Testing Framework for Zero Knowledge Programs

Wen-Ding Li

Motivation

- Zero knowledge programming now is practical
- Developers want to verify or test their programs as they want this to work in privacy or financial applications with **a lot at stake**

Background: Making ZK Available to Developers

- People developed DSL to let developers describe a function in the family $\mathscr{F}$
- What exactly are the statements can the system prove?

  Let’s look at a concrete system called zkSNARK

Background: Rank-1 Constraint System

- We say $(x_1, x_2, \ldots, x_n) \in F_p^n$ satisfied a Rank-1 Constraint System (R1CS) iff the following are all satisfied

$$(a_{11}x_1 + a_{12}x_2 + \ldots, a_{1n}x_n) \cdot (b_{11}x_1 + b_{12}x_2 + \ldots, b_{1n}x_n) = c_{11}x_1 + c_{12}x_2 + \ldots, c_{1n}x_n$$
$$(a_{21}x_1 + a_{22}x_2 + \ldots, a_{2n}x_n) \cdot (b_{21}x_1 + b_{22}x_2 + \ldots, b_{2n}x_n) = c_{21}x_1 + c_{22}x_2 + \ldots, c_{2n}x_n$$
$$(a_{31}x_1 + a_{32}x_2 + \ldots, a_{3n}x_n) \cdot (b_{31}x_1 + b_{32}x_2 + \ldots, b_{3n}x_n) = c_{31}x_1 + c_{32}x_2 + \ldots, c_{3n}x_n$$
$$\ldots$$
$$(a_{m1}x_1 + a_{m2}x_2 + \ldots, a_{mn}x_n) \cdot (b_{m1}x_1 + b_{m2}x_2 + \ldots, b_{mn}x_n) = c_{m1}x_1 + b_{m2}x_2 + \ldots, b_{mn}x_n$$

  where $a_{ij}, b_{i,j}, c_i$ are all in F_p

Background: Compiler

- People have developed some high level DSL and can compile to R1CS form
  - xJsnark by Elaine Shi et al. *in S&P 2018*
  - Viaduct by Andrew Myers et al. *in PLDI 2021*
  - Cairo by Starkware in 2021
  - CirC in 2021
  - Leo in 2021

Problem: Developers are still writing R1CS Form!

- Two main reasons:
  - Developers and tooling still not catch up yet
  - Manually optimize R1CS form is sometimes necessary to get efficient program because **proving time is proportional to the number of constraints**
    - Writing R1CS is like writing assembly level code!

Example

- Given an integer x compute the multiplicative inverse y corresponding in the underlying field F_p and assume 0 <= x < p
  - If we Implement this in a general programming language without field arithmetic: extended GCD algorithm or using Fermat's little theorem
  - R1CS form
    $x \cdot y = 1$

Example

- Given a number x in [0, 15] return its binary representation

  Input: x of type integer

  Output: four boolean variables y0, y1, y2, y3 corresponding to the binary representation of x

```
y0 := x % 2
x := x / 2
y1 := x % 2
x :=. x / 2
y2 := x % 2
x := x/2
y3 := x % 2
```

Example

- Given a number in [0, 15] return its binary representation
  - Optimized R1CS form

$$x = y_0 + 2 \cdot y_1 + 4 \cdot y_2 + 8 \cdot y_3$$

$$y_0 \cdot (y_0 - 1) = 0$$

$$y_1 \cdot (y_1 - 1) = 0$$

$$y_2 \cdot (y_2 - 1) = 0$$

$$y_3 \cdot (y_3 - 1) = 0$$

Example

- Input an integer x in F_p
- Output an integer y= $\begin{cases} 1, \text{ if x is zero} \\ 0, \text{ otherwise} \end{cases}$
- R1CS form

  $x \cdot y = 0$

  $y + x \cdot z = 1$

  (z is also a variable, and it's there to help to make it work)

Problem: Verifying or Testing R1CS form is hard

- People now are still writing this kind of low level R1CS form
- Given a spec, people want to verifying if their handwritten R1CS form is correct

Given a function $f : F_p \to F_p$ as our spec and

an R1CS $CS : F_p \times F_p^m \times F_p \to \{\text{True, False}\}$

- *(Existence)* $\forall x \,.\, \exists z_1, z_2, \ldots, z_m \,.$

$$CS(x, z_1, z_2, \ldots, z_m, f(x))$$

- *(Uniqueness)* $\forall x \,.\, \forall z_1, z_2, \ldots, z_m \,.\, \forall y \,.$

$$CS(x, z_1, z_2, \ldots, z_m, y) \Rightarrow y = f(x)$$

Given x as a test case input

- *(Testing Existence)* $\exists z_1, z_2, \ldots, z_m \,.$

$$CS(x, , z_1, z_2, \ldots, z_m, f(x_1, x_2, \ldots, x_n))$$

- *(Testing Uniqueness)* $\forall z_1, z_2, \ldots, z_m \,. \forall y \,.$

$$CS(x, z_1, z_2, \ldots, z_m, y) \Rightarrow y = f(x_1, x_2, \ldots, x_n)$$

Given x as a test case input

- *(Testing Existence)* $\exists z_1, z_2, \ldots, z_m \,.$

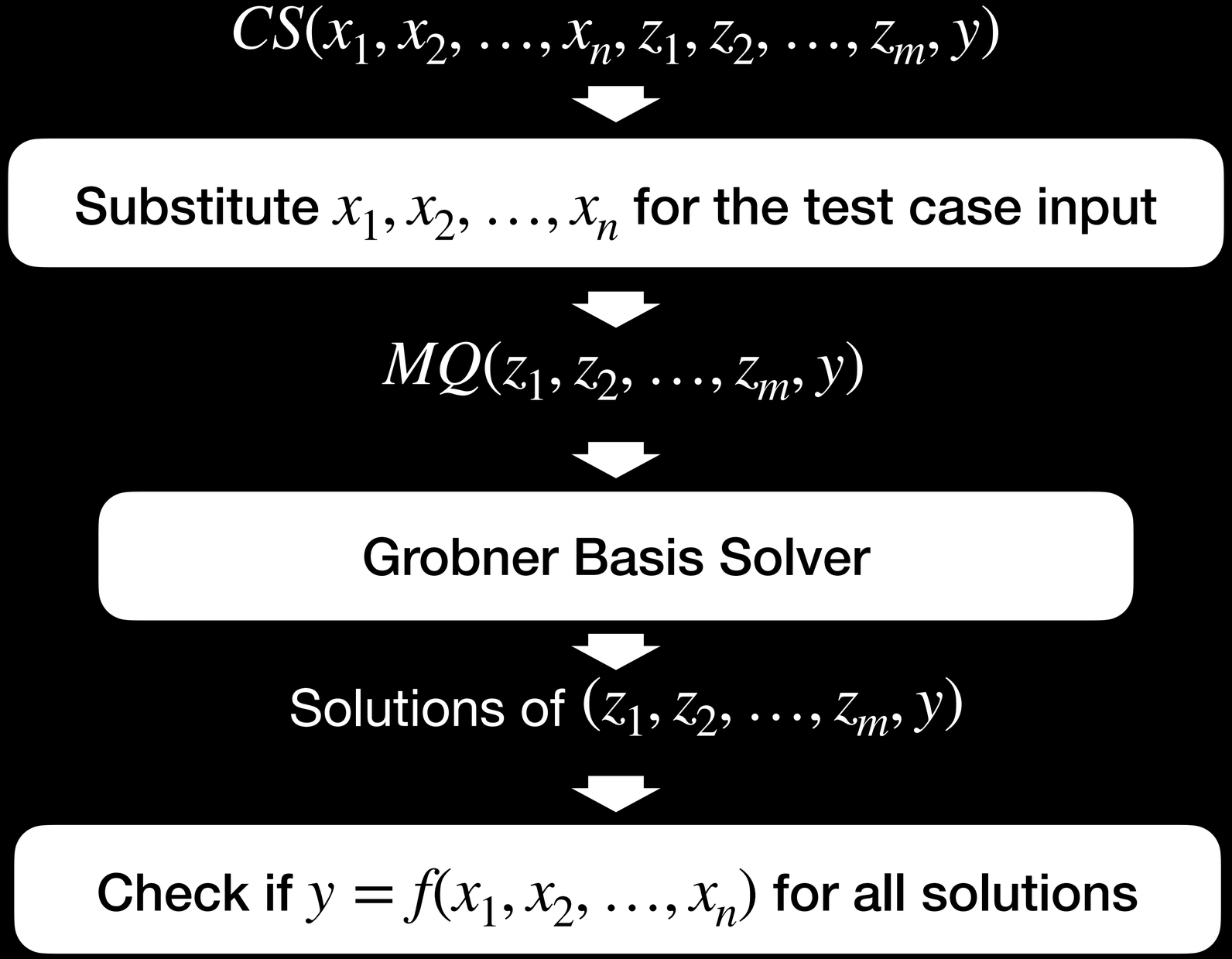
$$CS(x, , z_1, z_2, \ldots, z_m, f(x_1, x_2, \ldots, x_n))$$

People have written tests for this

- *(Testing Uniqueness)* $\forall z_1, z_2, \ldots, z_m \,. \forall y \,.$

$$CS(x, z_1, z_2, \ldots, z_m, y) \Rightarrow y = f(x_1, x_2, \ldots, x_n)$$

People don’t have tests for this


$CS(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m, y)$
Substitute $x_1, x_2, \ldots, x_n$ for the test case input
$MQ(z_1, z_2, \ldots, z_m, y)$
Grobner Basis Solver
Solutions of $(z_1, z_2, \ldots, z_m, y)$
Check if $y = f(x_1, x_2, \ldots, x_n)$ for all solutions

$CS(x_1, x_2, \ldots, x_n, z_1, z_2, \ldots, z_m, y)$

Substitute $x_1, x_2, \ldots, x_n$ for the test case input

$MQ(z_1, z_2, \ldots, z_m, y)$

Grobner Basis Solver

Assume $(z_1, z_2, \ldots, z_m)$ should also be unique

Check if the solution is unique

Demo

```
pragma circom 2.0.2;

template Num2FourBits() {
    signal input x;
    signal output bits[4];

    for(var i = 0; i < 4; i++){
        bits[i] <-- (x >> i) & 1;
    }
    x === bits[0] + bits[1] * 2 + bits[2] * 4  + bits[3] * 8;
    bits[0] * (bits[0]-1) === 0;
    bits[1] * (bits[1]-1) === 0;
    bits[2] * (bits[2]-1) === 0;
    bits[3] * (bits[3]-1) === 0;
}
```

Demo

```
thekev@zktest $ yarn test --grep "Num2FourBits"
yarn run v1.22.17
$ mocha -r ts-node/register 'test/**/*.ts' --grep Num2FourBits


  Num2FourBits
    ✔ Testing x: 12, expect output 0,0,1,1 (69ms)


  1 passing (159ms)

Done in 2.39s.
```

Demo: Buggy Code

```
pragma circom 2.0.2;

template Num2FourBits() {
    signal input x;
    signal output bits[4];

    for(var i = 0; i < 4; i++){
        bits[i] <-- (x >> i) & 1;
    }
    x === bits[0] + bits[1] * 2 + bits[2] * 4  + bits[3] * 8;
    bits[0] * (bits[0]-1) === 0;
    bits[1] * (bits[1]-1) === 0;
    bits[2] * (bits[2]-1) === 0;
    //bits[3] * (bits[3]-1) === 0;
}
```

Demo

```
thekev@zktest $ yarn test --grep "Num2FourBits"
yarn run v1.22.17
$ mocha -r ts-node/register 'test/**/*.ts' --grep Num2FourBits


  Num2FourBits
    ✔ Testing x: 12, expect output 0,0,1,1 (58ms)


  1 passing (129ms)

Done in 2.35s.
```

Demo: Running our Solver-based test

```
thekev@zktest $ yarn test --grep "Num2FourBits"
yarn run v1.22.17
$ mocha -r ts-node/register 'test/**/*.ts' --grep Num2FourBits


  Num2FourBits
    1) Testing x: 12, expect output 0,0,1,1


  0 passing (2s)
  1 failing

  1) Num2FourBits
       Testing x: 12, expect output 0,0,1,1:
     AssertionError: Error: solutions might Not be unique

      at Object.checkConstraints (node_modules/circom_r1cs_tester/index.js:23:9)
      at Context.<anonymous> (test/demo.test.ts:30:34)



error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

Limitations

- Grobner basis solver can only handle ~10 variables in reasonable amount of time

Future work

- Compilers for new proving systems
- Finding a good intermediate representation (IR)