```
pragma circom 2.0.0;

template Multiplier2(){
   //Declaration of signals
   signal input in1;
   signal input in2;
   signal output out;
   out <== in1 * in2;
}

component main {public [in1,in2]} = Multiplier2();
```

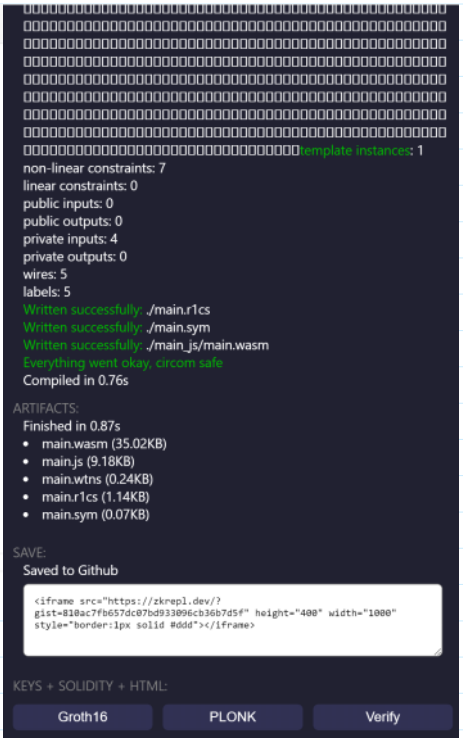
in1 and in2 are public inputs

Do it in circom

A simple piece of circom I wrote that represents the bipartite graph problem above:
https://zkrepl.dev/?gist=810ac7fb657dc07bd933096cb36b7d5f

main.circom + Add File

```
pragma circom 2.1.6;

// bipartite graph problem arithmetization

template Bipartite(n) {
    // coloring for 4 vertices x1, x2, x3, x4
    // in[0] -> x1
    // in[1] -> x2
    // in[2] -> x3
    // in[3] -> x4
    signal input in[n];

    // Condition 1: color is either 1 or 2 for each vertex
    (in[0] - 1) * (in[0] - 2) === 0;
    (in[1] - 1) * (in[1] - 2) === 0;
    (in[2] - 1) * (in[2] - 2) === 0;
    (in[3] - 1) * (in[3] - 2) === 0;

    // Condition 2: vertices from different "groups" have different colors
    in[0] * in[1] === 2;
    in[0] * in[3] === 2;
    in[1] * in[2] === 2;

}
component main = Bipartite(4);

/* INPUT = {"in": [1, 2, 1, 2]} */
```



Q: Can we control parameters other than INPUT?
A: Yes, but not in zkREPL. We will talk about it later.

base field (field_modulus) ✗

↕ ✓

Note: circom operates in **scalar field (curve_order)** as well:
https://docs.circom.io/circom-language/basic-operators/

2G + 3G = 5G

curve_order * G = O -> point at infinity

Field Elements

A field element is a value in the domain of Z/pZ, where p is the prime number set by default to

`p = 21888242871839275222246405745257275088548364400416034343698204186575808495617.`

As such, field elements are operated in arithmetic modulo p.

The circom language is parametric to this number, and it can be changed without affecting the rest of the language (using `GLOBAL_FIELD_P`).

```
ret2basic@PwnIsland: ~/Desktop/zero-knowledge-puzzles 79x22
>>> from py_ecc.bn128 import field_modulus, curve_order
>>> field_modulus
21888242871839275222246405745257275088696311157297823662689037894645226208583
>>> curve_order
21888242871839275222246405745257275088548364400416034343698204186575808495617
>>>
```

Circomlib - comparators.circom

https://github.com/iden3/circomlib/blob/master/circuits/comparators.circom

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

    signal inv;

    inv <-- in!=0 ? 1/in : 0;

    out <== -in*inv +1;
    in*out === 0;
}
```

Idea: non-zero field element has multiplicative inverse by definition.

<-- assign to signal

<== assign and add constraint

main.circom + Add File

```
pragma circom 2.1.6;

template IsZeroTheWrongWay() {
    signal input in;
    signal output out;

    if (in == 0) {
        out <== 1;
    } else {
        out <== 0;
    }
}

component main = IsZeroTheWrongWay();

/* INPUT = {
    "in": "5"
} */
```

Not that easy

```
template IsEqual() {
    signal input in[2];
    signal output out;

    component isz = IsZero();

    in[1] - in[0] ==> isz.in;

    isz.out ==> out;
}
```

Common pattern: When there are multiple inputs, store them into an array. Usually it is called in[].

component: instantiate another template and "wire" inputs.

Arrow direction can be <== or ==>

```
template LessThan(n) {
    assert(n <= 252);
    signal input in[2];
    signal output out;

    component n2b = Num2Bits(n+1);

    n2b.in <== in[0]+ (1<<n) - in[1];

    out <== 1-n2b.out[n];
}
```

```
template Num2Bits(n) {
    signal input in;
    signal output out[n];
    var lc1=0;

    var e2=1;
    for (var i = 0; i<n; i++) {
        out[i] <-- (in >> i) & 1;
        out[i] * (out[i] -1 ) === 0;
        lc1 += out[i] * e2;
        e2 = e2+e2;
    }

    lc1 === in;
}
```

accumulator

Compare 5 = 0101 and 7 = 0111 n=4

$e_2 = 1, 2, 4, 8, \cdots$

0101
+ 10000 ← 1<<4
10101

1. 5 = 0101 $e_2=1$ $lc_1=0$
& 1
1 → out[0]

$lc_1 += 1*1 \rightarrow lc_1=1$

$e_2 = 2$

10101
− 0111
01110

if MSB is 0, then a < b
if MSB is 1, then a > b

2. 0010 $e_2=2$ $lc_1=1$
& 1
0 → out[1]

$lc_1 += 0*1 \rightarrow lc_1=1$

$e_2 = 4$

```
// N is the number of bits the input  have.
// The MSF is the sign bit.
template LessEqThan(n) {
    signal input in[2];
    signal output out;

    component lt = LessThan(n);

    lt.in[0] <== in[0];
    lt.in[1] <== in[1]+1;
    lt.out ==> out;
}
```

```
// N is the number of bits the input  have.
// The MSF is the sign bit.
template GreaterThan(n) {
    signal input in[2];
    signal output out;

    component lt = LessThan(n);

    lt.in[0] <== in[1];
    lt.in[1] <== in[0];
    lt.out ==> out;
}
```

```
// N is the number of bits the input  have.
// The MSF is the sign bit.
template GreaterEqThan(n) {
    signal input in[2];
    signal output out;

    component lt = LessThan(n);

    lt.in[0] <== in[1];
    lt.in[1] <== in[0]+1;
    lt.out ==> out;
}
```

Side note: Circomlib is not bug-free. For instance, check out this audit report:
https://f8t2x8b2.rocketcdn.me/wp-content/uploads/2023/02/VAR-circom-bigint.pdf

Circom behind the scene https://www.rareskills.io/post/circom-tutorial

We are still working with this code:

First we check if it actually compiles:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite 80x24
ret2basic@PwnieIsland:~/Desktop/bipartite$ ll
total 12
drwxrwxr-x  2 ret2basic ret2basic 4096 Jun  5 22:47 ./
drwxr-xr-x 30 ret2basic ret2basic 4096 Jun  5 22:46 ../
-rw-rw-r--  1 ret2basic ret2basic  664 Jun  5 22:47 bipartite.circom
ret2basic@PwnieIsland:~/Desktop/bipartite$ circom bipartite.circom
template instances: 1
Everything went okay
ret2basic@PwnieIsland:~/Desktop/bipartite$
```

main.circom + Add File

```
pragma circom 2.1.6;

// bipartite graph problem arithmetization

template Bipartite(n) {
    // coloring for 4 vertices x1, x2, x3, x4
    // in[0] -> x1
    // in[1] -> x2
    // in[2] -> x3
    // in[3] -> x4
    signal input in[n];

    // Condition 1: color is either 1 or 2 for each vertex
    (in[0] - 1) * (in[0] - 2) === 0;
    (in[1] - 1) * (in[1] - 2) === 0;
    (in[2] - 1) * (in[2] - 2) === 0;
    (in[3] - 1) * (in[3] - 2) === 0;

    // Condition 2: vertices from different "groups" have different colors
    in[0] * in[1] === 2;
    in[0] * in[3] === 2;
    in[1] * in[2] === 2;

}
component main = Bipartite(4);

/* INPUT = {"in": [1, 2, 1, 2]} */
```

Generate R1CS file:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite 80x24
ret2basic@PwnieIsland:~/Desktop/bipartite$ circom bipartite.circom --r1cs --sym
template instances: 1
non-linear constraints: 7
linear constraints: 0
public inputs: 0
private inputs: 4
public outputs: 0
wires: 5
labels: 5
Written successfully: ./bipartite.r1cs
Written successfully: ./bipartite.sym
Everything went okay
ret2basic@PwnieIsland:~/Desktop/bipartite$
```

Here we generate the .sym file so that we can provide symbolic input.json later, such as {"in": [1, 2, 1, 2]}.

Check out R1CS file:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite 80x24
ret2basic@PwnieIsland:~/Desktop/bipartite$ ll
total 20
drwxrwxr-x  2 ret2basic ret2basic 4096 Jun  5 22:48 ./
drwxr-xr-x 30 ret2basic ret2basic 4096 Jun  5 22:46 ../
-rw-rw-r--  1 ret2basic ret2basic  664 Jun  5 22:47 bipartite.circom
-rw-rw-r--  1 ret2basic ret2basic 1136 Jun  5 22:48 bipartite.r1cs
-rw-rw-r--  1 ret2basic ret2basic   68 Jun  5 22:48 bipartite.sym
ret2basic@PwnieIsland:~/Desktop/bipartite$ snarkjs r1cs print bipartite.r1cs
[INFO]  snarkJS: [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 ] * [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[0] ] - [  ] = 0
[INFO]  snarkJS: [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[1] ] * [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[1] ] - [  ] = 0
[INFO]  snarkJS: [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[2] ] * [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[2] ] - [  ] = 0
[INFO]  snarkJS: [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[3] ] * [ 21888242871839275222246405745257275088548364400416034343698204186575808495615 1 +main.in[3] ] - [  ] = 0
[INFO]  snarkJS: [ main.in[0] ] * [ main.in[1] ] - [ 21 ] = 0
[INFO]  snarkJS: [ main.in[0] ] * [ main.in[3] ] - [ 21 ] = 0
[INFO]  snarkJS: [ main.in[1] ] * [ main.in[2] ] - [ 21 ] = 0
ret2basic@PwnieIsland:~/Desktop/bipartite$
```

Could be circom bug. Circom is printing 1 at the end of each huge number.

Generate wasm file as preparation for computing witness:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite 80x24
ret2basic@PwnieIsland:~/Desktop/bipartite$ circom bipartite.circom --r1cs --sym
--wasm
template instances: 1
non-linear constraints: 7
linear constraints: 0
public inputs: 0
private inputs: 4
public outputs: 0
wires: 5
labels: 5
Written successfully: ./bipartite.r1cs
Written successfully: ./bipartite.sym
Written successfully: ./bipartite_js/bipartite.wasm
Everything went okay
ret2basic@PwnieIsland:~/Desktop/bipartite$
```

In ./bipartite_js directory, create input.json:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite/bipartite_js 80x24
  GNU nano 6.2                          input.json *
{"in": [1, 2, 1, 2]}
```

Generate witness and check out:

```
ret2basic@PwnieIsland: ~/Desktop/bipartite/bipartite_js 80x24
ret2basic@PwnieIsland:~/Desktop/bipartite/bipartite_js$ node generate_witness.js
 bipartite.wasm input.json witness.wtns
ret2basic@PwnieIsland:~/Desktop/bipartite/bipartite_js$ snarkjs wtns export json
 witness.wtns
ret2basic@PwnieIsland:~/Desktop/bipartite/bipartite_js$ cat witness.json
[
 "1",
 "1",
 "2",
 "1",
 "2"
]ret2basic@PwnieIsland:~/Desktop/bipartite/bipartite_js$
```

Hacking underconstrained circuit https://www.rareskills.io/post/underconstrained-circom

Vulnerable code:

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 80x24
  GNU nano 6.2                          mul3.circom
pragma circom 2.1.8;

template Mul3() {

    signal input a;
    signal input b;
    signal input c;

    signal output out;

    signal i;

    a * b === 1;   // Force a * b === 1
    i <-- a * b;   // i must be equal 1
    out <== i * c; // out must equal c since i === 1
}

component main{public [a, b, c]} = Mul3();
```

i is assigned but not constrained

https://github.com/0xPARC/zk-bug-tracker?tab=readme-ov-file#8-assigned-but-not-constrained

8. Assigned but not Constrained

A very common bug and misunderstanding is the difference between assignments and constraints. For zk circuits, constraints are the mathematical equations that must be satisfied by any given inputs for a proof to be valid. If any of the inputs result in one of the constraint equations to be incorrect, a valid proof is not possible (well, extremely unlikely).

Assignments, on the other hand, simply assign a value to a variable during proof generation. Assignments do not have to be followed for a valid proof to be created. Often times, an assignment can be used, in combination with other constraints, to reduce the total number of constraints used.

Constraints actually add equations to the R1CS file whereas assignments do not.

Create correct input and store it in input.json:

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 80x24
  GNU nano 6.2                          input.json
{"a": "1", "b": "1", "c": "5"}
```

Compile the circuit into R1CS and generate witness:

```
ret2basic@PwnieIsland:~/Desktop/circom_hack$ circom mul3.circom --r1cs --wasm --sym
template instances: 1
non-linear constraints: 2
linear constraints: 0
public inputs: 3
private inputs: 0
public outputs: 1
wires: 6
labels: 6
Written successfully: ./mul3.r1cs
Written successfully: ./mul3.sym
Written successfully: ./mul3_js/mul3.wasm
Everything went okay
```

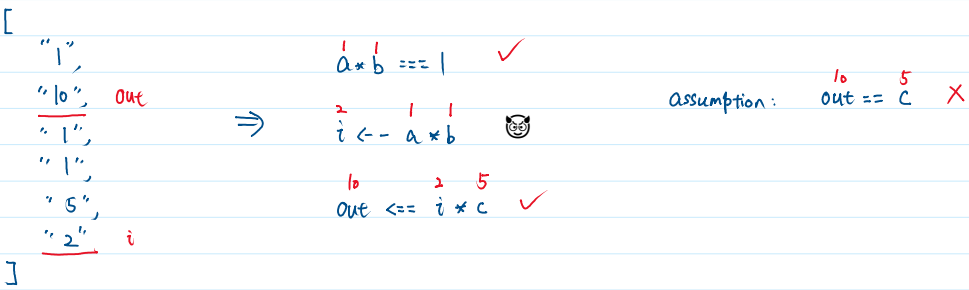
```
ret2basic@PwnieIsland:~/Desktop/circom_hack$ ll
total 28
drwxrwxr-x  3 ret2basic ret2basic 4096 Jul 27 17:45 ./
drwxr-xr-x 37 ret2basic ret2basic 4096 Jul 27 17:23 ../
-rw-rw-r--  1 ret2basic ret2basic   31 Jul 27 17:44 input.json
-rw-rw-r--  1 ret2basic ret2basic  320 Jul 27 17:42 mul3.circom
drwxrwxr-x  2 ret2basic ret2basic 4096 Jul 27 17:45 mul3_js/
-rw-rw-r--  1 ret2basic ret2basic  400 Jul 27 17:45 mul3.r1cs
-rw-rw-r--  1 ret2basic ret2basic   67 Jul 27 17:45 mul3.sym
ret2basic@PwnieIsland:~/Desktop/circom_hack$ cd mul3_js/
ret2basic@PwnieIsland:~/Desktop/circom_hack/mul3_js$ node generate_witness.js mul3.wasm
 ../input.json ../witness.wtns
ret2basic@PwnieIsland:~/Desktop/circom_hack/mul3_js$ ll
total 60
drwxrwxr-x 2 ret2basic ret2basic  4096 Jul 27 17:45 ./
drwxrwxr-x 3 ret2basic ret2basic  4096 Jul 27 17:46 ../
-rw-rw-r-- 1 ret2basic ret2basic   698 Jul 27 17:45 generate_witness.js
-rw-rw-r-- 1 ret2basic ret2basic 34417 Jul 27 17:45 mul3.wasm
-rw-rw-r-- 1 ret2basic ret2basic  9181 Jul 27 17:45 witness_calculator.js
ret2basic@PwnieIsland:~/Desktop/circom_hack/mul3_js$
```

Examine the generated witness:

```
ret2basic@PwnieIsland:~/Desktop/circom_hack/mul3_js$ cd ..
ret2basic@PwnieIsland:~/Desktop/circom_hack$ ll
total 32
drwxrwxr-x  3 ret2basic ret2basic 4096 Jul 27 17:46 ./
drwxr-xr-x 37 ret2basic ret2basic 4096 Jul 27 17:23 ../
-rw-rw-r--  1 ret2basic ret2basic   31 Jul 27 17:44 input.json
-rw-rw-r--  1 ret2basic ret2basic  320 Jul 27 17:42 mul3.circom
drwxrwxr-x  2 ret2basic ret2basic 4096 Jul 27 17:45 mul3_js/
-rw-rw-r--  1 ret2basic ret2basic  400 Jul 27 17:45 mul3.r1cs
-rw-rw-r--  1 ret2basic ret2basic   67 Jul 27 17:45 mul3.sym
-rw-rw-r--  1 ret2basic ret2basic  268 Jul 27 17:46 witness.wtns
ret2basic@PwnieIsland:~/Desktop/circom_hack$ snarkjs wtns export json witness.wtns witness
.json
ret2basic@PwnieIsland:~/Desktop/circom_hack$ ll
total 36
drwxrwxr-x  3 ret2basic ret2basic 4096 Jul 27 17:48 ./
drwxr-xr-x 37 ret2basic ret2basic 4096 Jul 27 17:23 ../
-rw-rw-r--  1 ret2basic ret2basic   31 Jul 27 17:44 input.json
-rw-rw-r--  1 ret2basic ret2basic  320 Jul 27 17:42 mul3.circom
drwxrwxr-x  2 ret2basic ret2basic 4096 Jul 27 17:45 mul3_js/
-rw-rw-r--  1 ret2basic ret2basic  400 Jul 27 17:45 mul3.r1cs
-rw-rw-r--  1 ret2basic ret2basic   67 Jul 27 17:45 mul3.sym
-rw-rw-r--  1 ret2basic ret2basic   38 Jul 27 17:48 witness.json
-rw-rw-r--  1 ret2basic ret2basic  268 Jul 27 17:46 witness.wtns
ret2basic@PwnieIsland:~/Desktop/circom_hack$
```

```
ret2basic@PwnieIsland:~/Desktop/circom_hack$ cat witness.json
[
 "1",  Constant
 "5",  Out
 "1",   a
 "1",   b
 "5",   c
 "1"    i
]ret2basic@PwnieIsland:~/Desktop/circom_hack$
```

Attack plan:



Q: How to manipulate i and out when we only have control over a, b, and c in input.json?
A: Figure out how snarkjs generates R1CS (the file format) by reading its source code:
https://github.com/iden3/circom_runtime/blob/master/build/main.cjs#L533
Then modify witness.wtns by writing a customized JS script.

```
533     async calculateWTNSBin(input, sanityCheck) {
534         const buff32 = new Uint32Array(this.witnessSize*this.n32+this.n32+11);
535         const buff = new  Uint8Array( buff32.buffer);
536         await this._doCalculateWitness(input, sanityCheck);
537
538         //"wtns"
539         buff[0] = "w".charCodeAt(0);
540         buff[1] = "t".charCodeAt(0);
541         buff[2] = "n".charCodeAt(0);
542         buff[3] = "s".charCodeAt(0);
543
544         //version 2
545         buff32[1] = 2;
546
547         //number of sections: 2
548         buff32[2] = 2;
549
550         //id section 1
551         buff32[3] = 1;
552
553         const n8 = this.n32*4;
554         //id section 1 length in 64bytes
555         const idSection1length = 8 + n8;
556         const idSection1lengthHex = idSection1length.toString(16);
557             buff32[4] = parseInt(idSection1lengthHex.slice(0,8), 16);
558             buff32[5] = parseInt(idSection1lengthHex.slice(8,16), 16);
```

...

```
580         // section 2 length
581         const idSection2length = n8*this.witnessSize;
582         const idSection2lengthHex = idSection2length.toString(16);
583         buff32[pos] = parseInt(idSection2lengthHex.slice(0,8), 16);
584         buff32[pos+1] = parseInt(idSection2lengthHex.slice(8,16), 16);
585
586         pos += 2;
587         for (let i=0; i<this.witnessSize; i++) {
588             this.instance.exports.getWitness(i);
589             for (let j=0; j<this.n32; j++) {
590                 buff32[pos+j] = this.instance.exports.readSharedRwMemory(j);
591             }
592             pos += this.n32;
593         }
594
595         return buff;
596     }
```

buff is retuned -> we should parse witness.wtns with Uint8Array.

Our customized script:

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 90x25
  GNU nano 6.2                          display_witness.js *
const fs = require('fs');

const filePath = 'witness.wtns';

const data = fs.readFileSync(filePath);

let data_arr = new Uint8Array(data);
console.dir(data_arr, {'maxArrayLength': null});
```

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 91x27
ret2basic@PwnieIsland:~/Desktop/circom_hack$ node display_witness.js
Uint8Array(268) [
  119, 116, 110, 115,   2,   0,   0,   0,   2,   0,   0,   0,
    1,   0,   0,   0,  40,   0,   0,   0,   0,   0,   0,   0,
   32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
  145, 112, 185, 121,  72, 232,  51,  40,  93,  88, 129, 129,
  182,  69,  80, 184,  41, 160,  49, 225, 114,  78, 100,  48,
    6,   0,   0,   0,   2,   0,   0,   0, 192,   0,   0,   0,
```

Side note: why not console.log()?

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 90x25
  GNU nano 6.2                          display_witness.js
const fs = require('fs');

const filePath = 'witness.wtns';

const data = fs.readFileSync(filePath);

let data_arr = new Uint8Array(data);
console.log(data_arr)
// console.dir(data_arr, {'maxArrayLength': null});
```

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 90x25
ret2basic@PwnieIsland:~/Desktop/circom_hack$ node display_witness.js
Uint8Array(268) [
  119, 116, 110, 115,   2,   0,   0,   0,   2,   0,   0,   0,
    1,   0,   0,   0,  40,   0,   0,   0,   0,   0,   0,   0,
   32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
```

```
  32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
 145, 112, 185, 121,  72, 232,  51,  40,  93,  88, 129, 129,
 182,  69,  80, 184,  41, 160,  49, 225, 114,  78, 100,  48,
   6,   0,   0,   0,   2,   0,   0,   0, 192,   0,   0,   0,
   0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   5,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   5,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0
]
ret2basic@PwnieIsland:~/Desktop/circom_hack$
```

```
ret2basic@PwnieIsland:~/Desktop/circom_hack$ node display_witness.js
Uint8Array(268) [
  119, 116, 110, 115,   2,   0,   0,   0,   2,   0,   0,   0,
    1,   0,   0,   0,  40,   0,   0,   0,   0,   0,   0,   0,
   32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
  145, 112, 185, 121,  72, 232,  51,  40,  93,  88, 129, 129,
  182,  69,  80, 184,  41, 160,  49, 225, 114,  78, 100,  48,
    6,   0,   0,   0,   2,   0,   0,   0, 192,   0,   0,   0,
    0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,
  ... 168 more items
]
ret2basic@PwnieIsland:~/Desktop/circom_hack$
```

Update customized script:

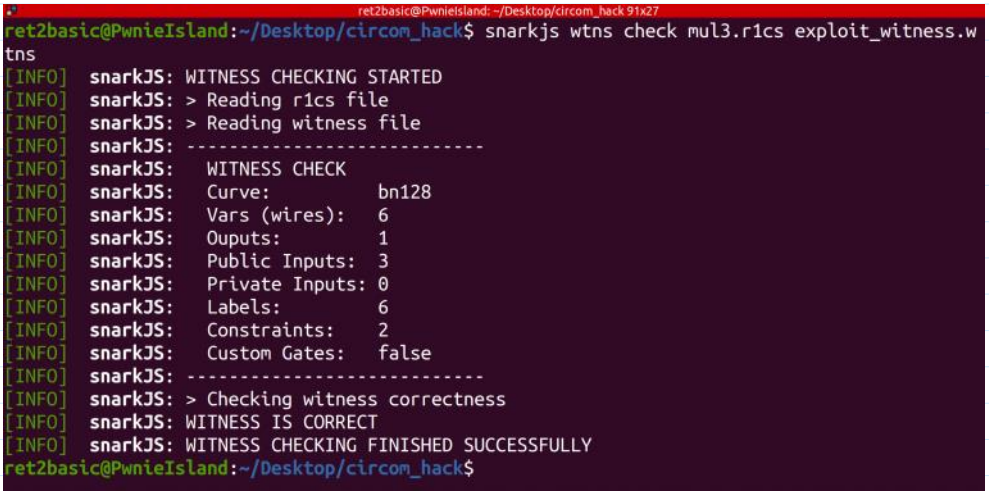
```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 91x27
  GNU nano 6.2                        display_witness.js *
const fs = require('fs');

const filePath = 'witness.wtns';

const data = fs.readFileSync(filePath);
console.log("Before");
console.dir(data, {'maxArrayLength': null});

data[108] = 10; // `out`
data[236] = 2;  // `i`

console.log("After");
console.dir(data, {'maxArrayLength': null});

fs.writeFileSync('exploit_witness.wtns', data);
```

Further improvement: how to automate it?

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 91x27
ret2basic@PwnieIsland:~/Desktop/circom_hack$ node display_witness.js
Before
Buffer(268) [Uint8Array] [
  119, 116, 110, 115,   2,   0,   0,   0,   2,   0,   0,   0,
    1,   0,   0,   0,  40,   0,   0,   0,   0,   0,   0,   0,
   32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
  145, 112, 185, 121,  72, 232,  51,  40,  93,  88, 129, 129,
  182,  69,  80, 184,  41, 160,  49, 225, 114,  78, 100,  48,
    6,   0,   0,   0,   2,   0,   0,   0, 192,   0,   0,   0,
    0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    5,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0
]
```

```
ret2basic@PwnieIsland: ~/Desktop/circom_hack 91x27
After
Buffer(268) [Uint8Array] [
  119, 116, 110, 115,   2,   0,   0,   0,   2,   0,   0,   0,
    1,   0,   0,   0,  40,   0,   0,   0,   0,   0,   0,   0,
   32,   0,   0,   0,   1,   0,   0, 240, 147, 245, 225,  67,
  145, 112, 185, 121,  72, 232,  51,  40,  93,  88, 129, 129,
  182,  69,  80, 184,  41, 160,  49, 225, 114,  78, 100,  48,
    6,   0,   0,   0,   2,   0,   0,   0, 192,   0,   0,   0,
    0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   10,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    5,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   2,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0
]
```

The modified witness works:


ret2basic@PwnieIsland: ~/Desktop/circom_hack 91x27
ret2basic@PwnieIsland:~/Desktop/circom_hack$ snarkjs wtns check mul3.r1cs exploit_witness.w
tns
[INFO] snarkJS: WITNESS CHECKING STARTED
[INFO] snarkJS: > Reading r1cs file
[INFO] snarkJS: > Reading witness file
[INFO] snarkJS: ---------------------------
[INFO] snarkJS: WITNESS CHECK
[INFO] snarkJS: Curve: bn128
[INFO] snarkJS: Vars (wires): 6
[INFO] snarkJS: Ouputs: 1
[INFO] snarkJS: Public Inputs: 3
[INFO] snarkJS: Private Inputs: 0
[INFO] snarkJS: Labels: 6
[INFO] snarkJS: Constraints: 2
[INFO] snarkJS: Custom Gates: false
[INFO] snarkJS: ---------------------------
[INFO] snarkJS: > Checking witness correctness
[INFO] snarkJS: WITNESS IS CORRECT
[INFO] snarkJS: WITNESS CHECKING FINISHED SUCCESSFULLY
ret2basic@PwnieIsland:~/Desktop/circom_hack$