zkSNARK 介绍

ASResearch

目录

1 ZkSNARK 相关介绍

1.1 总览

本文大致介绍 zkSNARK 的相关原理。鉴于已有大量相关的成熟资料，本文以科普形式传达个人理解为主，后续想到什么持续更新。

定义身份:

- $\mathcal{P}$ 证明者（Prover）, 他需要提供一段数据来证明自己知道验证者想要的信息。
- $\mathcal{V}$ 验证者 (Verifier)，他验证 $\mathcal{P}$ 提供的证明的正确性。

- 全称:Z(zero)k(knowledge)S(succinct)N(non-interactive)AR(argument)K(knowledge)
- 适用场景：较为通用的零知识证明/可信计算解决方案。
- 历史/发展过程：略。

在以下的函数描述中，当自变量为 x 的时候表示一个函数，如 $f(x)$。当自变量为其他字符的时候表示函数在该点的取值，如 $f(s)$ 表示函数 $f(x)$ 在 s 的取值。

$[n]$ 表示集合 $\{1, 2, ..., n\}$

1.2 参考资料

这里先介绍参考资料，其内容包含比本文更详尽的介绍。本文大部分内容源于下述材料的结合。

1. ZkSNARK 原始论文:“Pinocchio: Nearly Practical Verifiable Computation”Bryan Parno Jon Howell, Craig Gentry Mariana Raykova
2. QAP 原始论文:“Quadratic Span Programs and Succinct NIZKs without PCPs” Rosario Gennaro, Craig Gentry, Bryan Parno, Mariana Raykova
3. 最详尽的 ZkSNARK 解读报告:“Why and How zk-SNARK Works: Definitive Explanation”Maksym Petkus

4. V 神对 ZkSNARK 的介绍三部曲（第三部，含前两部链接，分别介绍 pairing 和 QAP）：
   https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6L
5. 主要从 P 与 NP 角度介绍 QSP，“zkSNARKs in a nutshell”
   https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/
6. ZCash 团队对 zkSNARK 的介绍，分 7 个小章节。
   https://z.cash/technology/zksnarks/
7. 对零知识证明的举例描述，介绍零知识证明三大性质（completeness, soundness, zero knowledgeness）
   https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/

1.3 证明知道一个多项式

几乎所有的 zkSNARK 资料都会从下面这个例子开始：$\mathcal{V}$ 心里有一个多项式，记作 $f(x)$。现在 $\mathcal{P}$ 需要向 $\mathcal{V}$ 证明他也“知道”这个多项式。

当然，最简单的方法就是 $\mathcal{P}$ 把这个多项式展示给 $\mathcal{V}$，但是这样首先交互的数据的长度过长，同时也不利于后续的扩展。（我们后面要求 $\mathcal{P}$ 证明他知道的这个多项式满足某些性质，同时会引入零知识的要求）

本章节所采用的方法，也是需要强调的核心思想描述如下：

- $\mathcal{V}$ 在大范围内随机生成一个数 s 并将其发送给 $\mathcal{P}$
- 随后 $\mathcal{P}$ 需要给出他所知道的多项式在 s 这点的值。
- $\mathcal{V}$ 收到 $\mathcal{P}$ 给出的这个值后，比较与 $f(s)$ 是否相同，若相同则认为 $\mathcal{P}$ 的证明合法，即他确实知道这个多项式。

这个协议的核心思想为，将证明两个多项式相同转化为证明两个多项式在某个点的取值相同。这看似不严谨但是在原理上是合法的：假设 $\mathcal{P}$ 知道的是另外一个多项式 $p(x) \neq f(x)$，且假设 $f(x)$ 和 $p(x)$ 的最高次数为 d 次，则根据代数理论可知，$f(x)$ 与 $p(x)$ 最多有 d 个交点，即它们最多在 d 个点的取值相同。那么，当 $\mathcal{V}$ 随机取一个 s，（其取值范围通常很大，比如为 $0 - 2^{256}$），s 恰好碰上这 d 个取值相同的点的概率为 $d/2^{256}$。因为通常 d 不是很大，$d/2^{256}$ 是一个可忽略的数（negligible），即 $\mathcal{P}$ 能给出的 $p(s) = f(s)$ 的概率可以忽略。这意味着，一旦 $\mathcal{P}$ 准确给出了 $f(s)$ 的值，则可以认为 $\mathcal{P}$ 精确知道 $f(x)$ 这个完整的多项式。

上述协议能够很好的实现证明知道一个多项式的目的，同时交互数据也很简洁（每次仅一个数）。但对于实现完整的 zkSNARK 还差很多。本章节的重点在于表达“**随机取值相等即相等**”的性质，这在后面的协议介绍中会经常用到。

1.4 证明知道的多项式被另一个整除

在后面的介绍中，我们通常会需要 $\mathcal{P}$ 证明他知道的多项式，记作 $p(x)$，满足某些性质，比如说，存在两个根（零点）1 和 2。这等价于，$p(x)$ 能够整除 $t(x) = (x-1)(x-2)$。这里的 $t(x)$ 是一个公共的，所有人都知道的多项式（common knowledge）。这等价于，$\mathcal{P}$ 需要证明存在一个多项式 $h(x)$ 使得 $p(x) = t(x)h(x)$。事实上，如果 $\mathcal{P}$ 的确知道符合条件的 $p(x)$，他可以通过多项式除法（long polynomial division）很快的算出 $h(x)$。[1]

为了实现这个目标，我们仍然可以沿用上一章节的思想，将需要证明的结论用一个随机数的值来概括:

- $\mathcal{V}$ 在大范围内随机生成一个数 s 并发送给 $\mathcal{P}$。
- $\mathcal{P}$ 计算 $h(s)$ 和 $p(s)$ 的值并发送给 $\mathcal{V}$。
- $\mathcal{V}$ 计算 $t = t(s)$ 验证 $h \stackrel{?}{=} p \cdot t$。其中 h, p 为 $\mathcal{V}$ 收到的数据的形式化表示。

上述交互过程中所有的变量都要求是整数。不难看出，如果 $p(x)$ 的确能整除 $t(x)$，那么 $p(s) = h(s)t(s)$ 一定成立，且每一部分均为整数。反过来，假设 $p(x)$ 不能整除 $t(x)$，我们将其写成 $p(x) = \hat{h}(x)t(x) + r(x)$，其中 $\hat{h}(x)$ 为商多项式，$r(x)$ 为余数多项式，是一个次数比 $t(x)$ 小的非零多项式。这时，当 $\mathcal{P}$ 拿到 s 后，他给出的 $h(s)$ 需要等于如下所述才能满足验证:

$$h(s) = \frac{p(s)}{t(s)} = \hat{h}(s) + r(s)/t(s),$$

由于 $r(s)/t(s)$ 非零，有大概率不是整数（但仍有非 negligible 的概率是整数），故无法通过验证。

然而，不同于上面一个章节，本章节介绍的协议存在大量的问题:

1. 如上所述，仍有非 negligible 的概率 $r(s)/t(s)$ 是整数，导致不合法的证明验证也能通过。

[1] https://www.purplemath.com/modules/polydiv2.htm 提供了一个计算多项式除法的工具。同时，V 神关于 STARK 资料的资料以及 QAP 论文均提到可以用 FFT 将这个过程加速到 quasi-linear 级别。

2. 上述协议只适用于多项式所有系数都为整数的情况。对于分数系数多项式的情形协议无法给出准确判断。

3. $\mathcal{P}$ 可能根本不知道一个合法的 $p(x)$，他可以直接计算出 $t(s)$ 的值（因为多项式 $t(x)$ 所有人都知道，s 已由 $\mathcal{V}$ 给出），然后随机取一个数字 h，或者随机构造一个多项式 $h(x)$，然后将 $h = h(s)$ 和 $p = h \cdot t(s)$ 发给 $\mathcal{V}$，仍会通过验证。

4. 后续我们仍需要对 $\mathcal{P}$ 给出的的 $p(x)$ 添加某些限定，如规定其次数不超过 d，或者不含某些项（如不含常数项，x^2 项等），上述协议无法扩展出这些限定要求。

接下来我们将引入一些密码学的知识来解决上面提到的几个问题。对于问题 3，4，解决办法为我们让 $\mathcal{P}$ 无法知道 s 的具体值。对于问题 1，2，解决方法为我们让 $\mathcal{P}$ 在 $r(s)$ 不为 0 的情况下无法给出 $r(s)/t(s)$ 的值！

1.5 同态加密

zkSNARK 中使用到的密码学工具较为专业但也不难理解，这里我们不过多的强调技术细节，而是简单的描述 zkSNARK 协议中用到的性质并不做证明。

这里我们以解决上一章节的 4 个问题为主，先介绍下面一些密码学假设：

- 本文之后所有运算都是在模 p 意义下进行（并省略 $\mod p$ 的描述），其中 p 为一个大质数。故当出现分数多项式时，在模 p 运算下均可以转化成整数。

（我们对所有的密码学假设描述描都采用非专业方式，实际的定义可以见参考资料）

假设 1 (离散对数的困难性). 给定 $y = g^x$，通过 y 计算出 x 的值是困难的。[2]

假设 2 (CDH 假设). 给定 g^a，g^b，算出 g^{ab} 的值是困难的。[3]

有了上述工具，定义加密函数

$$E(x) = g^x,$$

我们可以开始对 s 进行隐藏，其方法就是将 s 做一次加密 $E(s) = g^s$ 然后发给 $\mathcal{P}$。那么，$\mathcal{P}$ 如何返回多项式取值 $p(s)$ 和 $h(s)$ 的信息给 $\mathcal{V}$ 呢?

[2]这里的困难可以理解为除了暴力破解没有其他有效的算法。注意到在实数域下求离散对数是不困难的，而这里我们是指在模 p 意义下。

[3]相关的还有更强的 DDH 假设，给定 g^a，g^b，无法分辨 g^{ab} 与一个随机 g^s，其中 s 是一个随机数。后续会提到，DDH 假设对于某些特殊的椭圆曲线群并不成立。

如果 $\mathcal{P}$ 需要证明他知道一个不超过 3 次的多项式 $p(x)$ 整除 $t(x)$，事实上，光靠 $E(s)$ 无法计算出 $h(s)$ 的有效信息。这时，$\mathcal{V}$ 需要同时将 $E(s^0), E(s^2), E(s^3)$ 发送给 $\mathcal{P}$，这时对于 $\mathcal{P}$ 来说就足够计算出 $E(h(s))$ 的值了（即 $\mathcal{P}$ 给出的也是加密后的数据），因为 $E(x)$ 这个函数是满足加减同态性的，即 $E(a+b) = E(a) \cdot E(b)$，假设 $h(x) = x^3 + 2x^2 + 3x$, $\mathcal{P}$ 只需要通过下面的方式计算 [4]

$$E(h(s)) = E(s^3) + 2E(s^2) + 3E(s^1)$$

基于上述结论，若 $\mathcal{P}$ 需要向 $\mathcal{V}$ 证明他知道一个次数不超过 d 的多项式 $h(x)$ 能够被 $t(x)$ 整除，我们修改我们的协议如下：

- $\mathcal{V}$ 在大范围内随机生成一个数 s，并计算 $E(s^0), E(s^1), ..., E(s^d)$ 发送给 $\mathcal{P}$。
- $\mathcal{P}$ 通过收到的加密数据计算 $E(h(s))$ 和 $E(p(s))$，具体方法为计算 $E(s^0)^{c_0} \cdot E(s^1)^{c_1} ... E(s^d)^{c_d}$，其中 $c_0, ..., c_d$ 为对应的多项式系数（$h(x)$ 或 $p(x)$），并发送给 $\mathcal{V}$
- $\mathcal{V}$ 计算 $t(s)$ 并验证 $E_p \stackrel{?}{=} E_h^{t(s)}$。其中 E_h, E_p 为 $\mathcal{V}$ 收到的数据的形式化表示。

值得一提的是，$E(x)$ 这个加密函数满足加减同态性，但是却不满足乘除的同态性，即我们无法通过 $E(a), E(b)$ 计算出 $E(ab)$（否则与 CDH 假设矛盾）。这也是能保证协议安全性的关键之一。

我们接下来分析上述协议如何帮助我们解决上一章的几个问题。

- 对于问题 1，如果 $t(x)$ 不能整除 $h(x)$，如上一章节所述，存在非零余项 $r(s)$，且 $\mathcal{P}$ 需要提供 $E(\hat{h}(s)+r(s)/t(s)) = E(\hat{h}(s)) \cdot E(r(s)/t(s))$。其中 $E(\hat{h}(s)), E(r(s)), E(t(s))$ 均可以算出，但是 $E(r(s)/t(s))$ 却因为加密函数 $E(x)$ 不满足乘除同态性算不出！（即使 $r(s)/t(s)$ 是整数，在模 p 意义下整数和分数已没有区别）这意味着 $t(x)$ 必须完全整除（不能只对某些 s 取值整除）$h(x)$ 证明才能合法。
- 同样，对于问题 2，由于在模 p 意义下分数已经化成了整数，对于分数多项式也需要完全整除才能生成合法的证明。（一般而言，所有实数域上的运算法则在模 p 意义下也成立）
- 对于问题 3，首先，由于不知道 s 的值，$\mathcal{P}$ 不能直接算出 $t(s)$ 再随便乘个数发给 $\mathcal{V}$，但他仍然（根据 $\mathcal{V}$ 提供的加密数据）可以算出 $E(t(s))$ 的值然后再进行 h 次方得到 $E(p(s))$ 发给 $\mathcal{V}$，仍然通过验证。

[4] 在实际程序计算过程中，所有置于指数位置（即 $E(\cdot)$ 函数的自变量）的运算不是模 p，而是应该模这个群的阶 q（对于最简单的离散对数情形 $q = p - 1$），例如 $E(s^3) = E(s^3 \mod q)$。而指数之外的运算仍是模 p，如 $E(s) + E(s^2) = (E(s) + E(s^2)) \mod p$

- 对于问题 4，目前只能对多项式最高次数（不超过 d）做限定。

所以，本章介绍的同态加密技术主要让非完全整除情况下难以生成证明。对于问题 3，4，即假设我们需要对多项式 $h(x)$ 做某些限定的问题，我们在下一章介绍。

1.6 偏移量

本章节主要解决，如何解决给多项式 $p(x)$ 增加限定条件的问题，这里的限定条件主要是为不能包含某些项数，如二次项 x^2 常数项等 (比如，针对问题 3，直接以 $t(s) \cdot h$ 作为结果可能包含了常数项，不满足限定条件)。后续还会引入其他限定条件，如必须是某些给定多项式的线性组合。

解决这个问题的思路为限定 $\mathcal{P}$ 能使用的"原料"的集合，例如，$\mathcal{V}$ 提供加密数据集合 $E(s^i), i \in \{0, ..., d\}$ 时扣掉 $E(1), E(s^2)$ 这两项。这样的确禁止了 $\mathcal{P}$ 使用常数项和二次项来构造 $p(x)$（因为他不知道这两项的加密值，且没有办法算出），但同时他构造 $h(x)$ 时也不能使用常数项和二次项。而问题本身只要求我们限制 $p(x)$，而不能对 $h(x)$ 做任何限制。

为了只对 $p(x)$ 做限制而不限制 $h(x)$，我们需要引入下面的密码学假设。

假设 3 (KEA, Knowledge-of-Exponent Assumption). 给定一组 a, a' 满足 $a' = a^\alpha$（这个 a' 称作偏移量），在不知道 α 的情况下，如果有人能够给出另外一组数 (b, b') 使得 $b' = b^\alpha$，那么一定有 $b = a^c$。

该假设还有扩展形态：给定若干组 $(a_1, a'_1), ..., (a_n, a'_n)$ 满足 $a'_i = a_i^\alpha$，如果有人能够给出另外一组数 (b, b') 使得 $b' = b^\alpha$，那么一定有 $b = a_1^{c_1} a_2^{c_2} ... a_n^{c_n}$。（即 b 一定是 a_i 的指数线性组合）

KEA 这个假设的意思是，由于无法算出 α 的真实值，若要产生一组指数差距也为 α 的数，其唯一方式是通过已给出的，满足指数差距为 α 的数字来计算

- 对于假设的第一种情形，当且仅当通过 $b = a^c, b' = (a')^c$ 来计算，满足 $b' = (a')^c = (a)^{\alpha c} = b^\alpha$
- 对于假设的第二种情形，当且仅当通过 $b = a_1^{c_1} a_2^{c_2} ... a_n^{c_n}, b' = (a'_1)^{c_1} (a'_2)^{c_2} ... (a'_n)^{c_n}$，也满足 $b' = b^\alpha$，即对给定的数据及它们的偏移量做同样的运算。

基于 KEA 假设，$\mathcal{V}$ 可以只给 $\mathcal{P}$ 提供限定集合加密数据的偏移量（例如，不提供 $E(\alpha)$ 和 $E(\alpha^2)$），然后让 $\mathcal{P}$ 同时给出 $E(p(s))$ 和 $E(\alpha p(s))$ 的值，之后 $\mathcal{V}$ 会通过 α 验证 $\mathcal{P}$

提供的这两个数是否满足偏移量条件。这样为了通过验证，$\mathcal{P}$ 在构造 $h(x)$ 时只能使用限定集合的原料，同时由于 $\mathcal{P}$ 不需要提供 $h(s)$ 的偏移量，在构造 $h(x)$ 时没有限制（可以使用 $0, .., x^d$ 所有项）。

基于以上分析，我们可以更新我们的协议如下，

- $\mathcal{V}$ 在大范围内随机生成一个数 s，计算 $E(s^0), E(s^1), ..., E(s^d)$ 以及 $\{E(\alpha s^j)\}_{j\in\mathbb{S}}$（$\mathbb{S}$ 为限定集合）发送给 $\mathcal{P}$
- $\mathcal{P}$ 通过收到的加密数据计算 $E(h(s))$,$E(p(s))$ 和 $E(\alpha p(s))$，具体方法为计算 $E(s^0)^{c_0} \cdot E(s^1)^{c_1} \cdots E(s^d)^{c_d}$, 其中 $c_0, ..., c_d$ 为对应的多项式系数（$h(x)$ 或 $p(x)$），同时针对多项式 $p(x)$ 通过 $\Pi_{j\in\mathbb{S}} E(\alpha s^j)^{c_j}$ 计算偏移量 $E(\alpha p(s))$，并发送给 $\mathcal{V}$
- 计算 $t(s)$ 并验证 $E_p \stackrel{?}{=} E_h^{t(s)}$，$E_p^\alpha = E'_p$。其中 E_h, E_p, E'_p 为 $\mathcal{V}$ 收到的数据的形式化表示。

上述协议解决了多项式增加某些限定条件的问题（问题 4）：$\mathcal{V}$ 不想让 $\mathcal{P}$ 使用哪一项来构造 $h(x)$，就不给他提供对应项的偏移量。我们回到问题 3，现在 $\mathcal{P}$ 能通过 $E(t(s))$ 给出一个 $E(ht(s))$，无法通过 $E(t(s))$ 直接给出 $E(\alpha h(t(s)))$ 的值用于验证。但是，如果限定集合 $\mathbb{S}$ 包含所有 $0, ..., d$，$\mathcal{P}$ 仍然可以通过给定的 $E(\alpha s^j)$ 算出 $E(\alpha t(s))$ 的值，然后进行一次 h 次方作为偏移量。这样验证也会通过。但这个并不影响上面协议的有效性，一方面可以认为 $\mathcal{P}$ 的确知道一个合法的多项式 $ht(x)$，另一方面我们可以对 $\mathbb{S}$ 做限定来阻止上述 trivial 的构造。

上述协议还有个固有缺陷是交互性的，如果 $\mathcal{V}$ 与 $\mathcal{P}$ 合谋，将 α 泄露给 $\mathcal{P}$，那么 $\mathcal{P}$ 即使没有符合条件的多项式也能通过验证，或者 $\mathcal{V}$ 自己提交假证明自己验证自己。另外，作为区块链上的实现，我们希望所有人都能够公开的进行验证而不只是 $\mathcal{V}$ 一人。现在我们把要求再提高一点，

- 能否实现非交互性的协议?
- 能否让所有人（包括智能合约）都能够公开验证证明数据的正确性?（所有人手里的数据都是公开可见的，也就不存在泄露问题）

解决这个问题的核心思想为，我们让 $\mathcal{V}$（此时可以是任何人）在不知道 α 的情况下也能做验证！（对 s 也一样，但其实目前的协议没有 s 只靠 $E(s^i)$ 也能做验证）这样我们在一开始就可以不让 $\mathcal{V}$ 知道 α。具体细节我们将在下一章介绍。

1.7 Pairing: 加密数据的乘法

前面提到，同态加密函数 $E(x)$ 能够通过 $E(a), E(b)$ 算出 $E(a+b)$ 的值，但是无法计算 $E(ab)$ 的值。现在我们增加一项新功能，能够"一定程度"上反映 $E(ab)$ 的信息。现引入下面的 Pairing (bilinear mapping) 的定义。

定义 1. Pairing 是指一个函数 $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 满足

$$e(g^a, g^b) = e(g, g)^{ab}$$

其意义在于，将某个群中的一对元素映射到一个新的群，当且仅当两个原像对应指数的积相同其像也相同。并且，该函数要求是可计算的（即不需要用暴力搜索等方式）。

为了实现这个映射，我们需要重新定义加密所用的群（之前的群为一个模 p 整数群）。Pairing 函数只对某些具有特殊性质的椭圆曲线群满足，故我们需要引入一个已知 paring 函数的椭圆曲线群，然后把上述离散对数的所有乘法运算改为群运算（对于群的定义本文不做过多介绍，可以参考 V 神的资料或 [Elliptic Curves Number Theory and Cryptography]）。为了方便和统一表示，我们仍然用乘法 $\cdot$ 来表示群运算，g 表示群的生成元。故之前章节的加密函数仍为 $E(x) = g^x$，即对生成元做 x 次群运算。其他所有公式在群运算意义下均不需要做表述上的修改。

举例，$e(g^2, g^6) = e(g^3, g^4) = e(g, g)^{12}$。但由于函数的像是在一个新的群里，它不能再作为 e 函数的原像继续计算。

关于 Paring 映射的安全性有下面几点值得强调

- 在椭圆曲线群下离散对数问题仍然是困难的，且 Pairing 函数仍然无法通过 g^a, g^b 算出 g^{ab}（虽然定义 $e(g^a, g^b) = g^{ab}$ 是一个合法的 Pairing 映射，但由于给定原像无法算出 a, b，故这个定义是不可计算的）。但是之前提到的 DDH 假设已不成立。
- 即使引入 Pairing 函数，上述提到的所有假设（除了 DDH，可忽略）仍然是成立的（就目前研究而言），不影响协议的安全性。
- 常用的 Pairing 映射有 Weil 和 Tate-Lichtenbaum，均是可计算的。

所以，Pairing 函数给我们提供了一项新的功能且不影响安全性。我们下面介绍如何应用这项功能实现验证过程的公开化，其主要思想为加密 α, s 的具体值，但同时能够实现验证。

我们首先需要有一个可信的第三方进行 setup 过程：随机生成 α, s 的值，同时计算一系列数据，分为下面两部分。

- 证明钥匙：$E(s^i), E(\alpha s^i), i \in \{0, ..., d\}$（我们之后均假设 $\mathcal{S} = \{0, ..., d\}$，因为之后不需要用到此种限制条件）
- 验证钥匙：$E(t(s)), E(\alpha)$

上述数据统称为 CRS(common reference string)，分为两部分只是因为证明和验证分别只需要用到其中一部分，但整体是对所有人是公开可见的。

基于 CRS，我们可以再次升级我们的协议，让 $\mathcal{P}$ 和 $\mathcal{V}$ 仅通过 CRS 就能实现目标。

- $\mathcal{P}$ 因为 CRS 包含上一章生成证明所需要的所有内容，故此部分和上一章完全相同
- $\mathcal{V}$ 收到的数据形式的形式化表示为 E_h, E_p, E'_p，验证
  $e(E'_p, g) \stackrel{?}{=} e(E_p, E(\alpha))$,$e(E(t(s)), E_h) \stackrel{?}{=} e(E_p, g)$

注 1.1. 上述协议在实现公开化的同时又引入了新的问题：由于验证钥匙 $E(\alpha)$ 对于 $\mathcal{P}$ 也是可见的，这相当于 $\mathcal{P}$ 自动获得了常数项的偏移量！即 $\mathcal{P}$ 可以忽视不允许使用常数项的限定条件。

解决这个问题的思路为将验证钥匙 $E(\alpha)$ 进一步拆分成 $E(\gamma)$ 和 $E(\beta\gamma)$，让 $\mathcal{P}$ 无法从验证钥匙获得常数项的偏移量。具体的协议介绍我们留到之后介绍。

该协议在完全公开的情况下实现了验证目的，但有个缺陷是仍然需要第三方的 trusted setup：zkSNARK 要求第三方在 setup 完成后要彻底销毁 α 和 s 以免造成暴露，而一旦暴露任何人都可以伪造证明。但不能保证第三方有意或无意的没有进行销毁。为了降低这个风险，zkSNARK 采用的方式是由多个人共同实现 setup，每个人以自己的私钥（α, s）和前一个人生成的 CRS 作为输入生成一个新的 CRS。经过 n 个人这么做之后，只要保证 n 个人中至少有一个人彻底销毁了私钥即能保证最终的 CRS 是安全的。具体方式本文不多做介绍，见参考资料。无论如何，对 trusted setup 的依赖算是 zkSNARK 的缺陷之一，尽管有很多区块链项目采用了这种做法，但仍需付出成本以及损失去中心化程度。后续提出的新方案 zkSTARK 等可以不依赖这个过程。

值得一提的是，上述协议虽然引入了加密过程，但实际上还没有解决零知识证明的需求：零知识要求 $\mathcal{P}$ 不会泄露 $p(x)$ 的任何信息。但是对于上述协议，假设 $\mathcal{V}$ 知道 $\mathcal{P}$ 心中的 $p(x)$ 在一个较小的集合之内，比如只可能二者选一，他可以自己以 $\mathcal{P}$ 的身

份用两个候选多项式运行上述协议，然后看哪个返回的证明数据和真 $\mathcal{P}$ 提供的证明数据相同就能猜到哪个 $\mathcal{P}$ 心中的多项式。引入零知识性质并不复杂，只需要 $\mathcal{P}$ 将他提供的数据也做一次偏移同时也能保证验证通过即可。具体实施方式我们留到最后介绍。

至此，我们已经完成 zkSNARK 的准备工作，上述协议主要介绍了后续会用到的一些核心思想，还不能解决真正的实际问题。我们接下来将从实际问题出发，展示如何将指导一个问题的解转化为证明知道一个多项式，且满足该多项式是给定多项式的线性组合的问题。

1.8 从实际场景到方程

zkSNARK 中关于 $\mathcal{P}$ 向 $\mathcal{V}$ 证明一个东西的具体定义涉及 P 和 NP 的相关知识，具体见参考资料 [5]。但大致而言，zkSNARK 致力于下面的实际场景：

- **可信计算场景：$\mathcal{V}$ 有一个复杂的函数待计算，因自身算力不足将计算过程外包给 $\mathcal{P}$ 来计算。$\mathcal{P}$ 完成计算后需提交一个计算结果是正确的证明，$\mathcal{V}$ 用这个证明能够验证计算结果的正确性，且验证的计算难度远远小于自己重新计算。**
- **零知识证明场景：$\mathcal{P}$ 需要向 $\mathcal{V}$ 证明自己有合法的数据且不泄露数据的任何信息，如存款余额（已加密）大于某个数，某个地址的私钥等等。$\mathcal{V}$ 能够通过这个证明验证数据的合法性但获取不到数据的任何信息。**

大量可信计算和零知识证明要解决的问题本质上是一系列方程的问题，即，$\mathcal{P}$ 向 $\mathcal{V}$ 证明他知道某个方程（组）的解。例如下面这段代码：

Algorithm 1: 由输入决定运算符

```
Input: w, a, b
if w then
    return a × b
else
    return a + b
```

这段代码写成函数形式，就是 $f(w,a,b) = wab + (1-w)(a+b)$。进一步的，如果 $\mathcal{P}$ 想向 $\mathcal{V}$ 证明他知道一组输入变量的赋值使得代码的运行结果为 8，他就是要证明存在 (w,a,b) 使得 $f(a,b,w) = 8$。

参考资料中还列举了其他的例子，包括

- 证明知道方程 $x^3 + x + 5 = 35$ 的解（V 神资料）。

- 证明知道方程 $ab(a+c)=7$ 的解（ZCash 资料）。

这些都是 $\mathcal{P}$ 证明知道方程的解的问题。我们接下来将说明如何把证明知道方程的解问题和前文提到的证明知道特定性质多项式的问题结合起来。

1.9 R1CS

R1CS 是 Rank-1 Constraint System 的缩写，它将一个复杂的方程统一成约束向量的形式。

我们先以方程 $ab(a+c)=d$ 为例展示这个构造过程。（注意此时还没有限制 $d=7$）。

1. 我们首先要引入若干中间变量，将原方程转化成若干方程组，其中方程组的每个方程都是

$$(\text{若干变量相加})\times(\text{若干变量相加})=(\text{若干变量相加})$$

的形式。

对于上述例子，我们需要引入中间变量 e，同时将原方程转化成下面方程组

$$\begin{cases} a\times b=e \\ (a+c)\times e=d \end{cases}$$

这其实是将一个计算过程转化成一个算术电路的过程，其中，带三个颜色的部分分别称作电路一个门的左输入，右输入和输出。为了节省画图，我们仍用方程的形式来说明。

2. 随后，我们需要对上述方程组中的每一个方程（即，每一个电路门）都生成三个向量（分别称作左向量，右向量和出向量），其维度为不同变量的数目（例子中为 5 个，每一维分别对应 a,b,c,d,e）。记方程组方程的数目为 n。记 $\boldsymbol{l}_i,\boldsymbol{r}_i,\boldsymbol{o}_i),i\in[n]$ 分别为第 i 个方程对应的左向量，右向量和出向量。用向量 $\boldsymbol{v}=(v_1,...,v_m)$ 表示 m 个变量组成的向量（例如，$\boldsymbol{v}=(a,b,c,d,e)$），这些向量的构造方式如下：

对于第 $i\in[n]$，$j\in[m]$。如果变量 v_j 出现在了第 i 个方程组的左/右/出位置，则将 $\boldsymbol{l}_i/\boldsymbol{r}_i/\boldsymbol{o}_i$ 第 j 维置为 1，否则置为 0。

对于上述例子，$i=2, j=5$，不难得出构造的向量如下（变量 a 对应向量的第一位）：

$$\boldsymbol{l}_1=(1,0,0,0,0), \boldsymbol{r}_1=(0,1,0,0,0), \boldsymbol{o}_1=(0,0,0,0,1)$$
$$\boldsymbol{l}_2=(1,0,1,0,0), \boldsymbol{r}_2=(0,0,0,0,1), \boldsymbol{o}_2=(0,0,0,1,0)$$

3. 下一步，定义这些变量的好处是啥？我们考察 $\boldsymbol{v}$ 与上述向量的内积（$\circ$ 表示内积）。

   通过观察，或者直接根据定义，我们发现，$\boldsymbol{v}\circ\boldsymbol{l}_i$ 恰好是第 i 个方程的红色（左输入）部分！（对于 $\boldsymbol{v}\circ\boldsymbol{r}_i, \boldsymbol{v}\circ\boldsymbol{o}_i$ 结论相同，对应绿色和蓝色部分）。所以，整个方程组可以用向量表达成下面形式：

$$(\boldsymbol{v}\circ\boldsymbol{l}_i)\cdot(\boldsymbol{v}\circ\boldsymbol{r}_i)=(\boldsymbol{v}\circ\boldsymbol{o}_i), i\in[n], \tag{1}$$

   故 $\mathcal{P}$ 证明知道原方程的解等价于 $\mathcal{P}$ 证明知道一个向量 $\boldsymbol{v}$ 是的上述方程组成立。

   n 组向量 $(\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i)$ 构成 n 个满足特定形式的方程（电路门），这就是所谓的 R1CS。

1.10 QAP

数学家们总是期望用统一的形式，最低的复杂度来达成目的。上述 R1CS 虽然已经是 well define，但其交互过程涉及 $3n$ 个向量，共 $3mn$ 个数据，验证复杂度过高。本章我们将介绍如何通过多项式将 $3n$ 个向量结合起来，让验证过程能够一步到位，且与我们前几章一直在介绍的“证明知道一个多项式”的思想结合起来。其核心思想为，本来需要做 n 次验证，我们构造多项式使得该多项式在特定 n 个点的取值恰好为向量的值，然后只需要对多项式判断相等即可。

我们首先介绍下面的性质：

特征 1. 有且仅有一个度数为 $d-1$ 的多项式经过 d 个点 (x_i, y_i)，且该多项式可以通过拉格朗日插值方式求出，其时间复杂度为根据 QAP 论文的描述为线性，但根据 V 神描述似乎要用到 FFT 且为 $O(d\log d)$。其差别可能与多项式的表示方式有关：QAP 论文提到该多项式的表达方式为用 y_i 表示，并没有算出系数。这种表示方式是否足够待考证。

上一章提到，我们一共有 $3n$ 个向量，一组 $(\boldsymbol{l}_i, \boldsymbol{r}_i, \boldsymbol{o}_i)$ 对应一个限制条件，每个

向量有 m 维。我们现在按照下面的规则来构造多项式

1. 我们将分别针对左，右和出构造 m 个多项式，一共 $3m$ 个。一下我们均以左位置为例：基于向量 $\boldsymbol{l}_i, i \in [n]$，对于每个 $j \in [m]$，构造 $n-1$ 次多项式 $L_j(x)$ 满足

   $L_j(x)$ 在 z_i 处的取值为 $\boldsymbol{l}_i^{(j)}$，其中 $z_i, i = 1, 2, ..., n$ 为任意 n 个不同的数（实际就可取 $z_i = i$）。

   根据上述性质，多项式是唯一确定的，即点集 $\{i \in [n] | (z_i, \boldsymbol{l}_i^{(j)})\} \in L_j(x)$。（向量上标 (j) 表示该向量的第 j 维元素）

   举例如下：

	a	b	c	d	e
$(z_1,$	1	0	0	0	0
$(z_2,$	1	0	1	0	0
	$L_a(x)$	$L_b(x)$	$L_c(x)$	$L_d(x)$	$L_e(x)$

	a	b	c	d	e
$(z_1,$	0	1	0	0	0
$(z_2,$	0	0	0	0	1
	$R_a(x)$	$R_b(x)$	$R_c(x)$	$R_d(x)$	$R_e(x)$

	a	b	c	d	e
$(z_1,$	0	0	0	0	1
$(z_2,$	0	0	0	1	0
	$O_a(x)$	$O_b(x)$	$O_c(x)$	$O_d(x)$	$O_e(x)$

   （即，当要获得第 i 个向量在第 j 维的值，我们只需要计算多项式 $L_j(x)$ 在 z_i 点的取值。）

   对于右位置和出位置，我们同样方式构造出 $R_j(x), O_j(x), j \in [m]$

2. 一旦有了这些多项式 $L_j(x), R_j(x), O_j(x)$（这些多项式在本文之后都会被当做 common knowledge，因为原问题的计算过程是公开的），上一章的 R1CS 限制1就转变为下面的形式：

$$(\sum_{j \in m} L_j(x) v_j) \times (\sum_{j \in m} R_j(x) v_j) = \sum_{j \in m} O_j(x) v_j$$

   对于所有 $x = z_1, z_2, ..., z_n$ 均成立。

   此时我们可以巧妙地利用多项式的性质，如果我们把上面等式（左边-右边）看做一个整体的多项式，则这个多项式在 z_i 每一点都等于 0，这意味着该多项式能够被 $(x - z_1)...(x - z_n)$ 整除！这下就和前面证明知道一个多项式被另一个整除的思想结合了。

所以，如果记 $L(x)=\sum_{j\in m}L_j(x)\times v_j$ （类似定义 $R(x),O(x)$，为 $\mathcal{P}$ 提供的证明的一部分），则 R1CS 限制（1）能转化为下面的问题：

- **$L(x),R(x)$ 和 $O(x)$ 分别为 $L_j(x),R_j(x)$ 和 $O_j(x)$（$j\in[m]$）的线性组合，且三者的线性组合系数相同。**
- **$L(x)R(x)-O(x)$ 能被 $t(x)=(x-z_1)\cdots(x-z_n)$ 整除，即存在多项式 $h(x)$ 使得 $L(x)R(x)-O(x)=h(x)t(x)$。**

这就是所谓的 QAP（quadratic arithmetic program）。

3. 现在 $\mathcal{P}$ 的任务为，基于给定的 $L_j(x),R_j(x),O_j(x),j\in[m],t(x)$，提供合法的 $L(x),R(x),O(x),h(x)$ 满足 QAP 的两个限制条件。

   这里我们继续用（1）的结果来举例。

   不妨设 $z_1=1,z_2=2$，通过简单的计算不难算出，

$$L_a(x)=1,L_b(x)=0,L_c(x)=x-1,L_d(x)=0,L_e(x)=0$$

$$R_a(x)=0,R_b(x)=2-x,R_c(x)=0,R_d(x)=0,R_e(x)=x-1$$

$$O_a(x)=0,O_b(x)=0,O_c(x)=0,O_d(x)=x-1,O_e(x)=2-x$$

   假设 $\mathcal{P}$ 知道原问题的一组解 $(a,b,c,d,e)=(2,2,1,12,4)$（确实是一组解），那么构造的函数如下

$$\begin{aligned}
&L(x)=2+x-1=x+1,\\
&R(x)=2(2-x)+4(x-1)=2x,\\
&O(x)=12(x-1)+4(2-x)=8x-4,\\
&L(x)R(x)-O(x)=2x^2-6x+4=2(x-1)(x-2),
\end{aligned}$$

   确实能被 $t(x)=(x-1)(x-2)$ 整除。

上述交互过程已经是 zkSNARK 协议的雏形，然而到目前为止我们并没有解决任何实际问题：我们只能够证明存在几个数满足 $(a+b)ac=d$，这在任何时候都是显然的。而我们实际要完成的目标是证明存在几个数满足 $(a+b)ac=7$。显然，上面的构造 $(2,2,1,12,4)$ 能通过验证但是并不满足方程（因为 $d=12$ 而不是 7）。为了证明这个方程存在解，我们需要限定住 $d=7$，这就需要我们的 QAP 构造能够兼容出现常数的情形。这些扩展我们将在下一章介绍。

1.11 常数兼容

上一章介绍的 QAP 构造颇有局限性，我们在这一章对 QAP 进行稍微的扩展以便兼容方程中的常数。这主要包括下面两种情形：

- 常数作为变量的系数：例如，对于第 i 个方程，有

$$3a \times 2b = 6c$$

对于这种情形修正比较简单，在之前的构造中，$L_a(x)$ 这个多项式在 z_i 这点的取值为 1，现在我们将这个取值置为 3，即，让多项式通过 $(z_i, 3)$ 这个点。仍然可以用拉格朗日插值来构造。这样，$L_a(x)$ 与 a 相乘的时候 z_i 点取值为 $3a$，是我们想要的数字。对于右输入和输出部分以同样方式构造。

- 常数本身作为输入的一项，例如，对于第 i 个方程，有
$(a+5) \times b = c$ 或 $a + b = 5$（相当于 $(a+b) \times 1 = 5$）等等
这些本质上是常数单独作为电路输入或输出的一项，这时，我们对左输入部分新添加一个公共函数 $L_0(x)$，它用于记录左输入的常数部分：因为 5 出现在了方程 i 的左输入中，我们让 $L_0(x)$ 通过 $(z_i, 5)$ 这个点（对 $a + b = 5$，让 $O_0(x)$ 通过 $(z_i, 5)$ 这个点）。如果有其他常数 c 出现在了其他方程 k 的左输入，则让 $L_0(x)$ 同时也通过 (z_k, c) 这个点。如果方程没有常数项则将该点的值置为 0。同样用拉格朗日插值求出。最终 $L(x)$（之后的协议用 $\hat{L}(x)$ 来表示）的构造方法变为

$$\sum_{j \in m} L_j(x) \times v_j + L_0(x),$$

对右输入，输出部分同理。

- 如果有些方程存在多个独立的常数项，例如 $(12345 + 88888 + a) \times b = c$，正常来说（任何人）可以把这若干个常数直接加起来变成新的，但如果连常数的加法也不愿意做，这时可以将常数多项式 $L_0(x)$ 等拆分成多个，每个的取值对应一个常数输入项。

- 根据参考资料中介绍，zkSNARK 甚至支持有一部分输入甚至可以由 $\mathcal{V}$ 来制定！即，协议定义若干公共多项式 $L'_j(x)$，针对那些由 $\mathcal{V}$ 指定的变量。在计算 $L(x)$ 时，将 $L'_j(x)v'_j$ 作为添加项加在原 $L(x)$ 的后面，其中 v'_j 为 $\mathcal{V}$ 给出的变量值，$\mathcal{P}$ 无法改变。

这个功能的引入大大增加了协议的鲁棒性，同时也可能证明一些类似于：“存在 a, b, c 使得对任意的 x, y, z 都有 xxxxxx”的论点，但不是很确定。

为方便起见，本文将省略 $\mathcal{V}$ 指定变量的情形，且只引入三个常数多项式 $L_0(x), R_0(x), O_0(x)$。

1.12 QAP 限制条件

理解了电路到多项式的转化之后，我们可以开始介绍用于证明满足 QAP 两个限制的协议。

首先，沿用本文章节1.4的思想，为了证明存在 $L(x)R(x) - O(x) = h(x)t(x)$，只需要证明对于一个随机数 s，$\mathcal{P}$ 能给出相应的函数值满足 $L(s)R(s) - O(s) = h(s)t(s)$ 即可。(recall: 这个 s 由第三方 setup 时以加密的方式 $E(s) = g^s$ 给出)。

然后，第一个条件要求 $L(x)$ 必须是给定的 $L_j(x)$ 的线性组合。为了实现这个目标，我们用第1.6章的偏移量思想即可：将 $E(\alpha L_j(s))$ 作为证明钥匙的一部分，这样 $E(\alpha L(s))$ 只能从 $E(\alpha L_j(s))$ 的线性组合来获得。但是，为了防止 $\mathcal{P}$ 在构造 $L(x)$ 时使用 $R_i(x)$ 或 $O_i(x)$ 的信息，我们需要对左右输入和输出使用不同的偏移量，即随机选取 $\alpha_l, \alpha_r, \alpha_o$，并将 $E(\alpha_l L_j(s)), E(\alpha_r R_j(s)), E(\alpha_o O_j(s)), i \in [n]$ 作为证明钥匙的的一部分。相应的，$\mathcal{P}$ 需要提供满足偏移量的数对

$$(E(L(s)), E(\alpha_l L(s))), (E(R(s)), E(\alpha_r R(s))), (E(O(s)), E(\alpha_o O(s)))。$$

接下来，我们着重介绍如何让 QAP 的第二个限制条件满足，这也是 zkSNARK 协议的核心之一。

这个问题可以形式化描述成下面的形式：

给定三个向量 $\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{o}$，$\mathcal{P}$ 给出数字 L, R, O。$\mathcal{V}$ 需要能够验证存在向量 $\boldsymbol{v}$，使得

$$\boldsymbol{l} \circ \boldsymbol{v} = L, \boldsymbol{r} \circ \boldsymbol{v} = R, \boldsymbol{o} \circ \boldsymbol{v} = O。$$

解决方案为：

$\mathcal{V}$ 随机取三个数 $\beta_l, \beta_r, \beta_o$，用这个三个数对给定向量做一次线性组合 $\beta_l \boldsymbol{l} + \beta_r \boldsymbol{r} + \beta_o \boldsymbol{o}$（向量相加为每一维对应相加，结果仍为向量）并将这个结果发送给 $\mathcal{P}$。随后 $\mathcal{V}$ 要求 $\mathcal{P}$ 提供 L, R, O 的同时要提供这三个数的线性组合 $\beta_l L + \beta_r R + \beta_o O$。

这个解决方案为什么 work 呢?

- 首先我们将在下文提到，上述数据交互均是以同态加密过的，其关键在于，$\mathcal{P}$ **无法获取 $\beta_l, \beta_r, \beta_o$ 的值**。(如果是非加密的数据交互，我们可以从线性组合后的

向量推断出组合系数，但是对于加密后的数据不行）。

- $\mathcal{P}$ 要提供 L, R, O 的线性组合但他不知道具体的系数，他只能通过给定的向量 $\beta_l \boldsymbol{l} + \beta_r \boldsymbol{r} + \beta_o \boldsymbol{o}$ 来做文章。但由于其中涉及的系数也是不知道的，他唯一能做的就是取一个 v' 对这个向量做一次内积 $(\beta_l \boldsymbol{l} + \beta_r \boldsymbol{r} + \beta_o \boldsymbol{o}) \circ \boldsymbol{v}'$ 然后期望结果能通过验证。

  假设 $\mathcal{P}$ 用了不同的 $\boldsymbol{v}$ 对给定向量做内积，设为 $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$，那么想要验证通过的话要求

  $$\beta_l \boldsymbol{v}_1 \circ \boldsymbol{l} + \beta_r \boldsymbol{v}_2 \circ \boldsymbol{r} + \beta_o \boldsymbol{v}_3 \circ \boldsymbol{o} = (\beta_l \boldsymbol{l} + \beta_r \boldsymbol{r} + \beta_o \boldsymbol{o}) \circ \boldsymbol{v}'$$

  由于 $\beta_l, \beta_r, \beta_o$ 是随机给定，上述成立要求 $\boldsymbol{v}_1 = \boldsymbol{v}', \boldsymbol{v}_2 = \boldsymbol{v}', \boldsymbol{v}_3 = \boldsymbol{v}'$。[5]

- 假设 $\mathcal{P}$ 用了相同的 $\boldsymbol{v}$ 对给定向量做内积，不难发现只要用 $\boldsymbol{v}$ 对 $\beta_l \boldsymbol{l} + \beta_r \boldsymbol{r} + \beta_o \boldsymbol{o}$ 做内积即可得到满足验证要求的结果。

这个过程叫做变量一致性检验（variable consistency check）。

基于上述解决方案，下面我们具体这部分介绍协议实现方式（注意到这只是 QAP 解决限制条件的部分，省略了验证多项式整除的部分）。

- Setup 过程：
  - 随机选择 s 以及偏移量系数 $\alpha_l, \alpha_r, \alpha_o$ (代替之前只选择一个 α) 以及一致性系数 $\beta_l, \beta_r, \beta_o$。
  - 计算证明钥匙

    $$\big(\{E(s^k)\}_{k=\{0,1,\ldots,d\}},$$
    $$\{E(L_j(s)), E(R_j(s)), E(O_j(s)), E(\alpha_l L_j(s)), E(\alpha_r R_i(s)), E(\alpha_o O_j(s))\}_{j\in[m]}$$
    $$\{E(\beta_l L_j(s) + \beta_r R_j(s) + \beta_o L_j(s))\}_{j\in[m]}\big)$$

    上式最后一项即为随机线性组合后的结果。
  - 计算验证钥匙

    $$(E(\alpha_l), E(\alpha_r), E(\alpha_o), E(\beta_l), E(\beta_r), E(\beta_o))$$

- 证明过程：

[5] 不同的随机数可以看做不同的维度向量，好比两个复数 $a + bi$ 和 $c + di$ 相等必须 $a = c, b = d$。同样，上式左右相等必须 β_l 的左右两边的系数相同，这等价于存在向量 $\boldsymbol{v}_1 - \boldsymbol{v}'$ 与 $\boldsymbol{l}$ 正交。对于普通数据求出正交向量并不难，然而对于协议中使用到的加密数据，获得这个正交向量也是不可解的（否则 CDH 假设被攻破）。所以 $\mathcal{P}$ 让等式成立只能 $\boldsymbol{v}_1 = \boldsymbol{v}'$。

- 计算目标多项式的加密值

$$E(L(s)) = \Pi_{j=1}^{m} E(L_j(s))^{v_j},$$
$$E(R(s)) = \Pi_{j=1}^{m} E(R_j(s))^{v_j},$$
$$E(O(s)) = \Pi_{j=1}^{m} E(O_j(s))^{v_j},$$

（其中 v_j 为 $\mathcal{P}$ 对第 j 个变量的赋值，对应前面的 v_j。同时注意常数多项式 $L_0(s)$ 等没有包含在计算过程中，因为只有计算 $h(s)$ 时才需要用到，变量一致性检验只需对 $\mathcal{P}$ 能决定系数的多项式进行检测，即 $j \in [m]$）

- 计算目标多项式的偏移量

$$E(\alpha_l L(s)) = \Pi_{j=1}^{m} E(\alpha_l L_j(s))^{v_j},$$
$$E(\alpha_r R(s)) = \Pi_{j=1}^{m} E(\alpha_r R_j(s))^{v_j},$$
$$E(\alpha_o O(s)) = \Pi_{j=1}^{m} E(\alpha_o O_j(s))^{v_j}$$

- 计算一致性目标值

$$E(Z(s)) = \Pi_{j=1}^{m} E(\beta_l L_j(s) + \beta_r R_j(s) + \beta_o O_j(s))^{v_j}$$

相当于用 $\boldsymbol{v}$ 对给定的一致性向量做内积。

- 验证过程：
  - 假设收到证明的形式化表示为 $(E_l, E_r, E_o, E'_l, E'_r, E'_o, E_z)$
  - 偏移量验证：

$$e(E_l, E(\alpha_l)) = e(E'_l, g),$$

，对右边和输出位类似。

  - 一致性验证：

$$e(E_l, E(\beta_l))e(E_r, E(\beta_r))e(E_o, E(\beta_o)) = e(E_z, g)$$

即，对 $\mathcal{P}$ 提供的向量做线性组合等于 $\mathcal{P}$ 提供的一致性目标。

1.13 可塑性（Malleability）问题

正如我们前文提到过，直接引入偏移量 α_l, β_l 等会带来一些其他问题，即 $\mathcal{P}$ 在用 $L_i(x)$ 在做线性组合的之后可以任意加入常数项 $L(x)+c$，因为常数项的偏移量对 $\mathcal{P}$ 也是可见的。参考资料 [3] 着重分析了 $\mathcal{P}$ 如何通过添加常数项来伪造证明并且能通过验证，本文这里不做赘述。

解决方案为让常数的偏移量不可见，即在上章协议基础上做如下修改

- Setup 过程：
  - $\cdots$ 随机产生 γ
  - $\cdots$ 设定验证钥匙 $(\cdots, E(\beta_l\gamma), E(\beta_r\gamma), E(\beta_o\gamma), E(\gamma))$，代替原来的 $E(\beta_l), E(\beta_r), E(\beta_o)$ 等
- 证明过程：不变
- 验证过程：

  将一致性验证改为

  $$e(E_l, E(\beta_l\gamma))e(E_r, E(\beta_r\gamma))e(E_o, E(\beta_o\gamma)) = e(E_z, E(\gamma))$$

  这样由于 $\mathcal{P}$ 无法获知 $E(\beta_l)$ 的值，他无法给 $L(x)$ 添加合法的常数项以通过验证。

值得一提的是我们应该排除多项式为常数多项式的情况：假设 $L_1(x)=1, R_1(x)=0, O_1(x)=0$，那么证明钥匙 $E(\beta_l L_i(s)+\beta_r R_i(s)+\beta_o L_i(s))$ 将直接等于 $E(\beta_l)$，等同于本该隐藏的 $E(\beta_l)$ 暴露了。

如何排除这种情况呢（因为变量很稀疏这种情况还挺常见）？或许可以添加一个无用的电路门，包含所有变量作为输入，强行扰动原有的差值多项式

因为一致性验证条件已经保证了 $E(L(x))$ 等不能随便添加常数项，我们不需要对 $E(\alpha_l)$ 等验证钥匙再做调整（如变成 $E(\alpha_i\gamma)$）。

参考资料 [1] 介绍的 zkSNARK 方案用了个巧妙方式解决一致性验证，它用一个统一的 β 代替上文的 β_l 等，但是针对左/右输入和输出分别采用不同的生成元 $g_l = g^{\rho_l}, g_r = g^{\rho_r}, g_o = g^{\rho_o}$，其中 ρ_l, ρ_r 由第三方随机产生，且 $\rho_o = \rho_l\rho_r$。我们将在下章介绍该协议的完整形式。

1.14 可信计算协议

现在我们已经将 QAP 介绍完毕。下面我们开始描述实现 QAP 的完整协议内容。（因为我们使用了不同的生成元，原加密函数 $E(x) = g^x$ 失效。我们将直接用原始形式来表示，如 g_l^x 等。）

- Setup 过程：
  - 随机生成 $z_1, ..., z_n$，计算 $t(x) = (x - z_1) \cdots (x - z_n)$（可见）
  - 随机生成 $s, \alpha_l, \alpha_r, \alpha_o$ （不可见）
  - 根据章节1.10中介绍的方式构造出 $\{L_j(x), R_j(x), O_j(x)\}_{j\in\{0,...,m\}}$。具体的，$L_j(z_i) = c_{l,i,j}$，表示变量 v_j 在第 i 个方程左边的系数。$L_0(z_i)$ 等于 i 个方程左边的常数项。对右输入和输出部分同理。（可见）
  - 随机生成 $\beta, \gamma, \rho_l, \rho_r$，置 $\rho_o = \rho_l\rho_r$。（不可见）
  - 定义生成元 $g_l = g^{\rho_l}, g_r = g^{\rho_r}, g_o = g^{\rho_o}$（可见）
  - 计算证明钥匙（可见）：

$$\begin{aligned}&\big(\{g^{s^k}\}_{k=\{0,1,...,d\}},\\&\{g_l^{L_j(s)}, g_r^{R_j(s)}, g_o^{O_j(s)}\}_{j\in\{0,1,...,m\}},\\&\{g_l^{\alpha_l L_j(s)}, g_r^{\alpha_r R_j(s)}, g_o^{\alpha_o O_j(s)}, g_l^{\beta L_j(s)} \cdot g_r^{\beta R_j(s)} \cdot g_o^{\beta O_j(s)}\}_{j\in[m]}\big)\end{aligned}$$

  - 计算验证钥匙（可见）：

$$\big(g, g_o^{t(s)}, g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o}, g^{\beta\gamma}, g^{\gamma}, g_l^{L_0(s)}, g_r^{R_0(s)}, g_o^{O_0(s)}\big)$$

- 证明过程：（假定 $\mathcal{P}$ 有一组合法的 $(v_1, ..., v_m)$）
  - 计算目标多项式：

$$\begin{aligned}L(x) &= \Sigma_{j=1}^{m} L_j(x) v_j,\\R(x) &= \Sigma_{j=1}^{m} R_j(x) v_j,\\O(x) &= \Sigma_{j=1}^{m} O_j(x) v_j, \in [m]\end{aligned}$$

- 计算目标多项式的加密值：

$$
\begin{aligned}
g_l^{L(s)} &= \Pi_{j=1}^{m}(g_l^{L_j(s)})^{v_j}, \\
g_l^{R(s)} &= \Pi_{j=1}^{m}(g_r^{R_j(s)})^{v_j}, \\
g_l^{O(s)} &= \Pi_{j=1}^{m}(g_o^{O_j(s)})^{v_j}, j \in [m]
\end{aligned}
$$

- 计算目标多项式的偏移量：

$$
\begin{aligned}
g_l^{\alpha_l L(s)} &= \Pi_{j=1}^{m}(g_l^{\alpha_l L_j(s)})^{v_j}, \\
g_r^{\alpha_l R(s)} &= \Pi_{j=1}^{m}(g_r^{\alpha_r R_j(s)})^{v_j}, \\
g_o^{\alpha_l O(s)} &= \Pi_{j=1}^{m}(g_o^{\alpha_o O_j(s)})^{v_j}, j \in [m]
\end{aligned}
$$

- 计算一致性目标值：

$$
g^{Z(s)} = \Pi_{j=1}^{m}(g_l^{\beta L_j(s)} \cdot g_r^{\beta R_j(s)} \cdot g_o^{\beta O_j(s)})^{v_j}
$$

- 添加常数项：

$$
\begin{aligned}
\hat{L}(x) &= L(x) + L_0(x) \\
\hat{R}(x) &= R(x) + R_0(x) \\
\hat{O}(x) &= O(x) + O_0(x)
\end{aligned}
$$

- 计算 $h(x)$：

$$
h(x) = \frac{\hat{L}(x)\hat{R}(x) - \hat{O}(x)}{t(x)}
$$

- 计算 $h(x)$ 加密值 $g^{h(s)}$（利用 $\{g^{s^k}\}_{k=\{0,1,\ldots,d\}}$）
- 提供证明：

$$
(g_l^{L(s)}, g_r^{R(s)}, g_o^{O(s)}, g_l^{\alpha_l L(s)}, g_l^{\alpha_r R(s)}, g_o^{\alpha_l O(s)}, g^{Z(s)}, g^{h(s)})
$$

- 验证过程：
  - 假设收到证明的形式化表示为 $(g_l^L, g_r^R, g_o^O, g_l^{L'}, g_r^{R'}, g_o^{O'}, g^Z, g^h)$
  - 偏移量验证：$e(g_l^L, g^{\alpha_l}) = e(g_l^{L'}, g))$，其余类似。

– 一致性验证：

$$e(g_l^L g_r^R g_o^O, g^{\beta\gamma}) = e(g^Z, g^\gamma)$$

– 正确性验证：

$$e(g_l^L g_l^{L_0(s)}, g_r^R g_r^{R_0(s)}) = e(g_o^{t(s)}, g^h) e(g_o^O g_o^{O_0(s)}, g)$$

等价于

$$e(g,g)^{\rho_l \rho_r (L(s)+L_0(s))(R(s)+R_0(s))} = e(g,g)^{\rho_o t(s) h(s) + \rho_o (O(s)+O_0(s))}$$

为了理解这个一致性证明的正确性，只需要将 $\beta\rho_l$ 当做上一章的 β_l 即可，同样满足随机性且不可见。

这就是 zkSNARK 关于可信计算协议的全部，可以解决用来大量涉及可信计算场景的实际问题。为了增加零知识的部分，上式协议还要稍作修改，我们将在下一章进行介绍。

1.15 零知识证明协议

零知识证明的需求主要针对 $\mathcal{P}$ 不愿意透露变量 $\boldsymbol{v}$ 具体赋值的场景。（包括具体 $L(x), R(x), O(x), h(x)$ 等的构造方式）前文已经提到，现有的加密函数不足以实现零知识的需求。事实上，$\mathcal{P}$ 提供的任何**确定性的**的证明均不能满足零知识的需求（参考 CPA-secure）。所以，$\mathcal{P}$ 必须对自己的证明添加随机偏移，且这个偏移量只有自己知道。（关于零知识的需求的具体定义见参考资料 [7]）。

对证明添加随机偏移后，其核心性质可以概括为下面两点：

- 证明看起来和随机数没有区别。（故无法从中获取任何信息）
- 证明能够通过验证。

满足这两个条件的构造方式为：

- 随机选取 $\delta_l, \delta_r, \delta_o$，将 $L(x), R(x), O(x)$ 分别用 $L(x)+\delta_l t(x), R(x)+\delta_r t(x), O(x)+\delta_o t(x)$ 代替。
- 相应的，$h(x)$ 变成 $\delta_r L(x) + \delta_l R(x) + \delta_l \delta_r t(x) - \delta_o$

不难发现，调整之后的正确性能让能满足，由于 δ_l 随机且对其他人不可见，故新的证明没有暴露任何信息，满足条件。

关于这个构造有两点值得一提：

- 一定要选取不同的 δ_l, δ_r，否则假设它们相同，那么当 $L(x) = R(x)$ 时即使加入了偏移量它们也是相等的，暴露了信息。
- 还有一种偏移方法为做积，即用 $(L(x))^{\delta_l}$ 来代替 $L(x)$ 等，但是这样难以构造出合适的 $h(x)$。参考资料 [3] 对于为什么不能用做积来偏移做了详细介绍。

基于上述思想，我们可以开始介绍完整的零知识证明协议。注意到为了让 $\mathcal{P}$ 能够实现随机偏移，需要引入在证明钥匙中添加加密后的 $t(s)$ 及其对应的偏移量。

1.16 zkSNARK 协议

本章我们将介绍完整的 zkSNARK 协议，其中零知识证明组件作为可选部分用蓝色字体标出。

- Setup 过程：
  - 随机生成 $z_1, ..., z_n$，计算 $t(x) = (x - z_1) \cdots (x - z_n)$（可见）
  - 随机生成 $s, \alpha_l, \alpha_r, \alpha_o$ （不可见）
  - 根据章节1.10中介绍的方式构造出 $\{L_j(x), R_j(x), O_j(x)\}_{j\in\{0,...,m\}}$。具体的，$L_j(z_i) = c_{l,i,j}$，表示变量 v_j 在第 i 个方程左边的系数。$L_0(z_i)$ 等于 i 个方程左边的常数项。对右输入和输出部分同理。（可见）
  - 随机生成 $\beta, \gamma, \rho_l, \rho_r$，置 $\rho_o = \rho_l\rho_r$。（不可见）
  - 定义生成元 $g_l = g^{\rho_l}, g_r = g^{\rho_r}, g_o = g^{\rho_o}$（可见）
  - 计算证明钥匙（可见）：

$$
\begin{aligned}
&\big(\{g^{s^k}\}_{k=\{0,1,...,d\}},\\
&\{g_l^{L_j(s)}, g_r^{R_j(s)}, g_o^{O_j(s)}\}_{j\in\{0,1,...,m\}},\\
&\{g_l^{\alpha_l L_j(s)}, g_r^{\alpha_r R_j(s)}, g_o^{\alpha_o O_j(s)}, g_l^{\beta L_j(s)} \cdot g_r^{\beta R_j(s)} \cdot g_o^{\beta O_j(s)}\}_{j\in[m]},\\
&g_l^{t(s)}, g_l^{t(s)}, g_r^{t(s)}, g_o^{t(s)}, g_l^{\alpha_l t(s)}, g_r^{\alpha_r t(s)}, g_o^{\alpha_o t(s)}, g_l^{\beta t(s)}, g_r^{\beta t(s)}, g_o^{\beta t(s)}\big)
\end{aligned}
$$

  - 计算验证钥匙（可见）：

$$
\big(g, g_o^{t(s)}, g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o}, g^{\beta\gamma}, g^{\gamma}, g_l^{L_0(s)}, g_r^{R_0(s)}, g_o^{O_0(s)}\big)
$$

- 证明过程：（假定 $\mathcal{P}$ 有一组合法的 $(v_1, ..., v_m)$）
  - 计算目标多项式：

$$
\begin{aligned}
L(x) &= \Sigma_{j=1}^{m} L_j(x) v_j, \\
R(x) &= \Sigma_{j=1}^{m} R_j(x) v_j, \\
O(x) &= \Sigma_{j=1}^{m} O_j(x) v_j, \ \in [m]
\end{aligned}
$$

  - 随机选取偏移系数 $\delta_l, \delta_r, \delta_o$
  - 计算目标多项式的带偏移量的加密值：

$$
\begin{aligned}
g_l^{L(s)} &= (g_l^{t(s)})^{\delta_l} \Pi_{j=1}^{m} (g_l^{L_j(s)}) v_j, \\
g_l^{R(s)} &= (g_r^{t(s)})^{\delta_r} \Pi_{j=1}^{m} (g_r^{R_j(s)}) v_j, \\
g_l^{O(s)} &= (g_o^{t(s)})^{\delta_o} \Pi_{j=1}^{m} (g_o^{O_j(s)}) v_j, \ \in [m]
\end{aligned}
$$

  - 计算目标多项式的偏移量：

$$
\begin{aligned}
g_l^{\alpha_l L(s)} &= (g_l^{\alpha_l t(s)})^{\delta_l} \Pi_{j=1}^{m} (g_l^{\alpha_l L_j(s)}) v_j, \\
g_r^{\alpha_l R(s)} &= (g_r^{\alpha_r t(s)})^{\delta_r} \Pi_{j=1}^{m} (g_r^{\alpha_r R_j(s)}) v_j, \\
g_o^{\alpha_l O(s)} &= (g_l^{\alpha_o t(s)})^{\delta_o} \Pi_{j=1}^{m} (g_o^{\alpha_o O_j(s)}) v_j, j \in [m]
\end{aligned}
$$

  - 计算一致性目标值：

$$
g^{Z(s)} = (g_l^{\beta t(s)})^{\delta_l} (g_r^{\beta t(s)})^{\delta_r} (g_o^{\beta t(s)})^{\delta_o} \Pi_{j=1}^{m} (g_l^{\beta L_j(s)} \cdot g_r^{\beta R_j(s)} \cdot g_o^{\beta O_j(s)})^{v_j}
$$

  - 添加常数项：

$$
\begin{aligned}
\hat{L(x)} &= L(x) + L_0(x) \\
\hat{R(x)} &= R(x) + R_0(x) \\
\hat{O(x)} &= O(x) + O_0(x)
\end{aligned}
$$

  - 计算带偏移量的$h(x)$：

$$
h(x) = \frac{\hat{L}(x)\hat{R}(x) - \hat{O}(x)}{t(x)} + \delta_r L(x) + \delta_l R(x) + \delta_l \delta_r t(x) - \delta_o
$$

- 计算 $h(x)$ 加密值 $g^{h(s)}$（利用 $\{g^{s^k}\}_{k=\{0,1,\ldots,d\}}$）
- 提供证明：

$$(g_l^{L(s)}, g_r^{R(s)}, g_o^{O(s)}, g_l^{\alpha_l L(s)}, g_l^{\alpha_r R(s)}, g_o^{\alpha_l O(s)}, g^{Z(s)}, g^{h(s)})$$

- 验证过程：
  - 假设收到证明的形式化表示为 $(g_l^L, g_r^R, g_o^O, g_l^{L'}, g_r^{R'}, g_o^{O'}, g^Z, g^h)$
  - 偏移量验证：$e(g_l^L, g^{\alpha_l}) = e(g_l^{L'}, g))$，其余类似。
  - 一致性验证：

$$e(g_l^L g_r^R g_o^O, g^{\beta\gamma}) = e(g^Z, g^\gamma)$$

  - 正确性验证：

$$e(g_l^L g_l^{L_0(s)}, g_r^R g_r^{R_0(s)}) = e(g_o^t(s), g^h) e(g_o^O g_o^{O_0(s)}), g)$$

  等价于

$$e(g,g)^{\rho_l \rho_r (L(s)+L_0(s))(R(s)+R_0(s))} = e(g,g)^{\rho_o t(s) h(s) + \rho_o (O(s)+O_0(s))}$$

至此，zkSNARK 核心协议已全部介绍完毕。

1.17 讨论

首先我们讨论 zkSNARK 协议的复杂度问题：注意到虽然证明生成过程的长度是和变量总数线性相关的，但是证明的长度和验证的复杂度都是 $O(1)$ 的，具有很高的实用价值。

我们接下来讨论很多实际计算问题如何表示成算术电路的形式。这里主要介绍笔者个人的理解，有不足之处或者更好的方案欢迎提出。

- 加减乘运算用上文的技巧已经可以构造。对于除法运算，$a \div b = c$，可以考虑先转化成乘法的式子 $b \cdot c = a$ 再进行构造。
- 对于证明一个数 a 是 0 或 1，引入限制条件 $a(a-1) = 0$ 即可。
- 对于范围证明，如证明 $0 \le a \le 15$，引入限制新变量 $a = 8a_3 + 4a_2 + 2a_1 + 1a_0$ 以及限制条件 $a_i (i = 1, 2, 3, 4)$ 属于 0 或 1 即可。对于更大范围的范围证明同理，

先引入二进制拆分变量 a_i，再添加属于 0 或 1 的限制（注意到 2^i 这些项是常数）。

以下我们假设所有变量的取值相比于质数 p 以及群的阶 q 都在一个很小的范围。（一般而言，p, q 为 2^{256} 级别，而变量的取值最高为 2^{32}。且我们暂时忽略变量为负数的情形。）

注意到这里排除了变量为负数的情形，否则它模 q 意义下超出合理取值。对于允许绝对值在一定范围的负数变量的场景，如 $a \in [-10^9, 10^9]$，我们需要先对这些变量做一次变换 $a' = a + 10^9$，然后仍然用 $[0, 2^{32})$ 这个范围来限定所有变量。

基于范围证明也可以做大小比较。比如证明 $a > b$，只需要证明 $a - b$ 在合理取值范围（如 0 至 2^{32}）即可。

- 对于证明 a 被 b 整除，简单的提供 c 满足 $c \cdot b = a$ 可能不行，因为即使 b 不整除 a，在群的阶运算意义下也会存在这样的 c 满足条件（本质上是 $bc \mod q = a$）。

  解决方案对 c 添加一个合理取值范围证明），因为如果 b 不整除 a 且 $bc \mod q = a$，那么 b 或 c 至少有一个大于 $\sqrt{q}$，矛盾。

- 基于以上对于取余或整除函数我们也能构造：比如 $a \mod b = r$，提供变量 k，以及限制条件 $a = bk + r$，k 在合理取值范围，r 在 $[0, b)$ 之间的证明即可。

- 对于指数函数（$f(x) = a^x$），我们先考虑指数是常数的情况，如 $a^8 = b$。

  一个简单的构造方式为，引入新变量及限制方程：$a_2 = a \cdot a, a_3 = a_2 \cdot a, ..., a_7 = a_6 \cdot a, b = a_7 \cdot a$，但这样变量的总数为 8 的线性级别，不具有实用性。

  这里只需引入二分求幂的思想即可简化上述过程：$a_2 = a \cdot a, a_4 = a_2 \cdot a_2, b = a_4 \cdot a_4$，即只需要引入下标为二次幂的变量。这样变量的总数为 8 的对数级别，可以实际使用。

  对于指数不是 2 次幂的情形，只需将其做二进制拆分然后对每一位分别构造即可。

- 对于指数为变量的情形，如 $a^b = c$。如果仍然直接使用二分求幂的构造，则人们可以通过执行的次数来推断出 b 的值，这样就暴露了信息。

  为了不暴露 b 的信息，我们先考虑一种简单情况。如果 b 的取值只能是 0 或 1，那么方程可以用章节1.8中介绍的将判断函数转化成方程的情形：

$$c = ab + (1 - b), b(b - 1) = 0$$

如果范围稍微扩大，b 可以等于 $0,1,2,3$，我们引入若干中间变量也能实现：

$$a_1 = a \cdot a, b = 2b_1 + b_0, c = a_1 b_1 + a b_0, b_0(b_0 - 1) = 0, b_1(b_1 - 1) = 0$$

因为上述方程总是包含了 b_0, b_1 且别人不知道其具体值，他们只能知道 b 是 0 到 4 这个范围内但不知道 b 具体是哪个数。

上式可以推广到一般情形，先将 b 在合理范围内给出二进制拆分 $b_{31}b_{30}...b_0$，同时，构造方程 $a_k = a^{2^k}, k = 0, ..., 31$。这一项可以通过上面介绍的指数为常数的情形来构造，且显然对不同的 k 构造能复用。（具体略）最后构造方程

$$c = a_{31}b_{31} + a_{30}b_{30} + ... + a_0 b_0$$

有了指数函数，我们已经可以对 Bob Jenkins'96 bit Mix Function 这一简化哈希函数做 zkSNARK。

- 对于一般的判断函数情形：

  这里列举一些常见用法：

  –

  $$if \quad a = 5 \quad then \quad \text{表达式 } 1 \quad else \quad \text{表达式 } 2$$

  首先如果限定 a 只能是 0 或 5，构造方程

  $$c = \frac{(5-a)}{5} \text{表达式 } 2 + \text{表达式 } 1$$

  即可。

  如果不对 a 做取值限定，且不能暴露 a 的值呢？（**并且，最重要的，不能暴露 a 是不是等于 0！即，需要隐藏程序是按照哪个分支来执行的信息**）我们下面介绍不做其他限定的判断函数。

  –

  $$if \quad a = 0 \quad then \quad \text{表达式 } 1 \quad else \quad \text{表达式 } 2$$

  需要已知限定条件：a 在合理的取值范围（0 至 2^{32}）。

  我们需要构造一个变量 b，在 $a = 0$ 时取值为 1 否则为 0。

  将 a 做二进制拆分 $a_{31}...a_0$，构造

  $$b = (1 - a_{31})(1 - a_{30})...(1 - a_1)$$

即可。

构造完 b 之后，构造方程

$$(1-b)\text{表达式 } 2 + b\text{表达式 } 1$$

– 对于允许 a 为负数情形，且需要隐藏正负号怎么处理呢?（注意到虽然我们修改了原始数据不能为负数，但出现在判断条件里的可能是中间变量，如 $a-b$，在判断相等时需要使用。）

需要已知限定条件：a 的绝对值在合理范围内（这是合理的假设，因为 a, b 都正常取值时 $a-b$ 绝对值不会超过合理范围）。

解决方案：强行还原出 a 的绝对值：引入变量及限制 $c = a \cdot a, b \cdot b = c$，$b$ 在合理范围内（$b \in [0, 2^{32})$）。对 b 按照上面方案（判断条件 $a = 0$）构造证明。

Pinocchio（参考资料 [1]）给出了更为巧妙的构造。上述 $b = (a! = 0)?1 : 0$ 的取值可以仅由两个方程决定：

$$a(1-b) = 0, aM - b = 0$$

其中 M 为 $\mathcal{P}$ 提供的新变量。显然，如果 $a \neq 0$ 则 b 一定为 1，且 M 能够置为 $M = 1/a$ 使上式成立。反之如果 $a = 0$ 则 b 也一定为 0，M 可以为任何数。

–

$$if \;\; a > 0 \;\; then \;\; \text{表达式 } 1 \;\; else \;\; \text{表达式 } 2$$

需要已知限定条件：a 的绝对值在合理范围内。

注意到我们仍然不能暴露 a 是否大于 0。笔者能想到的一个解决方案如下：

(如上）构造 $c = a \cdot a, b \cdot b = c$，$b$ 在合理范围内（$b \in [0, 2^{32})$），即 $b = |a|$

(如上) 构造 $c_1 = (a! = b)?1 : 0$，对应 $a < 0$ 的情形，$c_2 = (-a! = b)?1 : 0$，对应 $a > 0$ 的情形。（注意到此时只有 $\pm a = b$ 两种情形）

注意到，此时还有 $a = 0$ 的情形，对应 c_1, c_2 都等于 0。我们再引入 $c_3 = (a = 0)?1 : 0$

构造 $d = c_2\text{表达式 } 1 + c_1\text{表达式 } 2 + c_3\text{表达式 } 2$

至此大部分判断函数已经解决。

- 对于程序常用的其他语句如 $while, for$ 等，需要视具体场景构造。但一个需要注意的重点是循环的次数一定要能够隐藏。一个通常的解决方案为根据合理取

值范围将循环次拉满。

至此我们分析了大量实际场景如何化为算术电路。值得一提的是，上述方案虽然通用但是远不是最优解决方案。最优方案永远是针对实际问题来做相应设计与优化。**同样** zkSNARK **作为一个较为通用的解决方案能满足大部分的需求，但最好的方案不一定局限于使用** zkSNARK。**根据不同场景针对性设计的零知识证明/可信计算方案永远是最佳的选择**。