ZK SHANGHAI
零知识证明工作坊

WORKSHOP!

CIRCOM基础电路

现代零知识密码学

Hosted by SutuLabs & Kepler42B-ZK Planet

课程资源：zkshanghai.xyz

个人介绍

梁爽

区块链 架构师

上海交大 计算机博士生
（休学创业中）

微信：icerdesign
微博：@wizicer
Github: @wizicer
Twitter: @icerdesign
LinkedIn: www.linkedin.com/in/icerdesign

1999年	• 正式开始学习写程序
2009年	• 在新媒传信（飞信）做高性能服务器程序架构及开发
2012年	• 在Honeywell工业控制部门做PLC、RTU上位机组态软件架构及开发
2017年	• 接触区块链，并开始创业开发区块链数据库
2020年	• 入学上海交大攻读博士学位，研究零知识证明数据库
2022年	• 获Chia全球开发大赛第一名，并开始Pawket钱包的开发
2023年	• 获得零知识链Mina的项目资助

zkSNARKs

什么是 zkSNARK?

- 一种新的密码学工具，可以针对任何问题高效地生成零知识协议。
- 特性：
  - zk 零知识：隐藏输入
  - Succinct 简洁：生成可以快速验证的简短证明
  - Noninteractive 非交互式：不需要来回交互
  - ARgument of Knowledge 知识论证：证明你知道输入

什么是 zkSNARK?

- 高层次的想法：
  - 将问题（图同构、离散对数等）转换为您要隐藏其输入的函数。
- 将该函数转换为等效的“R1CS”（或其他）方程组
  - 算术电路：一堆在素数域元素中的 + 和 * 操作
  - 简化：形式为 $x_i + x_j = x_k$ 或 $x_i * x_j = x_k$ 的方程
- 为 R1CS 的可满足性生成 ZKP

可满足性问题(Boolean Satisfiability Problem)，简称SAT问题，源于数理逻辑中经典命题逻辑关于公式的可满足性的概念，是理论计算机科学中一个重要的问题，也是第一个被证明的NP-complete问题。

Ref: https://zhuanlan.zhihu.com/p/432853785

范例

可满足：F=A^~B

(A=TRUE, B=FALSE => F=TRUE)

不可满足：F=A^~A

(A=TRUE, ~A=FALSE => F=FALSE)

zkSNARK 属性

- 一种新的密码学工具，可以针对任何问题高效地生成零知识协议。
- 特性：
  - zk：隐藏输入
  - Succinct 简洁：生成可以快速验证的简短证明
  - Noninteractive 非交互式：不需要来回交互
  - ARgument of Knowledge 知识论证：证明你知道输入

zkSNARKs

- 函数输入：x1，x2，x3，x4
- OUT = f(x) = (x1 + x2) * x3 - x4

- zkSNARK：我知道一些秘密（x1，x2，x3，x4），以及这个函数的计算结果 OUT。利用"签名"可以证明存在这样的秘密，但不需要告诉你秘密是什么。

zkSNARKs 证明约束

- 函数输入：x1，x2，x3，x4
- y1 := x1 + x2
- y2 := y1 * x3
- OUT := y2 - x4

- SNARK 证明器输入：x1，x2，x3，x4，y1，y2，OUT
- SNARK 证明器输出：仅当满足以下约束时的可验证“签名”：
  - y1 == x1 + x2
  - y2 == y1 * x3
  - y2 == OUT + x4

zkSNARKs 证明约束

- 函数输入：02, 04, 08, 05
- 06 := 02 + 04
- 48 := 06 * 08
- 043 := 48 - 05

- SNARK 证明器输入：02, 04, 08, 05, 06, 48, 043
- SNARK 证明器输出：仅当满足以下约束时的可验证“签名”：
  - 06 == 02 + 04
  - 48 == 06 * 08
  - 48 == 043 + 05

zkSNARKs 证明约束

- 函数输入：x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 * x3
- 043 := y2 - x4

- SNARK 证明器输入：x1, x2, x3, x4, y1, y2, 043
- SNARK 证明器输出：仅当满足以下约束时的可验证"签名"：
    - y1 == x1 + x2
    - y2 == y1 * x3
    - y2 == 043 + x4

zkSNARKs 证明约束（只用 + 和 *）

- 函数输入： x1， x2， x3， x4
- y1 := x1 + x2
- y2 := y1 / x3
- OUT := y2 - x4

- SNARK 证明器输入： x1， x2， x3， x4， y1， y2， OUT
- SNARK 证明器输出：仅当满足以下约束时的可验证“签名”：
  - y1 == x1 + x2
  - y1 == y2 * x3
  - y2 == OUT + x4

ZKRepl demo #1

课堂练习

```
pragma circom 2.1.4;

template Main () {
    signal input x1;
    signal input x2;
    signal input x3;
    signal input x4;

    signal y1;
    signal y2;

    signal output out;

    // f(x) = (x1 + x2) / x3 - x4
    y1 <== x1 + x2;
    y2 <== y1 / x3;
    out <== y2 - x4;
}

component main = Main();

/* INPUT = {
    "x1": "4",
    "x2": "6",
    "x3": "2",
    "x4": "1"
} */
```

```
STDERR:
  error[T3001]: Non quadratic constraints are not allowed!
     ┌─ "main.circom":16:5
     │
  16 │     y2 <== y1 / x3;
     │     ^^^^^^^^^^^^^^ found here
     │
     = call trace:
       ->Main

  previous errors were found

STDOUT:
  Compiled in 1.83s

FAIL:
  ENOENT: no such file or directory, open 'main_js/main.wasm'

KEYS + SOLIDITY + HTML:
```

Groth16 | PLONK | Verify

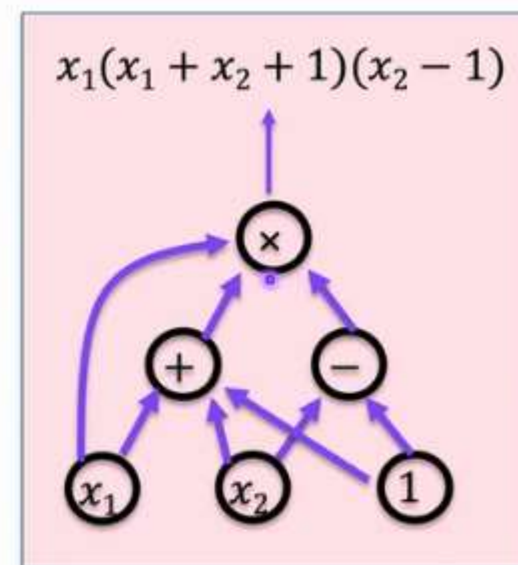
算术电路及其常见设计方法

算术电路

- 有限域 $\mathbb{F} = \{0, \ldots, p-1\}$ 基于某一素数$p > 2$
- 算术电路 $C: \mathbb{F}^n \rightarrow \mathbb{F}$
  - 是计算复杂性理论中的概念，与电子电路毫无关联
  - 有向无环图
  - 输入节点标记为$1, x_1, \ldots, x_n$
  - 内部节点标记为+,-,x
  - 每个内部节点也称为门(gate)

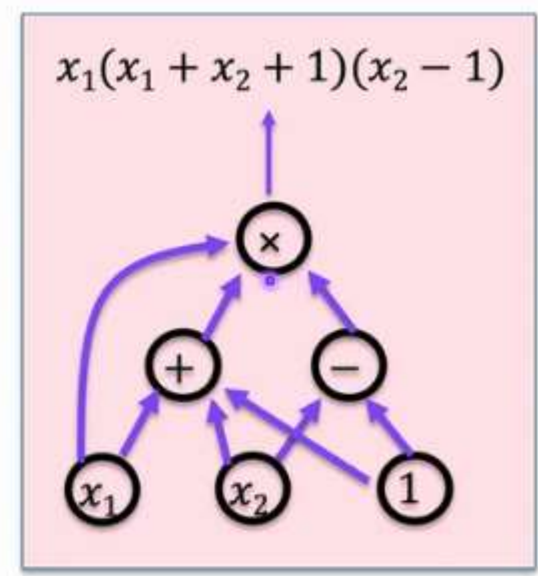
正是因为目前零知识领域正在使用算术电路这样的构造方式，与我们的传统程序逻辑不一样，因此过去那些成熟的程序逻辑，没办法轻易的移植到零知识证明器里

$|C_{sha256}| \approx$ 20K gates



算术电路

- 特性
  - 固定性：电路在证明过程不可动态增减
  - 丰富性：电路传递的是有限域的数字，比二进制具有丰富的表达能力
  - 约束性：电路既可用作计算，也用于约束流转信号的状态转换



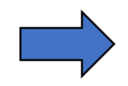
零知识证明电路常见设计方法

提示

由于算术电路的约束性，每个门电路的计算都会转换为约束，进而增加证明和验证的工作量，我们可以将复杂的计算过程变为预先计算的提示值，在电路中对提示值进行验证，从而降低证明和验证的工作量。

判零函数

```
let y = input > 0 ? 0 : 1;
```



提示值$inv = \begin{cases} 0, & input = 0 \\ 1/input, & otherwise \end{cases}$

(1) $output = -input \times inv + 1$

(2) $input \times output = 0$

示例

情况	输入 $input$	提示值 inv	输出 $output$	$input \times output$
非零输入	4	1/4	0	0
为零输入	0	0	1	0
非零时，恶意提示	4	1/5	1/5	4/5
为零时，恶意提示	0	1/5	1	0

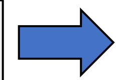
(1) 利用提示值inv计算输出$output$
(2) 验证输入输出符合约束

如果不用提示方法，需要用费马小定理$input^{p-1} = 1$证明，计算次数多

零知识证明电路常见设计方法

选择

```
let y = s ? (a + b) : (a * b);
```



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

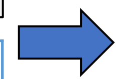
(2) $y = s \times (a + b) + (1 - s) \times (a \cdot b)$

(1) 由于算术电路的丰富性，需对s进行约束检查
(2) 利用一个二进制位s作为计算有效性的选择开关

二进制化

```
5 -> 101
```

输入 $input$	输出第1位out_1	输出第2位 out_2	输出第3位 out_3
5	1	0	1



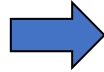
$$\begin{cases} out_1 \times (out_1 - 1) = 0 \\ out_2 \times (out_2 - 1) = 0 \\ out_3 \times (out_3 - 1) = 0 \\ out_1 \times 2^0 + out_2 \times 2^1 + out_3 \times 2^2 = input \end{cases}$$

由于算术电路的丰富性，输入均为有限域$\mathbb{F}$上的数字，将其转换为二进制表示，在很多方面（比如比较大小）都有很重要的作用。与传统思路不同地方在于，将数字转化为二进制的过程，实际上是利用提示技术对已经转化好的数字做约束验证的过程。

零知识证明电路常见设计方法

比较

```
let y = s1 > s2 ? 1 : 0;
```



(1) $y = s_1 + 2^n - s_2$

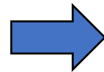
(2) y 二进制化取最高位

(1) 比较数字大小的朴素想法是将两个数字相减，将结果二进制化后，根据符号位进行判断。但由于数字均为群元素，没有负数，因此我们需要将数字加上最大值。
(2) 二进制并取最高位，例如输入分别为3和4，$n = 3$，$y = 3 + 2^3 - 4 = 7$，转为二进制：$(7)_{10} = (0111)_2$，最高位为0

循环

由于算术电路的固定性，电路只能设计为支持最大输入数量，根据实际输入数量的不同，利用选择器技术将部分计算功能关闭，以达到不同数量的循环功能。

```
for (let i = 0; i < N; i++) {
    y += 1;
}
```



(1) $s = \begin{cases} 1, i < n \\ 0, else \end{cases}$

(2) $y = s \times (y + 1) + (1 - s) \times (y)$

(1) 利用比较方法，为临时变量s赋值
(2) 利用选择方法，分别启用循环中的计算，或恒等原值，即未启用

零知识证明电路常见设计方法

交换

```
if (s) {
    output1 = input2;
    output2 = input1;
} else {
    output1 = input1;
    output2 = input2;
}
```

$$\begin{cases} output_1 = (input_2 - input_1) \times s + input_1 \\ output_2 = (input_1 - input_2) \times s + input_2 \end{cases}$$

通过一个交换标识s来标记是否要交换两个输入

逻辑

```
let y = a & b;
let y = !a;
let y = a | b;
let y = a ^ b;
```

(1) $y = a \cdot b$

(2) $y = 1 - a$

(3) $y = 1 - (1 - a) \times (1 - b)$

(4) $y = (a + b) - 2 \cdot a \cdot b$

- 逻辑运算可以通过简单的数学运算获得。
- 另外还需要使用类似于$a \cdot (1 - a) = 0$的方式检查二进制约束

零知识证明电路常见设计方法

排序

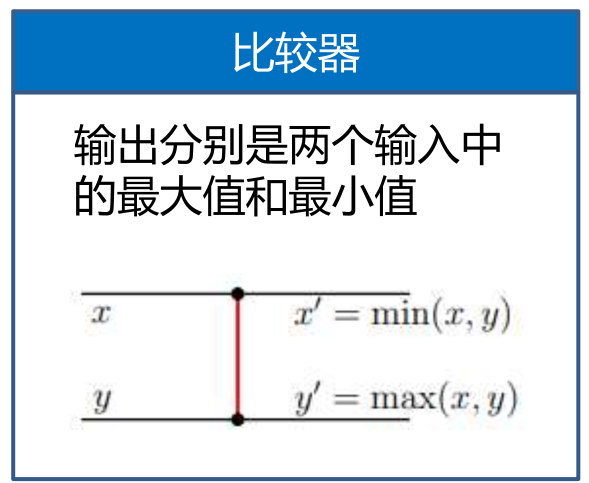
```
for (let i = 0; i <= array.length - 1; i++) {
    for (let j = 0; j < (array.length - i - 1); j++) {
        if (array[j] > array[j + 1]) {
            swap(array[j], array[j + 1])
        }
    }
}
```

冒泡排序

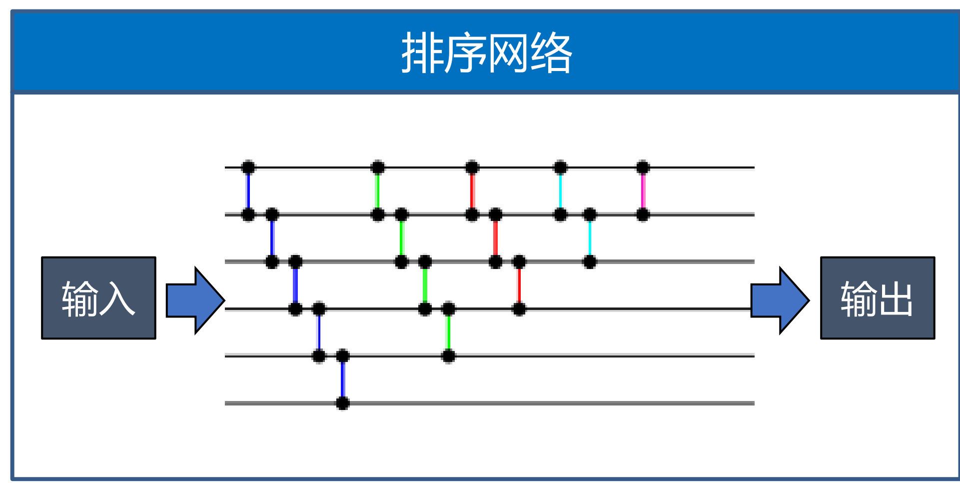
(1) 在算术电路上做排序，可以借用排序网络的概念。
(2) 利用多个比较器形成排序网络进行排序。

比较器

输出分别是两个输入中的最大值和最小值

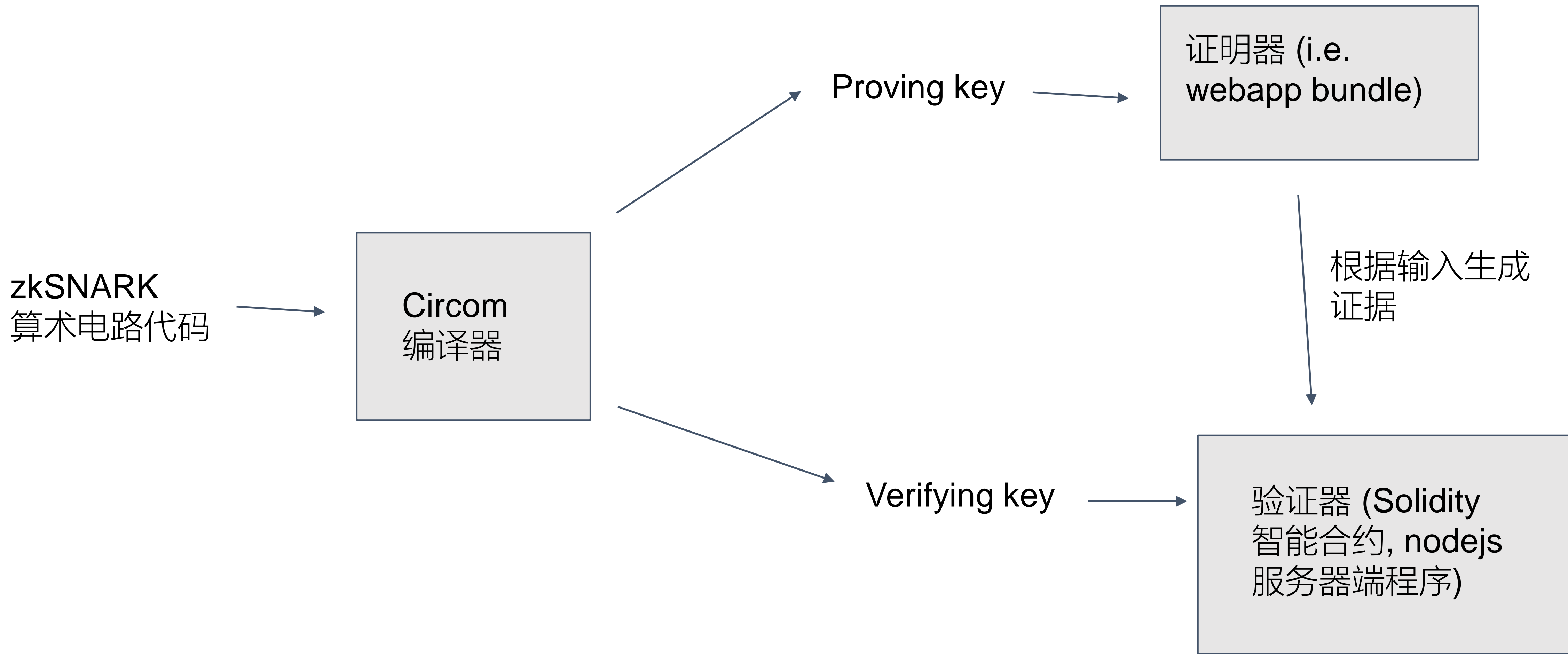


排序网络



ZKRepl demo #2: Num2Bits

ZK Dapp



各文件的功能

- Circuit .circom
- Input .json
- PoT .ptau
- Circuit .wasm
- Proving Key .zkey
- Verification Key .vkey
- Verifier .sol

snarkjs

circom-starter

https://github.com/0xPARC/circom-starter

未来：开放的探索领域

- 构建更好的电路
- 构建更好的协议
- 发掘新的用例
- 建设基础设施，加强现有电路的安全等等…