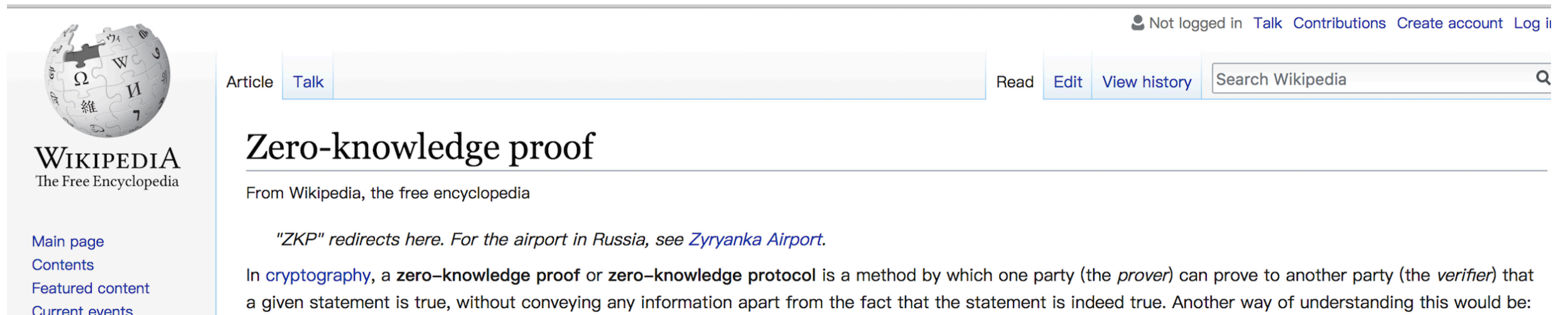


# 零知识证明算法之zkSNARKs

李康

中科院计算技术研究所

# 零知识证明是什么？



零知识证明是指一方(证明者)向另一方(验证者)证明一个陈述是正确的，而无需透露除该陈述是正确外的任何信息

# 零知识证明分类

- 交互式：证明者与验证者需进行多轮通信
- 非交互式：证明者按照协议向验证者发送一次消息，验证者根据协议即可验证。以 zkSNARKs 算法最为成熟实用，zcash 采用的便是 zkSNARKs 算法

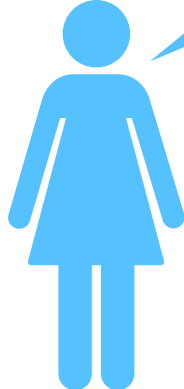
# zkSNARKs 算法

zk-SNARK 是 zero-knowledge succinct non-interactive arguments of knowledge 的简称，其中：


- **Succinct (简洁性)**：与实际计算的长度相比，生成的零知识证据消息很小
- **Non-interactive (非交互性)**：几乎没有任何交互。对于 zk-SNARK 算法来说，通常有一个构建阶段，构建阶段完成之后，证明者 (prover) 只需向验证者 (verifier) 发送一个消息即可。而且，SNARK 通常还有一个被称作是 "公开验证者" 的特性，意味着任何人无需任何交互即可验证零知识证据，这对区块链是至关重要的
- **Arguments (争议性)**：验证者只能抵抗计算能力有限的证明者的攻击。具有足够计算能力的证明者可以创建伪造的零知识证据以欺骗验证者 (注意刚提到的足够的计算能力，它足可以打破公钥加密，所以现阶段可不必担心)。这也通常被称为 "计算完好性 (computational soundness)"，而不是 "完美完好性 (perfect soundness)"
- **of Knowledge**：对于一个证明者来说，在不知晓特定证明 (witness) 的前提下，构建一个有效的零知识证据是不可能的

**zkSNARKs 能做什么？**

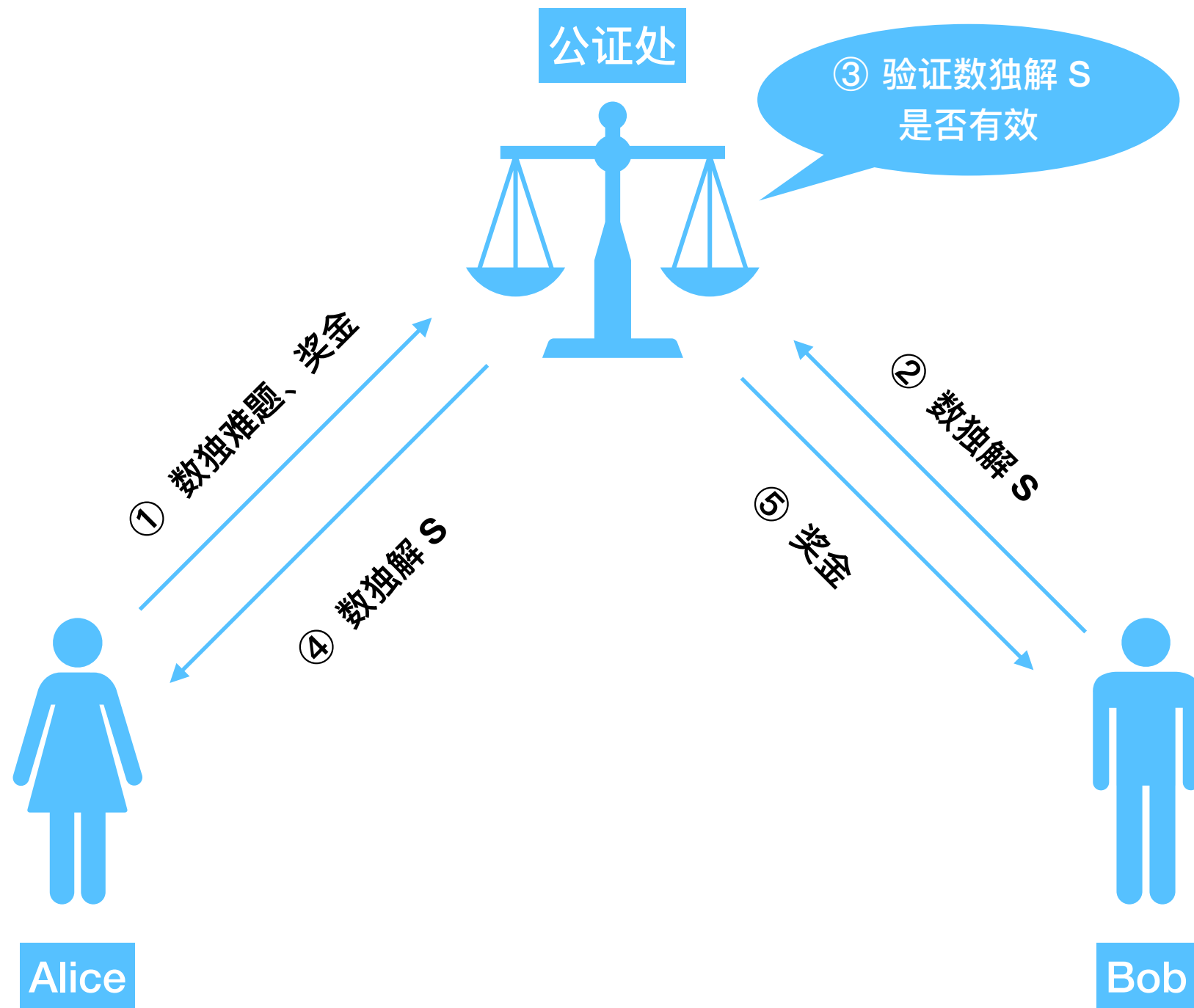
## 互不信任



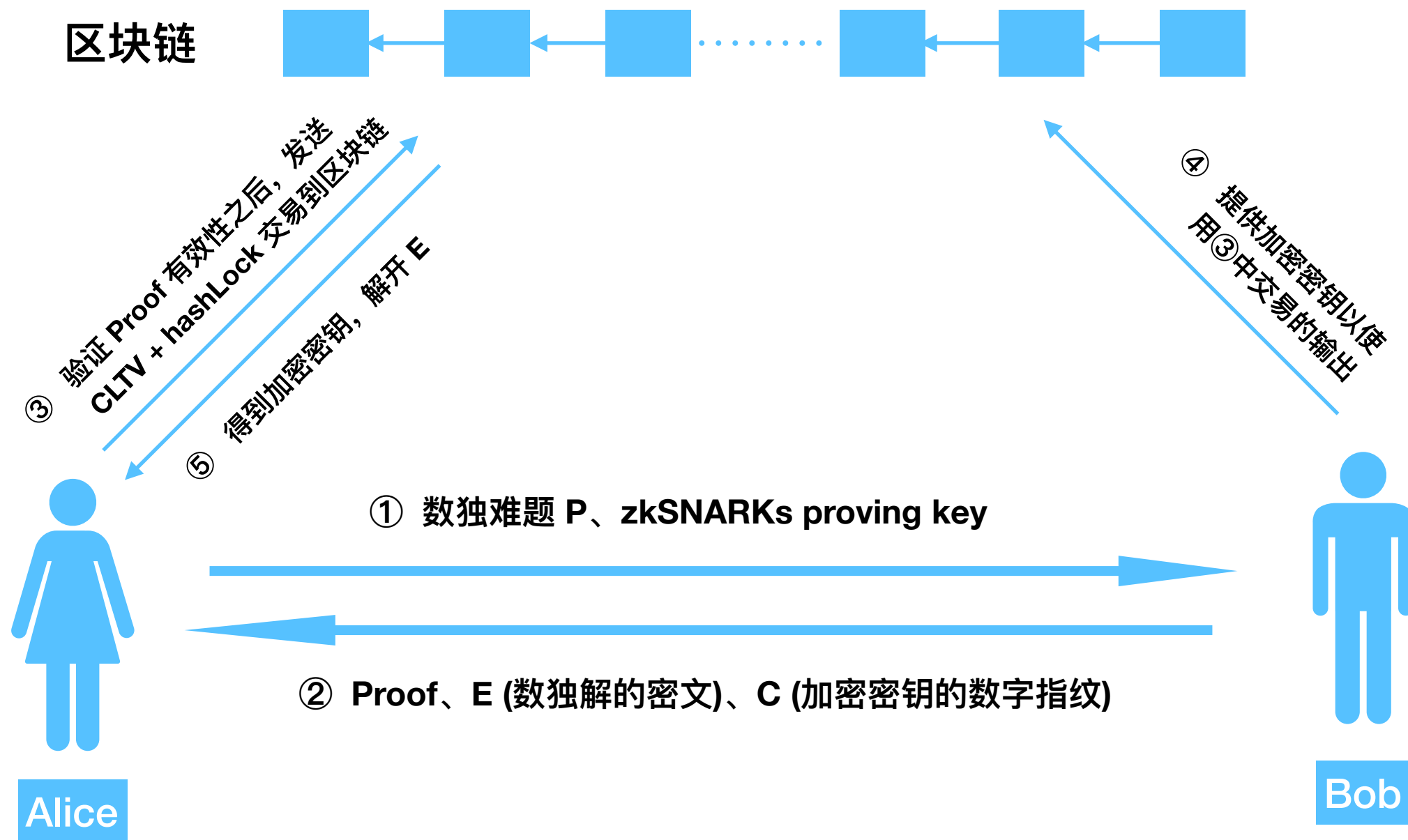
Alice（数独爱好者）：发布了一个数独难题，谁能提供一个有效的解，谁就获得一笔奖励



Bob（数独爱好者）：知道 Alice 数独难题的解 S，想获得奖励



# 数独 + zkSNARKs



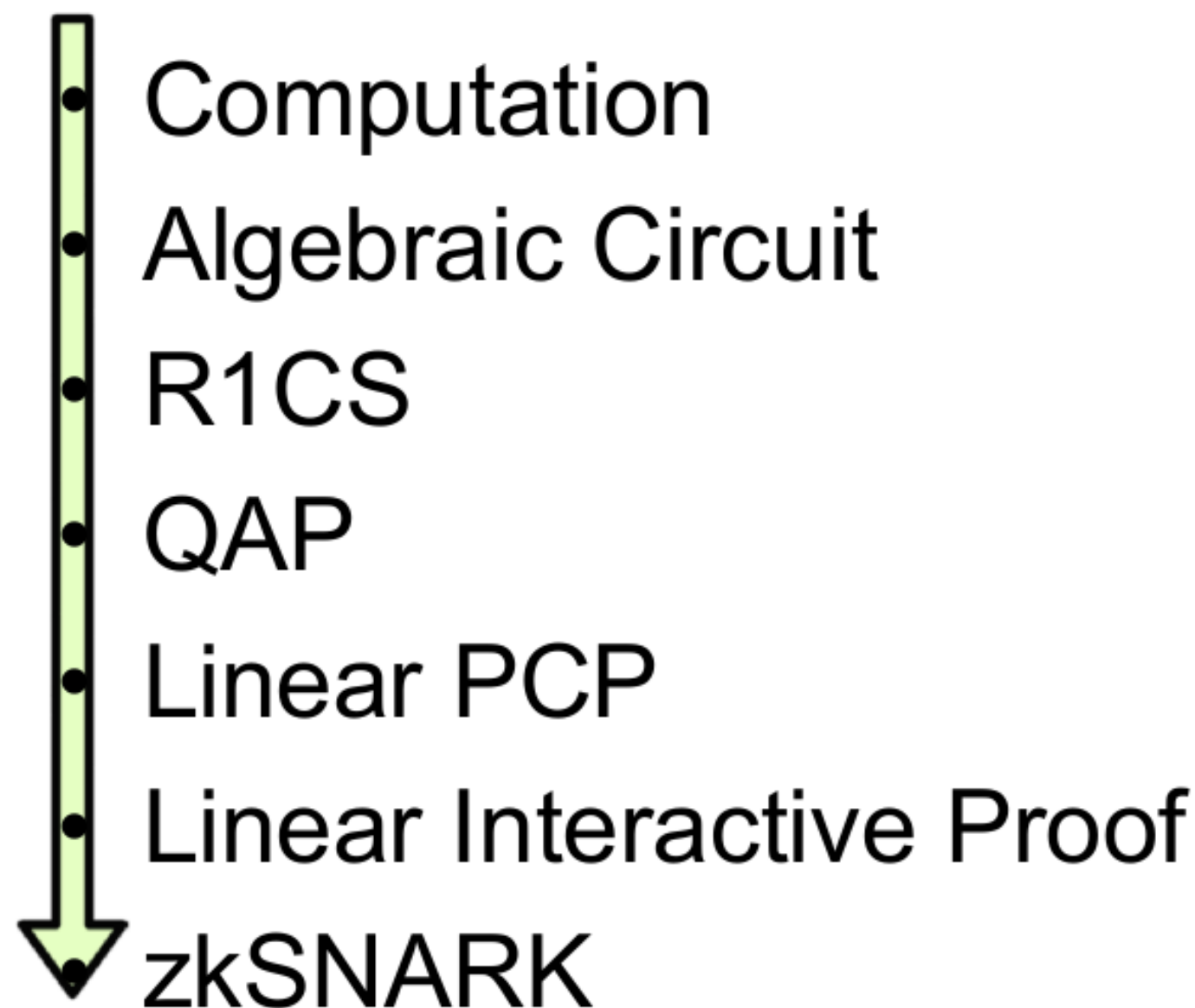


# 数独 + zkSNARKs

- 验证数独的解是否有效是一个 NP 问题，即能在多项式时间内验证
- Bob: 设数独难题为  $P$ ，加密后的解为  $E$ ，加密密钥数字指纹  $C$ ，数独解为  $S$ ，加密密钥为  $K$ ，其中  $C := \text{SHA256}(K)$ ， $E := \text{Encrypt}(K, S)$ ， $\text{Proof} := \text{Prove}(\text{pk}, P, E, C, K, S)$ ，将  $\langle \text{Proof}, P, E, C \rangle$  作为明文发送给 Alice
- Alice:  $\text{result} := \text{Verify}(\text{vk}, P, E, C, \text{Proof})$ ，若  $\text{result}$  为  $\text{true}$  则代表：
  - $S$  必须是  $P$  的一个有效解
  - $E$  必须是  $S$  通过  $K$  加密得到的
  - $C$  必须是  $\text{SHA256}(K)$

**zkSNARKs 是如何做到的？**

# zkSNARKs 原理



## 1. 将等式转换成最简单的表达式 (gate): $y = x$ or $y = x \text{ (op) } z$

$$x^3 + x + 5 = 35 \quad \left\{ \begin{array}{l} \text{sym\_1} = x * x \\ y = \text{sym\_1} * x \\ \text{sym\_2} = y + x \\ \sim\text{out} = \text{sym\_2} + 5 \end{array} \right.$$

## 2. 将上述所得 gates 转换为 R1CS

R1CS 是一系列三元向量组  $\langle a, b, c \rangle$ , 对 R1CS 的解  $s$  满足,  $s.a * s.b - s.c = 0$

R1CS 解  $s$  的第一个元素永远是常数 1, 在上述等式中, 我们可以将  $s$  看做是  $\langle 1, x, \sim\text{out}, \text{sym\_1}, y, \text{sym\_2} \rangle$

A	
1	5
3	0
35	0
9	0
27	0
30	1

B	
1	1
3	0
35	0
9	0
27	0
30	0

C	
1	0
3	0
35	1
9	0
27	0
30	0

$35 \quad * \quad 1 \quad - \quad 35 \quad = \quad 0$

$$\sim\text{out} = \text{sym\_2} + 5$$

**s := <1, x, ~out, sym\_1, y, sym\_2>**

**第一个 gate sym\_1 = x \* x:**

a = [0, 1, 0, 0, 0, 0]

b = [0, 1, 0, 0, 0, 0]

c = [0, 0, 0, 1, 0, 0]

**第二个 gate y = sym\_1 \* x:**

a = [0, 0, 0, 1, 0, 0]

b = [0, 1, 0, 0, 0, 0]

c = [0, 0, 0, 0, 1, 0]

**第三个 gate sym\_2 = y + x:**

a = [0, 1, 0, 0, 1, 0]

b = [1, 0, 0, 0, 0, 0]

c = [0, 0, 0, 0, 0, 1]

**第四个 gate ~out = sym\_2 + 5:**

a = [5, 0, 0, 0, 0, 1]

b = [1, 0, 0, 0, 0, 0]

c = [0, 0, 1, 0, 0, 0]

**A**

[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**

[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**

[0, 0, 0, 1, 0, 0]

[0, 0, 0, 0, 1, 0]

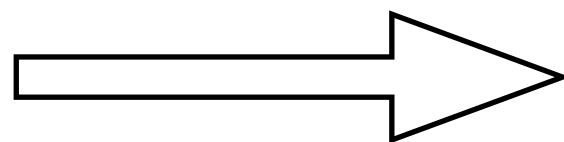
[0, 0, 0, 0, 0, 1]

[0, 0, 1, 0, 0, 0]

## 2. 将上述所得 R1CS 转换为 QAP

**A**

[0, 1, 0, 0, 0, 0]  
[0, 0, 0, 1, 0, 0]  
[0, 1, 0, 0, 1, 0]  
[5, 0, 0, 0, 0, 1]



**A 多项式**

[-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]

(1,0)、(2,0)、(3,0), (4,5)   $y = 0.833 * x^3 - 5 * x^2 + 9.166 * x - 5.0$

**B 多项式**

[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 多项式**

[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]

## 由 R1CS 到 QAP 的转换是为了什么？

相较于 R1CS 需要单独检查每一个等式约束，QAP 可以同时检查所有的约束

A			B			C	
1	$A_1(x)$		1	$B_1(x)$		1	$C_1(x)$
3	$A_2(x)$		3	$B_2(x)$		3	$C_2(x)$
35	$A_3(x)$		35	$B_3(x)$		35	$C_3(x)$
9	$A_4(x)$		9	$B_4(x)$		9	$C_4(x)$
27	$A_5(x)$		27	$B_5(x)$		27	$C_5(x)$
30	$A_6(x)$		30	$B_6(x)$		30	$C_6(x)$

$A(x) \quad * \quad B(x) \quad - \quad C(x) \quad = \quad H * Z(x)$

可以看出  $A(x) * B(x) - C(x)$  在  $x=1/2/3/4$  处均为 0，并且其必定是  $Z(x) := (x-1) * (x-2) * (x-3) * (x-4)$  的整多项式数倍

然而，在现实世界（密码学）中，我们都不是用来实数来做加减乘除的运算，而是采用有限域的元素来完成运算，有限域本身是自统一的，我们所熟知的各种实数定理在有限域里同样成立

# 有限域

- 有限域的个数只能是质数或者质数的幂次方
- 满足数学算术定理：交换律、结合律
- $a + b = (a + b) \bmod p$ 、 $a - b = (a - b) \bmod p$ 、 $a * b = (a * b) \bmod p$

$$a / b = (a * b^{(p-2)}) \bmod p, a^{(p-1)} = 1$$

$$2 + 3 = 5 \bmod 7 = 5$$

$$2 - 5 = -3 \bmod 7 = 4$$

$$6 * 3 = 18 \bmod 7 = 4$$

$$3 / 2 = 3 * 2^5 = 96 \bmod 7 = 5$$

$$5 * 2 = 10 \bmod 7 = 3$$

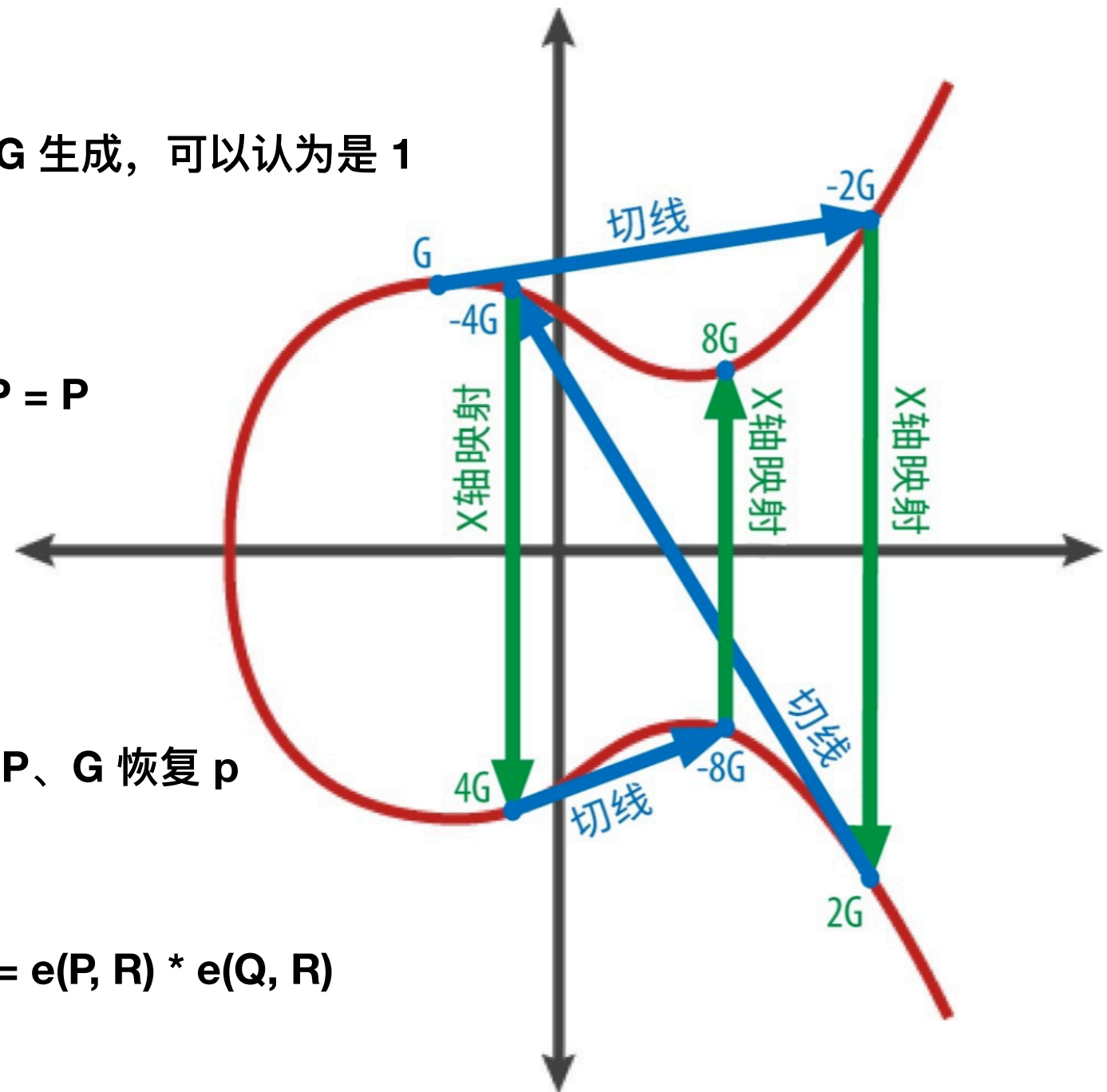
$$5^6 = 15625 \bmod 7 = 1$$



# 椭圆曲线

- 生成点  $G$ : 椭圆曲线上所有的点均可以由  $G$  生成, 可以认为是 1
- 椭圆曲线定义域及值域基于有限域  $F_p$
- 椭圆曲线的阶(order)  $n$ :  $P * n = O$ ,  $O + P = P$

- 椭圆曲线离散对数难题:  $P = G * p$ , 根据  $P$ 、 $G$  恢复  $p$
- 椭圆曲线配对(pairing):  
$$e(P, Q+R) = e(P, Q) * e(P, R), \quad e(P+Q, R) = e(P, R) * e(Q, R)$$



那么如何在不泄露  $s$  部分内容的情况下证明  $A(x) * B(x) - C(x) = H * Z(x)$  等式成立呢？

首先我们来看这样一个等式：  $Q = P * k$ ,  $S = R * k$ , 则  $e(Q, R) = e(P, S)$

证明:  $e(Q, R) = e(P * k, R) = e(P, R)^k = e(P, R * k) = e(P, S)$

再来看一些更有趣的例子：假设 Alice 提供给 Bob 10 个点  $(P_1, Q_1), (P_2, Q_2) \dots (P_{10}, Q_{10})$ , 其中  $Q_i = P_i * k$ ,  $k$  只有 Alice 知道；此时 Bob 提供给 Alice  $(R, S)$  两个点，并且  $e(Q_i, R) = e(P_i, S)$ ，那么此时 Alice 获取到了什么信息呢？

由于从  $P_i, Q_i$  获得  $k$  是离散对数难题，Alice 可以推断出，Bob 不可能得到  $k$ ，而是通过线性组合的方式获得了  $(R, S)$ ，即  $R = i_1 * P_1 + i_2 * P_2 + \dots + i_{10} * P_{10}$ ,  $S = i_1 * Q_1 + i_2 * Q_2 + \dots + i_{10} * Q_{10}$ ，此时  $e(Q_i, R) = e(P_i, S)$  成立

那 QAP 与 椭圆曲线配对到底有什么关联呢？

可以将  $s$  视为  $(i_1, i_2, \dots, i_{10})$ ，取  $A_i(x)$  在  $x = r$  处的值，通过椭圆曲线生成点  $G$ ，可得点  $P = A_i(x) * G$ ，因此可以使用椭圆曲线配对来表示上述 QAP

如何使用椭圆曲线配对来证明  $A(x) * B(x) - C(x) = H * Z(x)$  等式成立?

① 首先添加如下点到公共可信集合中

- $G * A_1(t), G * A_1(t) * k_a$
- $G * A_2(t), G * A_2(t) * k_a$
- ...
- $G * B_1(t), G * B_1(t) * k_b$
- $G * B_2(t), G * B_2(t) * k_b$
- ...
- $G * C_1(t), G * C_1(t) * k_c$
- $G * C_2(t), G * C_2(t) * k_c$
- ...

其中  $t$ 、 $k_a$ 、 $k_b$ 、 $k_c$  称之为“有毒废料”，一旦生成所需点之后，要立刻清除，这也是可信构建的来源

② 证明 prover 提供了  $A_i(x)$ 、 $B_i(x)$ 、 $C_i(x)$  的线性组合

- 证明者 (prover) 需提供:

$$\pi_a = G * A(t), \pi'_a = G * A(t) * k_a$$

$$\pi_b = G * B(t), \pi'_b = G * B(t) * k_b$$

$$\pi_c = G * C(t), \pi'_c = G * C(t) * k_c$$

- 验证者 (verifier) 验证:

$$e(\pi'_a, G * A_1(t)) \stackrel{?}{=} e(\pi_a, G * A_1 * k_a)$$

$$e(\pi'_b, G * B_1(t)) \stackrel{?}{=} e(\pi_b, G * B_1 * k_b)$$

$$e(\pi'_c, G * C_1(t)) \stackrel{?}{=} e(\pi_c, G * C_1 * k_c)$$

由于  $t$ 、 $k_a$ 、 $k_b$ 、 $k_c$  已经被删除，所以验证者验证通过后，获得的信息是，证明者提供了  $A_1(t)$ 、 $A_2(t)$ 、... 的一个线性组合 ( $B_1(t)$ 、 $B_2(t)$ 、...、 $C_1(t)$ 、 $C_2(t)$ 、...同理)。

### ③ 证明上述三个线性组合具有相同的系数

- 添加如下点到公共可信集合中

$$G * (A_i(t) + B_i(t) + C_i(t)) * b \text{ (有毒废料)}$$

- 证明者需提供

$$\pi'' = G * (A(t) + B(t) + C(t)) * b$$

- 验证者需验证

$$e(\pi'', G * (A_i(t) + B_i(t) + C_i(t))) \stackrel{?}{=} e(\pi_a + \pi_b + \pi_c, G * (A_i(t) + B_i(t) + C_i(t)) * b)$$

验证通过后，验证者可以获得的信息是：证明者提供的  $\pi_a$ 、 $\pi_b$ 、 $\pi_c$  确实由相同系数的多项式组合而成

### ④ 证明 $A * B - C = H * Z$

- 添加如下点到公共可信集合中

$$G * Z(t)$$

$$G, G * t, G * t^2, \dots$$

- 证明者需提供

$$\pi_h = G * H(t)$$

- 验证者需验证

$$e(\pi_a, \pi_b) / e(\pi_c, G) \stackrel{?}{=} e(\pi_h, G * Z(t))$$

$$\text{证明: } e(\pi_a, \pi_b) = e(A(t) * G, B(t) * G) = e(G, G)^{A(t)*B(t)}$$

$$e(\pi_c, G) = e(C(t) * G, G) = e(G, G)^{C(t)}$$

$$e(\pi_h, G * Z(t)) = e(G * H(t), G * Z(t)) = e(G, G)^{H(t)*Z(t)}$$

$$\text{因此 } e(\pi_a, \pi_b) / e(\pi_c, G) = e(G, G)^{A(t) * B(t) - C(t)}$$

## B The PGHR zk-SNARK protocol

For the purposes of completeness and to fix notation, in Figure 10 below we recall the zk-SNARK protocol of Parno et al. [PGHR13]. The zk-SNARK can be used to prove/verify satisfiability of  $\mathbb{F}_r$ -arithmetic circuits, where  $r$  is the order of the two cyclic groups  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , forming the domain of the pairing  $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ .

We refer the reader to [PGHR13] for further details regarding the intuition for the protocol, as well as the cryptographic assumptions on which its proof of security relies. (Briefly, security relies on the q-power Diffie–Hellman, q-power knowledge-of-exponent, and q-strong Diffie–Hellman assumptions [Gro10b, BB04, Gen04] for  $q$  that depends polynomially on the arithmetic circuit’s size.)

**Public parameters.** A prime  $r$ , two cyclic groups  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of order  $r$  with generators  $\mathcal{P}_1$  and  $\mathcal{P}_2$  respectively, and a pairing  $e: \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$  (where  $\mathbb{G}_T$  is also cyclic of order  $r$ ).

### (a) Key generator $G$

- INPUTS: circuit  $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$
  - OUTPUTS: proving key  $\text{pk}$  and verification key  $\text{vk}$
1. Compute  $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$ ; extend  $\vec{A}, \vec{B}, \vec{C}$  via
 
$$A_{m+1} = B_{m+2} = C_{m+3} = Z,$$

$$A_{m+2} = A_{m+3} = B_{m+1} = B_{m+3} = C_{m+1} = C_{m+2} = 0.$$
  2. Randomly sample  $\tau, \rho_A, \rho_B, \alpha_A, \alpha_B, \alpha_C, \beta, \gamma \in \mathbb{F}_r$ .
  3. Set  $\text{pk} := (C, \text{pk}_A, \text{pk}'_A, \text{pk}_B, \text{pk}'_B, \text{pk}_C, \text{pk}'_C, \text{pk}_K, \text{pk}_H)$  where for  $i = 0, 1, \dots, m+3$ :
 
$$\text{pk}_{A,i} := A_i(\tau)\rho_A\mathcal{P}_1, \quad \text{pk}'_{A,i} := A_i(\tau)\alpha_A\rho_A\mathcal{P}_1,$$

$$\text{pk}_{B,i} := B_i(\tau)\rho_B\mathcal{P}_2, \quad \text{pk}'_{B,i} := B_i(\tau)\alpha_B\rho_B\mathcal{P}_1,$$

$$\text{pk}_{C,i} := C_i(\tau)\rho_A\rho_B\mathcal{P}_1, \quad \text{pk}'_{C,i} := C_i(\tau)\alpha_C\rho_A\rho_B\mathcal{P}_1,$$

$$\text{pk}_{K,i} := \beta(A_i(\tau)\rho_A + B_i(\tau)\rho_B + C_i(\tau)\rho_A\rho_B)\mathcal{P}_1,$$
 and for  $i = 0, 1, \dots, d$ ,  $\text{pk}_{H,i} := \tau^i\mathcal{P}_1$ .
  4. Set  $\text{vk} := (\text{vk}_A, \text{vk}_B, \text{vk}_C, \text{vk}_\gamma, \text{vk}_{\beta\gamma}^1, \text{vk}_{\beta\gamma}^2, \text{vk}_Z, \text{vk}_{IC})$  where
 
$$\text{vk}_A := \alpha_A\mathcal{P}_2, \text{vk}_B := \alpha_B\mathcal{P}_1, \text{vk}_C := \alpha_C\mathcal{P}_2$$

$$\text{vk}_\gamma := \gamma\mathcal{P}_2, \quad \text{vk}_{\beta\gamma}^1 := \gamma\beta\mathcal{P}_1, \quad \text{vk}_{\beta\gamma}^2 := \gamma\beta\mathcal{P}_2,$$

$$\text{vk}_Z := Z(\tau)\rho_A\rho_B\mathcal{P}_2, \quad (\text{vk}_{IC,i})_{i=0}^n := (A_i(\tau)\rho_A\mathcal{P}_1)_{i=0}^n.$$
  5. Output  $(\text{pk}, \text{vk})$ .

**Key sizes.** When invoked on a circuit  $C: \mathbb{F}_r^n \times \mathbb{F}_r^h \rightarrow \mathbb{F}_r^l$  with  $a$  wires and  $b$  (bilinear) gates, the key generator outputs:

- $\text{pk}$  with  $(6a + b + n + l + 26)$   $\mathbb{G}_1$ -elements and  $(a + 4)$   $\mathbb{G}_2$ -elements;
- $\text{vk}$  with  $(n + 3)$   $\mathbb{G}_1$ -elements and 5  $\mathbb{G}_2$ -elements.

**Proof size.** The proof always has 7  $\mathbb{G}_1$ -elements and 1  $\mathbb{G}_2$ -element.

### (b) Prover $P$

- INPUTS: proving key  $\text{pk}$ , input  $\vec{x} \in \mathbb{F}_r^n$ , and witness  $\vec{a} \in \mathbb{F}_r^h$
  - OUTPUTS: proof  $\pi$
1. Compute  $(\vec{A}, \vec{B}, \vec{C}, Z) := \text{QAPinst}(C)$ .
  2. Compute  $\vec{s} := \text{QAPwit}(C, \vec{x}, \vec{a}) \in \mathbb{F}_r^m$ .
  3. Randomly sample  $\delta_1, \delta_2, \delta_3 \in \mathbb{F}_r$ .
  4. Compute  $\vec{h} = (h_0, h_1, \dots, h_d) \in \mathbb{F}_r^{d+1}$ , which are the coefficients of  $H(z) := \frac{A(z)B(z) - C(z)}{Z(z)}$  where  $A, B, C \in \mathbb{F}_r[z]$  are as follows:
 
$$A(z) := A_0(z) + \sum_{i=1}^m s_i A_i(z) + \delta_1 Z(z),$$

$$B(z) := B_0(z) + \sum_{i=1}^m s_i B_i(z) + \delta_2 Z(z),$$

$$C(z) := C_0(z) + \sum_{i=1}^m s_i C_i(z) + \delta_3 Z(z).$$
  5. Set  $\tilde{\text{pk}}_A :=$  “same as  $\text{pk}_A$ , but with  $\text{pk}_{A,i} = 0$  for  $i = 0, 1, \dots, n$ ”.
  6. Letting  $\vec{c} := (1 \circ \vec{s} \circ \delta_1 \circ \delta_2 \circ \delta_3) \in \mathbb{F}_r^{4+m}$ , compute
 
$$\pi_A := \langle \vec{c}, \tilde{\text{pk}}_A \rangle, \pi'_A := \langle \vec{c}, \text{pk}'_A \rangle, \pi_B := \langle \vec{c}, \text{pk}_B \rangle, \pi'_B := \langle \vec{c}, \text{pk}'_B \rangle,$$

$$\pi_C := \langle \vec{c}, \text{pk}_C \rangle, \pi'_C := \langle \vec{c}, \text{pk}'_C \rangle, \pi_K := \langle \vec{c}, \text{pk}_K \rangle, \pi_H := \langle \vec{h}, \text{pk}_H \rangle.$$
  7. Output  $\pi := (\pi_A, \pi'_A, \pi_B, \pi'_B, \pi_C, \pi'_C, \pi_K, \pi_H)$ .

### (c) Verifier $V$

- INPUTS: verification key  $\text{vk}$ , input  $\vec{x} \in \mathbb{F}_r^n$ , and proof  $\pi$
  - OUTPUTS: decision bit
1. Compute  $\text{vk}_{\vec{x}} := \text{vk}_{IC,0} + \sum_{i=1}^n x_i \text{vk}_{IC,i} \in \mathbb{G}_1$ .
  2. Check validity of knowledge commitments for  $A, B, C$ :
 
$$e(\pi_A, \text{vk}_A) = e(\pi'_A, \mathcal{P}_2), e(\text{vk}_B, \pi_B) = e(\pi'_B, \mathcal{P}_2),$$

$$e(\pi_C, \text{vk}_C) = e(\pi'_C, \mathcal{P}_2).$$
  3. Check same coefficients were used:
 
$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_{\vec{x}} + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^2) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$
  4. Check QAP divisibility:
 
$$e(\text{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \text{vk}_Z) \cdot e(\pi_C, \mathcal{P}_2).$$
  5. Accept if and only if all the above checks succeeded.

Figure 10: The zk-SNARK protocol of Parno et al. [PGHR13]. (More precisely, the protocol above differs from that in [PGHR13] in two ways. First, it does not assume that  $\mathbb{G}_1 = \mathbb{G}_2$ . Second, it obtains a verification key whose size grows as  $n + o(n)$ , rather than  $3n + o(n)$ , by leveraging the non-degeneracy property in Lemma 2.4.)

# zkSNARK-toy

- 项目地址: <https://github.com/lightning-li/zkSNARK-toy>
- 使用 libsnark 库构建
- 证明者知道 merkle 树根哈希值  $rt$ , 叶子节点  $leaf$ , 一个有效的 merkle 分支 (从  $leaf$  到  $rt$ ) 以及  $prev\_leaf$  ( $sha256(prev\_leaf) = leaf$ )
- 证明者将  $\langle proof, prev\_leaf, rt \rangle$  发给验证者, 验证者可验证证明者是否知晓  $leaf$  的哈希前数据, 以及是否拥有一个有效的分支, 使得计算出根哈希值  $rt$

# 参考文献

1. <https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>
2. <https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>
3. <https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6>
4. <https://github.com/zcash-hackworks/pay-to-sudoku/>
5. Parno B, Howell J, Gentry C, et al. Pinocchio: Nearly practical verifiable computation[C]//Security and Privacy (SP), 2013 IEEE Symposium on. IEEE, 2013: 238-252.
6. <https://github.com/lightning-li/zkSNARK-toy>

个人博客网站: <https://www.iethpay.com>

个人 github: <https://github.com/lightning-li/>

slides: <https://blockchain.iethpay.com/zero-knowledge-zkSNARKs.html>