联　　盟　　规　　范

PPCA 10 —2023

隐私计算 跨平台互联互通 开放协议 第 4 部分：Secure Gradient Boosting

Privacy-preserving computation cross-platform interconnection—

Open protocol Part4: Secure Gradient Boosting

2023-12-21 发布　　2023-12-21 实施

隐私计算联盟　发 布

目　次

前　　言

本文件是《隐私计算 跨平台互联互通》系列文件之一，该系列文件名称如下：

——开放协议 第1部分：ECDH-PSI；

——开放协议 第2部分：SS-LR；

——开放协议 第3部分：PHE-FLR；

——开放协议 第4部分：Secure Gradient Boosting。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本文件由隐私计算联盟提出并归口。

本文件起草单位：中国信息通信研究院、蚂蚁科技股份有限公司、中移动信息技术有限公司、天翼电子商务有限公司、联通数字科技有限公司、中国工商银行、上海浦东发展银行股份有限公司、深圳市洞见智慧科技有限公司。

本文件主要起草人：邵健、陆宇飞、杨靖世、章庆、孙林、毕剑锋、郭相林、高扬、何浩、彭晋、白玉真、陈思远、张锦锋、毛万葵、夏家骏、昌文婷、纪耀宗、靳新、袁鹏程、余超凡、袁博、赵原、茹志强、杨桂林、高峰、陶建萍、刘微、赵永坤、狄佳贵。

引　　言

当前多方安全计算、联邦学习等隐私计算技术快速发展，越来越多的产品从试点部署阶段转入落地应用，市场竞争火热。但是，不同技术厂商提供的产品和解决方案在设计原理和功能实现之间存在较大差异，使得部署于不同平台的隐私计算参与方之间无法跨平台完成同一计算任务，为实现与部署于不同平台的多个合作方之间的数据融合，用户往往不得不部署多套产品以逐一适配。作为促进跨机构间数据共享融合的关键技术，隐私计算有望成为支撑数据流通产业的基础设施，但高额的应用成本不利于隐私计算技术的推广应用。因此，解决不同产品之间的技术壁垒，实现计算任务在跨平台间的互联互通已成为产业内的迫切需求。

版权声明

邮箱：ppca@caictyds.cn

隐私计算 跨平台互联互通
开放协议 第 4 部分：Secure Gradient Boosting

1 范围

本文件规定了隐私计算平台间的SGB算法的互联互通协议和参考实现。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

3 术语和定义

GB/T 25069-2022界定的以及下列术语和定义适用于本文件。

3.1 隐私计算 privacy-preserving computation

在保证数据提供方不泄露原始数据的前提下，对数据进行分析计算的一系列信息技术，保障数据在流通与融合过程中的“可用不可见”。

3.2 开放协议 open protocol

通过规范算法执行流程中交互信息，各平台独立开发算法来实现平台间算法的互通。

3.3 算法 algorithm

为解决问题严格定义的有限的有序规则集。

[来源：GB/T 25069—2022, 3.581]

4 缩略语

下列缩略语适用于本文件。

XGBoost: Extreme Gradient Boosting[1]

SGB: Secure Gradient Boosting

主动参与方：持有样本标签列的参与方

被动参与方：不持有样本标签列的参与方

5 算法概述

XGBoost算法属于Boosting算法家族，而SGB算法则是将XGBoost算法扩展到纵向联邦学习任务中的一种变体。本节将简要介绍XGBoost算法的原理以及SGB算法的流程。

5.1 XGBoost 算法概述

XGBoost 模型是一种集成的树模型，其模型训练过程如图 1 所示。该模型采用加法策略训练 XGBoost 的子模型（每个子模型是一颗二叉树）。具体而言，每一步只训练一个子模型，最终预测值为每一步子模型预测值相加的结果。假设最终得到的模型包含 K 个子模型，用函数 f_i 表示第 i 个子模型的结构和叶子结点的权重，那么 $f_1(x)+f_2(x)+ ... +f_K(x)$ 就等于这个模型针对样本 x 的预测值。

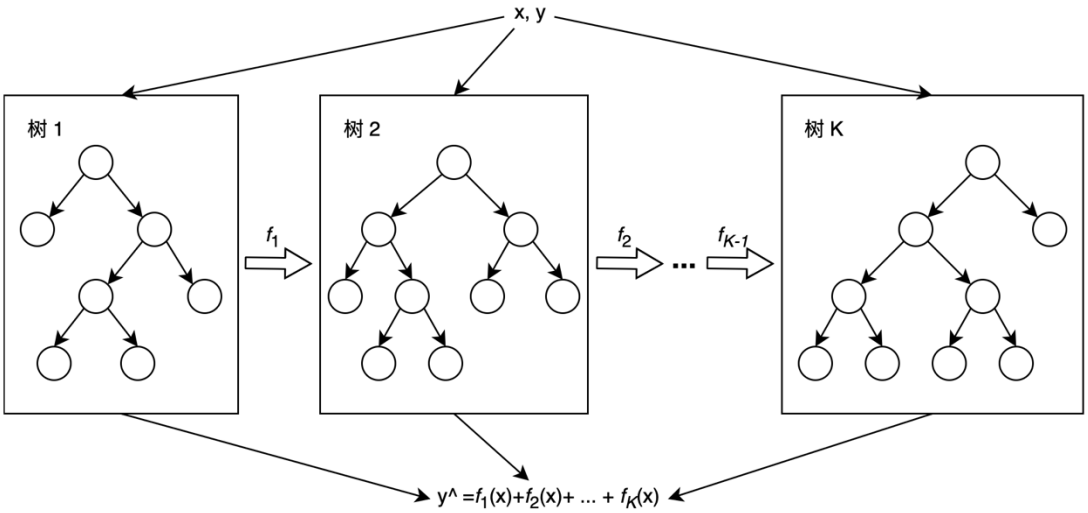

图 1 XGBoost 算法总体框架

用数学语言描述训练数据集$\mathcal{D}=\{(\mathbf{x}_i,y_i)\}(|\mathcal{D}|=n,\mathbf{x}_i\in\mathbb{R}^m,y_i\in\mathbb{R})$，样本数据集$\mathbf{x}$是 n×m 的矩阵，特征数据集$y$是 n×1 的矩阵，则算法模型如下：

$$\hat{y}_i=\phi(\mathbf{x}_i)=\sum_{k=1}^{K}f_k(\mathbf{x}_i),\quad f_k\in\mathcal{F}$$

其中，$\mathcal{F}=\{f(\mathbf{x})=w_{q(\mathbf{x})}\}(q:\mathbb{R}^m\to T,w\in\mathbb{R}^T)$。

为了训练每个子模型，首先需要定义子模型训练的目标函数。该目标函数由两部分组成，分别是损失项和正则项，具体表达如下：

$$\mathcal{L}(\phi)=\sum_i l(\hat{y}_i,y_i)+\sum_k\Omega(f_k)$$

其中，

$$\Omega(f)=\gamma T+\frac{1}{2}\lambda||w||^2$$

T 是叶子结点的数量，w 是每个叶子结点的权重，γ和λ是正则项中的系数。

训练第t个子模型的损失函数如下：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i)] + \frac{1}{2}h_i f_t^2(\mathbf{x}_i) + \Omega(f_t)$$

其中，$g_i = \partial_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$，$h_i = \partial^2_{\hat{y_i}^{(t-1)}} l(y_i, \hat{y_i}^{(t-1)})$

定义叶子结点j的实例集合为$I_j = \{i|q(\mathbf{x}_i) = j\}$，则方程可写为：

$$\tilde{\mathcal{L}}^{(t)} = \sum_{j=1}^{T}[G_j w_j + \frac{1}{2}(H_j + \lambda)w_j^2] + \gamma T$$

其中，$G_j = \sum_{i \in I_j} g_i$，$H_j = \sum_{i \in I_j} h_i$

若要使这个目标函数值最小，则

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\tilde{\mathcal{L}}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

在实际训练中，采用贪婪算法来划分树结构，即从单个叶结点开始，每次只优化子模型的一层。假设针对结点的某个划分，G_L、H_L和G_R、H_R分别是结点的左子结点和右子结点的G和H，则这个划分的增益为：

$$Gain = \frac{1}{2}[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma$$

结点的最优分裂点即为使增益最大的划分方式。

5.2 SGB 算法流程概述

在垂直联邦学习场景下，数据集的特征列分布在多个参与方，而数据集的标签列仅由一方持有。我们把持有标签列的参与方称为主动参与方，把不持有标签列的参与方称为被动参与方。SGB 算法支持多个被动参与方的场景，即对被动参与方的扩展。图 2 展示了仅有一个被动参与方的计算流程，该流程分为两个阶段：

a) 算法协商握手阶段，确定算法版本、XGBoost 算法参数、同态加密算法及相关参数等算法运行所需的信息，详见第 6 节；

b) 算法主体运行阶段，进行 SGB 模型训练，计算出模型结构和权重信息，详见第 7 节。

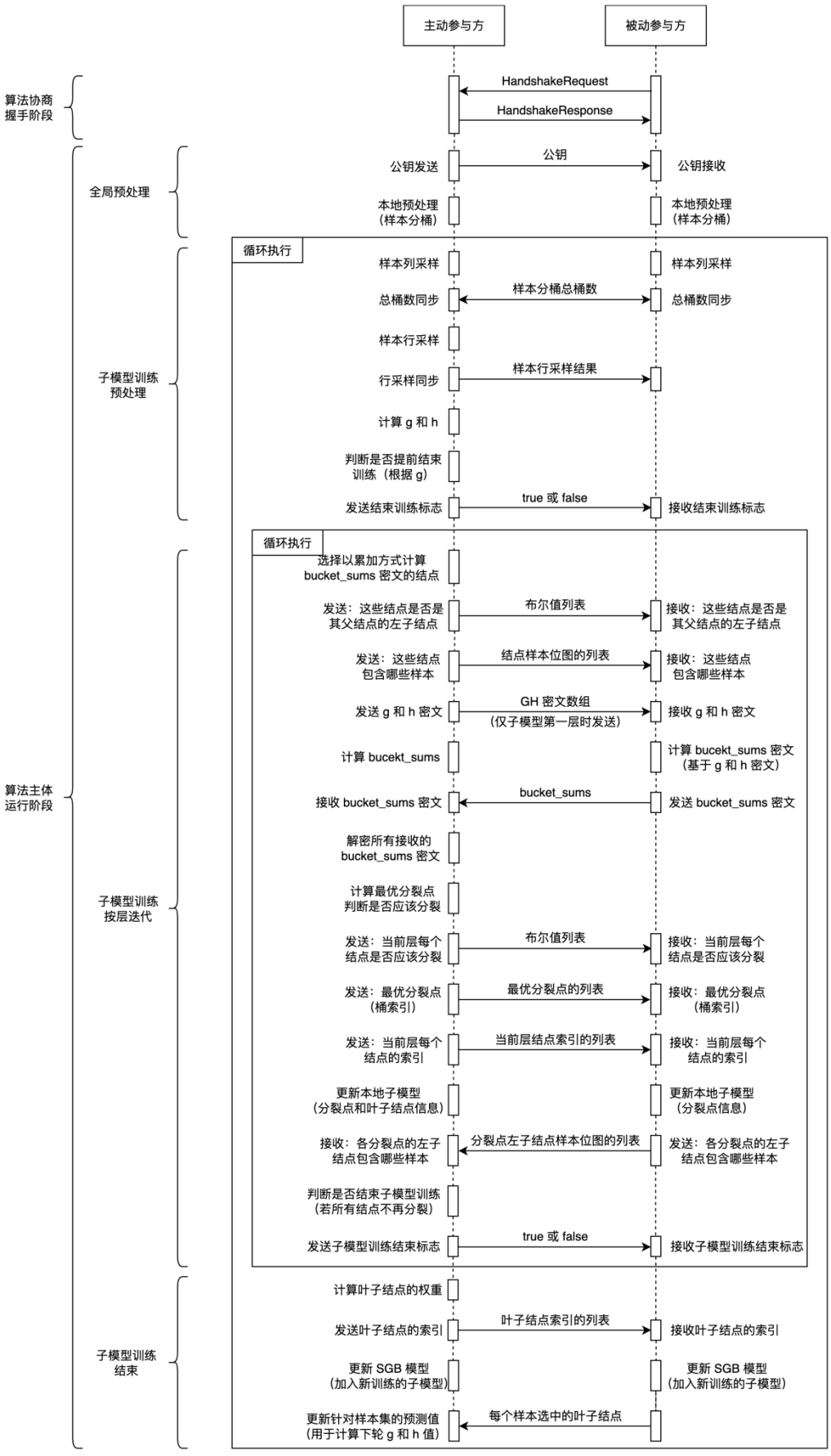

图 2 SGB 算法流程概述（仅有一个被动参与方的场景）

6 算法协商

在算法协商握手阶段，其主要目的是协调并对齐各个被动参与方和主动参与方之间所需的算法运行参数。具体操作如下：

a) 握手请求发送

每个被动参与方在程序启动时向主动参与方发送握手请求，握手请求中包含被动参与方所需的算法运行参数的初始信息。

b) 握手参数协商

主动参与方接收到握手请求后，根据收到的请求参数，进行参数协商。

c) 协商结果响应

如果主动参与方成功协商请求参数，将具体的参数信息放置在握手响应中。如果协商失败，主动参与方将在握手响应中包含详细的错误描述信息，指示协商失败的原因。

d) 握手响应发送

主动参与方将握手响应发送回其他参与方，包括成功协商的参数或错误描述信息。

有关算法协商握手的数据格式，请参考8.1 节。

7 算法运行

7.1 全局预处理

7.1.1 同态加密公钥分发

主动参与方根据算法协商阶段确定的同态加密算法类型和密钥长度，在本地生成私钥和公钥，并将生成的公钥发送给其他参与方。关于公钥传输时的数据格式，请参考 8.2.2.1 节。

7.1.2 样本数据分桶

每个参与方对其拥有的每个样本特征列进行排序和分桶，且每个特征列的桶数应相等。桶数的计算公式为 bucket_num = ceil(1 / bucket_eps) + 1，其中 bucket_eps 参数在算法协商握手阶段确定。桶的编号从 0 开始，依次累加，最大编号为 bucket_num - 1。

在对每个特征列进行排序和分桶时，会生成一组分割点，这些分割点的数值根据样本集中相应特征值的情况确定，用于将每个样本单元分割到该特征列的一个桶中。例如，如果一个样本的特征值为 x，并且 x 大于等于分割点 k-1 的值、小于分割点 k 的值，则该样本将被划分到桶 k 中。如图 3 所示，对于每个特征列的最后一个桶，分割点 n-1 的值为最大的样本特征值，因此最后这个桶仅包含特征值最大的样本。

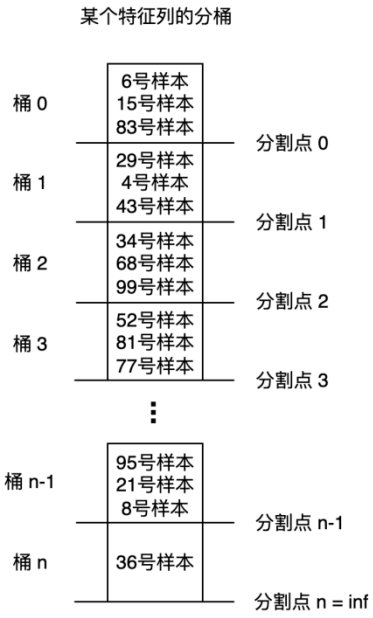

图 3 样本列按特征值排序和分桶

图 4 展示了样本分桶结果的矩阵表示。通过样本分桶，每个参与方在本地得到一个与本地样本特征数据集形状相同的矩阵，其中第 i 行第 j 列的元素值表示 i 号样本被分割到特征列 j 的哪个桶中。举例来说，假设参与方 A 的样本特征数据集有 14 行 3 列，bucket_eps = 0.18，则其样本分桶矩阵如图 4 所示。

	特征列 0	特征列 1	特征列 2
0号样本	5	2	6
1号样本	4	1	2
2号样本	5	6	0
3号样本	1	0	1
4号样本	0	2	4
5号样本	1	4	3
6号样本	2	3	5
7号样本	6	5	1
8号样本	6	1	0
9号样本	3	0	2
10号样本	4	6	4
11号样本	0	3	5
12号样本	3	5	3
13号样本	2	4	6

矩阵元素值：样本单元在这个特征列上的桶索引

图 4 参与方本地的样本分桶矩阵示例

7.2 子模型训练

7.2.1 子模型预处理

7.2.1.1 样本列采样

根据算法协商握手阶段确定的 colsample_by_tree 参数对样本特征列进行采样。假设参

与方持有 m 个特征列，则样本采样的列数 col_num = ceil(m * colsample_by_tree) 。

需要特别注意的是，如果当前正在训练第一个子模型，并且参数 completely_sgb 等于 true，那么仅对主动参与方进行列采样，被动参与方的 colsample_by_tree 参数直接置为 0。

7.2.1.2 总桶数同步

在各参与方进行本地列采样后，能够明确子模型训练所使用的本地特征列。由此，可以计算出这些特征列的总桶数，并将其传送给其它参与方。用变量 buckets_count 表示总桶数，它是一个整数类型的标量，传输时的数据格式详见 8.2.1 节。

各参与方在收到其它方的 buckets_count 后，将其保存在本地。若 Alice 、Bob、Carol 三方的 buckets_count 分别为 196 、 242 、 276 ， 则各方在本地拼接得到同一个数组 buckets_counts = [196, 242, 276]。

7.2.1.3 样本行采样

根据算法协商握手阶段确定的 rowsample_by_tree 参数对样本进行行采样。当 rowsample_by_tree < 1 时，主动参与方在本地执行随机行采样，然后将采样结果发送给其它的参与方。假设样本数据的总行数等于 n ， 则采样的行数 row_num = ceil(n * rowsample_by_tree)。

采样结果以一维数组形式表示，由主动参与方发送给被动参与方，数组元素表示采样行的索引。传输时的数据格式详见 8.2.1 节。

7.2.1.4 GH 矩阵计算

主动参与方利用本地持有的样本标签和计算得到的预测值，计算出样本的 g 和 h。针对不同的训练目标，g 和 h 的计算公式也有所不同。训练目标无需握手协商确定，本节仅提供回归任务和分类任务的参考公式。

如果训练目标是回归任务，则计算公式为：

$$\hat{y}_i = pred_i$$

$$g_i = \hat{y}_i - y_i$$

$$h_i = 1$$

其中，$\hat{y}_i$是第 i 个样本的实际预测值，y_i是对应的标签。

如果训练目标是分类任务，则计算公式为：

$$\hat{y}_i = sigmoid(pred_i)$$

$$g_i = \hat{y}_i - y_i$$

$$h_i = \hat{y}_i * (1 - \hat{y}_i)$$

其中，$pred_i$的初始值由主动参与方自行配置。

GH 矩阵有 n 行 2 列，每行对应一个样本的 g 值和 h 值。

7.2.1.5 预处理数据切分

7.2.1.5.1 本地切分 GH 矩阵

主动参与方根据行采样结果对 GH 矩阵进行切分，将切分得到的子矩阵作为本轮子模型训练的 GH 矩阵。

7.2.1.6 判断模型训练是否提前结束

主动参与方根据 G 矩阵中 g 值的大小和变化率来判断是否有必要提前结束整个模型的训练。判断逻辑由主动参与方自行决定，本节提供一种供参考的判断逻辑。

用数值 g_abs_sum 表示 G 矩阵中所有 g 值的绝对值之和，用数值 last_g_abs_sum 表示上一轮迭代计算的 g_abs_sum 值；主动参与方本地定义两个参数：g_threshold 和 g_ratio_threshold，分别用来限制 g_abs_sum 的大小和变化率，当满足以下任意一个条件时，可提前结束训练：

a) g_abs_sum <= g_threshold

b) abs((last_g_abs_sum – g_abs_sum) / g_abs_sum) <= g_ratio_threshold

主动参与方将判断结果同步给其它几个参与方，用布尔值 true 表示提前结束，布尔值 false 表示不要提前结束，传输时的数据格式详见8.2.1 节。

若被动参与方收到布尔值 false，则与主动参与方同时结束模型训练。

7.2.1.7 GH 矩阵加密和同步

如果当前正在训练第一个子模型且参数 completely_sgb 为 true，则跳过这一步；否则，主动参与方对GH矩阵进行编码和加密，得到GH密文矩阵，随后将这个密文矩阵发送给其他参与方。有关密文矩阵的通信格式，请参考8.2.2.2 节。

7.2.2 子模型按层迭代

7.2.2.1 结点索引和样本位图

SGB 模型的子模型是一颗二叉树，我们可以通过编号来索引这个二叉树的每个结点，如图 5 所示。根结点的索引值是 0。如果子模型中某个结点的索引值等于 i，则它的左子结点的索引值是 2 * i + 1，右子结点的索引值是 2 * i + 2。

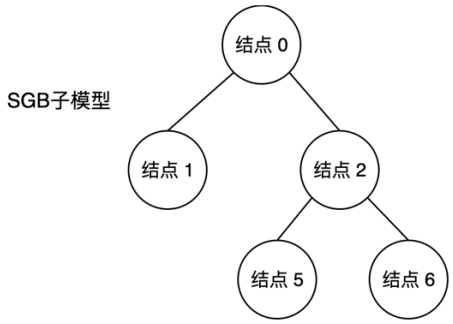

图 5 用索引值对树结点编号

对于每个结点，可以使用一个位图（bitmap）来表示在该结点上分配了哪些样本，如图 6 所示。位图的比特长度等于输入样本的数量，位图的比特索引值等于输入样本的索引值，而位图的比特值表示对应索引的样本是否划分在这个结点上。例如，如果某个结点的位图等于[0, 1, 0, 1, 0, 0, 0]，那么说明输入样本总数为 7，且只有样本 1 和样本 3 被划分到这个结点上。

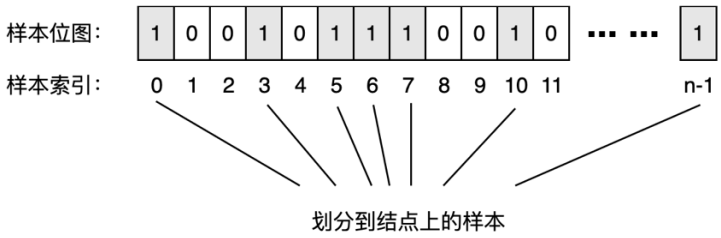

图 6 用位图表示划分到结点上的样本

由于根结点包含了所有的输入样本，因此它的位图中的每个比特值都被设置为 1。

7.2.2.2 当前层结点的 bucket_sums

当前层结点的 bucket_sums 矩阵如图 7 所示。持有样本特征的每个参与方都需要在本地计算自己的 bucket_sums 矩阵。Bucket_sums 矩阵的列数等于 2，分别对应 g 和 h；行数等于参与本轮训练的本地特征列的总桶数（buckets_count），每一行对应本地特征列的一个桶，元素值等于从这个特征列的第 1 个桶到这个桶包含的所有样本的 g 值（和 h 值）之和。与主动参与方不同，被动参与方的 bucket_sums 计算是针对 g 和 h 同态密文的加法运算。

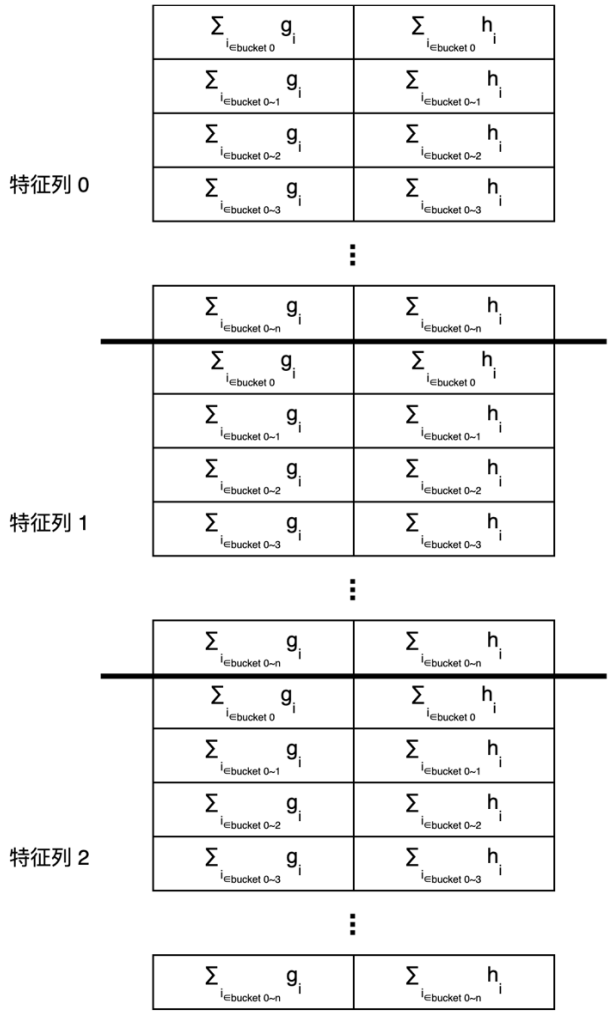

图 7 当前层结点的 bucket_sums 矩阵（以本地 3 个特征列为例）

7.2.2.3 从每对兄弟结点中选择一个结点

由于被动参与方计算每个结点的 bucket_sums 时采用的是同态密文计算，为了减少密文加法计算次数，对被动参与方的计算过程进行调整，可以将密文计算量减少接近一半。因为主动参与方计算 bucket_sums 时采用的是明文计算，所以可以不采用这个优化。

在子模型中，除了最顶层只有一个根结点外，其它各层都有偶数个结点，这些结点成对出现，每一对中的两个结点具有相同的父结点，称为兄弟结点（如图 5 所示）。

在被动参与方，对于两个兄弟结点而言，它们的 bucket_sums 密文矩阵相加的结果应当等于它们的父结点的 bucket_sums 密文矩阵。由于上一层迭代已经计算出父结点的 bucket_sums 密文，因此在当前层只需要先计算第一个兄弟结点的 bucket_sums 密文，然后通过将父结点的 bucket_sums 密文减去第一个兄弟结点的 bucket_sums 密文，即可得到第二个兄弟结点的 bucket_sums 密文。因此，第一个兄弟结点的 bucket_sums 密文是通过累加的方式计算的，而第二个兄弟结点的 bucket_sums 密文是通过减法计算得出的。

主动参与方负责从每对兄弟结点中选择一个结点，并把选择结果发送给其它参与方。其他参与方可以根据这个选择结果确定采用累加方式计算 bucket_sums 的结点。一种推荐的选

择方式是，在一对兄弟结点中，如果左节点的样本数量小于等于右结点的样本数量，则选择左结点，否则选择右结点，如图 8 示例所示。

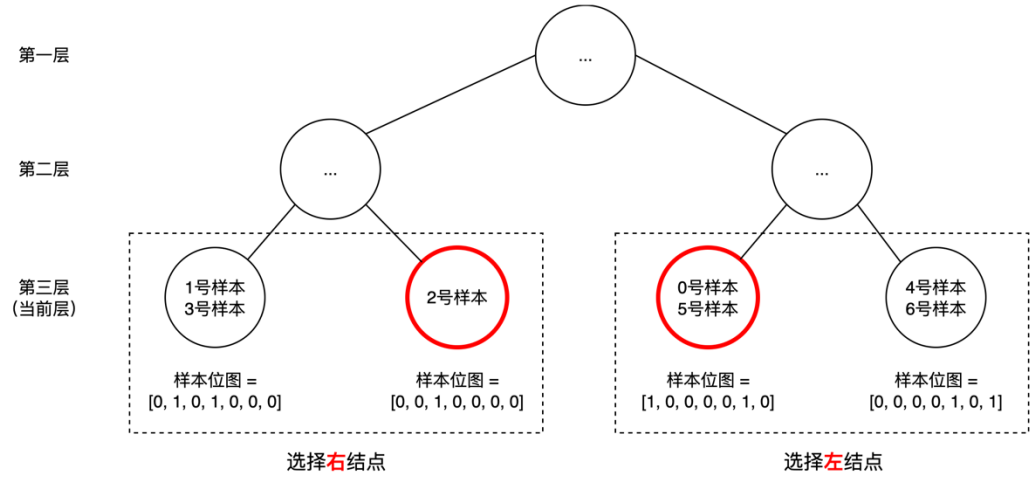

图 8 根据结点上的样本数量从每对兄弟中选择一个结点的示例

7.2.2.4 主动参与方向被动参与方同步结点选择结果

当主动参与方向其它参与方同步结点选择结果时，使用布尔值列表来表示这个结果。例如，对于图 8 的示例，用 [false, true] 列表表示结点选择结果，其中 true 的意思是选择兄弟中的左结点，false 的意思是选择兄弟中的右结点。布尔值列表传输时的数据格式详见 8.2.1 节。

被动参与方在计算选中结点 bucket_sums 时需要使用到这些结点的样本位图，因此主动参与方还需要把选中结点的样本位图发送给其它参与方。我们用 uint8 一维数组表示结点的样本位图，例如，图 8 中第一个选中结点的样本位图 [0, 0, 1, 0, 0, 0] 可以用一维数组 [0b100000] 表示（用二进制整数展示列表元素）。因为这个样本位图的长度不超过 8，所以数组中只有一个元素。第二个选中结点的样本位图 [1, 0, 0, 0, 1, 0] 可以用一维数组 [0b10001000] 表示。传输时，主动参与方将这两个 uint8 一维数组放在一个数组列表中，一次性发送给其它参与方。数组列表传输的数据格式详见 8.2.1 节。

7.2.2.5 被动参与方向主动参与方发送 bucket_sums 密文矩阵

如果当前正在训练第一个子模型且参数 completely_sgb 等于 true，那么结点划分不会使用到被动参与方的 bucket_sums，因此应该跳过本节。否则，被动参与方需要将本地计算得到的 bucket_sums 密文矩阵发送给主动参与方，用于确定最优分裂点。

为了增加安全攻击的难度（参考[3]），被动参与方在发送 bucket_sums 密文矩阵时，应该在发送前按行进行乱序。本节介绍一种分别对每个特征列进行乱序的方式，如图 9 所示，特征列最后一个桶的位置保持不变，其余桶的位置则随机排列。此外，也可以考虑在特征列之间进行乱序操作。

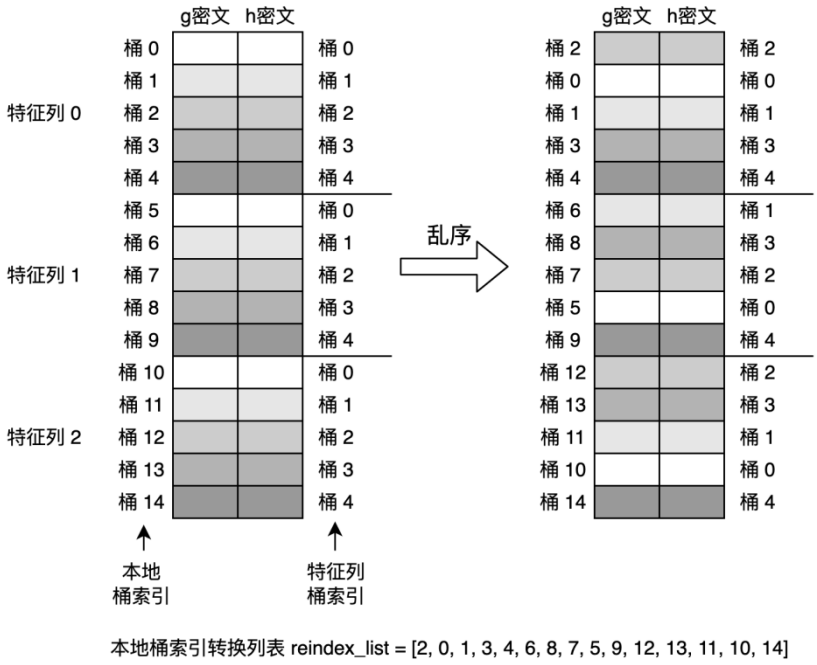

图 9 被动参与方将 bucket_sums 密文矩阵在本地打乱（以 3 个特征列，共 15 个桶为例）

被动参与方在乱序后，按照结点从左到右的顺序，逐一将当前层所有结点的 bucket_sums 密文矩阵发送给主动参与方。有关传输时的数据格式，请参考 8.2.2.2 节。

主动参与方在接收到被动参与方发送的 bucket_nums 密文矩阵后，需要进行解密操作，以获取相应的明文数据。

7. 2. 2. 6 主动参与方对 bucket_sums 拼接和转换得到 GL、HL 矩阵

主动参与方在收到所有的 bucket_sums 后，需要对其进行拼接和整理，得到 GL 矩阵和 HL 矩阵，如图 10 所示。这两个矩阵具有相同的形状，因此仅用一张矩阵图表示。

GL 矩阵和 HL 矩阵的行对应当前层的每个结点，矩阵的列对应来自所有参与方的所有特征列的每个桶，矩阵元素的值等于 bucket_sums 矩阵中对应的元素值，即以这个桶为分裂点的左子结点包含的样本 g / h 值之和。

在同一行上，每个特征列最后一个桶对应的元素值（图 10 灰色方块）相同，都等于这个结点包含的所有样本 g /h 值之和。因此，若取 GA 矩阵为 GL 矩阵的最后一列，计算 GR = GA - GL，则 GR 矩阵的元素值为以每个桶为分裂点的右子节点的样本 g / h 值之和。

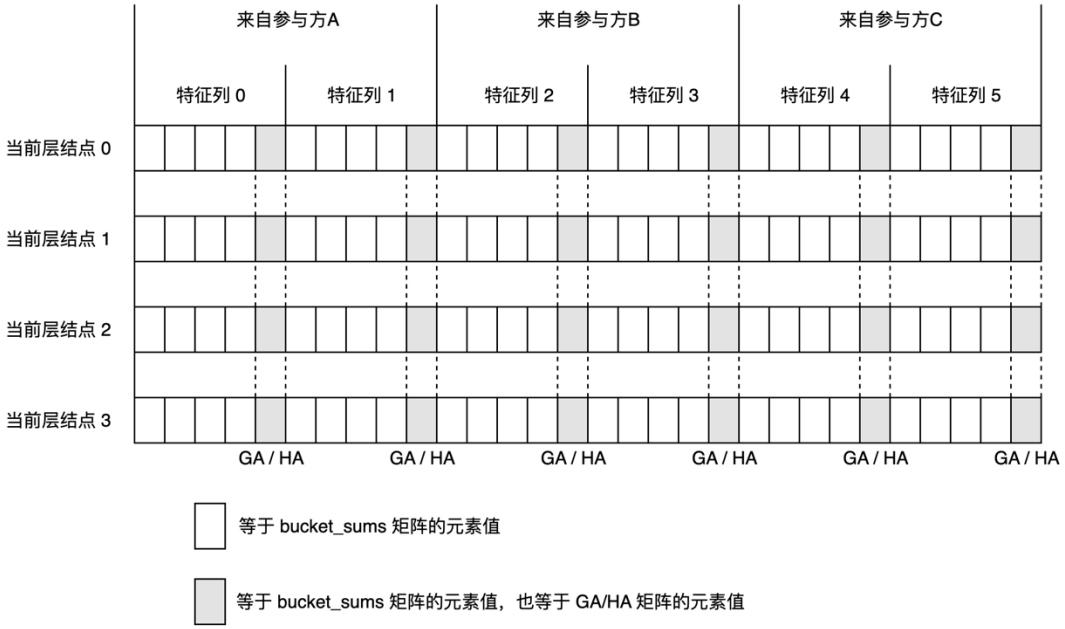

图 10 GL（或 HL）矩阵示例

7.2.2.7 主动参与方计算增益矩阵、确定最优分裂点

为简化增益公式，可以省略公式中的系数，因为增益（Gain）仅用于比较。以下是省略系数后的新增益公式：

$$Gain = [\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}] - \gamma$$

其中，G_L、G_R、H_L、H_R分别是 GL、GR、HL、HR 矩阵。

主动参与方可以根据此增益公式计算增益矩阵。增益矩阵的形状与 GL、GR、HL、HR 矩阵的形状相同，如图 11 所示。每一行对应当前层的一个结点，从每行中选取元素值最大的单元，即可得到对应结点的最优分裂点。

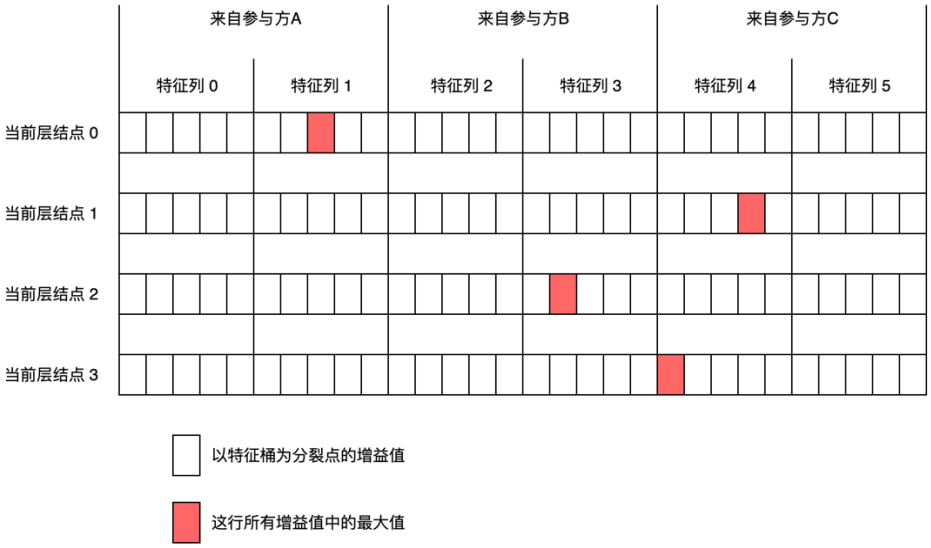

图 11 增益矩阵示例

对于每个结点，主动参与方可以采用以下原则来决策每个结点是否需要分裂：

a) 若$\gamma < 0$，则最优增益大于 0 的结点需要分裂；

b) 若$\gamma <= 0$，则所有结点都要分裂。

7.2.2.8 主动参与方向被动参与方同步决策结果

主动参与方在决策过程中获得当前层每个结点的最优分裂点后，需同步决策结果至其他参与方。具体内容包括当前层每个结点是否应继续分裂，以及每个结点的最优分裂点信息。

采用一维布尔数组表示当前层每个结点是否应继续分裂。例如 [True, True, True, False] 表示第一、二、三个结点应继续分裂，第四个结点不应分裂。有关一维布尔数组传输时的数据格式，请参阅 8.2.1 。

同时，采用一维整型数组表示当前层每个结点的最优分裂点索引。以图 11 中增益矩阵示例为例，红色方块表示计算出的最优分裂点，因此一维整型数组为 [7, 23, 16, 20]，其中元素值为红色方块在增益矩阵中的列索引。有关一维整型数组传输时的数据格式，请参阅 8.2.1 。

7.2.2.9 计算最优分裂点的本地桶索引

7.2.2.8 节结点最优分裂点列表中的每个元素都是一个索引值，是最优分裂点对应的桶的全局索引。各参与方需要根据 7.2.1.2 节在本地保存的全局总桶数列表（即 buckets_counts 数组），推导出最优分裂点对应的本地桶索引，如图 12 所示。例如，若 buckets_counts = [100, 120, 150]，最优分裂点列表中某个元素的值等于 190，则这个分裂点属于参与方 B，是参与方 B 的本地索引值为 90 的桶；对于参与方 A 和 参与方 C，本地不存在这个桶，因此本地索引值被标记为 -1。

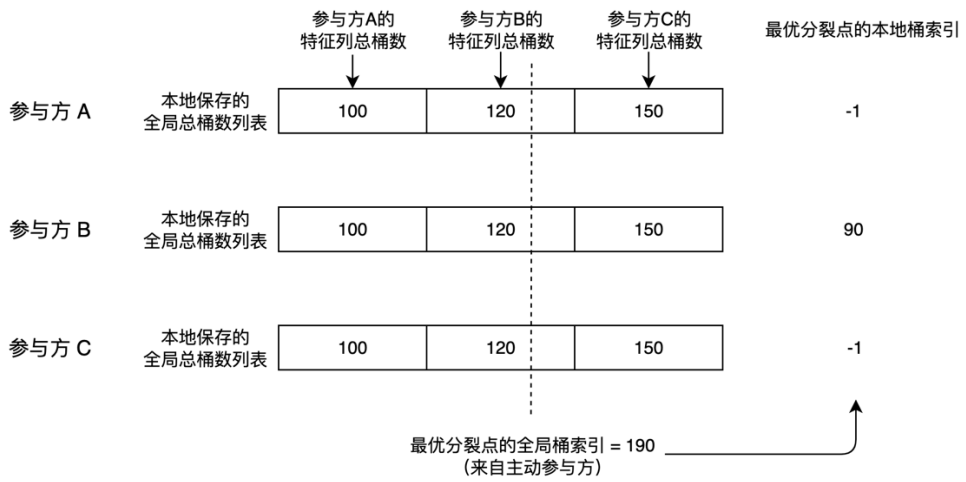

图 12 本地计算最优分裂点的本地桶索引

对于被动参与方，还需要参考图 9 中本地桶索引转换列表（reindex_list）记录的打乱结果，将最优分裂点的本地桶索引转换为打乱前的索引，作为最终的本体桶索引。具体而言：

a) 若本地桶索引值 i 等于 -1，则最终本地桶索引值为-1；

b) 若本地桶索引值 i 不等于 -1，则最终本地桶索引值为 reindex_list[i]。

7.2.2.10 计算最优分裂点的特征列桶索引

各参与方需要根据样本列采样结果和样本分桶信息，来计算最优分裂点属于本地的哪个特征列的，以及在该特征列上的桶索引。

计算过程如图 13 所示，假设参与方本地有 4 个特征列，每个特征列分成 50 个桶，子模型训练时采样了其中 3 个特征列，分别是第 1 列、第 3 列和第 4 列。以本地桶索引 80 为例，该索引对应于第 3 个特征列的第 30 个桶（80 - 50 = 30，不考虑第 2 个特征列）。

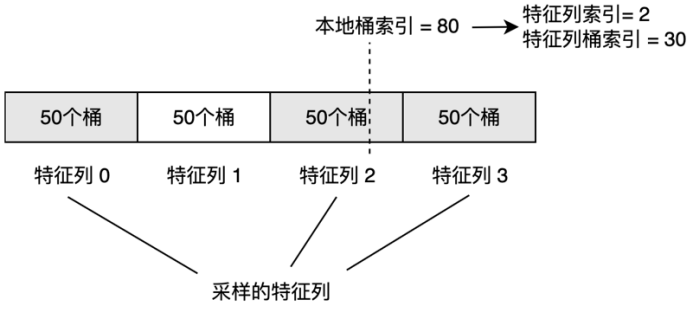

图 13 本地计算最优分裂点的特征列索引和在特征列上的桶索引

7.2.2.11 记录分裂点信息

对于当前层的每个结点，若需要分裂，各参与方将分别在本地记录该结点的分裂信息，如图 14 所示。结点分裂信息包括三个要素：特征列索引、分割值大小和结点索引。特征列索引通过 7.2.2.10 节计算得到。分割值是在 7.1.2 节样本分桶时确定的分割点数值，仅当

分裂点属于参与方自己时，才需要记录分割值大小。结点索引在主动参与方一侧维护，需要由主动参与方提前发送给被动参与方；用整数类型列表表示当前层结点的索引，其在传输时的数据格式详见 8.2.1　节。

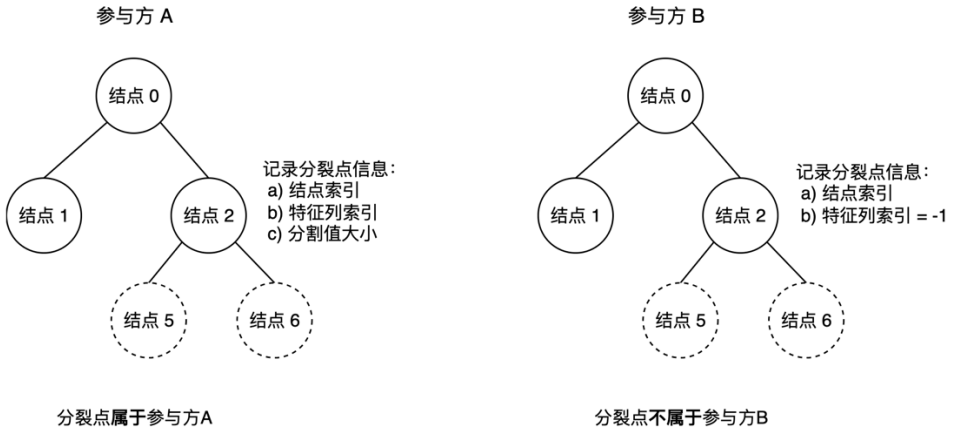

图 14 结点分裂示例

此外，由于子模型的第一层仅有一个根结点，而且根结点的结点索引固定为 0，因此在对第一层结点进行分裂时，主动参与方无需向被动参与方发送当前层结点索引列表。

7.2.2.12　计算新结点的样本位图

对于每个参与方，如果分裂点属于它自己，可以利用 7.2.2.10　节计算的分裂点信息（特征列和桶索引）以及 7.1.2　节的样本分桶矩阵，结合样本行采样结果，计算出结点分裂后左子节点和右子结点的样本位图。这里的样本位图是针对所有采样样本的位图，即将所有采样样本分配给了左子结点或右子结点。

然后各参与方将每个分裂点的左子结点的样本位图发送给主动参与方。这里的样本位图的传输格式与 7.2.2.4　节相似，唯一的不同之处在于：对于不属于参与方自己的分裂点，用空数组表示左子结点样本位图。例如，若当前层有两个分裂点，第一个分裂点的左子结点的样本位图是[0, 1, 0, 1, 1, 0, 0]，而第二个分裂点属于其他参与方，则传输给主动参与方的样本位图列表是 [[0, 1, 0, 1, 1, 0, 0], [ ]]。

主动参与方本地保存着当前层所有分裂点的样本位图。假设某个分裂点的样本位图等于 parent_bits,前面计算出它分裂后左子节点的样本位图等于 received_bits,则它的左子结点的实际样本位图为 left_bits = parent_bits * received_bits，右子结点的实际样本位图为 right_bits = parent_bits - left_bits。通过这种方式，下一层每个新结点的样本位图就都计算出来了，可以用作子模型下一层迭代的输入。

7.2.3　子模型迭代结束

7.2.3.1　子模型迭代结束条件

子模型迭代结束有两个判断条件：

a) 主动参与方判断这层所有结点都不需要分裂，并将判断结果发给其他参与方。判断结果用一个布尔类型的数值表示。布尔值传输时的数据格式详见 8.2.1 节。

b) 若树模型深度到达 max_depth，则各参与方同时结束训练。

若这两个条件都不满足，则继续进行树模型下一层训练。

7.2.3.2 记录叶子结点的信息

叶子结点的信息包含索引值和权重值。

叶子结点的索引值由主动参与方计算，然后发送给被动参与方。根据 7.2.2.11 节的分裂点信息和本节的叶子结点索引信息，各参与方可以在本地完整地描述子模型的结构。在子模型中，所有不再继续分裂的结点都被视为叶子结点，包含 7.2.2.7 节中决策的不应分裂的结点和 7.2.3.1 节中子模型迭代结束后最后一层的结点。用整数类型列表表示主动参与方向被动参与方传输的叶子结点索引列表，传输时的数据格式详见 8.2.1 节。

叶子结点的权重值由主动参与方计算，且仅由主动参与方持有。主动参与方根据叶子结点的样本位图，计算每个叶子结点的G_j和 H_j，然后使用以下公式计算每个叶子结点的权重。

$$w_j^* = -\frac{G_j}{H_j + \lambda} * learning\_rate$$

7.3 子模型训练结束

根据算法协商握手阶段确定的参数 num_round，判断是否结束模型训练。当子模型的个数等于 num_round 时，所有参与方结束模型训练；当子模型的个数小于 num_round 时，所有参与方继续训练下一个子模型。在训练下一个子模型之前，主动参与方需要更新针对样本数据的预测值，该预测值计算的过程详见 7.4 节。

7.4 预测

7.4.1 模型训练阶段的预测

根据 7.2 节训练得到的新子模型，可以计算出每个样本的权重。由于子模型是通过加法策略训练得到的（详见图 1），将这些样本权重加到上次计算的样本预测值上，就得到新的预测值。这些样本预测值用于计算下一个子模型的 GH 矩阵（详见 7.2.1.4 节）

在利用子模型计算每个样本的权重值时，需要根据子模型的结构确定样本可以被分配到子模型的哪个叶子结点上，而叶子结点的权重则是这个样本的权重。图 15 描述了如何将每个样本分配到子模型的叶子结点上。对于每个样本，从子模型的根节点开始向下寻找叶子结点。寻找方式为：若当前分裂点属于参与方自己（特征列索引不等于-1），则根据分割值的大小选择它的左子结点或右子结点；若当前分裂点不属于参与方自己（特征列索引等于-1），则它的左子结点和右子结点都会被选择；若当前结点不是分裂点，则这个叶子结点被选中。

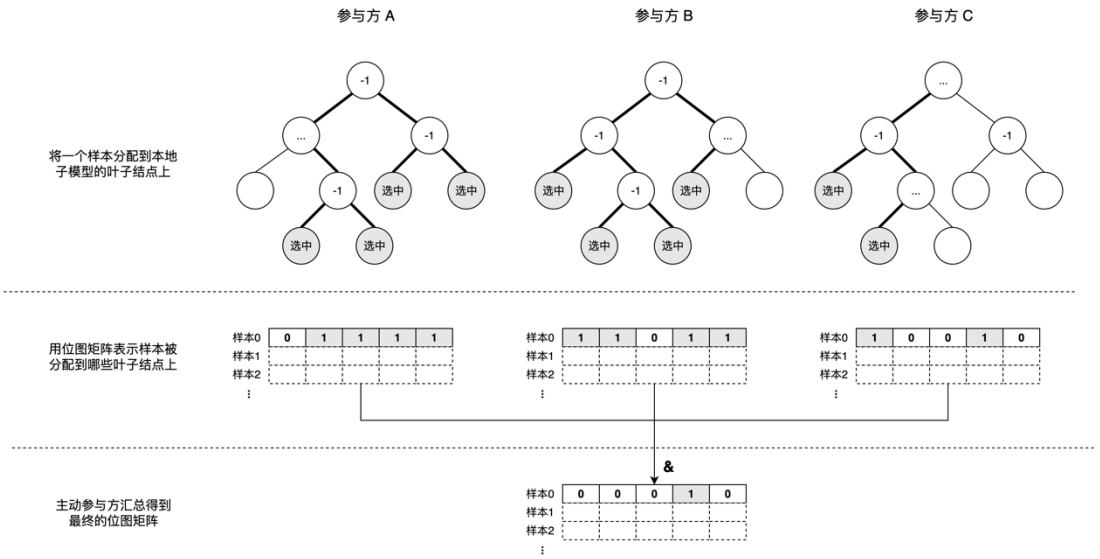

图 15 预测每个样本分配到子模型的哪个叶子结点

每个参与方在本地用一个位图矩阵表示每个样本选中了哪些叶子结点，如图 15 所示。位图矩阵的行数等于样本总数，列数等于子模型叶子结点的总数，矩阵元素值为 1 表示样本选中了对应的叶子结点。

被动参与方需要将本地计算的位图矩阵发送给主动参与方。样本位图采用与 7.2.2.4 节相同的传输方式。主动参与方收到的位图矩阵具有相同的行数和列数，将这些矩阵按元素相乘，得到最终的位图矩阵。最终位图矩阵的每行只有一个元素的值是 1，如图 15 所示，说明对应的样本被分配到了这个元素表示的叶子结点上。

7.4.2 模型使用阶段的预测

使用 SGB 模型的每个子模型对输入样本计算一次预测值，然后将这些预测值相加，得到最终的预测值。子模型的预测过程与 7.4.1 节中描述的过程相同。

8 数据通信格式

8.1 算法协商握手请求和响应

8.1.1 HandshakeRequest 消息

HandshakeRequest包括算法协商握手请求的基本信息，其数据结构如表1所示。

表 1 HandshakeRequest 数据结构

属性名称	数据类型	数据说明	示例	备注
version	int32	握手请求版本号	2	必选
requester_rank	int32	发送方在传输层的编号，预先给每个参与方分配好	1	必选

属性名称	数据类型	数据说明	示例	备注
supported_algos	int32 list	支持的开放协议的 enum 值，如 SGB	见表 2	必选
algo_params	google.protobuf.Any list	相应的算法详细握手参数，与 supported_algos 对应。SGB 算法的类型是 SgbParamsProposal	见表 3	必选
protocol_families	int32 list	支持的协议族的 enum 值，如 PHE 协议族	见表 4	必选
protocol_family_params	google.protobuf.Any list	相应的协议族详细握手参数，与 protocol_families 对应。实际类型随协议族类型而变，PHE 协议族的类型是 PheProtocolProposal	见表 5	必选

其中：

- supported_algos 列表元素的取值和说明如表 2 所示。

表 2 supported_algos 的值域表

取值	数据说明
1	ALGO_TYPE_ECDH_PSI，ECDH-PSI 算法
2	ALGO_TYPE_SS_LR，SS-LR 算法
3	ALGO_TYPE_SGB，SGB 算法

- SgbParamsProposal 包括 SGB 算法参数协商的基本信息，如表 3 所示。

表 3 SgbParamsProposal 数据结构

属性名称	数据类型	数据说明	示例	备注
supported_versions	int32 list	支持的版本列表	[1]	必选
support_completely_sgb	bool	训练第一棵树时是否仅采用主动参与方的样本特征列	true	必选
support_row_sample_by_tree	bool	是否支持行采样	true	必选
support_col_sample_by_tree	bool	是否支持列采样	true	必选

- protocol_families元素的取值和说明如表4所示。

表 4 protocol_families 元素的值域表

取值	数据说明
1	PROTOCOL_FAMILY_ECC，PSI 协议族
2	PROTOCOL_FAMILY_SS，SS 协议族
3	PROTOCOL_FAMILY_PHE，PHE 协议族

- PheProtocolProposal包括PHE协议族参数协商的基本信息，如表5。

表 5 PheProtocolProposal 数据结构

属性名称	数据类型	数据说明	示例	备注
supported_versions	int32 list	支持的版本列表	[1]	必选
supported_phe_algos	int32 list	支持的 PHE 算法种类的 enum 值，如 Paillier、Elgamal 等	见表 6	必选
supported_phe_params	google.protobuf.Any list	相应的 PHE 算法参数，与 supported_phe_algos 对应。Paillier 算法的类型是 PaillierParamsProposal，EC ElGamal 算法的类型是 EcElGamalProposal	见表 7、表 8	必选

- supported_phe_algos元素的取值和说明如表6所示。

表 6 supported_phe_algos 元素的值域表

取值	数据说明
1	PHE_ALGO_PAILLIER，Paillier 算法
3	PHE_ALGO_EC_ELGAMAL，EC ElGamal 算法

- PaillierParamsProposal包括Paillier算法参数协商的基本信息，如表7所示。

表 7 PaillierParamsProposal 数据结构

属性名称	数据类型	数据说明	示例	备注
key_sizes	int32 list	支持的密钥长度	[2048, 3072]	必选

- EcElGamalProposal包括 Exponential EC Elgamal 算法参数协商的基本信息，如表8所示。

表 8 EcElGamalProposal 数据结构

属性名称	数据类型	数据说明	示例	备注
curve_name	int32 list	支持的椭圆曲线编号	[2, 3]	必选

其中椭圆曲线的编号与曲线名称的对应关系如表9所示。

表 9 curve_name 元素的值域表

取值	数据说明
2	SM2
3	ED25519

8.1.2 HandshakeResponse 消息

HandshakeResponse消息结构包括算法协商握手响应的基本信息，其数据结构如表10所示。

表 10 HandshakeResponse 数据结构

属性名称	数据类型	数据说明	示例	备注
header	ResponseHeader	握手请求响应消息头	见表 11	必选
algo	int32	决策下来的算法的 enum 值，如 SGB	见表 2	必选
algo_param	google.protobuf.Any	决策下来的算法详细参数，实际类型随算法而变，SGB 算法的实际类型是 SgbParamsResult	见表 13	必选
protocol_families	int32 list	决策下来的协议族	见表 4	必选
protocol_family_params	google.protobuf.Any list	决策下来的协议族详细参数，实际类型随协议族而变，PHE 协议族的类型是 PheProtocolResult	见表 14	必选

其中：

- ResponseHeader消息结构包括的基本信息如表11所示。

表 11 ResponseHeader 数据结构

属性名称	数据类型	数据说明	备注
error_code	int	握手响应的结果，其值域如表12 所示。	必选
error_msg	string	用户自定的消息字符串	可选

- error_code的取值和说明如表12所示。

表 12 error_code 的值域表

错误码	数据说明
0	成功
31100000	GENERIC_ERROR，通用错误
31100001	UNEXPECTED_ERROR，状态不符合预期错误
31100002	NETWORK_ERROR，网络通信错误
31100100	INVALID_REQUEST，非法请求
31100101	INVALID_RESOURCE，运行资源不满足
31100200	HANDSHAKE_REFUSED，握手拒绝
31100201	UNSUPPORTED_VERSION，不支持的版本
31100202	UNSUPPORTED_ALGO，不支持的算法
31100203	UNSUPPORTED_PARAMS，不支持的算法参数

- SgbParamsResult包括SGB算法参数协商的基本信息，如表13所示。

表 13 SgbParamsResult 数据结构

属性名称	数据类型	数据说明	示例	备注
version	int32	算法版本	1	必选
num_round	int32	迭代次数	10	必选
max_depth	int32	树的最大深度	5	必选
row_sample_by_tree	double	树训练的行采样率	0.9	必选
col_sample_by_tree	double	树训练的列采样率	0.9	必选

属性名称	数据类型	数据说明	示例	备注
bucket_eps	double	样本分桶的 eps 参数	0.08	必选
use_completely_sgb	bool	训练第一棵树时是否仅采用主动参与方的样本特征列，即主动参与方单方训练	true	必选

- PheProtocolResult包括PHE协议族参数协商的基本信息，如表14所示。

表 14 PheProtocolResult 数据结构

属性名称	数据类型	数据说明	示例	备注
version	int32	算法版本	1	必选
phe_algo	int32	决策下来的 PHE 算法种类的 enum 值，如 Paillier 算法	见表 6	必选
phe_param	google.protobuf.Any	决策下来的 PHE 算法详细参数，实际类型随 PHE 算法而变，Paillier 算法的类型是 PaillierParamsResult，EC ElGamal 算法的类型是 EcElGamalResult	见表 15、表 16	必选

- PaillierParamsResult包括Paillier算法参数协商的基本信息，如表15所示。

表 15 PaillierParamsResult 数据结构

属性名称	数据类型	数据说明	示例	备注
key_size	int32	密钥长度	2048	必选

- EcElGamalResult包括Exponential EC Elgamal算法参数协商的基本信息，如表16所示。

表 16 EcElGamalResult 数据结构

属性名称	数据类型	数据说明	示例	备注
curve_name	int32	协商得到的椭圆曲线编号，编号对应的曲线名称见表 9	2	必选

8.2 算法主体运行时通信数据

8.2.1 DataExchangeProtocol 消息

在算法主体运行阶段，参与方之间通过DataExchangeProtocol消息进行通信数据的传送。DataExchangeProtocol消息的数据结构如表17所示。

表 17 DataExchangeProtocol 数据结构

属性名称	数据类型	数据说明	示例	备注
scalar_type	int32	container 结构中的标量的类型	见表 18	必选
scalar_type_name	string	当 scalar_type 是 object 类型时，用该字段描述类型的具体名字	“public_key”	可选
container	-	具体的数据结构	见表 19	必选

其中：

- scalar_type的值域如表18所示。

表 18 scalar_type 的值域表

取值	数据说明
1	SCALAR_TYPE_BOOL，bool 类型
2	SCALAR_TYPE_INT8，int8 类型
3	SCALAR_TYPE_UINT8，uint8 类型
4	SCALAR_TYPE_INT16，int16 类型
5	SCALAR_TYPE_UINT16，uint16 类型
6	SCALAR_TYPE_INT32，int32 类型
7	SCALAR_TYPE_UINT32，uint32 类型
8	SCALAR_TYPE_INT64，int64 类型
9	SCALAR_TYPE_UINT64，uint64 类型
10	SCALAR_TYPE_INT128，int128 类型
11	SCALAR_TYPE_UINT128，uint128 类型
15	SCALAR_TYPE_FLOAT16，float16 类型
16	SCALAR_TYPE_FLOAT32，float32 类型
17	SCALAR_TYPE_FLOAT64，float64 类型

取值	数据说明
20	SCALAR_TYPE_OBJECT，object 类型

- container的数据结构可以是表19所示数据结构中的任意一种。

表 19 container 可能的数据结构

container 名称	数据类型	数据说明	示例	备注
scalar	Scalar	单个标量	见表 20	选择其中一种
f_scalar_list	FScalarList	标量列表，每个元素的长度相等	见表 21	
v_scalar_list	VScalarList	标量列表，每个元素的长度不一定相等	见表 22	
f_ndarray	FNdArray	标量的 N 维数组，每个元素的长度相等	见表 23	
v_ndarray	VNdArray	标量的 N 维数组，每个元素的长度不一定相等	见表 24	
f_ndarray_list	FNdArrayList	N 维数组的列表，元素类型为 FNdArray	见表 25	
v_ndarray_list	VNdArrayList	N 维数组的列表，元素类型为 VNdArray	见表 26	

其中：

- Scalar 数据结构的属性如表 20 所示。

表 20 Scalar 数据结构

属性名称	数据类型	数据说明	备注
buf	bytes	原始字节序列，类型见表 18，字节序为小端序	必选

- FScalarList 数据结构的属性如表 21 所示。

表 21 FScalarList 数据结构

属性名称	数据类型	数据说明	备注
item_count	int64	元素数量	必选
item_buf	bytes	元素地址连续的原始字节序列。元素类型见表 18，字节序为小端序	必选

- VScalarList 数据结构的属性如表 22 所示。

表 22 VScalarList 数据结构

属性名称	数据类型	数据说明	备注
items	bytes list	每个元素的原始字节序列。元素类型见表 18，字节序为小端序	必选

- FNdArray 数据结构的属性如表 23 所示。

表 23 FNdArray 数据结构

属性名称	数据类型	数据说明	备注
shape	int64 list	形状	必选
item_buf	bytes	元素地址连续的原始字节序列。元素内存布局为行主序，元素类型见表 18，字节序为小端序	必选

- VNdArray 数据结构的属性如表 24 所示。

表 24 VNdArray 数据结构

属性名称	数据类型	数据说明	备注
shape	int64 list	形状	必选
items	bytes list	每个元素的原始字节序列。元素顺序为行主序，元素类型见表 18，字节序为小端序	必选

- FNdArrayList 数据结构的属性如表 25 所示。

表 25 FNdArrayList 数据结构

属性名称	数据类型	数据说明	备注
ndarrays	FNdArray list	每个元素的类型都是 FNdArray	必选

- VNdArrayList 数据结构的属性如表 26 所示。

表 26 VNdArrayList 数据结构

属性名称	数据类型	数据说明	备注
ndarrays	VNdArray list	每个元素的类型都是 VNdArray	必选

8.2.2 同态加密密文数据

8.2.2.1 公钥

同态加密的公钥同样是通过DataExchangeProtocol消息（见8.2.1 节）传送的，如表27所示。

表 27 公钥的 DataExchangeProtocol 消息实例

属性名称	取值	取值说明	备注
scalar_type	SCALAR_TYPE_OBJECT	见表 18	-
scalar_type_name	“paillier_public_key” 或者 “ec_elgamal_public_key”	-	根据实际所用算法选择
container	Scalar 结构	见表 20	Scalar.buf 保存公钥序列化之后的二进制数据

其中，Scalar结构中的buf字段表示公钥的protobuf协议字节序列，具体的数据格式因算法的不同而不同。如果握手协商选定的是 Paillier，则 PublicKey 属性如表28所示，如果握手协商选定的是 EC ElGamal，则PublicKey属性如表30所示。

表 28 Paillier PublicKey 数据结构

属性名称	数据类型	数据说明	备注
n	Bigint	含义见附 录 A	必选
hs	Bigint	含义见附 录 A	必选

其中，Bigint表示大整数类型的数据结构，其属性如表29所示。

表 29 Bigint 数据结构

属性名称	数据类型	数据说明	备注
is_neg	bool	是否是负整数	必选
little_endian_value	bytes	以小端序表示的绝对值的原始字节	必选

属性名称	数据类型	数据说明	备注
		序列	

表 30 EC ElGamal PublicKey 数据结构

属性名称	数据类型	数据说明	备注
curve_name	int32	椭圆曲线编号，编号的含义见表 9	必选
h	bytes	存储以 X962_COMPRESSED 格式序列化之后的点，含义见附 录 A	必选

8.2.2.2 密文矩阵

同态加密的密文矩阵也是通过DataExchangeProtocol消息（见8.2.1 节）传送的，如表31所示。

表 31 密文矩阵的 DataExchangeProtocol 消息实例

属性名称	取值	取值说明	备注
scalar_type	SCALAR_TYPE_OBJECT	见表 18	-
scalar_type_name	“paillier_ciphertext” 或者 “ec_elgamal_ciphertext”	-	根据实际所用算法选择
container	VNdArray 结构	见表 24	VNdArray.items 依次保存每个密文元素序列化之后的二进制数据

其中，VNdArray结构中的items字段表示每个密文的protobuf协议字节序列，具体的密文格式与握手协商选定的同态加密算法有关，如果加密算法为 Paillier，则密文属性如表32所示；如果加密算法为 EC ElGamal，则密文属性如表33所示。

表 32 Paillier Ciphertext 数据结构

属性名称	数据类型	数据说明	备注
c	Bigint	含义见附 录 A	必选

表 33 EC ElGamal Ciphertext 数据结构

属性名称	数据类型	数据说明	备注
c1	bytes	存储以 X962_COMPRESSED 格式序列化之后的点，含义见附　录　A	必选
c2	bytes	存储以 X962_COMPRESSED 格式序列化之后的点，含义见附　录　A	必选

9　传输层实现参考

9.1　通信框架

异构隐私计算技术平台间进行互联互通时，通信框架需能满足兼容性、通用性以及安全性的要求。通信框架与编码方式如表34所示。

表 34 通信框架范围

RPC框架	GRPC
编码方式	ProtoBuf

9.2　初始化通信协议

初始化通信协议在 SGB 任务开始前执行一次。

每个参与者向其它参与者通知自己的存在性，即向他人发送 connect_{self_rank}：

For i in 0..word_size [注1]:

　if i == self_rank:

　　continue

　P2P send to rank i: {key: connect_{self_rank}, value: ""}

每个参与者检查他人的存在性，即依次检查 connect_{rank} 消息已经收到：

For i in 0..word_size:

　if i == self_rank:

　　continue

　P2P receive on key connect_{i}

注 1：word_size 表示参与者数量

9.3　Protobuf 消息

不同隐私计算的参与者之间使用 Protobuf 协议传递信息。

```
service ReceiverService
{
```

rpc Push(PushRequest) returns (PushResponse);
}

9.3.1 PushRequest 消息结构

PushRequest包括传输的基本信息，其数据结构如表35所示。

表 35 PushRequest 数据结构

属性名称	数据类型	数据说明	示例	备注
sender_rank	uint64	发送者的编号，如 0 指 rank	0	必选
key	string	消息唯一 ID，生成规则见 9.4 节	“root:P2P-0:0->1”	必选
value	bytes	消息体，ECDH-PSI 中的 protobuf 序列化成二进制 string，然后把整个 string 放到 value 中	需要传输的实际信息	必选
trans_type	TransType	传输模式，如全量传输、分块传输，其值域如表 36 所示	见表 36	必选
chunk_info	ChunkInfo	消息大小，其数据结构如表 37 所示	见表 37	必选

其中：

- TransType 的值域如表 36 所示。

表 36 TransType 的值域表

数值	数据说明
MONO	全量传送模式
CHUNKED	分块传送模式

- ChunkInfo 的值域如表 37 所示。

表 37 ChunkInfo 的数据结构

属性名称	数据类型	数据说明		备注
message_length	uint64	数据总大小，单位是字节 Byte	1048576	必选
chunk_offset	uint64	当前分块的偏移量	0	必选

9.3.2 PushResponse 消息结构

PushResponse包括传输的基本信息，其数据结构如表38所示。

表 38 PushResponse 消息的数据结构

属性名称	数据类型	数据说明	备注
header	ResponseHeader	返回消息，如表 11 所示	必选

9.4 通信模式

9.4.1 信道

信道是一个逻辑概念，用于区分通信的上下文。每一个信道有一个全局唯一名称，命名规则为：\w+，即信号名称由字母、数字、下划线组成。信道的名字由通信组双方约定，在初始化阶段由用户传入。

信道唯一的作用就是影响 message key 的生成，信道名称会作为 message key 一部分，因此，不同信道中的消息一定不会有相同的 key，因此不同信道的消息在逻辑上不会混淆。

当上层算法需要多个信道时，第一个信道称为主信道，其它信道称为子信道。子信道的命名规则为：主信道名称-子信道编号。

举例：假设主信道名称为 root，则 0 号子信道名称为 root-0，1 号子信道名称为 root-1，以此类推。

9.4.2 P2P 通信

P2P 通信允许在任意两个参与者之间发送信息。P2P 通信key的命名规则为：{信道名称}:P2P-{计数器}:{发送者 RANK}->{接收者 RANK}，其中每一个信道、每一对 <sender，receiver> 都有一个独立的计数器。

举例：假设信道名称为 root，以下消息依次发送：

Rank 0 → 1 发送消息，key 为：root:P2P-0:0->1

Rank 1 → 0 发送消息，key 为：root:P2P-0:1->0

Rank 0 → 2 发送消息，key 为：root:P2P-0:0->2

Rank 0 → 1 发送消息，key 为：root:P2P-1:0->1

附 录 A
（资料性）
同态加密算法

Secure Gradient Boost 算法主体部分的 GH 矩阵需要使用同态加密，本附录展示两种推荐的同态加密算法 Paillier 和 Exponential EC Elgamal 的算法步骤。

A.1 Paillier crypto system with DJN optimization

密钥生成:

- 选择一个合适的模数 $n = pq$，其中 n 的比特数为 k，满足 $p \equiv q \equiv 3 \pmod 4$ 并且 $gcd(p-1, q-1) = 2$
- 随机选择 $x \leftarrow \mathbb{Z}_n^*$，计算 $h = -x^2$ 和 $h_s = h^n \mod n^2$
- 计算 $\lambda = (p-1)(q-1)/2$
- 公钥是 (n, h_s)，私钥是 λ

加密 (pk, m):

- 随机生成掩码 $r \leftarrow \mathbb{Z}_{2^{k/2}}$ 其中 k 表示 n 的比特长度
- 计算密文 $c = (1 + mn)h_s^r \mod n^2$

解密 (sk, C):

- 计算 $\mu = \lambda^{-1} \mod n$
- 计算明文 $m = L(c^\lambda \mod n^2) \cdot \mu \mod n$，其中 L 表示函数 $L(x) = \frac{x-1}{n}$

密文加法 (c_1, c_2):

- 计算 $c = c_1 \cdot c_2 \mod n^2$

密文减法 (c_1, c_2):

- 计算 $c = \frac{c_1}{c_2} \mod n^2$

A.2 Exponential EC Elgamal crypto system

参数说明：

- E 是定义在 $\mathbb{F}_p$ 上的椭圆曲线
- G 是 E 的生成元

密钥生成(1^λ):

- 随机选取$x \leftarrow \mathbb{Z}_p$

- 计算 $h := xG$
- 公钥是 (E, h) ，私钥是 x

加密 (pk, m):
- 随机生成掩码 $r \leftarrow \mathbb{Z}_p$
- 计算 $c_1 = rG$
- 计算 $c_2 = mG + rh$
- 密文是 (c_1, c_2)

解密 (sk, ciphertext):
- 计算 $M = mG = c_2 - xc_1$
- 通过解 ECDLP 问题从 M 中恢复 m，一般需要预计算表

密文加法 (C_A, C_B):
- 输出 $(C_{A1} + C_{B1}, C_{A2} + C_{B2})$

密文减法(C_A, C_B):
- 输出 $(C_{A1} - C_{B1}, C_{A2} - C_{B2})$

参 考 文 献

[1] Chen, T., and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, 785–794. ACM.

[2] Cheng, K., Fan, T., Jin, Y., et al. (2019). Secureboost: A lossless federated learning framework. arXiv preprint arXiv:1901.08755.

[3] Weng, H., Zhang, J., Ma, X., et al. (2022). Practical Privacy Attacks on Vertical Federated Learning. arXiv preprint arXiv:2011.09290.

[4] Jurik, M. (2003). Extensions to the paillier cryptosystem with applications to cryptological protocols. Brics, August. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.4.2396&rep=rep1&type=pdf

[5] Damgård, I., Jurik, M., & Nielsen, J. B. (2010). A generalization of Paillier's public-key system with applications to electronic voting. International Journal of Information Security, 9(6), 371–385. https://doi.org/10.1007/s10207-010-0119-9

[6] Chatzigiannis, P., Chalkias, K., & Nikolaenko, V. (2022). Homomorphic Decryption in Blockchains via Compressed Discrete-Log Lookup Tables. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 13140 LNCS, 328–339. https://doi.org/10.1007/978-3-030-93944-1_23

[7] Bauer, J. (2004). Elliptic EcGroup Cryptography Tutorial. 1–13.

[8] Logarithms, D. (1976). a Public Key Cryptosystem and a Signature based on Discrete Logarithms. System, 10–18.