理解 Plonk（六）：实现 Zero Knowledge

在前文的 Plonk 协议中，所有的多项式承诺都没有混入额外的随机数进行保护，因此当一个未被随机化的多 项式承诺 $f(X)$ 经过一次或者多次 Open，会泄露 $f(X)$ 自身的信息，这会限制协议在需要隐私保护的 场景中应用。

考虑一个 3 次多项式 $f(X)$，只要它在四个不同的点上 Open ，多项式就可以通过 Lagrange 插值来复原。 然而即使一个次数超过一百万的多项式，哪怕被打开一次也会泄漏关于原多项式的部分信息。

为了实现 Zero Knowledge 性质的 Plonk，我们需要在多项式中加入足够多的随机因子，确保在多项式 打开 k 次之后，仍然不会泄漏原多项式的信息，保证没有知识泄漏。

Plonk 协议的大致流程为：Prover 构造多项式，然后发送多项式的承诺给 Verifier。然后 Verfier 挑战两个随机挑战点 $X = \zeta$ 与 $X = \omega \cdot \zeta$，其中 ω 为 子群 H 的生成元。下面是 Prover 需要构造的多项式列表：

- Witness 多项式： $w_a(X), w_b(X), w_c(X)$
- 置换累乘多项式： $z(X)$
- 商多项式： $t_{low}(X)$， $t_{mid}(X)$， $t_{high}(X)$

其中三个 Witness 多项式要在 $X = \zeta$ 这一个点处打开，置换累乘多项式 $z(X)$ 要在 $X = \zeta$， $X = \omega \cdot \zeta$ 两个点处打开，而三个商多项式则不需要被打开。

Prover 要混入两类随机因子，第一类是保护承诺本身，满足信息隐藏 Hiding，一个承诺一般只需要混入一个随机数即可； 第二类是保护多项式承诺在打开之后仍然保证原多项式信息不会泄漏。如果多项式打开的次数越多（假设每次打开的位置都不同），Prover 就要混入越多的随机因子。

第一类的随机因子，也可以用多项式承诺方案来实现，比如 Bulletproof-IPA，或者 KZG10-with-Hiding，这些多项式承诺方案本身已经支持 Hiding 。如果 Plonk 后端采用的是朴素的 KZG10，那么就需要在 Plonk 协议层面增加足够的随机因子，不仅保证承诺自身的 Hiding 性质，还要保护承诺的打开。

下面我们介绍两个不同的混入随机因子方案实现 Zero Knowledge 的方法。第一个方法比较经典，是为多项式加上一个盲化（Blinding）用途的多项式，GWC19 论文[3]（或其它学术论文）中正是采用的这种方法。而第二个方法是在向量的对齐填充空间里面填入随机数，再插值产生多项式的，这是工程实现中的常见方法。

方法一：Blinding 多项式

我们先看 Witness 多项式 $w_a(X)$，它是由下面的等式计算：

$$w_a(X) = w_{a,0}L_0(X) + w_{a,1}L_1(X) + w_{a,2}L_2(X) + \cdots + w_{a,n-1}L_{n-1}(X)$$

我们假设 $n \leq N$，其中 $N = |H|$。

在 Plonk 协议中，Prover 需要计算 $w_a(\zeta)$ 的取值，其中 ζ 为 Verifier 给出的随机挑战点。

如果我们直接鲁莽地在 $w_a(X)$ 中混入随机数 b_0，比如 $w'_a(X) = w_a(X) + b_0$，那么 $w'_a(X)$ 可能就不再满足算术约束：

$$q_L(X)w'_a(X) + q_R(X)w_b(X) + q_M(X)w_a(X)w_b(X) - q_O(X)w_c(X) + q_C(X$$

而且也无法满足置换约束。

如果要让随机化后的多项式 $w'_a(X)$ 满足「算术约束」和「置换约束」，那么我们可以考虑在乘法子群 H 之外增加一些随机的点，这样可以让随机化后的多项式 $w'_a(X)$ 在 H 整个乘法子群上的取值仍然与 $w_a(X)$ 完全相等，但是整个多项式却已经被随机化了。所谓的在 H 上的取值相等，就是保证随机化后的多项式仍然可以被 $z_H(X)$ 整除。下面是随机化多项式的构造：

$$w'_a(X) = (b_1X + b_0) \cdot z_H(X) + w_a(X)$$

这里 $b_1X + b_0$ 为 Blinding 多项式，包含两个随机因子 (b_0, b_1)，它们恰好是自变量的不同次数的系数，这样可以保证线性不相关。换个方式理解，只有对这个 Blinding 多项式打开两次以上，才可以计算出所有的随机因子。如果只打开一次，Blinding 多项式会被消耗掉一个随机因子，还剩下一个起作用的随机因子。

简单检查下，我们可以发现新定义的 $w'_a(X)$ 符合要求，能满足算术约束。同时因为 $w'_a(X) = w_a(X), \forall x \in H$，因此 $w'_a(X)$ 也一定满足置换关系。

这里 $w'_a(X)$ 被混入了两个随机因子，其中一个随机因子可以保护 $[w'_a(x)]$ 被打开一次，另一个随机因子用来实现承诺 $[w'_a(x)]$ 本身的信息隐藏。

考虑下置换累乘多项式 $z(X)$，假如多项式承诺 $[z(X)]$ 被打开两次的话，那么就需要混入三个随机因子，构造一个次数为 2 的 Blinder 多项式，$b_0 + b_1X + b_2X^2$，然后混入到 $z(X)$ 中：

$$z'(X) = (b_0 + b_1X + b_2X^2) \cdot z_H(X) + z(X)$$

最后考虑商多项式 $t_{low}(X)$， $t_{mid}(X)$， $t_{high}(X)$，由于他们不需要在任何点打开，因此只要加上随机因子即可，不过这几个商多项式有额外的要求，即他们三个需要一起能拼出真正的商多项式 $t(X)$:

$$t(X) = t_{low}(X) + t_{mid}(X) \cdot X^N + t_{high}(X) \cdot X^{2N}$$

我们可以采用下面的方式，为每一个多项式分片混入一个随机因子，并且保证他们拼起来之后仍然等于 $t(X)$:

$$\begin{aligned}
t'_{low}(X) &= t_{low}(X) + b_0 X^N \\
t'_{mid}(X) &= t_{mid}(X) - b_0 + b_1 X^N \\
t'_{high}(X) &= t_{high}(X) - b_1
\end{aligned}$$

容易检验:

$$\begin{aligned}
&t'_{low}(X) + t'_{mid}(X) \cdot X^N + t'_{high}(X) \cdot X^{2N} \\
&= t_{low}(X) + b_0 X^N + (t_{mid}(X) - b_0 + b_1 X^N) \cdot X^N + (t_{high}(X) - b_1) \cdot X^{2n} \\
&= t_{low}(X) + t_{mid}(X) \cdot X^N + t_{high}(X) \cdot X^{2N} \\
&= t(X)
\end{aligned}$$

同理，如果 $t(X)$ 的次数达到了 $4N$，那么就需要三个随机数给四个 $t(X)$ 分段加上随机数，实现 Hiding。

这个方法存在一个问题，就是 Blinding 多项式的次数会超过 N ，这里 $N = |H|$。因为 $z_H(X)$ 的次数为 N，因此 $(b_1 X + b_0) \cdot z_H(X)$ 次数为 $N + 1$。如果 Plonk 后端采用的是 Bulletproof-IPA 这类的多项式承诺，承诺会要求多项式的次数按 2^k 对齐，这样盲化之后的多项式的次数刚刚超出 N，只能对齐到 $2N$。一些 Plonk 变种协议可能会把 Witness table 的列数增加，稍稍超出的多项式次数会使 $t(X)$ 的计算在一个更大的子群上完成。

方法二：随机因子对齐

下面介绍的第二种方法不会推高多项式的次数。考虑到 H 子群的大小 N 是按 2^k 对齐，在实际电路中，一般情况下需要把 Witness Table 的长度对齐到 N，为了对齐，需要把空余的空间用零填满。

那么这里可以用随机数来代替零填充对齐空间，好处是这些随机数可以保护表中的其它正常数据。

Daniel Lubarov 按照这个思路给出了第二种随机数填充实现 Zero-Knowledge 性质的办法[1]。

对于商多项式，因为方法一不会推高他们的次数，因此我们下面只考虑剩下的两类多项式：

- Witness 多项式： $w_a(X), w_b(X), w_c(X)$
- 置换累乘多项式： $z(X)$

先看第一类多项式，以 $w_a(X)$ 为例，它编码了 $w_{a,i}$ 向量。如果本身向量长度不足 N，一般情况下是用零补齐，我们现在可以考虑让 Prover 额外用两个随机数补齐，这样做的效果和方法一的 Blinding 多项式完全一样。 如下所示：

$$w'_a(X) = w_a(X) + (b_0 \cdot L_{N-2}(X) + b_1 \cdot L_{N-1}(X))$$

其中 $b(X) = b_0 \cdot L_{N-2}(X) + b_1 \cdot L_{N-1}(X)$ 也可以看成是利用 Lagrange Basis 产生的 Blinding 多项式。这里假设 $\{w_{a,i}\}$ 的长度为 $N-2$， (b_0, b_1) 为两个随机数。假设 $w_a(X)$ 的系数为固定值，那么当 $w'_a(X)$ 被打开两次之后， $b(X) = b_0 \cdot L_{N-2}(X) + b_1 \cdot L_{N-1}(X)$ 的系数即可被求解，从而失去随机化的能力。因此，$w'_a(X)$ 只能承受一次安全的打开操作（假设协议基于 Non-hiding 的多项式承诺）。

对于置换累乘多项式 $z(X)$，则需要在累乘向量 $\vec{z}$ 的尾部引入随机值。考虑下 $\vec{z}$ 的计算方式：

$$z_{i+1} = z_i \cdot \frac{(w_a(X) + \beta \cdot X + \gamma)(w_b(X) + \beta \cdot k_1 X + \gamma)(w_a(X) + \beta \cdot k_2 X}{(w_a(X) + \beta \cdot \sigma_a(X) + \gamma)(w_b(X) + \beta \cdot \sigma_b(X) + \gamma)(w_a(X) + \beta \cdot \sigma}$$

列出所有的 z_i 的计算如下：

i	H_i	z_i
0	$\omega^0 = 1$	1
1	ω^1	$1 \cdot \frac{f_0}{g_0}$
2	ω^2	$\frac{f_0}{g_0} \cdot \frac{f_1}{g_1}$
3	ω^3	$\frac{f_0 f_1}{g_0 g_1} \cdot \frac{f_2}{g_2}$
$\vdots$		$\vdots$
$N-2$	ω^{N-2}	$\frac{f_0 f_1 \cdots f_{N-4}}{g_0 g_1 \cdots g_{N-4}} \cdot \frac{f_{N-3}}{g_{N-3}}$
$N-1$	ω^{N-1}	$\frac{f_0 f_1 \cdots f_{N-3}}{g_0 g_1 \cdots g_{N-3}} \cdot \frac{f_{N-2}}{g_{N-2}}$
N	$\omega^N = 1$	$\frac{f_0 f_1 \cdots f_{N-1}}{g_0 g_1 \cdots g_{N-1}} = 1$

假如我们想设置 z_{N-1} 为随机值，我们需要让 $w_{a,N-1}$ 和 $w_{a,N-2}$ 这两个元素设置一个 Copy Constraint，并填上同一个随机数 ρ_1。如果 $w_{b,N-1}$ 和 $w_{b,N-2}$ 设置为零，那么

$$\frac{f_{N-2}}{g_{N-2}} = \frac{(\rho_1 + \beta \cdot \omega^{N-2} + \gamma)}{(\rho_1 + \beta \cdot \omega^{N-1} + \gamma)}$$

又因为

$$z_{N-1} = z_{N-2} \cdot \frac{f_{N-2}}{g_{N-2}}$$

那么 z_{N-1} 的概率分布与 ρ_1 相同。这样我们通过把 Witness Table 的最后两行用来填入随机数 ρ_1，并且设置一个 Copy Constraint 来随机化 z_{N-1}。如果要再引入一个随机数 ρ_2，一种方法是我们再征用 Witness table 的两行，$i = N-4, N-3$，可以让 z_{N-4} 随机化。或者我们节省下空间，利用 $w_{b,N-3}$ 与 $w_{b,N-2}$ 来构造一个随机数 ρ_2 的 Copy Constraint。同理，我们可以再用两行 $i = N-4, N-3$ 来引入 ρ_3。这样，我们总共征用了四行，引入了三个随机数 ρ_1, ρ_2, ρ_3：

i	w_a	w_b	w_c
0	$\cdots$	$\cdots$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-4$	ρ_3	0	0
$N-3$	ρ_3	ρ_2	0
$N-2$	ρ_1	ρ_2	0
$N-1$	ρ_1	0	0

i	σ_a	σ_b	σ_c
0	$\cdots$	$\cdots$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N-4$	ω^{-3}	$k_1\omega^{-4}$	$k_2\omega^{-4}$
$N-3$	ω^{-4}	$k_1\omega^{-2}$	$k_2\omega^{-3}$
$N-2$	ω^{-1}	$k_1\omega^{-3}$	$k_2\omega^{-2}$
$N-1$	ω^{-2}	$k_1\omega^{-1}$	$k_2\omega^{-1}$

最后我们推导一下 $z_{N-3}, z_{N-2}, z_{N-1}$，请注意 $z_{N-4} = 1$，因为前面的 Permutation 项都已经消完。

$$z_{N-3} = \frac{(\rho_3 + \beta \cdot \omega^{N-4} + \gamma)}{(\rho_3 + \beta \cdot \omega^{N-3} + \gamma)}$$

$$
\begin{aligned}
z_{N-2} &= z_{N-3} \cdot \frac{(\rho_3 + \beta \cdot \omega^{N-3} + \gamma)(\rho_2 + \beta \cdot k_1\omega^{N-3} + \gamma)}{(\rho_3 + \beta \cdot \omega^{N-4} + \gamma)(\rho_2 + \beta \cdot k_1\omega^{N-2} + \gamma)} \\
&= \frac{(\rho_2 + \beta \cdot k_1\omega^{N-3} + \gamma)}{(\rho_2 + \beta \cdot k_1\omega^{N-2} + \gamma)}
\end{aligned}
$$

$$
\begin{aligned}
z_{N-1} &= z_{N-2} \cdot \frac{(\rho_1 + \beta \cdot \omega^{N-2} + \gamma)(\rho_2 + \beta \cdot k_1\omega^{N-2} + \gamma)}{(\rho_1 + \beta \cdot \omega^{N-1} + \gamma)(\rho_2 + \beta \cdot k_1\omega^{N-3} + \gamma)} \\
&= \frac{(\rho_1 + \beta \cdot \omega^{N-2} + \gamma)}{(\rho_1 + \beta \cdot \omega^{N-1} + \gamma)}
\end{aligned}
$$

于是 $z_{N-3}, z_{N-2}, z_{N-1}$ 中各自包含了一个随机数。请注意这个方法需要在 Witness table 中留有足够的 padding 空间，并且 $z(X)$ 的盲化因子不能与 $w_a(X), w_b(X), w_c(X)$ 的重复，那么总共需要留出 6 排空间，并且把 $w'_a()$ 盲化因子提前到第 $N-5$ 与 $N-6$ 排：

$$
\begin{aligned}
w'_a(X) &= w_a(X) + (b_0 \cdot L_{N-6}(X) + b_1 \cdot L_{N-5}(X)) \\
w'_b(X) &= w_b(X) + (b_2 \cdot L_{N-6}(X) + b_3 \cdot L_{N-5}(X)) \\
w'_c(X) &= w_c(X) + (b_4 \cdot L_{N-6}(X) + b_5 \cdot L_{N-5}(X))
\end{aligned}
$$

满足 Hiding 性质的 KZG10

在 Daniel Lubarov 的 Blog 中讲述的方案是基于带有 Hiding 性质的多项式承诺 IPA（Inner product argument）。因此在 $w_a(X), w_b(X), w_c(X)$ 中只需要混入一个随机因子， $z(X)$ 中只混入两个随机因子。

但是我们也可以选择一个带有 Hiding 性质的 KZG10 承诺方案，这样也可以按照 Halo2 方式混入较少的随机数实现 Zero-knowledge。

这个方案参考了 Marlin 论文[2]的 Appendix B.3，基于 AGM 模型的 KZG10-with-hiding。

在 Setup 阶段，我们需要产生两倍长的 srs：

$$
\boldsymbol{srs} = \begin{pmatrix} [1]_1, & [\chi]_1, & [\chi^2]_1, & \cdots, & [\chi^D]_1, \\ [\rho]_1, & [\rho\chi]_1, & [\rho\chi^2]_1, & \cdots, & [\rho\chi^D]_1, \end{pmatrix}, ([1]_1, [\rho]_1, [1]_1, [\chi]_2)
$$

如果我们要承诺一个多项式 $f(X) = f_0 + f_1X + \cdots + f_{n-1}X^{n-1}$，那么需要额外产生一个次数相同的 Blinder 多项式：

$$r(X) = r_0 + r_1 X + \cdots + r_{n-1} X^{n-1}$$

然后计算承诺：

$$C_f = \sum_{i=0}^{n-1} f_i \cdot [\chi^i]_1 + \sum_{i=0}^{n-1} r_i \cdot [\rho\chi^i]_1 = [f(\chi) + \rho \cdot r(\chi)]_1$$

如果我们要在 $X = \zeta$ 处打开一个多项式承诺，先计算 $y = f(\zeta)$，还要计算盲化多项式 $r(X)$ 在 $X = \zeta$ 的求值， $y' = r(\zeta)$，然后产生这两个多项式的求值证明：

$$q(X) = f(X) + \rho \cdot r(X) = \frac{f(X) - f(\zeta)}{X - \zeta} + \rho \cdot \frac{r(X) - r(\zeta)}{X - \zeta}$$

$$\pi_{f(\zeta)} = ([q_(X)]_1, y')$$

检查求值证明的方式如下：

$$e(C_f - y \cdot [1]_1 - y' \cdot [\rho]_1, [1]_2) \stackrel{?}{=} e([q(X)]_1, [\chi]_2 - \zeta \cdot [1]_2)$$

我们可以看到为了实现 Hiding，计算承诺和打开承诺的成本会加倍。如果我们限定多项式只能被打开一次（或者有限次），那么我们可以采用更低次数的盲化多项式 $r(X)$。假如我们只考虑多项式最多被打开一次的情况，那么 $r(X)$ 只需要是一个一次多项式，同时也可以减少 srs 的尺寸。

$$r(X) = r_0 + r_1 X$$

最后请注意的是，仅有实现 Hiding 的多项式承诺不足以实现 Plonk 的 Zero-knowledge，仍然需要在 Plonk 协议层面混入足够的随机的盲化因子。

参考文献

- [1] Adding zero knowledge to Plonk-Halo https://mirprotocol.org/blog/Adding-zero-knowledge-to-Plonk-Halo
- [2] Chiesa, Alessandro, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. "Marlin: Preprocessing zkSNARKs with universal and updatable SRS." In Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39, pp. 738-768. Springer International Publishing, 2020. https://eprint.iacr.org/2019/1047.

- [3] Gabizon, Ariel, Zachary J. Williamson, and Oana Ciobotaru. “Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge.” *Cryptology ePrint Archive* (2019).
