知乎 首发于
区块链和隐私计算

隐私计算关键技术：隐私集合求交（PSI）的性能扩展

甘露
「原本区块链」创始人CTO，隐私计算、区块链和机器学习

已关注

25 人赞同了该文章

更新：本文介绍的PSI方法的Python版完整代码已在Github开源：

delta-mpc/python-psi
github.com/delta-mpc/python-psi

作者：Delta - 开箱即用的区块链隐私计算框架（deltampc.com）

在上一篇文章中，我们介绍了如何使用不经意传输（OT）来构建不经意伪随机函数（OPRF），并使用不经意伪随机函数来构造隐私集合求交（PSI）。

甘露：隐私计算关键技术：隐私集合求交（PSI）原理介绍
74 赞同 · 51 评论 文章

但是，上文介绍的方法，速度很慢，因为在构造不经意伪随机函数时，需要使用 l 次不经意传输（l 是输入数据的长度），进而导致隐私集合求交需要使用 $O(nl)$ 次不经意传输（n 为集合大小）。我们知道，由于不经意传输使用公私钥加密技术，所以不经意传输的速度是很慢的。所以，我们要对上文介绍的方法进行改进，而改进的主要目标，就是减少使用的不经意传输数量。

要做到这一点、我们就需要引入一个新的方法、不经意传输扩展。能够少量（常数次）"慢速"的不

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

知乎 首发于 区块链和隐私计算

这里介绍的不经意传输扩展方法，来自文章[2]

不经意传输扩展的目标，是使用少量"慢速"的基础不经意传输，配合对称加密，来实现大量"快速"的不经意传输。形式化的来说，我们以符号 OT_l^m 来表示进行 m 次不经意传输，每次传输 l 个比特，则不经意传输扩展的定义为：使用 OT_k^k 来实现 OT_l^m，其中 k 是一个比较小的安全参数，且 $m \gg k$。

下面，我们来介绍如何使用 OT_k^k 来实现 OT_l^m。我们首先使用 OT_m^k 来实现 OT_l^m，因为使用 OT_m^k 比使用 OT_k^k 更加简单直观，同时使用 OT_k^k 可以很简单地实现 OT_k^m。之后，我们再简单地介绍如何使用 OT_k^k 实现 OT_k^m。

我们使用 S 来表示 OT_m^k 中的发送者，R 表示 OT_m^k 中的接收者。S 有 m 对输入 $(x_{j,0}, x_{j,1}), 1 \le j \le m$，$R$ 有 m 个选择比特 $r = (r_1, \dots, r_m)$。S 和 R 之间有一个统一的随机函数 $H : [m] \times \{0,1\}^k \to \{0,1\}^l$，即将长度为 k 的比特串随机映射到长度为 l 的比特串。之后的步骤如下：

1. R 先初始化一个随机的比特矩阵 T，矩阵 T 的大小为 $m \times k$，即 m 行 k 列，矩阵的每个元素都是0或1。S 随机初始化 k 个选择比特 $s = (s_1, \dots, s_k)$
2. R 作为发送者，S 作为接收者，执行 OT_m^k。对于第 i 次长度为 m 的不经意传输，R 的输入为 $(t^i, t^i \bigoplus r)$，其中 t^i 表示矩阵 T 的第 i 列，长度为 m；S 的输入为 s_i。当 $s_i = 0$ 时，S 得到 t^i，当 $s_i = 1$ 时，S 得到 $t^i \bigoplus r$。将 S 收到的所有列组合成一个 $m \times k$ 的矩阵，称为 Q
3. 现在，R 作为接收者，S 作为发送者。S 要执行 m 次传输，对于 $1 \le j \le m$，S 发送一对数据 $(y_{j,0}, y_{j,1})$，其中 $y_{j,0} = x_{j,0} \bigoplus H(j, q_j)$，$y_{j,1} = x_{j,1} \bigoplus H(j, q_j \bigoplus s)$，$q_j$ 为矩阵 Q 的第 j 行
4. 对于 $1 \le j \le m$，R 输出 $z_j = y_{j,r_j} \bigoplus H(j, t_j)$，其中 t_j 表示矩阵 T 的第 j 行

现在，我们来证明方法的正确性，即 $z_j = x_{j,r_j}$。

从步骤2中我们可以得知，$s_i = 0$ 时，$q^i = t^i$；$s_i = 1$ 时，$q^i = t^i \bigoplus r$。我们可以将它写成如下形式：

$$q^i = t^i \bigoplus (s_i \cdot r)$$

其中，符号 $\cdot$ 表示按位与运算。那么，对于矩阵 Q 中的第 i 列，第 j 行的元素 q_j^i，有如下等式：

$$q_j^i = t_j^i \bigoplus (s_i \cdot r_j)$$

我们把 s_i 当作选择比特，当 $s_i = 0$ 时，$q_j^i = t_j^i$；当 $s_i = 1$ 时，$q_j^i = t_j^i \bigoplus r_j$。我们换一个角度，把 r_j 当作选择比特，当 $r_j = 0$ 时，$q_j^i = t_j^i$；当 $r_j = 1$ 时，$q_j^i = t_j^i \bigoplus s_i$。这一结果，对于所有 $1 \le i \le k$ 都成立，那么我们可以得到：当 $r_j = 0$ 时，$q_j = t_j$;当 $r_j = 1$ 时，$q_j = t_j \bigoplus s$，它可以写成如下形式

$$q_j = t_j \bigoplus (r_j \cdot s)$$

从上式，我们可以得到

$$t_j = q_j \bigoplus (r_j \cdot s)$$

这时我们来看 z_j。由于 $y_{j,0} = x_{j,0} \bigoplus H(j, q_j)$，$y_{j,1} = x_{j,1} \bigoplus H(j, q_j \bigoplus s)$，那么 $y_{j,r_j} = x_{j,r_j} \bigoplus H(j, q \bigoplus (r_j \cdot s))$，将 y_{j,r_j} 和 t_j 带入到公式 $z_j = y_{j,r_j} \bigoplus H(j, t_j)$ 中，可得 $z_j = y_{j,r_j} \bigoplus H(j, t_j) = x_{j,r_j} \bigoplus H(j, q \bigoplus (r_j \cdot s)) \bigoplus H(j, q \bigoplus (r_j \cdot s)) = x_{j,r_j}$，证毕。

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

输。由于 $1 \leq j \leq m$，因此我们总共构造了 m 个不经意传输。

这种不经意传输所传输的数据 $(q_j, q_j \bigoplus s)$ 是随机的，如果想要传输特定的数据，也就是 $(x_{j,0}, x_{j,1})$，我们可以把 $(q_j, q_j \bigoplus s)$ 当作加密函数的key，用来加密 $(x_{j,0}, x_{j,1})$，得到 $(y_{j,0}, y_{j,1})$，然后接收者 R 就可以使用 t_j，解密出 r_j 对应的 x_{j,r_j}。这里的加密与解密操作是很简单的，使用对称加密即可，这相比公私钥加密要快的多。

由此，我们通过 OT^k_m，外加 $2m$ 次的对称加密，就实现了 OT^m_l。OT^k_m 的开销是 k 次OT，加上 $2m$ 次的对称加密，相比 OT^m_l 的 m 次OT，还是要小很多的。尤其是当 $m \gg k$ 时，这种差距就更加明显。

之前还说到，我们可以进一步使用 OT^k_k 来实现 OT^k_m，缩减开销。从 OT^k_k 到 OT^k_m 是很简单的，步骤如下：

1. 发送者 S 随机初始化 k 对长度为 k 的密钥 $(s_{i,0}, s_{i,1})$
2. 接收者 R 有 k 个选择比特 $r = (r_1, \ldots, r_k)$，通过 OT^k_k，得到 k 个密钥 s_{i,r_i}
3. 对于 $1 \leq i \leq k$，发送者 S 发送 $(y_{i,0}, y_{i,1})$，其中 $y_{i,b} = x_{i,b} \bigoplus G(s_{i,b})$，
   G：$\{0,1\}^k \to \{0,1\}^m$ 为一个伪随机函数
4. 对于 $1 \leq i \leq k$，接收者 R 得到 $z_i = y_{i,r_i} \bigoplus G(s_{i,r_i})$

我们可以看到，主要的思路就是通过 OT^k_k 传输长度为 k 的密钥，作为对称加密的密钥，然后加密长度为 m 的数据，再进行传输。这样做其实并没有减少传输的数据量，因为实际上都要传输长度为 m 的数据给接收方。

更进一步

在有了不经意传输扩展之后，我们可以用它来改进之前的隐私集合求交算法。最简单的想法，就是将之前隐私集合求交算法中所有的不经意传输，都使用不经意传输扩展来替换。这样的确能够提升算法的效率，因为我们将大量“慢速”的不经意传输，替换为了少量的“慢速”不经意传输加大量的对称加密。但是，算法的传输量并没有减少，仍然需要进行 $O(nl)$ 次传输，每次传输2个长度为 k 的比特串。单纯使用不经意传输扩展来替换不经意传输，只是减少了每次传输的计算开销，并没有减少传输上的开销。那么，能不能继续减少传输上的开销呢？答案是可以的。我们需要在不经意传输扩展的基础上更进一步，继续扩展。

观察之前的不经意传输与不经意传输扩展，它们都是 $1-2$ 不经意传输，也就是二选一。基于 $1-2$ 不经意传输的范式，对于一个长为 l 的输入数据，我们需要为它的每一个比特进行一次不经意传输，所以传输量是 $O(l)$。不经意传输除了 $1-2$ 不经意传输，还有 $1-n$ 不经意传输，也就是n选一。如果我们使用 $1-n$ 不经意传输，那么，对于一个长为 l 的输入数据，传输量变为了 $O(l/n)$。如果我们用 $1-n$ 不经意传输来替换 $1-2$ 不经意传输，很明显，隐私集合求交算法的传输量会缩减，但是渐进复杂度依然没变，还是 $O(nl)$，也就是说，传输量依然与输入数据的长度 l 有关。但是，如果我们能实现 $1-\infty$ 的不经意传输呢？那么，隐私集合求交总体的传输量就会缩减到 $O(n)$，因为我们只需要一次 $1-\infty$ 不经意传输，就能实现不经意伪随机函数（OPRF），也就是隐私比较了。

那么，如何构建 $1-\infty$ 不经意传输呢？我们要在刚刚介绍的不经意传输扩展的基础上，实现大量的 $1-\infty$ 不经意传输。

由不经意传输扩展到 $1-\infty$ 不经意传输

现在我们回头看不经意传输扩展，接收方 R 会先随机初始化一个 $m \times k$ 的比特矩阵 T，矩阵 T 的每一列 t^i 都会和选择比特向量 r 进行异或，通过 $1-2$ 不经意传输发送给发送方 S。我们将

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

我们可以发现，T 的第 j 行 t_j，与 U 的第 j 行 u_j 进行异或 $t_j \bigoplus u_j$，得到的都是全零或全一，是零还是一取决于选择 r_j。现在我们假设一个编码函数 $C:\{0,1\}\rightarrow\{0,1\}^k$，$C(b)=b^k$，也就是将输入的比特 b 重复 k 次,即重复编码。那么，我们可以得到 $t_j \bigoplus u_j = C(r_j)$。现在观察发送者 S 通过不经意传输得到的矩阵 Q，我们知道 $q_j = t_j \bigoplus (r_j \cdot s)$，那么，结合编码函数 C，我们可以得到：

$$q_j = t_j \bigoplus (C(r_j) \cdot s)$$

思考一下，如果我们改变编码函数 C，上面的等式是否会成立？很显然，只要 $q^i = t^i$ 或 $q^i = u^i$ 成立，上面的等式就是成立的。那么，我们就可以对编码函数 C 进行改动。

之前的编码函数 $C:\{0,1\}\rightarrow\{0,1\}^k$ 是重复编码，输入是一个比特，我们可以将它改为随机编码，且能接受任意长比特的输入，即：

$$C:\{0,1\}^*\rightarrow\{0,1\}^k$$

那么，现在接收者 R 的选择位 r_j 就不需要限制在一个比特了，可以为一个任意长度的数据，即 $r_j:\{0,1\}^*$。我们知道，接收者 R 现在有比特矩阵 T，发送者 S 有矩阵 Q，每一行 $t_j = q_j \bigoplus (C(r_j)\cdot s)$。对于发送者 S 来说，它可以对任意长度的输入 r' 计算 $q_j \bigoplus (C(r')\cdot s)$，只有当 $r_j = r'$ 时，$t_j = q_j \bigoplus (C(r')\cdot s)$ 才会成立。从不经意传输的角度来看，这就是一种 $1-\infty$ 的不经意传输，因为接收者 R 只能获得 t_j 一个输出，发送者 S 却能对任意长的 r' 计算 $q_j \bigoplus (C(r')\cdot s)$，也就是说，发送者 S 能产生任意多个输出，但是接收者 R 只能知道其中的一个。

实现不经意伪随机函数（OPRF）

那么，我们就可以使用 $1-\infty$ 不经意传输，来实现隐私比较。让 r_j 和 r' 分别为 R 和 S 需要比较的数据，发送者 S 输出 $H(j, q_j \bigoplus (C(r')\cdot s))$，接收者 R 输出 $H(j,t_j)$，H 是一种哈希函数。只需要比较 R 与 S 的输出，就可以实现隐私比较。

从上一篇文章，我们可以得知，隐私比较可以看作不经意伪随机函数（OPRF）。那么，我们就可以通过对不经意传输扩展进行改造，构造出大量不经意伪随机函数。形式化的描述如下：

接收者 R 有 m 个选择字符串 $r=(r_1,\ldots,r_m)$，$r_i\in\{0,1\}^*$，接收者 R 与发送者 S 有一个共同的随机编码函数 $C:\{0,1\}^*\rightarrow\{0,1\}^k$，$C$ 的编码长度为 k，有一个共同的哈希函数 $H:[m]\times\{0,1\}^k\rightarrow\{0,1\}^v$

1. R 先初始化一个 $m\times k$ 随机的比特矩阵 T。S 随机初始化 k 个选择比特 $s=(s_1,\ldots,s_k)$
2. R 构建矩阵 U，$u_j = t_j \bigoplus C(r_j)$，其中 u_j 与 t_j 分别表示 U 与 T 的第 j 行
3. R 作为发送者，S 作为接收者，执行 OT_m^k。对于第 i 次长度为 m 的不经意传输，R 的输入为 (t^i,u^i)，t^i 和 u^i 分别表示矩阵 T 和 U 的第 i 列；S 的输入为 s_i。当 $s_i=0$ 时，S 得到 t^i，当 $s_i=1$ 时，S 得到 u^i。将 S 收到的所有列组合成一个 $m\times k$ 的矩阵，称为 Q
4. 现在，R 作为接收者，S 作为发送者，可以执行 m 次OPRF。对于 $1\le j\le m$，R 输出 $H(j,t_j)$，S 输出 $H(j, q_j \bigoplus (C(r')\cdot s))$，$r'\in\{0,1\}^*$，$r'$ 为 S 选择的任意值。

由此，我们就通过改造不经意传输扩展，实现了大量的不经意伪随机函数。我们只需要进行固定次数（k 次）的 $1-2$ 不经意传输，就能实现 m 次（$m \gg k$）不经意伪随机函数。每次不经意伪随机函数，只需要使用一个哈希函数，进行一次传输即可完成，与输入的长度 l 无关，开销很小。

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

假设接收者 R 是诚实且好奇的，也就是说，R 会正确地执行协议，但是同时它会尽可能地尝试从协议的输出中窥探发送者 S 的信息。假设 S 发给 R 的信息是 $H(q_j \bigoplus (C(r') \cdot s))$，$r' \neq r_j$，$R$ 想要去暴力碰撞 r'，我们可以发现：

$$q_j \bigoplus (C(r') \cdot s) = t_j \bigoplus (C(r_j) \cdot s) \bigoplus (C(r') \cdot s) = t_j \bigoplus ((C(r') \bigoplus C(r_j)) \cdot s)$$

其中，只有 s 对 R 来说是未知的。如果要保证安全，我们需要让 $C(r') \bigoplus C(r_j)$ 的汉明重量大于 κ，这样的情况下，R 需要至少猜测出 s 中的 κ 位，才能完成碰撞攻击。κ 是一个安全参数，κ 的值越大，攻击者就越难完成攻击。一般来说，κ 的值需要至少大于等于128。

我们知道随机编码 C 的输出长度是 k，要使 $C(r') \bigoplus C(r_j)$ 的汉明重量大于 κ，就需要 $k > \kappa$。根据文章[1]的结论，只要 $3\kappa < k < 4\kappa$，就能保证 $C(r') \bigoplus C(r_j)$ 的汉明重量小于 κ 的概率是可以忽略的。具体的证明这里不做介绍，感兴趣的读者，可以自行查看论文。

在隐私集合求交中的应用

回到隐私集合求交上，我们使用这种改进过的不经意伪随机函数，可以大大提升算法的效率。

上一篇文章中，已经介绍了如何使用不经意伪随机函数来实现隐私隐私集合求交。隐私集合求交中，需要使用 $1.2n + s$，也就是 $O(n)$ 次的不经意伪随机函数。现在，改进过的不经意伪随机函数的开销与输入长度无关，是一个均摊的常数。使用改进过的不经意伪随机函数后，隐私集合求交的渐进复杂度也就变为 $O(n)$ 了，只与需要比较的集合大小相关，与集合中元素的长度无关。

从整体上看，隐私集合求交算法只需要进行 k 次 $1-2$ 不经意传输，外加 $O(n)$ 次的对称加密即可完成，其中 k 是个常数。我们可以将这 k 次不经意传输看作算法的初始化阶段，后续阶段称为在线阶段，初始化阶段的执行时间基本是固定的，在线阶段的时间会随着集合大小的增大而增大。 根据文章[1]中的试验结果，当集合大小为 2^{20} 时，在内网环境下（延时0.2ms），使用改进过的不经意伪随机函数的隐私集合求交，在线阶段只需要不到4s时间，离线阶段只需要600ms。与朴素的哈希方法相比，朴素哈希方法的执行时间在700ms左右。可以看出，这种方法的速度是很快的。

结论

综上所述，我们使用固定次数的 $1-2$ 不经意传输，外加对称加密，就可以实现隐私求交算法了。这样实现的隐私求交算法，速度是很快的，即使和朴素的哈希方法相比，也不会很慢。算法的传输量是 $O(n)$，n 是发送者的集合大小，因此更加适合参与双方进行比较的集合大小相差不大的情况。

本文只介绍了算法的大致思路，如果需要详细的了解，推荐阅读原文章[1]。同时，原文章还有对应的开源代码，可以作为参考：

github.com/osu-crypto/B...

另外，本文介绍的PSI方法是一个两方PSI方法，无法用于多方之间计算PSI。本方法的作者在后续的文章中，用同样的思路加以扩展，支持了多方PSI。详情可阅读这篇介绍的文章：

甘露：隐私计算关键技术：多方隐私集合求交（PSI）从原理到实现
22 赞同 · 3 评论 文章

参考文献：

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

on Computer and Communications Security. 2016: 818-829.

[2] Ishai Y, Kilian J, Nissim K, et al. Extending oblivious transfers efficiently[C]//Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2003: 145-161.

DNA 3GTCHEIO

本文经「原本」原创认证，作者一个洋葱，点击"阅读原文"或访问yuanben.io查询【3GTCHEIO】获取授权

编辑于 2021-11-12 20:26

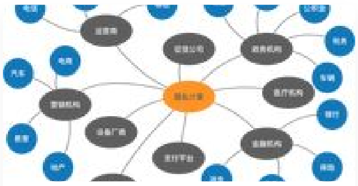
联邦学习 隐私保护 多方安全计算

文章被以下专栏收录

区块链和隐私计算
区块链技术前沿探索。关注隐私计算、区块链性能等方向

推荐阅读

多方安全计算之秘密分享
周辉



隐私计算企业和产品列表-国内篇
开放隐私计... 发表于开放隐私计...

隐私计算实例：详解一个纵向联邦学习的场景和技术实现
甘露 发表于区块链和隐...

隐私计 (MPC)
唔讲粗

23 条评论 切换为时间排序

写下你的评论...

Death 02-23
您好，我看代码实现里base ot的执行次数固定都是128，请问会有安全风险么？我之前看到一个说法是如果安全系数为128，那么k一般要3-4倍的安全系数；如果我要自己改写实现的话，请问需要调整底层base ot的执行次数么？
赞

lin 01-17
为什么：如果我们使用n选1不经意传输，那么，对于一个长为l的输入数据，传输量变为了

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

知乎 首发于 区块链和隐私计算

统一回复一下这个问题，如果是一个1-2的OT，那么对长为l 的数据的每一位都要进行一次1-2的OT， 如果是一个1-2的OT，那么对长位l 的数据，每两位进行一次1-4的OT，推广下去，一个1-n的OT，对长位l 的数据，每logn 位进行一次1-n的OT，所以传输量应该从O(l)变成了O(l/logn)。当实现了1-inf的OT之后，就可以一次传输l 位，传输开销降低到O(1)

1

马萨 回复 缪弘 02-24

您好，1-2 OT 里边的"2"，指的是每次传输的数据对里有两个数据吧？所以是2选1。那对于1-n OT，不就应该变成 每次传输的数据对里有n个数据吗？但是对于长为l位的数据，不还是要传输l次？

赞

小胖头 01-04

如果我们使用1-n不经意传输，那么，对于一个长为l的输入数据，传输量变为了O(l/n)。 请问这是为什么？

赞

小胖头 2021-12-01

"因为我们只需要一次 1-无穷不经意传输，就能实现不经意伪随机函数（OPRF），也就是隐私比较了。"这里是为什么呢？ 1-无穷不经意传输 每次不也只能传输一个数据么？

赞

小胖头 2021-12-01

"如果我们使用1-n不经意传输，那么，对于一个长为l的输入数据，传输量变为了O(l/n)"。请问这是为什么？1-n不经意传输，每次不也只能传输一个数据么？为什么整体的传输量会减少呢？

赞

小胖头 2021-11-29

作者你好，请教一个问题，OTE步骤描述中的第2步："R作为发送者,S作为接收者，执行OT_m^k"，请问这里的OT_m^k具体执行哪一个协议呢？这里第2步应该是实现OT_m^k的一小步吧，这样的话，这里岂不是就套娃了？

赞

小胖头 2021-11-29

OTE下面的第三段开始，OT_k^m和OT_m^k 会混着使用，亲问哪一个是正确的？

赞

缪弘 回复 小胖头 2021-11-29

OT_k^m是正确的

赞

小胖头 回复 缪弘 2021-11-30

我看了原始论文，是OT_m^k啊

赞

Charles 2021-11-24

开源代码是实现了这篇文章的优化吗，还是只是上一篇的实现。

赞

缪弘 回复 Charles 2021-11-29

你指的是GitHub - delta-mpc/python-psi: Private set intersection implemented in

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···

请问一下，OT可以用于多方，不仅仅是两方吗

赞

缪弘 回复 lemon 2021-07-02

OT一般都是两方之间的

1

嘿嘿嘿 2021-06-24

还有一个问题是，我看过作者关于本文的PPT介绍，说是用了128公钥加密，以及大量对称加密，用在哪里了，我懵了

赞

缪弘 回复 嘿嘿嘿 2021-06-24

论文中C(r)的实现就是AES128加密，大量的对称加密指的是C(r)

赞

为你而来 回复 缪弘 2021-06-24

emmm，我一直把C（r）当成一个hash，将任意长数据转换为定长数据。那么128次公钥加密就是执行128次的1-∞的ot么

赞

为你而来 回复 缪弘 2021-06-24

如果C（r）是AES加密，密钥怎么通知，还有很多问题，方便加个联系方式嘛，看你私信

赞

缪弘 回复 为你而来 2021-06-24

128次公钥加密就是一开始执行的128次base OT

赞

嘿嘿嘿 2021-06-24

作者，有个问题，如果发送方控制随机的选择比特s，第二次PSI过程中选择s的反位（0变1，1变0），这不会暴露矩阵T以及C（r）么

赞

缪弘 回复 嘿嘿嘿 2021-06-24

Emmm，第二次PSI的时候，接收方还会重新生成T矩阵的啊

赞

嘿嘿嘿 2021-06-24

第一次阅读PSI相关的文章，希望作者指点

赞

▲ 赞同 25 ▼ 23 条评论 分享 喜欢 收藏 申请转载 ···