Notes on Hyrax-PCS

这篇文章简要介绍 Hyrax-PCS 的原理，其安全假设为 Discrete Log。它的主要思想是证明一个 Inner Product，并且采用了递归折叠的思路逐步把长度为 n 的两个向量折叠成两个长度为 $n/2$ 的两个向量，并且把内积计算归约到长度只有一半的向量的内积计算。递归折叠的思路主要来自于 [BCC+16] 与 [BBB+18]，其主要问题是 Verifier 的计算复杂度为 $O(n)$。为了使 Hyrax 能达到 zkSNARK 的要求，Hyrax 将向量重新整理成 $\sqrt{n} \times \sqrt{n}$ 的矩阵，然后把内积计算归约到向量长度为 $\sqrt{n}$ 的向量的内积计算。这样导致 Verifier 的计算复杂度可以优化到 $O(\sqrt{n})$。同时，经过 [BCC+16] 的优化，Hyrax 的 Proof 尺寸（通讯复杂度）也优化到 $O(\log n)$。

1. MLE 多项式的求值证明

不管 MLE 多项式是 Coefficients 形式还是 Evaluation 形式，我们都可以通过「内积证明」 Inner Product Argument 来证明多项式在某些点上的求值。

$$f(X_0, X_1, \ldots, X_{n-1}) = \sum_{i=0}^{2^n-1} a_i \cdot \tilde{eq}(\mathsf{bits}(i), X_0, X_1, \ldots, X_{n-1}) \tag{1}$$

$$f(X_0, X_1, \ldots, X_{n-1}) = \sum_{i=0}^{2^n-1} c_i \cdot X_0^{i_0} \cdot X_1^{i_1} \cdot \ldots \cdot X_{n-1}^{i_{n-1}}, \quad \text{where } \mathsf{bits}(i) = (i_0, i_1, \ldots, i_{n-1}) \tag{2}$$

如果我们有一个内积证明协议，那么我们可以轻松构造一个 MLE 多项式的求值证明

公共输入

1. $\vec{a}$ 的承诺： $C_a = \mathsf{cm}(a_0, a_1, \ldots, a_{2^n-1})$
2. $\vec{u} = (u_0, u_1, \ldots, u_{n-1})$
3. $v = \tilde{f}(u_0, u_1, \ldots, u_{n-1})$

Witness

1. $\vec{a}$

内积协议

Prover 计算向量 $\vec{e}$，长度为 2^n，

$$\begin{aligned} e_0 &= \tilde{eq}(\mathsf{bits}(0), u_0, u_1, \ldots, u_{n-1}) \\ e_1 &= \tilde{eq}(\mathsf{bits}(1), u_0, u_1, \ldots, u_{n-1}) \\ &\cdots \\ e_{2^n-1} &= \tilde{eq}(\mathsf{bits}(2^n-1), u_0, u_1, \ldots, u_{n-1}) \end{aligned} \tag{3}$$

Prover 与 Verifier 通过一个 Inner Product Argument 协议来证明 $\vec{a}$ 与 $\vec{e}$ 的内积等于 v。下面我们介绍一个简单的内积证明，它证明两个隐藏向量的内积值等于一个公开值。

2. Mini-IPA

我们先从最简单的情况入手，假如 Prover 拥有两个向量 $\vec{a}$ 与 $\vec{b}$，满足 $\langle \vec{a}, \vec{b} \rangle = c$（注意：这里的 c 是一个 *公开值* ）。

证明目标

Prover 拥有知识 $(\vec{a}, \vec{b})$（ 两个向量长度相等，记为 m，则总共有 $2m$ 个 witness），并且 $\langle \vec{a}, \vec{b} \rangle = c$

公开参数

我们计算向量的 Pedersen Commitment，需要选取一组随机的群元素 $g_1, g_2, g_3, \ldots, g_m, h \in \mathbb{G}$ 。

公共输入

1. 内积计算结果 c
2. 向量 $\vec{a}$ 的承诺 $C_a = \mathsf{cm}(\vec{a}; \rho_a)$,
3. 向量 $\vec{b}$ 的承诺 $C_b = \mathsf{cm}(\vec{b}; \rho_b)$

Witnesses

1. $(\vec{a}, \rho_a)$
2. $(\vec{b}, \rho_b)$

协议基本思路

Prover 引入两个「致盲向量」Blinder factors， $\vec{r}$ 与 $\vec{s}$，这两个向量通过一个挑战数 μ （来自 Verifier）来拍扁成一个向量，然

$$\vec{a}' = \vec{r} + \mu \cdot \vec{a} \qquad \vec{b}' = \vec{s} + \mu \cdot \vec{b} \tag{4}$$

然后计算$\vec{a}' \cdot \vec{b}'$ 内积（或点乘）。

$$\langle \vec{a}', \vec{b}' \rangle = (\vec{r} + \mu \cdot \vec{a})(\vec{s} + \mu \cdot \vec{b}) = \mu^2(\langle \vec{a}, \vec{b} \rangle) + \mu(\langle \vec{a}, \vec{s} \rangle + \langle \vec{b}, \vec{r} \rangle) + \langle \vec{r}, \vec{s} \rangle \tag{5}$$

观察下 $\vec{a}'$ 与 $\vec{b}'$，我们会发现这两个向量中都有 μ 项，拍扁向量的内积运算后得到一个关于 μ 的**二次多项式**，其中 μ^2 项的「系数」恰好是向量$\vec{a}$ 与 $\vec{b}$ 的内积（应该等于 c），常数项 $\vec{r} \cdot \vec{s}$ 恰好是两个「致盲向量」的内积。不过 μ 的系数看起来似乎有些 *混乱*。 我们可以先不管混乱的 μ 项的系数，只关注 μ^2 项的系数。根据 Schwartz-Zippel 定理，只要 Prover 能成功答复 Verifier 的挑战，那么多项式所有项的系数都必须（极大概率地）正确无误。

我们让 Prover 在协议第一步中不仅需要对「致盲向量」承诺，同时还要对内积展开后的（关于 μ 的）多项式系数进行承诺。然后在第三步中，Prover 只要发送两个拍扁向量 $\vec{a}'$ 与 $\vec{b}'$ 就刚刚好，Verifier 先验证 a' 是否能打开 A'，再验证 b' 是否能打开 B'， 最后 Verifier 验证 $\vec{a}' \cdot \vec{b}'$ 能否打开承诺 C' 。我们看看协议具体怎么定义：

协议

Round 1

Prover 发送两个「致盲向量」 $\vec{r}$，$\vec{s}$ 的承诺 C_r 与 C_s；还要发送多项式系数承诺 C_0 与 C_1：

$$\begin{aligned} C_r &= \mathsf{cm}(\vec{r}; \rho_r) \\ C_s &= \mathsf{cm}(\vec{s}; \rho_s) \\ C_0 &= \mathsf{cm}(\langle \vec{r}, \vec{s} \rangle; \rho_0) \\ C_1 &= \mathsf{cm}(\langle \vec{a}, \vec{s} \rangle + \langle \vec{b}, \vec{r} \rangle; \rho_1) \end{aligned} \tag{6}$$

Round 2

1. Verifier 回复一个挑战数 μ
2. Prover 发送两个拍扁向量 $\vec{a}'$ ， $\vec{b}'$，三个混入 μ 的随机数 ρ'_a， ρ'_b， ρ'_{ab}

$$\begin{aligned} \vec{a}' &= \vec{r} + \mu \cdot \vec{a} \\ \vec{b}' &= \vec{s} + \mu \cdot \vec{b} \end{aligned} \tag{7}$$

$$
\begin{aligned}
\rho'_a &= \rho_r + \mu \cdot \rho_a \\
\rho'_b &= \rho_s + \mu \cdot \rho_b \\
\rho'_{ab} &= \rho_0 + \mu \cdot \rho_1
\end{aligned} \tag{8}
$$

验证

Verifier 在群 $\mathbb{G}$ 上同态验证： $\vec{a}'$ 与 $\vec{b}'$，以及他们的内积

$$
\begin{aligned}
\mathsf{cm}(\vec{a}'; \rho'_a) &\overset{?}{=} C_r + \mu \cdot C_a \\
\mathsf{cm}(\vec{b}'; \rho'_b) &\overset{?}{=} C_s + \mu \cdot C_b
\end{aligned} \tag{9}
$$

$$
\mathsf{cm}(\langle \vec{a}', \vec{b}' \rangle; \rho'_{ab}) \overset{?}{=} \mu^2 \cdot \mathsf{cm}(c; 0) + \mu \cdot C_1 + C_0 \tag{10}
$$

这个协议的最大问题是 Verifier 的计算复杂度为 $O(n)$，因为 Verifier 要计算 $\langle \vec{a}', \vec{b}' \rangle$。另外其中 $\vec{b}$ 为隐藏的信息，但是对于 MLE 的 Evaluation 证明协议来说，$\vec{e}$ （由 $\vec{u}$ 计算得来）是一个公开值，所以我们需要调整下协议。

3. Square-root inner product argument

Hyrax 论文提出了一个简单直接的思路，将 Verifier 的计算复杂度降到 $O(\sqrt{n})$，Sublinear Verifier 是 zkSNARK 的一个基本要求。我们仍然只考虑 $\tilde{f}$ 的 Coefficients 形式，即 $\vec{c}$ 是 $\tilde{f}$ 的系数。

我们假设 $n = 4$，那么 $\vec{a}$ 长度为 16，然后我们可以把这个向量排成一个矩阵：

$$
\begin{bmatrix}
c_0 & c_1 & c_2 & c_3 \\
c_4 & c_5 & c_6 & c_7 \\
c_8 & c_9 & c_{10} & c_{11} \\
c_{12} & c_{13} & c_{14} & c_{15}
\end{bmatrix} \tag{11}
$$

以 $\vec{a}$ 为表示的 MLE 多项式可以表示为下面的形式：

$$
\tilde{f}(X_0, X_1, X_2, X_3) = \begin{bmatrix} 1 & X_2 & X_3 & X_2X_3 \end{bmatrix}
\begin{bmatrix}
c_0 & c_1 & c_2 & c_3 \\
c_4 & c_5 & c_6 & c_7 \\
c_8 & c_9 & c_{10} & c_{11} \\
c_{12} & c_{13} & c_{14} & c_{15}
\end{bmatrix}
\begin{bmatrix} 1 \\ X_0 \\ X_1 \\ X_0X_1 \end{bmatrix} \tag{12}
$$

这个矩阵的计算结果如下：

$$
\tilde{f}(X_0, X_1, X_2, X_3) = c_0 + c_1X_0 + c_2X_1 + c_3X_2 + c_4X_0X_1 + \cdots + c_{14}X_1X_2X_3 + c_{15}X_0X_1X_2X_3 \tag{13}
$$

我们先把 $\vec{u}$ 拆成两个短向量：

$$
\vec{u} = (u_0, u_1, u_2, u_3) = (u_0, u_1) \parallel (u_2, u_3) \tag{14}
$$

那么 $\tilde{f}(u_0, u_1, u_2, u_3)$ 可以表示为：

$$
\tilde{f}(u_0, u_1, u_2, u_3) = \begin{bmatrix} 1 & u_2 & u_3 & u_2u_3 \end{bmatrix}
\begin{bmatrix}
c_0 & c_1 & c_2 & c_3 \\
c_4 & c_5 & c_6 & c_7 \\
c_8 & c_9 & c_{10} & c_{11} \\
c_{12} & c_{13} & c_{14} & c_{15}
\end{bmatrix}
\begin{bmatrix} 1 \\ u_0 \\ u_1 \\ u_0u_1 \end{bmatrix} \tag{15}
$$

然后，我们把 $\vec{c}$ 组成的矩阵按行进行承诺的计算，得到 C_0, C_1, C_2, C_3，

$$
\begin{aligned}
C_0 &= \mathsf{cm}(c_0, c_1, c_2, c_3; \rho_0) \\
C_1 &= \mathsf{cm}(c_4, c_5, c_6, c_7; \rho_1) \\
C_2 &= \mathsf{cm}(c_8, c_9, c_{10}, c_{11}; \rho_2) \\
C_3 &= \mathsf{cm}(c_{12}, c_{13}, c_{14}, c_{15}; \rho_3)
\end{aligned} \tag{16}
$$

然后我们可以使用 $(1, u_2, u_3, u_2u_3)$ 与 (C_0, C_1, C_2, C_3) 进行内积运算，得到 C^*:

$$C^* = C_0 + u_2C_1 + u_3C_2 + u_2u_3C_3 \tag{17}$$

那么 C^* 可以看成是 M_c 矩阵的列向量与 $(1, u_2, u_3, u_2u_3)$ 的内积，记为 $\vec{d} = (d_0, d_1, d_2, d_3)$，

$$\begin{aligned} d_0 &= c_0 + c_4 \cdot u_0 + c_8 \cdot u_2 + c_{12} \cdot u_2u_0 \\ d_1 &= c_1 + c_5 \cdot u_0 + c_9 \cdot u_2 + c_{13} \cdot u_2u_0 \\ d_2 &= c_2 + c_6 \cdot u_0 + c_{10} \cdot u_2 + c_{14} \cdot u_2u_0 \\ d_3 &= c_3 + c_7 \cdot u_0 + c_{11} \cdot u_2 + c_{15} \cdot u_2u_0 \end{aligned} \tag{18}$$

可以轻易验证:

$$C^* = \mathsf{cm}(d_0, d_1, d_2, d_3; \rho^*) \tag{19}$$

这里 $\rho^* = \rho_0 + \rho_1u_2 + \rho_2u_3 + \rho_3u_2u_3$ 。

采用这个思路，我们构造一个简单的 MLE 多项式承诺方案。

公共输入

1. $\vec{a}$ 的承诺： $C_a = \mathsf{cm}(a_0, a_1, \ldots, a_{2^n-1})$
2. $\vec{u} = (u_0, u_1, \ldots, u_{n-1})$
3. $v = \tilde{f}(u_0, u_1, \ldots, u_{n-1})$

Witness

1. $\vec{a}$

承诺

1. Prover 把 $\vec{a}$ 重排成一个矩阵 $M_a \in \mathbb{F}_p^{l \times h}$:

$$M_a = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{l-1} \\ a_l & a_{l+1} & a_{l+2} & \cdots & a_{2l-1} \\ a_{2l} & a_{2l+1} & a_{2l+2} & \cdots & a_{3l-1} \\ a_{(h-1)l} & a_{(h-1)l+1} & a_{(h-1)l+2} & \cdots & a_{hl-1} \end{bmatrix} \tag{20}$$

2. Prover 按行计算承诺 $C_0, C_1, \ldots, C_{h-1}$

$$\begin{aligned} C_0 &= \mathsf{cm}(a_0, a_1, \ldots, a_{l-1}; \rho_0) \\ C_1 &= \mathsf{cm}(a_l, a_{l+1}, \ldots, a_{2l-1}; \rho_1) \\ C_2 &= \mathsf{cm}(a_{2l}, a_{2l+1}, \ldots, a_{3l-1}; \rho_2) \\ \cdots &= \quad \cdots \\ C_{h-1} &= \mathsf{cm}(a_{(h-1)l}, a_{(h-1)l+1}, \ldots, a_{hl-1}; \rho_{h-1}) \end{aligned} \tag{21}$$

Evaluation证明协议

1. Prover 与 Verifier 把 $\vec{u}$ 拆成两个短向量，分别用 $\vec{u}_L$ 与 $\vec{u}_R$ 表示:

$$\begin{aligned} \vec{u}_L &= (u_0, u_1, \ldots, u_{\log(l)-1}) \\ \vec{u}_R &= (u_{\log(l)}, u_{\log(l)+1}, \ldots, u_{n-1}) \end{aligned} \tag{22}$$

显然

$$\vec{u} = \vec{u}_L \parallel \vec{u}_R \tag{23}$$

Round 1

1. Prover 计算 $\vec{e} = (e_0, e_1, \ldots, e_{h-1})$，长度为 h：

$$\begin{aligned} e_0 &= \tilde{eq}(\mathsf{bits}(0), \vec{u}_R) \\ e_1 &= \tilde{eq}(\mathsf{bits}(1), \vec{u}_R) \\ \cdots &= \quad \cdots \\ e_{h-1} &= \tilde{eq}(\mathsf{bits}(h-1), \vec{u}_R) \end{aligned} \tag{24}$$

2. Prover 计算 $\vec{e}$ 与 M_a 的矩阵乘法，得到一个新的向量 $\vec{b}$，长度为 l

$$\begin{aligned} b_0 &= \langle \vec{e}, (a_0, a_l, \ldots, a_{(h-1)l}) \rangle \\ b_1 &= \langle \vec{e}, (a_1, a_{l+1}, \ldots, a_{(h-1)l+1}) \rangle \\ \cdots &= \quad \cdots \\ b_{l-1} &= \langle \vec{e}, (a_{l-1}, a_{l+l-1}, \ldots, a_{(h-1)l+l-1}) \rangle \end{aligned} \tag{25}$$

2. Prover 计算 $\vec{b}$ 的承诺 C^*

$$C^* = \mathsf{cm}(\vec{b}; \rho^*) \tag{26}$$

这里 $\rho^* = \langle \vec{e}, (\rho_0, \rho_1, \ldots, \rho_{h-1}) \rangle$

Round 2.

Prover 和 Verifier 进行一个 Inner Product Argument 协议，完成 $\vec{b}$ 与 $\vec{d}$ 的内积证明。

$$\begin{aligned} d_0 &= \tilde{eq}(\mathsf{bits}(0), \vec{u}_L) \\ d_1 &= \tilde{eq}(\mathsf{bits}(1), \vec{u}_L) \\ \cdots &= \quad \cdots \\ d_{h-1} &= \tilde{eq}(\mathsf{bits}(h-1), \vec{u}_L) \end{aligned} \tag{27}$$

1. Prover 先抽样一个随机数向量 $\vec{r}$，用来保护 $\vec{b}$ 的信息，然后计算它的承诺：

$$C_r = \mathsf{cm}(\vec{r}; \rho_r) \tag{28}$$

2. Prover 计算 $\vec{r}$ 与 $\vec{b}$ 的内积，得到 v

$$s = \vec{r} \cdot \vec{d} \tag{29}$$

$$C_0 = \mathsf{cm}(s; \rho_s) \tag{30}$$

$$C_1 = \mathsf{cm}(0; \rho_t) \tag{31}$$

Round 3.

1. Verifier 发送一个随机数 μ
2. Prover 计算并发送下面的值：

$$\vec{z_b} = \vec{r} + \mu \cdot \vec{b} \tag{32}$$

$$z_\rho = \rho_r + \mu \cdot \rho^* \tag{33}$$

$$z_t = \rho_s + \mu^{-1} \cdot \rho_t \tag{34}$$

Verification

Verifier 验证：

$$C_r + \mu \cdot C^* \stackrel{?}{=} \mathsf{cm}(\vec{z}_b; z_\rho) \tag{35}$$

$$C_0 + \mu^{-1} \cdot C_1 + \mu \cdot \mathsf{cm}(v; 0) \stackrel{?}{=} \mathsf{cm}(\langle \vec{z}_b, \vec{d} \rangle; z_t) \tag{36}$$

4. Bulletproofs 的优化

在上文中的「内积证明」协议（Mini-IPA）的 Round-2，由于 Prover 要发送长度为 l 的向量（$\vec{z}_b$），因此整体协议的通讯量为 $O(l)$。如果向量比较长，那么最后的证明尺寸就会比较大。J. Bootle 等人 在[BCC+16] 论文中提出来了一个非常有趣的思路，用递归的方式来逐步地折叠证明，实现**证明尺寸的压缩**。

假设有一个长度为 4 的向量 $\vec{a} = (a_1, a_2, a_3, a_4)$，我们可以对半把它切分成两个向量 $\vec{a}_1 = (a_1, a_2)$ 与$\vec{a}_2 = (a_3, a_4)$，然后把它们纵向迭在一起，形成一个矩阵：

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \tag{37}$$

然后我们对这个 2×2 的矩阵（用一个随机数向量来辅助）进行「拍扁」操作：

$$(x, x^{-1}) \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} = (a_1 x + a_3 x^{-1}, \quad a_2 x + a_4 x^{-1}) = \vec{a}' \tag{38}$$

方法如上式，我们把矩阵左乘上一个随机数向量 (x, x^{-1})，然后得到一个长度为 2 的向量 $\vec{a}'$。我们可以把这个动作看成是一种特殊的纵向拍扁。这个技巧和前文中的纵向拍扁略有不同，没有采用朴素的拍扁向量 (x^0, x^1) 。我们把这个动作叫做「折叠」。

我们能注意到，折叠后的向量 $\vec{a}'$ 的长度仅为 $\vec{a}$ 的一半。这样递归地做下去，就通过 Verifier 不断地发送挑战数，Prover 不停地递归折叠，最终把向量折叠成一个素数长度的向量。但是，如果允许我们为向量附加一些冗余值，使得向量的长度按 2^k 对齐， 那么经过 k 次折叠后，我们可以把向量折叠到一个长度仅为 1 的数。这就相当于对一个矩阵进行**横向拍扁**。

这个初步思路面临**第一个问题**，就是当向量切成两半之后，原始向量 $\vec{a}$ 的承诺 A 好像就没办法用了。这样当 Prover 进行一次折叠动作之后，如何让 Verifier 得到折叠之后的向量承诺呢？换个思路，Pedersen 承诺从某种意义上看，也是一种内积，即「待承诺的向量」与「基向量」的「内积」：

$$\mathsf{cm}_{\vec{G}}(\vec{a}) = a_1 G_1 + a_2 G_2 + \cdots + a_m G_m \tag{39}$$

接下来的技巧很关键，我们把基向量 $\vec{G}$ 做同样的切分，然后同样折叠，但是用一个**不同的**「挑战向量」：

$$(x^{-1}, x) \begin{bmatrix} G_1 & G_2 \\ G_3 & G_4 \end{bmatrix} = (G_1 x^{-1} + G_3 x, \quad G_2 x^{-1} + G_4 x) = \vec{G}' \tag{40}$$

请注意上面两个挑战向量 (x, x^{-1}) 与 (x^{-1}, x) 看起来是*对称的*。这么做的目的，是为了凑一个特殊的常数项。当把新的向量 $\vec{a}'$ 与 $\vec{G}'$ 在一起计算内积时，其常数项正好就是原始向量 $\vec{a}$ 与 $\vec{G}$ 的内积。但是对于非常数项（ x^2 与 x^{-2} 项）的系数，是一些看着*比较混乱的值*。

我们展开计算下，首先把 $\vec{a}$ 切成两个子向量 $\vec{a} = (\vec{a}_1, \vec{a}_2)$，$\vec{G} = (\vec{G}_1, \vec{G}_2)$，然后分别进行折叠，得到 $\vec{a}' = \vec{a}_1 x + \vec{a}_2 x^{-1}$ ，$\vec{G}' = \vec{G}_1 x^{-1} + \vec{G}_2 x$，

$$\vec{a}' \cdot \vec{G}' = (1, 1) \left(\begin{bmatrix} \vec{a}_1 \cdot \vec{G}_1 & \vec{a}_1 \cdot \vec{G}_2 \\ \vec{a}_2 \cdot \vec{G}_1 & \vec{a}_2 \cdot \vec{G}_2 \end{bmatrix} \circ \begin{bmatrix} 1 & x^2 \\ x^{-2} & 1 \end{bmatrix} \right) \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{41}$$

上面式子右边是一个关于 x 的多项式，于是我们可以清晰地看到 x^2 与 x^{-2} 的系数。

接下来我们来解决上面提出的问题：如何让 Verifier 能算出折叠后向量 $\vec{a}'$ 的承诺：

$$A' = A + (\vec{a}_1 \cdot \vec{G}_2)\, x^2 + (\vec{a}_2 \cdot \vec{G}_1)\, x^{-2} \tag{42}$$

我们让 Verifier 计算 $A' = \vec{a}' \cdot \vec{G}'$，很显然 Verifier 可以自行计算出 $\vec{G}'$，然后 Verifier 就可以根据 Prover 发送的 $\vec{a}'$ 来计算 A'。

承诺 A' 是一个关于 x 的多项式，常数项是原始向量的承诺 A，而 x^2 项与 x^{-2} 的系数可以让 Prover 计算并发送。那么 x^1 与 x^{-1} 项的系数呢？他们恰好为零。因为我们巧妙采用了两个对称的挑战向量分别对 $\vec{a}$ 与 $\vec{G}$ 进行折叠操作，于是正好消去 x 与 x^{-1} 两个项的系数，同时让常数项等于原始向量的内积。

新的承诺 A' 很容易计算 $A' = A + Lx^2 + Rx^{-2}$，其中 $L = (\vec{a}_1 \cdot \vec{G}_2)$， $R = (\vec{a}_2 \cdot \vec{G}_1)$。承诺 L 与 R 看上去是将两个子向量交叉内积的结果。这样一来，问题得到了解决，当 Prover 需要发送一个长度为 m 的向量 $\vec{v}$ 时，Prover 可以选择发送将其**对折并拍扁后的向量** $\vec{v}'$，它的长度只有 $m/2$。然后 Verifier 同样可以通过验证打开（Open） 对折拍扁后的向量，从而保证原始向量的正确性。

递归折叠也有相应的代价，除了Verifier 要额外计算 $\vec{G}'$，Prover 也要额外计算 L 与 R，两者也要增加一轮的交互。我们可以通过下面一个递归折叠的内积证明协议来看下递归折叠的完整过程。

公开参数

$\vec{G}, \vec{H} \in \mathbb{G}^n;\ U, T \in \mathbb{G}$

公共输入

$P = \vec{a}\vec{G} + \vec{b}\vec{H} + cU + \rho T$

Witnesses：$\vec{a}, \vec{b} \in \mathbb{Z}_p^n;\ c \in \mathbb{Z}_p$

第一步（承诺步）： Prover 发送两个遮罩向量的承诺 P_0 与 C_1，C_2：

1. $P_0 = \vec{a}_0\vec{G} + \vec{b}_0\vec{H} + \rho_0 T$
2. $C_1 = \vec{a}_0 \cdot \vec{b}_0 \cdot U + \rho_1 T$
3. $C_2 = (\vec{a} \cdot \vec{b}_0 + \vec{a}_0 \cdot \vec{b}) \cdot U + \rho_2 T$

第二步（挑战）： Verifier 回复一个挑战数 z

第三步：Prover 计算 z 拍扁后的向量 $\vec{a}'$，$\vec{b}'$，ρ'，并发送 ρ_c

1. $\vec{a}' = \vec{a} + z\vec{a}_0$
2. $\vec{b}' = \vec{b} + z\vec{b}_0$
3. $\rho' = \rho + z\rho_0$
4. $\rho_c = z^2\rho_1 + z\rho_2$

验证

Verifier 可以计算得到 P'

1. $P' = P + zP_0 + zC_2 + z^2C_1 - \rho_c T$

于是，Prover 和 Verifier 可以接着运行 rIPA 协议，证明 $\vec{a}' \cdot \vec{b}' \overset{?}{=} c'$，这三个值的承诺合并为 $P' = \vec{a}'\vec{G} + \vec{b}'\vec{H} + (\vec{a}' \cdot \vec{b}')U + \rho' T$。

5. 完整协议

下面是结合了递归折叠与 Square-root IPA 的完整协议，这个协议支持 Zero-Knowledge 性质。如果不需要的 ZK 性质，直接去除 H 部分关于 ρ 相关的值即可。

公开参数

- $G_0, G_1, G_2, \ldots, G_{2^n-1}, H, U \in \mathbb{G}$ 。

计算承诺

1. Prover 把 $\vec{a}$ 重排成一个矩阵 $M_a \in \mathbb{F}_p^{l \times h}$:

$$M_a = \begin{bmatrix} a_0 & a_1 & a_2 & \cdots & a_{l-1} \\ a_l & a_{l+1} & a_{l+2} & \cdots & a_{2l-1} \\ a_{2l} & a_{2l+1} & a_{2l+2} & \cdots & a_{3l-1} \\ a_{(h-1)l} & a_{(h-1)l+1} & a_{(h-1)l+2} & \cdots & a_{hl-1} \end{bmatrix} \tag{43}$$

2. Prover 按行计算承诺 $C_0, C_1, \ldots, C_{h-1}$

$$\begin{aligned} C_0 &= \mathsf{cm}(a_0, a_1, \ldots, a_{l-1}; \rho_0) \\ C_1 &= \mathsf{cm}(a_l, a_{l+1}, \ldots, a_{2l-1}; \rho_1) \\ C_2 &= \mathsf{cm}(a_{2l}, a_{2l+1}, \ldots, a_{3l-1}; \rho_2) \\ \cdots &= \quad \cdots \\ C_{h-1} &= \mathsf{cm}(a_{(h-1)l}, a_{(h-1)l+1}, \ldots, a_{hl-1}; \rho_{h-1}) \end{aligned} \tag{44}$$

这里的 $\mathsf{cm}(\vec{a}; \rho)$ 的定义如下:

$$\mathsf{cm}(\vec{a}; \rho) = \sum_{i=0}^{l-1} a_i G_i + \rho H \tag{45}$$

Evaluation证明协议

公共输入

1. $\vec{a}$ 的承诺: $(C_0, C_1, \ldots, C_{h-1})$
2. $\vec{u} = (u_0, u_1, \ldots, u_{n-1}) = \vec{u}_L \parallel \vec{u}_R$ ，其中 $|\vec{u}_L| = \log(h)$ ， $|\vec{u}_R| = \log(l)$
3. $v = \tilde{f}(u_0, u_1, \ldots, u_{n-1})$

Witness

1. $\vec{a}$
2. $(\rho_0, \rho_1, \ldots, \rho_{h-1})$

证明协议

Round 1

1. Prover 计算 $\vec{e}$:

$$\begin{aligned} e_0 &= \tilde{eq}(\mathsf{bits}(0), \vec{u}_R) \\ e_1 &= \tilde{eq}(\mathsf{bits}(1), \vec{u}_R) \\ \cdots &= \quad \cdots \\ e_{h-1} &= \tilde{eq}(\mathsf{bits}(h-1), \vec{u}_R) \end{aligned} \tag{46}$$

2. Prover 计算 $\vec{e}$ 与 M_a 的矩阵乘法，得到 $\vec{b}$，长度为 l

$$\begin{aligned} b_0 &= \langle \vec{e}, (a_0, a_l, \ldots, a_{(h-1)l}) \rangle \\ b_1 &= \langle \vec{e}, (a_1, a_{l+1}, \ldots, a_{(h-1)l+1}) \rangle \\ \cdots &= \quad \cdots \\ b_{l-1} &= \langle \vec{e}, (a_{l-1}, a_{l+l-1}, \ldots, a_{(h-1)l+l-1}) \rangle \end{aligned} \tag{47}$$

3. Prover 计算 $\vec{b}$ 的承诺 C^*

$$C^* = \mathsf{cm}(\vec{b}; \rho^*) \tag{48}$$

Round 2.

Prover 和 Verifier 进行一个 IPA 协议，完成 $\vec{b}$ 与 $\vec{d}$ 的内积证明，$\vec{d}$ 计算如下：

$$\begin{aligned} d_0 &= \tilde{eq}(\mathsf{bits}(0), \vec{u}_L) \\ d_1 &= \tilde{eq}(\mathsf{bits}(1), \vec{u}_L) \\ \cdots &= \quad \cdots \\ d_{h-1} &= \tilde{eq}(\mathsf{bits}(h-1), \vec{u}_L) \end{aligned} \tag{49}$$

1. Verifier 发送一个随机数 γ
2. Prover 和 Verifier 计算 $U' \in \mathbb{G}$

$$U' = \gamma \cdot U \tag{50}$$

Round 3 (Repeated $i = 0, 1, \ldots, n-1$).

先引入下面的符号，比如 $\vec{b}_L$ 表示 $\vec{b}$ 的前半部分，$\vec{b}_R$ 表示 $\vec{b}$ 的后半部分。

$$\begin{aligned} \vec{b}_L^{(i)} &= (b_0^{(i)}, b_1^{(i)}, \ldots, b_{2^{n-1}-1}^{(i)}) \\ \vec{b}_R^{(i)} &= (b_{2^{n-1}}^{(i)}, b_{2^{n-1}+1}^{(i)}, \ldots, b_{2^n-1}^{(i)}) \\ \vec{d}_L^{(i)} &= (d_0^{(i)}, d_1^{(i)}, \ldots, d_{2^{n-1}-1}^{(i)}) \\ \vec{d}_R^{(i)} &= (d_{2^{n-1}}^{(i)}, d_{2^{n-1}+1}^{(i)}, \ldots, d_{2^n-1}^{(i)}) \\ \vec{G}_L^{(i)} &= (G_0^{(i)}, G_1^{(i)}, \ldots, G_{2^{n-1}-1}^{(i)}) \\ \vec{G}_R^{(i)} &= (G_{2^{n-1}}^{(i)}, G_{2^{n-1}+1}^{(i)}, \ldots, G_{2^n-1}^{(i)}) \end{aligned} \tag{51}$$

注意这里的初始值，$\vec{b}^{(0)} = \vec{b}$，$\vec{d}^{(0)} = \vec{d}$，$\vec{G}_L^{(0)} = \vec{G}_L$，$\vec{G}_R^{(0)} = \vec{G}_R$。

1. Prover 发送 $L^{(i)}$ 与 $R^{(i)}$:

$$\begin{aligned} L^{(i)} &= \mathsf{cm}_{\vec{G}_L^{(i)}}(\vec{b}^{(i)}; \rho_L^{(i)}) + \langle \vec{b}_R^{(i)}, \vec{d}_L^{(i)} \rangle \cdot U' \\ R^{(i)} &= \mathsf{cm}_{\vec{G}_R^{(i)}}(\vec{b}^{(i)}; \rho_R^{(i)}) + \langle \vec{b}_L^{(i)}, \vec{d}_R^{(i)} \rangle \cdot U' \end{aligned} \tag{52}$$

2. Verifier 发送一个随机数 $\mu^{(i)}$,
3. Prover 计算并发送下面的值：

$$\begin{aligned} \vec{b}^{(i+1)} &= \vec{b}_L^i + \mu^{(i)} \cdot \vec{b}_R^i \\ \vec{d}^{(i+1)} &= \vec{d}_L^i + {\mu^{(i)}}^{-1} \cdot \vec{d}_R^i \end{aligned} \tag{53}$$

4. Prover 和 Verifier 计算 $\vec{G}^{(i+1)}$

$$\vec{G}^{(i+1)} = \vec{G}_L^{(i)} + {\mu^{(i)}}^{-1} \cdot \vec{G}_R^{(i)} \tag{54}$$

5. Prover 和 Verifier 递归地进行 Round 3，直到 $i = n-1$
6. Prover 计算

$$\hat{\rho} = \rho^* + \sum_{i=0}^{n-1} \mu^{(i)} \cdot \rho_L^{(i)} + {\mu^{(i)}}^{-1} \cdot \rho_R^{(i)} \tag{55}$$

Round 4.

1. Prover 计算并发送 R，其中 $r, \rho_r \in \mathbb{F}_p$ 为 Prover 随机抽样的随机数

$$R = r \cdot (G^{(n-1)} + b^{(n-1)} \cdot U') + \rho_r \cdot H \tag{56}$$

Round 5.

1. Verifier 发送一个随机数 $\zeta \in \mathbb{F}_p$
2. Prover 计算 z 与 z_r

$$z = r + \zeta \cdot b^{(n-1)} \tag{57}$$

$$z_r = \rho_r + \zeta \cdot \hat{\rho} \tag{58}$$

Verification

1. Verifier 计算 C^* 与 P

$$C^* = d_0 C_0 + d_1 C_1 + \ldots + d_{h-1} C_{h-1} \tag{59}$$

$$P = C^* + \sum_{i=0}^{n-1} \mu^{(i)} L^{(i)} + {\mu^{(i)}}^{-1} R^{(i)} \tag{60}$$

2. Verifier 验证下面的等式是否成立

$$R + \zeta \cdot P \stackrel{?}{=} z \cdot (G^{(n-1)} + b^{(n-1)} \cdot U') + z_r \cdot H \tag{61}$$

References

1. [WTSTW16] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. "Doubly-efficient zkSNARKs without trusted setup." In 2018 IEEE Symposium on Security and Privacy (SP), pp. 926-943. IEEE, 2018. https://eprint.iacr.org/2016/263
2. [BBB+18] Bünz, Benedikt, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short proofs for confidential transactions and more." In 2018 IEEE symposium on security and privacy (SP), pp. 315-334. IEEE, 2018. https://eprint.iacr.org/2017/1066
3. [BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. "Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting." In Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35, pp. 327-357. Springer Berlin Heidelberg, 2016. https://eprint.iacr.org/2016/263