

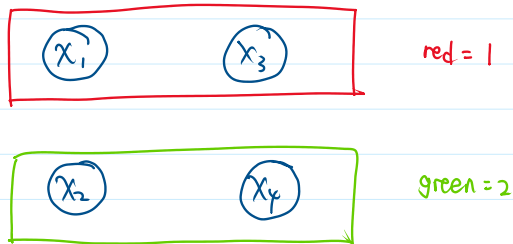
Reference

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>

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":

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

\uparrow how to "2-color" this graph
 Constant term (private input)
 (public 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:

$$x_1 x_2 - 2 = 0 \quad \textcircled{1}$$

$$x_1 x_4 - 2 = 0$$

$$x_2 x_3 - 2 = 0 \quad \textcircled{2}$$

($x_1 x_3 - 2 = 0$ is implied by $\textcircled{1}$ and $\textcircled{2}$ implicitly)

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:

$$x_1 x_2 = 2$$

$$x_1 x_4 = 2$$

$$x_2 x_3 = 2$$

Turn system of equations into matrix form

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: $x_1 x_1 = 3x_1 - 2$

$$\Rightarrow (x_1) \cdot (x_1) = \underline{3x_1 - 2}$$

↑ ↑ ↑
L R O

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

L · a o R · a = O · a

Check if this transformation is done correctly:

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

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

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

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

```
groth16.py X
groth16.py > ...
1 import numpy as np
2 import galois
3 from functools import reduce
4 from py_ecc.bn128 import G1, G2, multiply, add, curve_order, Z1, pairing, neg, final_exponentiate, FQ12
5
6 # curve_order = 1151
7 GF = galois.GF(curve_order) # we work with bn128/bn254 curve
8
```

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:

```
58 # R1CS matrices
59
60 L = np.array([
61     [0, 1, 0, 0, 0],
62     [0, 0, 1, 0, 0],
63     [0, 0, 0, 1, 0],
64     [0, 0, 0, 0, 1],
65     [0, 1, 0, 0, 0],
66     [0, 1, 0, 0, 0],
67     [0, 0, 1, 0, 0],
68 ])
69
70 R = np.array([
71     [0, 1, 0, 0, 0],
72     [0, 0, 1, 0, 0],
73     [0, 0, 0, 1, 0],
74     [0, 0, 0, 0, 1],
75     [0, 0, 1, 0, 0],
76     [0, 0, 0, 0, 1],
77     [0, 0, 0, 1, 0],
78 ])
```

```
80 0 = np.array([
81     [curve_order-2, 3, 0, 0, 0],
82     [curve_order-2, 0, 3, 0, 0],
83     [curve_order-2, 0, 0, 3, 0],
84     [curve_order-2, 0, 0, 0, 3],
85     [2, 0, 0, 0, 0],
86     [2, 0, 0, 0, 0],
87     [2, 0, 0, 0, 0],
88 ])
89
90 L_galois = GF(L)
91 R_galois = GF(R)
92 0_galois = GF(0)
93
```

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

Prover computes witness:

```
94 # In reality this witness is prover's secret, only prover knows it
95 x1 = GF(1)
96 x2 = GF(2)
97 x3 = GF(1)
98 x4 = GF(2)
99 # a is the witness
100 a = GF(np.array([1, x1, x2, x3, x4]))
101
102 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):

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

Do it in circom

A simple piece of circom I wrote to represent the bipartite graph problem

above: <https://zkrepl.dev/?gist=810ac7fb657dc07bd933096cb36b7d5f>

main.circom × + Add File

```
1 pragma circom 2.1.6;
2
3 // bipartite graph problem arithmetization
4
5 template Bipartite(n) {
6     // coloring for 4 vertices x1, x2, x3, x4
7     // in[0] -> x1
8     // in[1] -> x2
9     // in[2] -> x3
10    // in[3] -> x4
11    signal input in[n];
12
13    // Condition 1: color is either 1 or 2 for each vertex
14    (in[0] - 1) * (in[0] - 2) === 0;
15    (in[1] - 1) * (in[1] - 2) === 0;
16    (in[2] - 1) * (in[2] - 2) === 0;
17    (in[3] - 1) * (in[3] - 2) === 0;
18
19    // Condition 2: vertices from different "groups" have different colors
20    in[0] * in[1] === 2;
21    in[0] * in[3] === 2;
22    in[1] * in[2] === 2;
23
24 }
25 component main = Bipartite(4);
26
27 /* INPUT = {"in": [1, 2, 1, 2]} */
```

template instances: 1
non-linear constraints: 7
linear constraints: 0
public inputs: 0
public outputs: 0
private inputs: 4
private outputs: 0
wires: 5
labels: 5
Written successfully: ./main.r1cs
Written successfully: ./main.sym
Written successfully: ./main.js/main.wasm
Everything went okay, circom safe
Compiled in 0.76s

ARTIFACTS:
Finished in 0.87s

- main.wasm (35.02KB)
- main.js (9.18KB)
- main.wtns (0.24KB)
- main.r1cs (1.14KB)
- main.sym (0.07KB)

SAVE:
Saved to Github

<iframe src="https://zkrepl.dev/?gist=810ac7fb657dc07bd933096cb36b7d5f" height="400" width="1000" style="border:1px solid #ddd"></iframe>

KEYS + SOLIDITY + HTML:

Groth16 PLONK Verify

Note: circom operates in scalar field (curve_order) as well:

<https://docs.circom.io/circom-language/basic-operators/>

Field Elements

A field element is a value in the domain of $\mathbb{Z}/p\mathbb{Z}$, where p is the prime number set by default to

$p = 2188824287183927522246405745257275088548364400416034343698204186575808495617$.

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@PwnieIsland: ~/Desktop/zero-knowledge-puzzles 79x22
>>> from py_ecc.bn128 import field_modulus, curve_order
>>> field_modulus
2188824287183927522246405745257275088696311157297823662689037894645226208583
>>> curve_order
2188824287183927522246405745257275088548364400416034343698204186575808495617
>>>
```

Circom behind the scene

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$
```

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: [ 2188824287183927522224640574525727508854836440041603434369820
41865758084956161 +main.in[0] ] * [ 21888242871839275222246405745257275088548364
4004160343436982041865758084956151 +main.in[0] ] - [ ] = 0
[INFO] snarkJS: [ 2188824287183927522224640574525727508854836440041603434369820
41865758084956161 +main.in[1] ] * [ 21888242871839275222246405745257275088548364
4004160343436982041865758084956151 +main.in[1] ] - [ ] = 0
[INFO] snarkJS: [ 2188824287183927522224640574525727508854836440041603434369820
41865758084956161 +main.in[2] ] * [ 21888242871839275222246405745257275088548364
4004160343436982041865758084956151 +main.in[2] ] - [ ] = 0
[INFO] snarkJS: [ 2188824287183927522224640574525727508854836440041603434369820
41865758084956161 +main.in[3] ] * [ 21888242871839275222246405745257275088548364
4004160343436982041865758084956151 +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:


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

Compare $5 = 0101$ $n=4$
and $7 = 0111$

$$\begin{array}{r} 0101 \\ + 10000 \leftarrow 1 \ll 4 \\ \hline 10101 \end{array}$$
$$\begin{array}{r} 10101 \\ - 0111 \\ \hline 01110 \end{array}$$

If MSB is 0, then $a < b$

if MSB is 1, then $a > b$

```
lc1 += out[1] * e2;
e2 = e2 + e2;
}

lc1 == in;
}
```

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

1. $5 = 0101$ $e2 = 1$
 $LC1 = 0$
 $\& \quad 1$

 $1 \rightarrow out[0]$

$$|c| \neq 1 \rightarrow |c| = 1$$
$$e_2 = 2$$

2.
$$\begin{array}{cccc} & 0 & 0 & 1 & 0 & e_2 = 2 \\ & \underline{0} & \underline{0} & \underline{1} & \underline{0} & w_1 = 1 \\ 8 & & & & 1 & \\ \hline & & & & 0 & \rightarrow \text{out}[1] \end{array}$$

$$|C| + 2 \cdot 0 + 1 \rightarrow |C| = 1$$
$$e_2 = 4$$

```
// 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;
}
```