> 本文主要记录隐私计算中的秘密共享（Secret Sharing，SS）技术，包括加性秘密共享、Shamir 秘密共享、复制秘密共享（Replicated Secret Sharing，RSS）等技术，**仅供参考**。

一、秘密共享技术介绍

秘密共享是安全多方计算中的一项关键技术，它的核心思想是将一个秘密（隐私数据）**随机拆分**为若干份额，随后将这些**份额分发**给不同的参与方，只有持有足够份额数量的参与方聚集在一起才可以**重建秘密**。此外，不同的数据在秘密共享的状态下可以通过一些算法实现**安全的运算**。

秘密共享技术主要用于多方计算的**算术部分**（加法和乘法），且对于非线性运算往往只能采用**多项式近似**将其转化为线性运算。秘密共享主要由三类算法组成：**分发**、**运算**、**重建**。本文从**算法实现角度**将秘密共享分为**加性秘密共享**、**Shamir 秘密共享**以及**复制秘密共享**进行介绍。

符号说明：一个秘密 s 在共享状态下通常表示为 $[\![s]\!]$，而 $[\![z]\!]=[\![x]\!]+[\![y]\!]$ 代表两个秘密 x,y 在共享状态下进行加法运算得到结果 $z=x+y$ 的共享 $[\![z]\!]$。

二、加性秘密共享

加性秘密共享通常在**环**上实现（例如 $\mathbb{Z}_{2^k}$），后文介绍中的加法和乘法运算均为**环上的运算**（模 2^k），使用环或有限域中的定点数表示是为了保证安全性。加性秘密共享支持扩展至任意数量的参与方，本文中为便于表示以**两个参与方**（Alice 和 Bob）为例进行介绍。

1. 秘密分发

对于一个秘密 $s\in\mathbb{Z}_{2^k}$，不妨设该秘密由 Alice 拥有，则秘密分发过程为：Alice 生成一个随机数 $r\in\mathbb{Z}_{2^k}$，并将其发送给 Bob 作为 Bob 的秘密份额 $s_2=r$，则 Alice 的秘密份额为 $s_1=s-r$，即 $[\![s]\!]=(s-r,r)$。

2. 秘密重建

在加性秘密共享中，只有**集齐所有的参与方的份额**才能重建秘密。重建过程很简单：Alice 将 s_1 发送给 Bob，Bob 将 s_2 发送给 Alice，双方在本地计算秘密 $s=s_1+s_2$。

3. 加法运算

设两个秘密 x,y 在加性秘密共享状态下的份额为：Alice 拥有 x_1,y_1，Bob 拥有 x_2,y_2，即 $[\![x]\!]=(x_1,x_2)$，$[\![y]\!]=(y_1,y_2)$。

则这两个秘密进行加法运算后的结果 $z=x+y$ 在秘密共享状态下的份额为：Alice 在本地计算 $z_1=x_1+y_1$，Bob 在本地计算 $z_2=x_2+y_2$，得到 $[\![z]\!]=(z_1,z_2)$，正确性显然成立。

4. 乘法运算

设两个秘密 x,y 在加性秘密共享状态下的份额为：Alice 拥有 x_1,y_1，Bob 拥有 x_2,y_2，即 $[\![x]\!]=(x_1,x_2)$，$[\![y]\!]=(y_1,y_2)$。记两个秘密进行乘法运算后的结果为 $[\![z]\!]=[\![x]\!]\cdot[\![y]\!]$。

乘法运算需要利用 **Beaver 三元组**实现，该三元组 (a,b,c) 满足 $a\cdot b=c$，并以加性秘密共享的形式分享在 Alice 和 Bob 之间，满足：$[\![a]\!]=(a_1,a_2),[\![b]\!]=(b_1,b_2),[\![c]\!]=(c_1,c_2)$，其中 Alice 持有 (a_1,b_1,c_1)，Bob 持有 (a_2,b_2,c_2)。

假设已经得到了这个三元组，那么乘法运算后的结果可以通过如下运算得到：

$$
\begin{aligned}
z=x\cdot y&=(x-a)\cdot(y-b)+x\cdot b+a\cdot y-a\cdot b\\
&=(x-a)\cdot(y-b)+(x-a)\cdot b+a\cdot(y-b)+a\cdot b
\end{aligned}
$$

Alice 和 Bob 可以首先计算 $\hat{x}=x-a$ 和 $\hat{y}=y-b$ 并将它们公开重建，则乘法可以表示为：

$$
\begin{aligned}
z &= \hat{x} \cdot \hat{y} + \hat{x} \cdot b + a \cdot \hat{y} + a \cdot b \\
&= (\hat{x} \cdot \hat{y} + \hat{x} \cdot b_1 + a_1 \cdot \hat{y} + c_1) + (\hat{x} \cdot b_2 + a_2 \cdot \hat{y} + c_2)
\end{aligned}
$$

因此 Alice 和 Bob 可以在本地计算 z 在加性秘密共享下的份额 $[\![z]\!]$：

$$
\begin{aligned}
z_1 &= \hat{x} \cdot \hat{y} + \hat{x} \cdot b_1 + a_1 \cdot \hat{y} + c_1 \\
z_2 &= \hat{x} \cdot b_2 + a_2 \cdot \hat{y} + c_2
\end{aligned}
$$

由于三元组是随机生成的，公开重建 $\hat{x} = x - a$ 和 $\hat{y} = y - b$ 并不会暴露 x, y 的信息。

三元组的生成方式通常可以分为三种：通过**可信第三方**生成、利用**同态加密**生成、利用**不经意传输**生成。

4.1 可信第三方生成 Beaver 三元组

假设存在一个所有参与方都能信任，且不会与任何参与方合谋的第三方服务器存在，那么只需要在这个服务器中生成好三元组与其对应的份额，再通过安全信道传输给对应的参与方即可。

通过可信第三方生成三元组的方式效率极高，但现实世界一般无法找到这样的第三方。

4.2 同态加密生成 Beaver 三元组

利用 Paillier 或 DGK 同态加密技术可以在不依赖第三方的情况下生成三元组，这两种同态加密技术都具备**加法同态**和**常量乘法同态**性质。设要生成的三元组为 (a, b, c) 满足 $a \cdot b = c$，且 Alice 应当持有 (a_1, b_1, c_1)，Bob 应当持有 (a_2, b_2, c_2)。该方法生成三元组本质上是**利用同态加密实现加法秘密共享下的乘法运算**：

1. Alice 在本地随机生成 a_1, b_1，Bob 在本地随机生成 a_2, b_2，且只有 Alice 具有同态加密算法的私钥。
2. Alice 将 a_1, b_1 利用同态加密算法**加密**后得到 $Enc(a_1), Enc(b_1)$ 并将它们发送给 Bob。
3. Bob 在本地生成一个随机数 r，并利用同态加密算法**同态计算** $Enc(a_1 \cdot b_2 + a_2 \cdot b_1 + r)$ 并将其发送给 Alice。
4. Alice 使用私钥**解密** Bob 发来的消息，并计算份额 $c_1 = a_1 \cdot b_1 + a_1 \cdot b_2 + a_2 \cdot b_1 + r$。
5. Bob 在本地计算份额 $c_2 = a_2 \cdot b_2 - r$。

既然上述步骤可以利用同态加密实现加法秘密共享下的乘法运算，为什么还要生成三元组后再让参与方进行若干计算？这是因为上述步骤利用了公私钥加解密方案，直接使用效率较低，但三元组的生成**不依赖于真实数据**，可以在需要计算乘法前**预先生成**，在真实计算时直接使用之前生成的三元组即可。

4.3 不经意传输生成 Beaver 三元组

利用不经意传输技术也可以在不依赖第三方的情况下生成三元组。

首先 Alice 在本地随机生成 a_1, b_1，Bob 在本地随机生成 a_2, b_2，它们要计算：

$$
c_1 + c_2 = (a_1 + a_2) \cdot (b_1 + b_2) = a_1 \cdot b_1 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_2 \cdot b_2
$$

显然 $a_1 \cdot b_1$ 和 $a_2 \cdot b_2$ 可以在本地计算，只需要利用不经意传输计算 $a_1 \cdot b_2$ 和 $a_2 \cdot b_1$ 即可，将它们统一表示为计算 $x \cdot y$，其中 Alice 拥有 x，Bob 拥有 y。由于 $x, y \in \mathbb{Z}_{2^k}$，则有：

$$
x \cdot y = x[0] \cdot 2^0 \cdot y + x[1] \cdot 2^1 \cdot y + \cdots + x[k-1] \cdot 2^{k-1} \cdot y
$$

其中 $x[i]$ 代表 x 在二进制表示下第 i 位的取值。Bob 可以构造 k 对消息 $(s_i, 2^i \cdot y + s_i)$ 作为发送方，其中 s_i 为随机数，而 Alice 分别以 $(x[0], x[1], \cdots, x[k-1])$ 为选择比特利用 2 选 1 不经意传输技术作为接收方按顺序对 k 对消息进行选择，并将结果求和作为自己的份额 z_1，Bob 在本地计算份额 $z_2 = -\sum_{i=0}^{k-1} s_i$。

类似于 4.2，不经意传输生成三元组也是预先生成的，且可以利用 OT 扩展技术执行大量的 OT 操作，效率较高。

三、Shamir 秘密共享

Shamir 秘密共享[1]是一种 (k, n) **门限秘密共享**，即秘密被随机拆分为 n 份，拥有其中 k 份及以上即可还原秘密，否则无法获取关于秘密的任何额外信息。

Shamir 秘密共享基于多项式的**拉格朗日插值法**实现，本质上是在二维平面上 k 个不同的点即可唯一确定一个 $k-1$ 阶的多项式，而少于 k 个不同的点则存在无穷多种 $k-1$ 阶的多项式。

为了使加密有意义，k 作为门限应当满足 $k \geq 2$，且 $n \geq k$（若需**乘法计算**，必须满足 $n \geq 2k-1$，原因在乘法部分介绍）。在实际应用中，通常取 $n \geq 2k-1$，这样既可以进行乘法运算，也可以在少数份额丢失 $(\leq \lfloor n/2 \rfloor = k-1)$ 的场景下还原秘密。

1. 秘密分发

Shamir 秘密共享通常在**有限域** $\mathbb{F}_p$ 上实现，其中 p 往往是一个大于秘密值的质数。以下所有的运算默认在该有限域上进行 (mod p)。

对于一个秘密 $s \in \mathbb{F}_p$，一个 (k, n) 门限的 Shamir 秘密共享需要构造一个 $k-1$ 阶的随机多项式：

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

其中令 $a_0 = s$，而 $a_1, \cdots, a_{k-1}$ 均为 $\mathbb{F}_p$ 的随机数。

秘密持有者构造完多项式后，随机选择该多项式上 n 个**不同的点** (x_i, y_i) 并将其对应发送给 n 个参与方 P_i（通常取 $x_i = i$），其中 $i = 1, 2, \cdots, n$。这样每个参与方拥有一个点，即一个秘密份额，完成秘密分发。

注意：在秘密分发时，每个参与方份额对应的横坐标应当**公开且保持不变**。

2. 秘密重建

在进行秘密重建时，需要至少 k 个参与方参加，不妨设 k 个参与方提供的 k 个点为 (x_i, y_i)，其中 $i = 1, 2, \cdots, k$。则根据拉格朗日插值法，有：

$$f(x) = \sum_{i=1}^{k} (y_i \cdot l_i(x))$$

其中，$l_i(x)$ 为拉格朗日基多项式：

$$l_i(x) = \prod_{j=1, j \neq i}^{k} \frac{x - x_j}{x_i - x_j}$$

原理可以参考拉格朗日插值法的证明，当 $f(x)$ 被唯一确定后，其常数项 a_0 就是重建得到的秘密 s。

注意到，Shamir 秘密共享在秘密重建时引入了除法运算，这是其在域而非环上实现的原因，环中的元素**不保证都存在乘法逆元**。

3. 加法运算

在 Shamir 秘密共享下，加法运算较为简单。对于两个秘密 a, b，每个秘密都对应一个 $k-1$ 阶多项式，不妨记为 $A(x), B(x)$。每个参与方都持有每个秘密对应的一个份额（点坐标），不妨设 P_i 持有 $\big(x_i, A(x_i)\big)$ 和 $\big(x_i, B(x_i)\big)$，其中 $i = 1, 2, \cdots, n$。

由于在不同秘密分享时，每个份额对应的横坐标应当公开且保持不变，即对于 P_i 来说，所有秘密在它这里的份额横坐标都为 x_i。

显然 P_i 可以计算 $a + b$ 的份额为 $\big(x_i, A(x_i) + B(x_i)\big)$。

4. 乘法运算

Shamir 秘密共享的乘法运算较为复杂。类似地，对于两个秘密 a, b，每个秘密都对应一个 $k-1$ 阶多项式，不妨记为 $A(x), B(x)$。参与方 P_i 持有 $\big(x_i, A(x_i)\big)$ 和 $\big(x_i, B(x_i)\big)$，其中 $i = 1, 2, \cdots, n$。设：

$$A(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}$$
$$B(x) = b_0 + b_1 x + \cdots + b_{k-1} x^{k-1}$$

其中 $a = a_0, b = b_0$。不妨设 $D(x) = A(x) \cdot B(x)$：

$$D(x) = d_0 + d_1 x + \cdots + d_{2k-2} x^{2k-2}$$

显然有 $d_0 = a_0 \cdot b_0 = a \cdot b$，因此每个参与方 P_i 在本地计算 $D(x_i) = A(x_i) \cdot B(x_i)$。

这样一来，所有参与方总共拥有 n 个点 $(x_i, D(x_i))$，而为了还原 $D(x)$ 这个 $2k-2$ 阶多项式，至少需要 $2k-1$ 个点，因此必须满足 $n \geq 2k-1$。

即使满足 $n \geq 2k-1$，这个结果仍然存在一些问题，最重要的是门限由 k 变为了 $2k-1$。如果乘法计算会导致**门限上升**，经过有限次乘法后一定会超过 n 导致**无法还原秘密**。

经典的解决方法是一种“**二次共享**“的思想。我们的目标是基于上述信息构建一个 $k-1$ 阶的多项式 $C(x)$，其常数项为 $d_0 = a \cdot b$，其余项为随机数，且这个过程不能泄露 $a, b, a \cdot b$ 的信息。

由拉格朗日插值法可知：

$$D(x) = \sum_{i=1}^{n} (D(x_i) \cdot l_i(x))$$

$$l_i(x) = \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

则有 $a \cdot b = d_0 = D(0) = \sum_{i=1}^{n} (D(x_i) \cdot l_i(0))$。

由于横坐标是固定且公开的，$l_i(0)$ 也可以公开计算得到。

$D(x_i)$ 由参与方 P_i 持有，这时每个参与方对自己的 $D(x_i)$ 进行一次 (k, n) 门限的 Shamir 共享，得到一个多项式 $f_i(x)$：

$$f_i(x) = D(x_i) + r_{i,1} x + r_{i,2} x^2 + \cdots + r_{i,k-1} x^{k-1}$$

其对应的 n 个点坐标（份额）为 $\big(x_1, f_i(x_1)\big)$，$\big(x_2, f_i(x_2)\big)$，$\cdots$，$\big(x_n, f_i(x_n)\big)$。

总共有 n 个这样的多项式，因此总共有 n^2 个点坐标，每个参与方拥有其中 n 个，例如参与方 P_1 拥有 $\big(x_1, f_1(x_1)\big)$，$\big(x_1, f_2(x_1)\big)$，$\cdots$，$\big(x_1, f_n(x_1)\big)$。

注意到：

$$\begin{aligned}
\sum_{i=1}^{n} \big(l_i(0) \cdot f_i(x)\big) &= \sum_{i=1}^{n} \big(l_i(0) \cdot D(x_i)\big) + \sum_{i=1}^{n} l_i(0) \cdot r_{i,1} x + \cdots + \sum_{i=1}^{n} l_i(0) \cdot r_{i,k-1} x^{k-1} \\
&= a \cdot b + r'_{i,1} x + r'_{i,2} x^2 + \cdots + r'_{i,k-1} x^{k-1} \\
&= C(x)
\end{aligned}$$

其中因为 $r_{i,1}$ 是随机数，所以 $r'_{i,1}$ 也为随机数。因此可以看出上面这个多项式就是我们最终要构造的多项式 $C(x)$，它是 $f_i(x)$ 以 $l_i(0)$ 为系数的一个线性组合。

我们找出了一种 $C(x)$ 的构造方法，最后一步就是获得 $C(x)$ 的 n 个点坐标，即秘密分享的份额。 由于对 n 个多项式进行线性组合，等价于**在点坐标表示下**对这 n 个多项式的**点坐标**进行线性组合。因此我们要获取的最终的多项式 $C(x)$ 的 n 个点坐标为：

$$\Big(x_1, \sum_{i=1}^{n} \big(l_i(0) \cdot f_i(x_1)\big)\Big), \Big(x_2, \sum_{i=1}^{n} \big(l_i(0) \cdot f_i(x_2)\big)\Big), \cdots, \Big(x_n, \sum_{i=1}^{n} \big(l_i(0) \cdot f_i(x_n)\big)\Big)$$

显然每个参与方对应的点坐标（份额）可以直接在本地计算得出，这样 Shamir 秘密共享的乘法计算就完成了，计算结果 $[\![a \cdot b]\!]$ 就是上述 n 个点坐标（份额），仍然保持 (k, n) 门限。

Shamir 秘密共享的乘法计算看上去较为繁琐，但思想其实很简单，“二次共享“的本质可以理解为利用 Shamir 秘密共享**加密** Shamir 秘密共享的**解密过程**，这个思想和同态加密的 Bootstrapping 类似，都是一种“**同态解密**“。在这个过程中，$D(x_i)$ 可以看成每个参与方的“解密密钥“，将这些解密密钥利用 Shamir 秘密共享“加密“，进行“密文状态下的解密运算“，最终得到计算结果的“密文“，即 n 个份额。

四、复制秘密共享

复制秘密共享通常是指**三方**复制秘密共享，即三个参与方 P_0, P_1, P_2 参与计算。复制秘密共享也可以扩展至多方，其思想相同，可以参考[2]的实现，本文以三方复制秘密共享为例介绍。

复制秘密共享也是一种**门限**秘密共享，在三方复制秘密共享中门限通常为 2。复制秘密共享一般不涉及除法，因此可以在环上实现（例如 $\mathbb{Z}_{2^k}$ ），后续介绍中的运算默认都在环上进行（模 2^k）。

后续介绍中会经常提到某个参与方的上一个/下一个参与方，这里的顺序可以理解为模 3 环绕，例如 P_0 的上一个参与方是 P_2，P_2 的下一个参与方是 P_0，其他变量的下标序号也遵循这个规范。

复制秘密共享利用伪随机数生成器优化随机数在多个参与方上的生成，首先对这个优化作一个介绍。

1. 伪随机数生成器优化

一个伪随机数生成器 PRG，给定一个随机数种子 s，可以生成**看似随机的确定序列**，即在不知道种子 s 的取值时，PRG 每次生成的数字都可以看成随机数，而知道 s 取值的情况下，PRG 生成的数字序列都是确定的。

在三方复制秘密共享中，每个参与方 P_i 本地生成一个随机数种子 s_i，并将其发送给上一个参与方，使得 P_0 拥有 (s_0, s_1)，P_1 拥有 (s_1, s_2) ，P_2 拥有 (s_2, s_0)。

这样一来，任意两个参与方可以使用它们共有的随机数种子生成一个相同的随机数，例如 P_0, P_1 可以通过 $PRG(s_1)$ 生成一个随机数 r，这样就避免了一次通信（不使用 PRG 时必须一方生成随机数后发送给另一方）。在安全多方计算中，**通信开销非常影响性能**，尤其在低带宽高时延网络环境下，减少一次通信有利于整体效率的提升。

利用这种技巧，三个参与方可以**不通过任何通信交互**构造出三个零和随机数 $\alpha_0, \alpha_1, \alpha_2$，其中 α_i 只有 P_i 拥有，且三者满足 $\alpha_0 + \alpha_1 + \alpha_2 = 0$。构造方式如下：

1. 利用伪随机数生成器，(P_0, P_1) 生成一个随机数 r_1，(P_1, P_2) 生成一个随机数 r_2，(P_2, P_0) 生成一个随机数 r_0。
2. P_i 在本地计算得到 $\alpha_i = r_i - r_{i+1}$。

显然可以验证 $\alpha_0 + \alpha_1 + \alpha_2 = (r_0 - r_1) + (r_1 - r_2) + (r_2 - r_0) = 0$。

不妨记上述步骤为 $(\alpha_0, \alpha_1, \alpha_2) = PRGShare(0)$。

2. 秘密分发

三方复制秘密共享的秘密分发会使用到伪随机数生成器。不妨设待共享的秘密 x 由 P_0 持有，三个参与方计算 $(\alpha_0, \alpha_1, \alpha_2) = PRGShare(0)$，随后三个参与方分别计算三个份额：$x_0 = \alpha_0 + x$，$x_1 = \alpha_1$，$x_2 = \alpha_2$。

参与方 P_i 将份额 x_i 发送给上一个参与方 P_{i-1} 完成秘密分发，使得 P_0 拥有 (x_0, x_1)，P_1 拥有 (x_1, x_2)，P_2 拥有 (x_2, x_0)，即 $[\![x]\!] = \big((x_0, x_1), (x_1, x_2), (x_2, x_0)\big)$。

易证 $x = x_0 + x_1 + x_2$。可以看出任意两个参与方都可以还原秘密 x，因此这是一个 $(2, 3)$ 门限秘密共享。

3. 秘密重建

三方复制秘密共享在重建秘密 x 时，每个参与方 P_i 将 x_i 发送给下一个参与方即可。这样一来所有参与方都有拥有完整的 (x_0, x_1, x_2)，从而还原秘密。

在重建时可以指定在参与方 P_i 处重建秘密，只需让 P_{i-1} 将 x_{i-1} 发送给 P_i 即可。

4. 加法运算

对于两个在三方复制秘密共享下的秘密 $[\![x]\!]$ 和 $[\![y]\!]$，要计算 $[\![z]\!] = [\![x]\!] + [\![y]\!]$，只需每个参与方 P_i 在**本地计算** $(z_i, z_{i+1}) = (x_i + y_i, x_{i+1} + y_{i+1})$ 即可。

5. 乘法运算

对于两个在三方复制秘密共享下的秘密 $[\![x]\!]$ 和 $[\![y]\!]$，要计算 $[\![z]\!] = [\![x]\!] \cdot [\![y]\!]$，首先可以观察到：

$$
\begin{aligned}
z = x \cdot y &= (x_0 + x_1 + x_2) \cdot (y_0 + y_1 + y_2) \\
&= x_0 \cdot (y_0 + y_1) + x_1 \cdot y_0 && (1) \\
&+ x_1 \cdot (y_1 + y_2) + x_2 \cdot y_1 && (2) \\
&+ x_2 \cdot (y_2 + y_0) + x_0 \cdot y_2 && (3)
\end{aligned}
$$

注意上述等式中的 $(1)(2)(3)$ 行的内容均可由三个参与方分别在本地计算，即 P_i 可以在本地计算 $z_i' = x_i \cdot (y_i + y_{i+1}) + x_{i+1} \cdot y_i$。

这样一来我们似乎没有进行任何通信交互就得到了乘法运算的结果，但仍然存在两个问题。一是 z_i' **并非完全随机**，其取值依赖于输入数据；二是每个参与方 P_i **只拥有一个份额** z_i'，不满足三方复制秘密共享 $(2, 3)$ 门限共享的规范，无法进行后续乘法计算。

解决办法仍然是利用伪随机数生成器得到 $(\alpha_0, \alpha_1, \alpha_2) = PRGShare(0)$，随后每个参与方 P_i 在本地计算 $z_i = z_i' + \alpha_i$（解决随机性），并将 z_i 发送给上一个参与方 P_{i-1}（解决份额不足），最终得到 $[\![z]\!] = \big((z_0, z_1), (z_1, z_2), (z_2, z_0)\big)$。

可以看出三方复制秘密共享在进行秘密分发、秘密重建和乘法运算时都**只需要一轮通信**，进行加法运算**不需要通信**，在不合谋的三方半诚实敌手安全模型下，效率极高，是最常用的秘密共享技术之一。

参考文献

- [1] Adi Shamir. 1979. How to share a secret. Commun. ACM 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176
- [2] A. Baccarini, M. Blanton, and C. Yuan, 'Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning', Proceedings on Privacy Enhancing Technologies, 2023, doi: 10.56553/popets-2023-0035.

本文为作者在学习相关知识时的一种记录，便于以后的回顾。作者并没有系统地学习过密码学，因此在表述上可能会存在不严谨甚至出错的地方，文章仅供参考，欢迎大家与我交流，一起进步！

其他平台：

- 知乎（Totoro）：https://www.zhihu.com/people/totoro-14-60
- CSDN（_Totoro_）：https://blog.csdn.net/orz_Totoro
- B站（Totoro_134）：https://space.bilibili.com/279377771
- Github（Totoro134）：https://github.com/Totoro134
- 公众号（知识长生所）