
# zkPLONK 介绍

ASResearch
YeeZTech

# 目录

# 1 ZkPLONK 相关介绍

ZkPLONK，全称 Permutations(P) over Lagrange-bases(L) for Oecumenical(O) Noninteractive(N) Arguments(A) of Knowledge(K)，是于 2019 年提出的零知识证明体系，是基于 2018 年 sonic 的改进。在此之前还有 zkSNARK，zkSTARK，Bulletproof 等。ZkPLONK 的最大特点是，可信第三方构造证明/验证钥匙所需要的 common refenrence string(CRS) 是通用的（该性质叫做 universal），即可信第三方只需要生成一个 CRS，即可用于不同的零知识证明任务（更准确来说是针对不同的电路，见 zkSNARK 介绍。当然这里需要电路的最大规模等参数固定）。这大大提高了零知识证明系统的实用性。

本文简要介绍 zkPLONK 实现 univeral CRS 的核心思想，以及相比于 zkSNARK 其有何本质不同。

## 1.1 参考资料

[1] V 神博客：https://vitalik.ca/general/2019/09/22/plonk.html

[2] 原始论文："PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge", Ariel Gabizon, Zachary J. Williamson and Oana Ciobotaru.

## 1.2 问题描述和 zkSNARK 回顾

zkPLONK 与 zkSNARK 解决的都是给方程赋值一组解的问题，例如下面这个方程组。

$$
\begin{aligned}
2a + b &= c \\
c + 3d &= e \\
4a + 5e &= f
\end{aligned}
$$

$\mathcal{P}$ 需要向 $\mathcal{V}$ 证明他知道一组满足上述方程组的变量赋值（可选的零知识性质：$\mathcal{P}$ 给出的证明不会暴露具体的变量值。但本文不做讨论）。

我们先来回顾 zkSNARK 的做法：上述方程组一共三个方程，包含 6 个变量。

- zkSNARK 把每一个方程对应于多项式在某一点的取值，如 $x = 1, 2, 3$。这一点和 zkPLONK 类似（后面介绍）。这样三个方程相等等价于一个多项式在 1,2,3 这三个点取值为 0，亦即被多项式 $Z(x) = (x-1)(x-2)(x-3)$ 整除。

- 这个多项式如何构造呢? 因为要处理同一变量出现在不同方程中的情况，zkSNARK 的解决方法为，将每个方程视作同样的结构：包含所有变量的系数，例如，第一个方程左边对应的向量为 (2,1,0,0,0,0)。

- 这样，zkPLONK 给出了 6 个所谓"系数"多项式 $L_i(x), i = 1, 2, 3, 4, 5, 6$，其中 $L_i(j), j = 1, 2, 3$ 的取值为第 $i$ 个变量在第 $j$ 个方程的系数。然后问题转化为对这 6 个多项式进行线性组合，线性组合赋值的系数就是 6 个变量的值：$L(x) = aL_1(x) + ... + fL_6(x)$，使得结果多项式能被 $Z(x)$ 整除（事实上，是左多项式乘以右多项式减去出多项式被 $Z(x)$ 整除，这里我们省略细节）。

为了实现上述目标，我们发现针对方程特定的（非通用）CRS 对 zkSNARK 是必须的：我们需要确保结果多项式是给定多项式的线性组合。特别是当方程数大于变量数时，一个任选的 $L(x)$ 显然不一定能表达成 $L_i(x)$ 的线性组合。

为了确保这一性质（$\mathcal{P}$ 给出的 $L(x)$ 是给定多项式的线性组合得到）可验证，zkSNARK 采取的方式是引入偏移量 $\alpha$，即给出 $g^{L_i(x)}$ 和 $g^{\alpha L_i(x)}$，并要求 $\mathcal{P}$ 提供 $g^{L(x)}$ 和 $g^{\alpha L(x)}$。之后 $\mathcal{V}$ 通过 pairing 验证 $\mathcal{P}$ 给出结果满足偏移量设定。其原理在于，如果 $\mathcal{P}$ 能成功给出某个满足偏移量方程的数对，那么这个数对一定是根据已经公开的满足偏移量方程的数对进行线性组合得出。

咋看之下，zkSNARK 恰好完美的通过这个偏移量的性质解决了上述验证线性组合的问题，确实非常巧妙。然而其一个弊端是，有可信第三方提供的带偏移量的"具体"多项式，即 $\alpha L_i(x)$ 等，是必须的：第三方不能直接直接给出 $\alpha$，否则偏移量方程可以被任意破解。所以，zkSNARK 相当于限定了 CRS 必须是针对具体的 $L_i(x)$ 给出，但对于不同的任务，其对应的方程是不同的，所需的 $L_i(x)$ 也不同。这就造成了 zkSNARK 在 setup 上的局限性。

## 1.3 zkPLONK 方案

zkPLONK 采用了完全不同的构造方案。还是如上章同样的方程，zkPLONK 采用的做法关键点如下

- 首先，zkPLONK 要求每个方程左边包含的变量数目只能为 2 个，右边为 1 个（事实上，zkPLONK 还可以让每个方程左边包含一个乘积项。这里本文省略细节）。对于含多个变量的情况，可引入中间变量化为多个方程来表示。（注意到，

即使对于上述每个方程的变量只有两个的情况，zkSNARK 所需向量维度仍然为 6 个，包含所有变量，即用到的 $L_i(x)$ 数目为 6，对此情形并没有简化问题）。

- 其次，zkPLONK 给出的公共多项式，这里用 $P(x)$ 表示，代表的是三个方程的第一个变量对应的系数，而不管这个变量是什么。（相对应的，zkSNARK 用的公共多项式是三个方程中 $a$ 这个变量对应的系数）例如上面这个例子，$P(1) = 2, P(2) = 1, P(3) = 4$。

- 给出公共多项式后，$\mathcal{P}$ 要给出的也是一个多项式，$f(x)$，他在三个给定点的取值对应三个方程中第一个变量的取值，即 $f(1) = a, f(2) = c, f(3) = a$。需要达到的目标为：结果多项式（包含 $f(x)P(x)$ 等）能被 $Z(x)$ 整除。（事实上，结果多项式是 $P(x)f(x) + Q(x)g(x) - O(x)h(x)$，其中 $Q(x), O(x)$ 分别为方程的第二个变量和方程右边的变量的系数多项式，$g(x), h(x)$ 分别为方程的第二个变量和方程右边的变量的取值多项式。这里我们省略细节）。

我们不能让 $\mathcal{P}$ 直接给出 $f(x)$ 的所有系数，一方面其交互复杂度过高——我们希望证明的长度为常数级别，另一方面我们希望 $\mathcal{P}$（可选择的）不暴露变量复制的信息——零知识证明。和 zkSNARK 的思想类似，我们让 $\mathcal{P}$ 给出和 $f(x)$ 有关的一个数（可能不止一个，总之是常数规模），而不是完整的 $f(x)$，且这个数能实现验证功能。

首先，我们介绍 zkPLONK 的 CRS 的形式：和 zkSNARK 类似，可信第三方随机生成一个 $s$，然后把 $E[1], E[s], E[s^2], ..., E[s^d]$ 作为 CRS 给出，其中 $E[x] = g^x$。(recall: 同态加密）

这就是 zkPLONK 的 CRS，很明显它是通用的——它不包含任何没有偏移量，更不用针对公开多项式 $P(x)$ 等来生成。

接下来我们要介绍一个新概念，叫 polynomial commitment。本质而言，是一种通过 CRS 将需要给出的多项式浓缩成一个数的算法。

**$f(x)$ 的 commitment 为 $E[f(s)]$，可通过 CRS 算出。**

$\mathcal{P}$ 为了完成证明任务，需要提供两个 commitment，一个针对要给出的 $f(x)$，即 $E[f(s)]$，另外一个针对商多项式，$t(x) = f(x)P(x)/Z(x)$[1]，即 $E[t(s)]$。

那么，有了这两个 commitment 之后，$\mathcal{V}$ 如何验证呢？他也可以通过 CRS 算出 $E[Z(s)], E[P(s)]$ 的值，因为这些多项式都是公开的，然后根据 $e(E[t(s)], E[Z(s)]) = e(E[f(s)], E[P(s)])$ 来验证。

根据所谓“随机数相等即相等”的原则（见 zkSNARK 那篇），上述过程的确是难以伪造的，即 $\mathcal{P}$ 在不知道满足条件的 $f(x), t(x)$ 的情况下难以给出能通过验证的证

[1]为了方便起见我们就假设结果多项式为 $f(x)P(x)$，而省略第二个变量和方程右边变量对应的多项式

明。但现在的问题是，我们能认为 $s$ 是“真随机的”吗?

zkPLONK 系统的一个很大的不同在于，它所用的 CRS 是通用的，即在没有更新整个系统的情况下所有人都用的是同一个 $s$ 来给出 commitment。这样带来的问题是，$\mathcal{P}$ 可能自己不知道 $f(x)$ 长什么样，但是他可以查找并使用“别人用过的”commitment 去验证。而且，如果每个任务都使用同一个随机数来验证，也与随机数的本质相违背。（这是我能想到的唯一解释）而 zkSNARK 不会有这个问题：因为它是对每个任务都生成一组独特的 CRS，故每次可以产生不同的随机数。

所以，为了实现 zkPLONK 的验证，我们需要每次使用一个不同的随机数，再来判断“多项式在这个数的取值相等”。zkPLONK 假设在 $\mathcal{P}$ 给出 commitment 之后，再由 $\mathcal{V}$ 给出另外一个随机数 $r$，然后 $\mathcal{P}$ 给出 $f(r)$ 的值（注意到这里没有用加密）。这个时候，$\mathcal{V}$ 可以直接算出 $P(r), Z(r)$ 的值（不需要 CRS），然后 $\mathcal{V}$ 自己算出 $t(r) = f(r)P(r)/Z(r)$，即 $t(r)$ 应该等于的值。

那么，既然 $\mathcal{V}$ 什么都能算出来，他还需要做什么验证呢? 问题的关键在于，如何保证 $\mathcal{P}$ 给出的函数在 $r$ 点的取值是正确的。

这时，就需要和 $\mathcal{P}$ 和前面给出的 commitment $E[f(s)], E[t(s)]$ 进行联动了。一般而言，和 commitment 有关的一个对应概念叫“open”，而 polynomial commitment 的 open 的作用就是证明 commit 的多项式在某个点的取值是正确的。即 $\mathcal{P}$ 通过对 commitment $E[f(s)]$ 给出一个”Open”，证明函数 $f(x)$ 在 $r$ 点的取值就是他给出的 $f_r = f(r)$。

Polynomial commitment 用到的一个事实为，如果 $f(r) = f_r$，那么多项式 $f(x) - f_r$ 必被多项式 $x - r$ 整除。这时 $\mathcal{P}$ 可以通过 CRS 计算出 $w(x) = \frac{f(x)-f_r}{x-r}$ 在 $s$ 处的加密值 $g[w(s)]$ 并发送出去。

所以，所谓的 open 就是对多项式 $w(x)$ 的一个 commitment $E[w(s)]$。一旦给出，$\mathcal{V}$ 可以通过 $e(E[w(s)], E[s]/E[r]) = e(E[f(s)]/E[f_r], E[1])$ 来验证（即验证等式 $\frac{f(s)-f_r}{s-r} = w(s)$）。一旦验证通过则可以说明 $f(r)$ 的取值高概率是 $f_r$。否则，$\mathcal{P}$ 无法给出整多项式 $w(s)$，故无法通过 CRS 给出满足上述关系的 $E[w(s)]$。

注: commitment 的 open 一定要求 $r$ 足够随机。同时上述过程虽然是交互的——需要由 $\mathcal{V}$ 给出随机数 $r$，但我们可以使用 Fiat-Shamir 过程将其转化为非交互的，即令 $r$ 等于某些 transcript 的 hash 值即可，注意到这里的 transcript 一定要包含 $\mathcal{P}$ 之前给出的 commitment $E[f(s)], E[t(s)]$ 等以保证“时序性”。

所以，$\mathcal{P}$ 的证明包括了两个 commitment $E[f(s)], E[t(s)]$，两个分别针对他们的 open，以及一个取值 $f_r$。至于多项式 $t(x)$ 在 $r$ 的取值 $t_r$ 就不用给了，因为 $\mathcal{V}$ 可以自己算出来（即使给了 $\mathcal{V}$ 也要验证），然后 $\mathcal{V}$ 只需通过上述过程验证两个 open 均合法即可。

注：实际实现中可以将多个待验证的 open rollup 到一起，只需一次验证即可，大大降低验证复杂度。具体见原始论文。

## 1.4 增加排列限制条件

上述算法还忽略了一个重要的限制条件：$\mathcal{P}$ 在构造 $f(x)$ 时，有些取值对应的是同一个变量！例如在上面那个方程中，必须要有 $f(1) = f(3)$（都等于变量 $a$ 的值）。但上述算法在对不满足此条件的 $f(x)$ 也能验证通过。

zkPLONK 通过引入额外的限制条件来证明 $f(x)$ 在某些取值下函数值相等。它用到如下事实：

如果函数 $f(x)$ 在某些取值 $x \in T$ 上相等（$T$ 可以有多个），令 $\sigma : [n] \to [n]$ 为一个排列，这个排列定义为（包含所有的）$T$ 上的置换。例如，假设 $f(1) = f(3)$，则定义 $\sigma = (3, 2, 1)$。$\sigma$ 这一排列是完全公开的信息，可通过方程组的形式获取。

基于这个定义，我们不难发现下面的等式成立：

$$\Pi_{i=1}^{n}\big(f(i) + \beta i + \gamma\big) = \Pi_{i=1}^{n}\big(f(\sigma(i)) + \beta i + \gamma\big) \tag{1}$$

其中 $\beta, \gamma$ 为随机给定。因为对于 $i \in T$ 有 $f(i) = f(\sigma(i))$。

那么，其逆命题成不成立呢？事实上，有结论表明，如果 (1) 对于足够随机的 $\beta, \gamma$ 成立，那么以高概率有 $f(x)$ 在 $x \in T$ 上相等。

所以，$\mathcal{P}$ 同时需要给出一个证明保证 (1) 成立，且可以认为 $\mathcal{P}$ 难以构造出在 $f(x)$ 在 $x \in T$ 上不等的且 (1) 成立。这就是 zkPLONK 中 P——permutation 的由来：证明函数经过排列变换后函数值相同。

在实际实现中，zkPLONK 巧妙的将 (1) 成立的问题转化为某两个特定多项式被 $Z(x)$ 整除的问题，这样可以用同样的 commitment 的技巧解决，且能和前面的验证 rollup 到一起来完成！这里我们不介绍细节。同时，zkPLONK 在定义固定点多项式 $Z(x)$ 时也用了一定技巧，用有限域单位根 $\omega, \omega^2, ..., \omega^n$ 来代替取点 $1, 2, 3, ..., n$，这样可以用 FFT 等手段加速计算过程。参见原文。

## 1.5 总结

至此，我们介绍了 zkPLONK 的基本思想以及和 zkSNARK 的对比。它能实现通用的 trusted setup，且证明长度为 $O(1)$，具有很高的实用性。其依赖的假设需要引入抗碰撞 hash 函数（用于 FS 过程），这方面要求比 zkSNARK 更多。总之我们认为

zkPLONK 具有很高的实用价值，相信以后会被广泛利用。