Contents

1 Week 2

2 Week2 Assignments

2.1 Assignment Selection

We offer 3 assignments, so you can choose one or more depending on your situation.

Untitled

Bonus points for completing multiple assignments !!!

2.2 Assignment 1. Polynomial Solving and Proving

Define the following function with 3 variables a, b, c.

```
def f(a, b, c):
  if a == 1:
    return b*c
  return 2b - c
}
```

Polynomial solving and polynomial proving are very important parts of zkSNARKs, while Groth16 and Plonk representations of circuit constraints differently, completing the following polynomial-related problem solving and deduction.

Tips: Refer to Why and How zk-SNARK Works: Definitive Explanation for help.

2.2.1 Q1

To convert the function to circuit of R1CS form (flatten), use polynomial interpolation with values$(1, 2, 3, ...)$ to solve variable polynomials for each variables (including intermediate variables).

2.2.2 A1

Since x is only part of the conditional, we can make a strong assumption that $x \in \{0, 1\}$. The function is equivalent to this function:

$f(x, y, z) = x \cdot y \cdot z + (x - 1) \cdot (2y - z)$

s.t.

Figure 1: Untitled

$x \cdot (x - 1) = 0$

The circuit looks like below:

i.e. for variable mapping:

$\sim one, x, y, z, xminu1, ytimesz, twoyminusz, l, r, \sim out$

s.t.

1. $xminus1 = x \cdot \sim one + (-1) * \sim one$

2. $x * xminus1 = 0$

3. $ytimesz = y * z$

4. $twoyminusz = 2 \cdot \sim one \cdot y - z$

5. $l = ytimesz * x$

6. $r = twoyminusz * xminus1$

7. $\sim out = l + r$

For each numbered gate:

1.

$a = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$
$b = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$c = [1, 0, 0, 0, 1, 0, 0, 0, 0, 0]$

1.

$a = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$
$b = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$
$c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$

1.

$a = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$
$b = [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$
$c = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$

1.

$a = [2, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$b = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$
$c = [0, 0, 0, 1, 0, 0, 1, 0, 0, 0]$

1.

$a = [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$
$b = [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$
$c = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]$

1.

$a = [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$
$b = [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]$
$c = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$

1.

$a = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]$
$b = [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$
$c = [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$

The complete R1CS put together is:

A

$$\begin{vmatrix} 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 \\ 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 \end{vmatrix}$$

B

$$\begin{vmatrix} 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 \\ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 \\ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 \end{vmatrix}$$

C

$$\begin{vmatrix} 1, 0, 0, 0, 1, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 \\ 0, 0, 0, 1, 0, 0, 1, 0, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 \\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 \end{vmatrix}$$

Use Lanrange interpolation to get the variable polynomials for:

```
:'
For A:
x    1,2,3,4,5,6,7
  +-----------------
1 |[[0,0,0,2,0,0,0],
2 | [1,1,0,0,1,0,0],
3 | [0,0,1,0,0,0,0],
4 | [0,0,0,0,0,0,0],
5 | [0,0,0,0,0,1,0],
6 | [0,0,0,0,0,0,0],
7 | [0,0,0,0,0,0,0],
8 | [0,0,0,0,0,0,1],
9 | [0,0,0,0,0,0,0],
10| [0,0,0,0,0,0,0]]

For B:
x    1,2,3,4,5,6,7
  +-----------------
1 |[[1,0,0,0,0,0,0],
2 | [0,0,0,0,0,0,0],
3 | [0,0,0,1,0,0,0],
4 | [0,0,1,0,0,0,0],
5 | [0,1,0,0,0,0,0],
6 | [0,0,0,0,1,0,0],
7 | [0,0,0,0,0,1,0],
8 | [0,0,0,0,0,0,0],
9 | [0,0,0,0,0,0,1],
10| [0,0,0,0,0,0,0]]

For C:
x    1,2,3,4,5,6,7
  +-----------------
1 |[[1,0,0,0,0,0,0],
2 | [0,0,0,0,0,0,0],
3 | [0,0,0,0,0,0,0],
4 | [0,0,0,1,0,0,0],
5 | [1,0,0,0,0,0,0],
6 | [0,0,1,0,0,0,0],
7 | [0,0,0,1,0,0,0],
8 | [0,0,0,0,1,0,0],
```

```
9 | [0,0,0,0,0,1,0],
10| [0,0,0,0,0,0,1]]
,
```

Use the following Python code to compute:

```python
from scipy.interpolate import lagrange
import numpy as np
from numpy.polynomial import polynomial as P

np.set_printoptions(precision=2)

def poly_interpolate(index, matrics):
    res = []
    for row in matrics:
        poly = lagrange(index, row).coef[::-1]
        res.append(poly)
    return np.array(res, dtype=object)

def polymuln(lst):
    if len(lst) == 0:
        return 1
    return P.polymul(lst[0], polymuln(lst[1:]))

def QAP(s, A, B, C):
    L = np.matmul(s, A)
    R = np.matmul(s, B)
    O = np.matmul(s, C)
    print(f"L: {L}")
    print(f"R: {R}")
    print(f"O: {O}")
    Q = P.polysub(P.polymul(L, R), O)
    print(f"T:\n {Q}")
    assert len(L) == len(R) == len(O)
    return P.polydiv(Q, Z(len(L)))

def Z(n):
    ns = [[-(1 + i), 1] for i, _ in enumerate(range(n))]
    return polymuln(ns)
```

7

```python
def main(s, A, B, C):
    x = np.arange(1, len(A) + 1)

    resA = poly_interpolate(x, np.transpose(A))
    resB = poly_interpolate(x, np.transpose(B))
    resC = poly_interpolate(x, np.transpose(C))

    print(f"resA: \n {resA}")
    print(f"resB: \n {resB}")
    print(f"resC: \n {resC}")

    return QAP(s, resA, resB, resC)

# variables: \sim{one},x,y,z,xminu1,ytimesz,twoyminusz,l,r,\sim{out}
s = np.array([1, 0, 11, 5, 0, 55, 17, 0, -17, -17])

A = np.array(
    [
        [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 1, 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, 1, 0, 0],
    ]
)


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

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

# print(main(s, A, B, C))

# A = np.array(
#     [
#         [0, 1, 0, 0, 0, 0],
#         [0, 0, 0, 1, 0, 0],
#         [0, 1, 0, 0, 1, 0],
#         [5, 0, 0, 0, 0, 1],
#     ]
# )

# B = np.array(
#     [
#         [0, 1, 0, 0, 0, 0],
#         [0, 1, 0, 0, 0, 0],
#         [1, 0, 0, 0, 0, 0],
#         [1, 0, 0, 0, 0, 0],
#     ]
# )

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

```
# A_ = np.array(
#     [
#         [-5.0, 9.166, -5.0, 0.833],
#         [8.0, -11.333, 5.0, -0.666],
#         [0.0, 0.0, 0.0, 0.0],
#         [-6.0, 9.5, -4.0, 0.5],
#         [4.0, -7.0, 3.5, -0.5],
#         [-1.0, 1.833, -1.0, 0.166],
#     ]
# )
# B_ = np.array(
#     [
#         [3.0, -5.166, 2.5, -0.333],
#         [-2.0, 5.166, -2.5, 0.333],
#         [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, 0.0],
#     ]
# )
# C_ = [
#     [0.0, 0.0, 0.0, 0.0],
#     [0.0, 0.0, 0.0, 0.0],
#     [-1.0, 1.833, -1.0, 0.166],
#     [4.0, -4.333, 1.5, -0.166],
#     [-6.0, 9.5, -4.0, 0.5],
#     [4.0, -7.0, 3.5, -0.5],
# ]
# s = np.array([1, 3, 35, 9, 27, 30])

# print(main(s, A, B, C))
```

For variables $\sim one(1), x(2), y(3), z(4), xminu1(5), ytimesz(6), twoyminusz(7), l(8), r(9), \sim out(10)$ in R1CS flatten format:

A variable polynomials with coefficients:

$$
\begin{pmatrix}
\sim one \\
x \\
y \\
z \\
xminu1 \\
ytimesz \\
twoyminusz \\
l \\
r \\
\sim out
\end{pmatrix}
=
\begin{pmatrix}
-7.00e+01 & 1.64e+02 & -1.41e+02 & 5.87e+01 & -1.26e+01 & 1.33e \\
7.00e+00 & -1.74e+01 & 1.90e+01 & -9.75e+00 & 2.47e+00 & -3.00e \\
3.50e+01 & -7.91e+01 & 6.48e+01 & -2.54e+01 & 5.15e+00 & -5.21e \\
0 & 0 & 0 & 0 & 0 & 0 \\
-7.00e+00 & 1.70e+01 & -1.54e+01 & 6.83e+00 & -1.58e+00 & 1.83e \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
1.00e+00 & -2.45e+00 & 2.26e+00 & -1.02e+00 & 2.43e-01 & -2.92e \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

$$
\begin{pmatrix}
1 \\
x \\
x^2 \\
x^3 \\
x^4 \\
x5 \\
x6
\end{pmatrix}
$$

B variable polynomials with coefficients:

$$
\begin{pmatrix}
\sim one \\
x \\
y \\
z \\
xminu1 \\
ytimesz \\
twoyminusz \\
l \\
r \\
\sim out
\end{pmatrix}
=
\begin{pmatrix}
7.00e+00 & -1.12e+01 & 7.09e+00 & -2.31e+00 & 4.10e-01 & -3.75e \\
0 & 0 & 0 & 0 & 0 & 0 \\
-3.50e+01 & 8.20e+01 & -7.07e+01 & 2.93e+01 & -6.28e+00 & 6.67e \\
3.50e+01 & -7.91e+01 & 6.48e+01 & -2.54e+01 & 5.15e+00 & -5.21e \\
-2.10e+01 & 4.40e+01 & -3.27e+01 & 1.18e+01 & -2.25e+00 & 2.17e \\
2.10e+01 & -5.02e+01 & 4.47e+01 & -1.93e+01 & 4.31e+00 & -4.79e \\
-7.00e+00 & 1.70e+01 & -1.54e+01 & 6.83e+00 & -1.58e+00 & 1.83e \\
0 & 0 & 0 & 0 & 0 & 0 \\
1.00e+00 & -2.45e+00 & 2.26e+00 & -1.02e+00 & 2.43e-01 & -2.92e \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

$$
\begin{pmatrix}
1 \\
x \\
x^2 \\
x^3 \\
x^4 \\
x5 \\
x6
\end{pmatrix}
$$

C variable polynomials with coefficients:

11

$$
\begin{pmatrix}
\sim one \\
x \\
y \\
z \\
xminu1 \\
ytimesz \\
twoyminusz \\
l \\
r \\
\sim out
\end{pmatrix}
=
\begin{pmatrix}
7.00e+00 & -1.12e+01 & 7.09e+00 & -2.31e+00 & 4.10e-01 & -3.75e \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
-3.50e+01 & 8.20e+01 & -7.07e+01 & 2.93e+01 & -6.28e+00 & 6.67e \\
7.00e+00 & -1.12e+01 & 7.09e+00 & -2.31e+00 & 4.10e-01 & -3.75e \\
3.50e+01 & -7.91e+01 & 6.48e+01 & -2.54e+01 & 5.15e+00 & -5.21e \\
-3.50e+01 & 8.20e+01 & -7.07e+01 & 2.93e+01 & -6.28e+00 & 6.67e \\
2.10e+01 & -5.02e+01 & 4.47e+01 & -1.93e+01 & 4.31e+00 & -4.79e \\
-7.00e+00 & 1.70e+01 & -1.54e+01 & 6.83e+00 & -1.58e+00 & 1.83e \\
1.00e+00 & -2.45e+00 & 2.26e+00 & -1.02e+00 & 2.43e-01 & -2.92e
\end{pmatrix}
$$

$$
\begin{pmatrix}
1 \\
x \\
x^2 \\
x^3 \\
x^4 \\
x5 \\
x6
\end{pmatrix}
$$

2.2.3 Q2

Continuing with Q1, solve for the left operand polynomial L(X), the right operand polynomial R(X), the output operand polynomial O(X), and the target polynomial T(X) = (x-1)(x-2)(x-3)... (interpolation of Q1 solution process), after assigning values to the variables (x=1, y=2, z=5)... , verify if $L(X) * R(X) - O(X)$ is divisible by T(X) and why?

> Note: The polynomial solution problems involved in Q1, Q2 are the very core of Groth16, Pinocchio

2.2.4 A2

Run the above Python code and the logs returned polynomials with coefficients:

```
L: [ 3.15e+02 -7.06e+02  5.72e+02 -2.21e+02  4.40e+01 -4.40e+00  1.74e-01]
R: [ 8.16e+02 -1.94e+03  1.71e+03 -7.33e+02  1.63e+02 -1.80e+01  7.81e-01]
O: [ 1.26e+03 -2.80e+03  2.24e+03 -8.53e+02  1.68e+02 -1.66e+01  6.54e-01]
T: [-5.04e+03  1.31e+04 -1.31e+04  6.77e+03 -1.96e+03  3.22e+02 -2.80e+01
  1.00e+00]
```

$L * R - O$ should be divisible by T with zero (almost due to floating point). The reason is:

1. is defined as `(x - 1) * (x - 2) * (x - 3) ...` - the simplest polynomial that is equal to zero at all points that correspond to logic gates. It is an elementary fact of algebra that *any* polynomial that is equal to zero at all of these points has to be a multiple of this minimal polynomial, and if a polynomial is a multiple of Z then its evaluation at any of those points will be zero

Since $1, 2, \ldots, 10$ are x values used in Lagrange interpolation, the curve $L * R - T$ should have them as roots. As quoted above, all polynomials should be divisible by the product of roots.

2.2.5 Q3

Convert the function into a system of circuit constrained equations (addition, multiplication, input, output, etc.), then convert these circuit constraints into a system of circuit constrained equations of the form defined by Plonk as "(qL) - a + (qR) - b + (qO) - c + (qM) - (a * b) + (qC) = 0", and finally perform a polynomial interpolation (1,2,3,...) Solve for: qL(x), qR(x), ... , qM(x)

2.2.6 A3

Considering all inputs as private, we get these 7 PLONK gates:
$$\begin{pmatrix} 0a_1 + 0b_1 + -1c_1 + -1a_1b_1 + 0 = 0 \\ 0a_2 + 0b_2 + 0c_2 + 1a_2b_2 + 0 = 0 \\ 0a_3 + 0b_3 + -1c_3 + 1a_3b_3 + 0 = 0 \\ 2a_4 + -1b_4 + -1c_4 + 0a_4b_4 + 0 = 0 \\ 0a_5 + 0b_5 + -1c_5 + 1a_5b_5 + 0 = 0 \\ 0a_6 + 0b_6 + -1c_6 + 1a_6b_6 + 0 = 0 \\ 1a_7 + 1b_7 + -1c_7 + 0a_7b_7 + 0 = 0 \end{pmatrix}$$
We collect parameters into vectors:
$$q_L = (0, 0, 0, 2, 0, 0, 1)^T$$
$$q_R = (0, 0, 0, -1, 0, 0, 1)^T$$
$$q_O = (-1, 0, -1, -1, -1, -1, -1)^T$$
$$q_M = (-1, 1, 1, 0, 1, 1, 0)^T$$
$$q_C = (0, 0, 0, 0, 0, 0, 0)^T$$
$$\mathbf{a} = (0, 0, 11, 11, 55, 17, 0)^T$$
$$\mathbf{b} = (1, -1, 5, 5, 0, -1, -17)^T$$
$$\mathbf{c} = (-1, 0, 55, 17, 0, -17, -17)^T$$
Use the following Python code:

```python
from scipy.interpolate import lagrange
import numpy as np
from numpy.polynomial import polynomial as P
from a1 import poly_interpolate, Z

np.set_printoptions(precision=2)

Q = np.array(
    [
        [0, 0, 0, 2, 0, 0, 1],
        [0, 0, 0, -1, 0, 0, 1],
        [-1, 0, -1, -1, -1, -1, -1],
        [-1, 1, 1, 0, 1, 1, 0],
        [0, 0, 0, 0, 0, 0, 0],
    ]
)
ABC = np.array(
    [
        [0, 0, 11, 11, 55, 17, 0],
        [1, -1, 5, 5, 0, -1, -17],
        [-1, 0, 55, 17, 0, -17, -17],
    ]
)


def PLONK(Q, ABC):
    index = np.arange(1, np.shape(Q)[1] + 1)
    q_L, q_R, q_O, q_M, q_C = [
        P.Polynomial(coeffs) for coeffs in poly_interpolate(index, Q)
    ]

    a, b, c = [P.Polynomial(coeffs) for coeffs in poly_interpolate(index, ABC)]

    # qL(x) - a(x) + qR(x) - b(x) + qO(x) - c(x) + qM(x) - (a(x) * b(x)) + qC(x)
    T = q_L - a + q_R - b + q_O - c + q_M - (a * b) + q_C
    z = Z(len(index))
    return P.polydiv(T.coef, z)
```

we get:

$$\begin{pmatrix} q_L(x) \\ q_R(x) \\ q_O(x) \\ q_M(x) \\ q_C(x) \end{pmatrix} = \begin{pmatrix} -6.90e+01 & 1.62e+02 & -1.39e+02 & 5.76e+01 & -1.23e+01 & 1.30e+00 \\ 3.60e+01 & -8.45e+01 & 7.30e+01 & -3.04e+01 & 6.52e+00 & -6.96e-01 \\ -2.20e+01 & 4.40e+01 & -3.27e+01 & 1.18e+01 & -2.25e+00 & 2.17e-01 \\ 2.10e+01 & -5.72e+01 & 5.43e+01 & -2.37e+01 & 5.22e+00 & -5.62e-01 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{pmatrix}$$

2.2.7   Q4

Continuing with Q3, assign the variables (x=1, y=2, z=5) and then interpolate (1,2,3,...) Solve for: a(x), b(x), c(x), then verify if qL(x) - a(x) + qR(x) - b(x) + qO(x) - c(x) + qM(x) - (a(x) * b(x)) + qC(x) can be obtained by the target polynomial T(x) = (x-1)(x-2)(x-3)...

> Note: The polynomial solution problems involved in Q3, Q4 are
> fundamentals of Plonk.

2.2.8   A4

Run the code above, we can get $a(x), b(x), c(x)$:

$$\begin{pmatrix} a(x) \\ b(x) \\ c(x) \end{pmatrix} = \begin{pmatrix} 1.04e+03 & -2.44e+03 & 2.13e+03 & -9.00e+02 & 1.98e+02 & -2.16e+01 & 9. \\ 1.80e+01 & -1.58e+01 & -1.24e+01 & 1.61e+01 & -5.55e+00 & 7.87e-01 & -4 \\ 1.42e+03 & -3.19e+03 & 2.58e+03 & -9.95e+02 & 1.99e+02 & -1.99e+01 & 7. \end{pmatrix}$$

$$\begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \end{pmatrix}$$

And we can check the divisibility of $Z \mid qL(x) - a(x) + qR(x) - b(x) + qO(x) - c(x) + qM(x) - (a(x) * b(x)) + qC(x)$ by

```
P.polydiv(T.coef, z) # z = T(x) = (x-1)(x-2)(x-3)...
```

2.3 Assignment 3. Classical Protocol Engineering Practice

The following function is known, with 3 variables x, y, z.

```
def f(x, y, z):
  if x == 1:
    return y*z
  return 2y - z
}
```

   prover owns private variables x, y, z with values (x = 1, y = 2, z = 5), please choose one open source libraries or toolkits of Groth16, Plonk, Stark protocol, and use one of the protocols to complete the verifiable computation of function `f(x,y,z)` as above.
   Engineering Assignment Requirements:

1. please briefly describe the use of the protocol code base or framework

2. please explain the implementation idea in steps

3. if there is an output of the calculation step, please give a screenshot of the result

4. please provide the project practice repository, specify the commit hash

2.3.1 Answer

1. I use `circom 2` with `snarkjs`.

2. I follow the steps:

   (a) Post input signals in `input.json`
   (b) Generate supplementary files following procedures described here
   (c) Generate witness by running `./circuit_js/witness_calculator.js`
   (d) Generate proof and public signals using PLONK
   (e) Use `verification_key.json` to verify the proof

3. Log the proof and verification result

4. `main.js`: Commit hash `4bba17dbb299f0ac05096441554e0b8db24ab7fe` from Github repo: `https://github.com/tlkahn/zkworkshop-w2-a3`

```
proof:ircuit_js                          JS main.js > [@] wc
{ ■ circuit.wasm                          3    const snarkjs = require('snarkjs')
 A: [
   '2688704540252528704292058653638129469477832575026398846122328681668312241381',
   '13386016281384020957751886669409370247147468045812789079857144053325573060599',
   '1'
 ], node_modules                          4    const wc = require('./circuit_js/witness_cal
 B: [ircuit_final.zkey                     1    const wasm = './circuit_js/circuit.wasm'
   '13543033197325718127687354891627387427888818465273735237338774114971996083008',
   '12343004014869996850946880851729046993600200491537215259797164009563482432637',
   '1'                                     3    const INPUTS_FILE = './input.json'
 ], circuit.r1cs
 C: [circuit.r1cs.json                     4    const WITNESS_FILE = './witness.wtns'
   '14892355985813793125542568674893333897065908812188973714760532904990655485547',
   '73537652352151780701202421900849357754886630494740776900280724944733002716 25'  ts) => {
   '1'
 ], circuit.sym                           7       const buffer = fs.readFileSync(wasm)
 Z: [input.json                           8       const witnessCalculator = await wc(buffe
   '18423498219526506094453969366025727674459800753460538561115406104397524697036'  tor.cal
   '10884444433911389064924641273899800729553484231910706385925651865604807985223',
   '1'                                              ())
 ], package-lock.json
 T1: [package.json                        10      fs.writeFileSync(WITNESS_FILE, buff)
   '14461522760010978110071018619297480123432696433429303853669157819359850884483',
   '13674586389730134785297774846388197388687160421720947984091723203935709653213',
   '1'pot12_0001.ptau
                                          13    const main = async () => {
 ], pot12_0002.ptau
 T2: [                                     14      const inputSignals = JSON.parse(fs.readF
   '15180879072871270250675439436723420422391842654349774409651395498457039470693',
   '21766104876857024656620821925801061801473989658363868319455596741812640163419'
   '1'pot12_final.ptau                     15      await generateWitness(inputSignals)
 ], proof.json                            16      const { proof, publicSignals } = await s
 T3: [public.json                                  (zkey, WITNESS_FILE)
   '4701300466570190130015800747258017817606394813698198343452006762840160834824',
   '16978783720225010910453779106190918792775197749057552787770836842479063901269',
   '1'                                     18      const vkey = JSON.parse(fs.readFileSync(
 ], witness.wtns
 eval_a: '5327827115816381893759647321562386491218097646213634212861953796667804607025',
 eval_b: '4521583154211037167111960668024875615874885976958346785376536401327452230187',  y(v
 eval_c: '17866722008669314876285040090705907064354403528092230842733006106368576222818',
 eval_s1: '14801802111833685506630655069738641039116502347312770505709842381043574810357',
 eval_s2: '6181062214335597582895265231159333682627944386116168818170192684469940852107',
 eval_zw: '13729000815616769420878426791801927946937814307570666501973077248595960700 62',
 eval_r: '4300023916231751257193653541671511747665804315970338675413388589775171081221',
 Wxi: [                                    22           } else {
   '16687718634755293933374998791334536457029558131532652668649874658536687673325',
   '45899806366130472357707764161918388200030542502746672433729433297288856900 23',
   '1'                                     24           }
 ],                                        25      }
 Wxiw: [
   '16663950425298083041711167217106600015039157613323457588923397865667855723704',
   '21085094157367782481038682418513333599087935334477059046064248293235488997901',
   '1'
 ],                                        28      process.exit(0)
 protocol: 'plonk',                        29    })
 curve: 'bn128'
}
Verification OK
```

Figure 2: Screen Shot 2022-07-20 at 15.51.42.png