WPPCC-基因赛道

方案设计文档

**队长**

韩培义

**队员**

段少明、郭蕴哲、靳小鹏

（按姓氏拼音排序）

2022年1月

# 技术路线

在本比赛中，我们解决**T1 + PG1 + PP1**场景下的新冠病毒进化树构建任务。具体技术场景如下：

* **T1**：甲乙双方采用同态加密隐私计算技术；
* **PP1**：甲乙双方基因序列对输入前已经完成对齐(alignment)，大赛主办方预先采用成熟的软件mafft进行比对，确保序列对齐的准确性，然后进行数据截取；
* **PG1**：任一参与方的基因序列集数据明文不能被其他参与方以截获的方式获知。但是甲乙双方的序列之间的相似矩阵可以作为结果的一部分输出，结果NJ树可以保留每条边的权重信息。

# 方案概述

在本方案中，我们使用了同态加密技术完成隐私计算过程。使用已经完成对齐的基因序列数据集，使用同态加密技术，在互补泄漏双方明文数据集的前提下，计算双方持有基因数据的距离（pdistance），并使用基因序列距离结果和ID名称，构建基因数据所对应的相似性矩阵，最后以相似性矩阵作为输入，构建出新冠病毒的进化树。

在下文中，为了能让大赛评委清晰了解本文所述技术方案，我们称两个计算方为甲方与乙方，其中：

* 甲方获取最终的结果；
* 甲方拥有私钥，并且将公钥和加密的基因数据发送给乙方；
* 乙方在本地利用甲方的加密数据和自己持有的数据，进行运算得到加密结果，发送回甲方；
* 甲方解密乙方的结果，得到矩阵。

由此，在上述计算过程中，甲方发给乙方和乙方传回甲方的内容，均不含双方持有的基因明文数据，双方也无法通过传输内容，推知或反解出明文的基因序列内容，具体描述将在后续章节中进行分析。

# 基因距离度量的同态改造思路

## 同态加密的基本思路

首先，同态加密运算包含：加、减法（其实还是加法）、乘三种运算，根据全同态和半同态之分，乘又分为数乘和同态乘法，但是同态加密运算不包含比较：

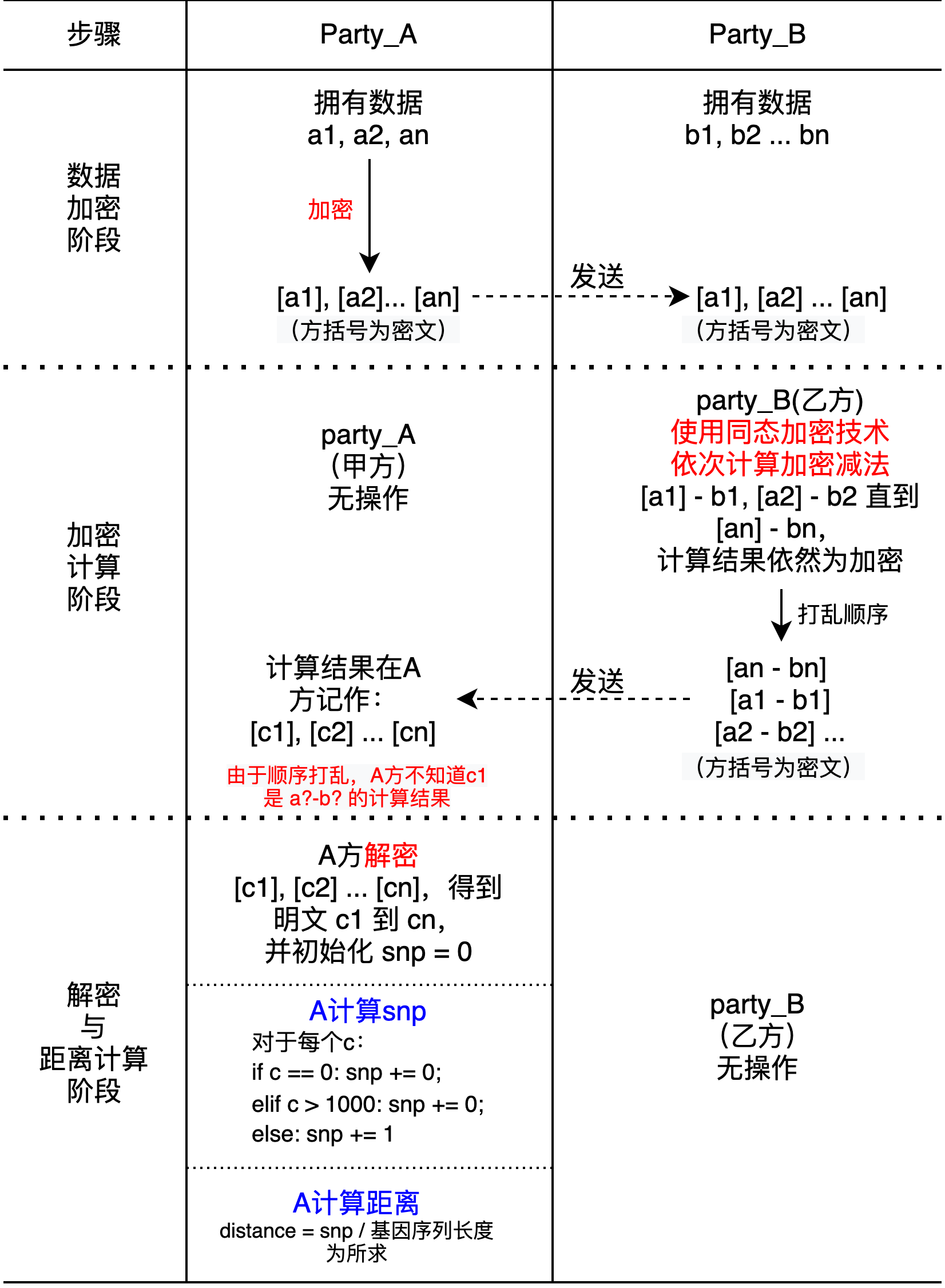
* 密文和明文之间无法比较：如果密文和明文可以比较，那么加密在安全性上就没有任何意义；
* 密文和密文之间无法比较：即使用相同的明文、相同的公钥，分别加密同两个相同的明文，得到的密文也是不同的。

那么现在有两个数A和B，直接想到的比较方式，就是**减法或者除法**：

* 如果 A-B == 0，那么两个数相等；
* 如果 A/B == 1，那么两个数相等； 这两种方法遇到的后续问题是类似的，由于减法的运算比除法简单，我们以减法为例子进行推进，除法应该也可能存在相应的解决方案。

## 从基因序列比较入手

我们的总体方法如下图所示，在展示总体方法的步骤之后，我们介绍设计此方法的思路：



我们首先需要了解基因比较（计算距离）函数的逻辑，这是比较函数的源码，如下图所示：

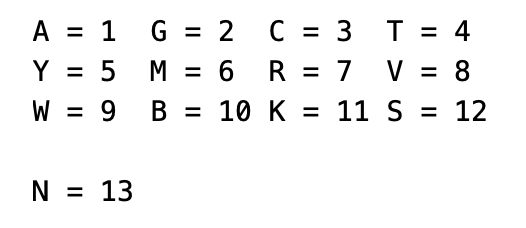
一張含有 文字 的圖片

自動產生的描述

梳理之后发现，基因序列的比较需要满足以下**三个原则**：

1. 两个序列都是有效碱基，且碱基相同，则 snp 不变，或者 snp += 0；
2. 两个序列都是有效碱基，且碱基不同，则 snp += 1；
3. 如果两个序列有任何一个碱基位置是缺失，则 snp 不累加；

而作为基因序列，碱基都是以字符形式出现的，但是作为同态加密的输入数据，肯定需要编码为数字，我们初步对基因进行如下编码（以下编码针对L1/目录中的fasta文件，L3中还存在新的碱基，并未在下图中显示），编码方式如下图所示：



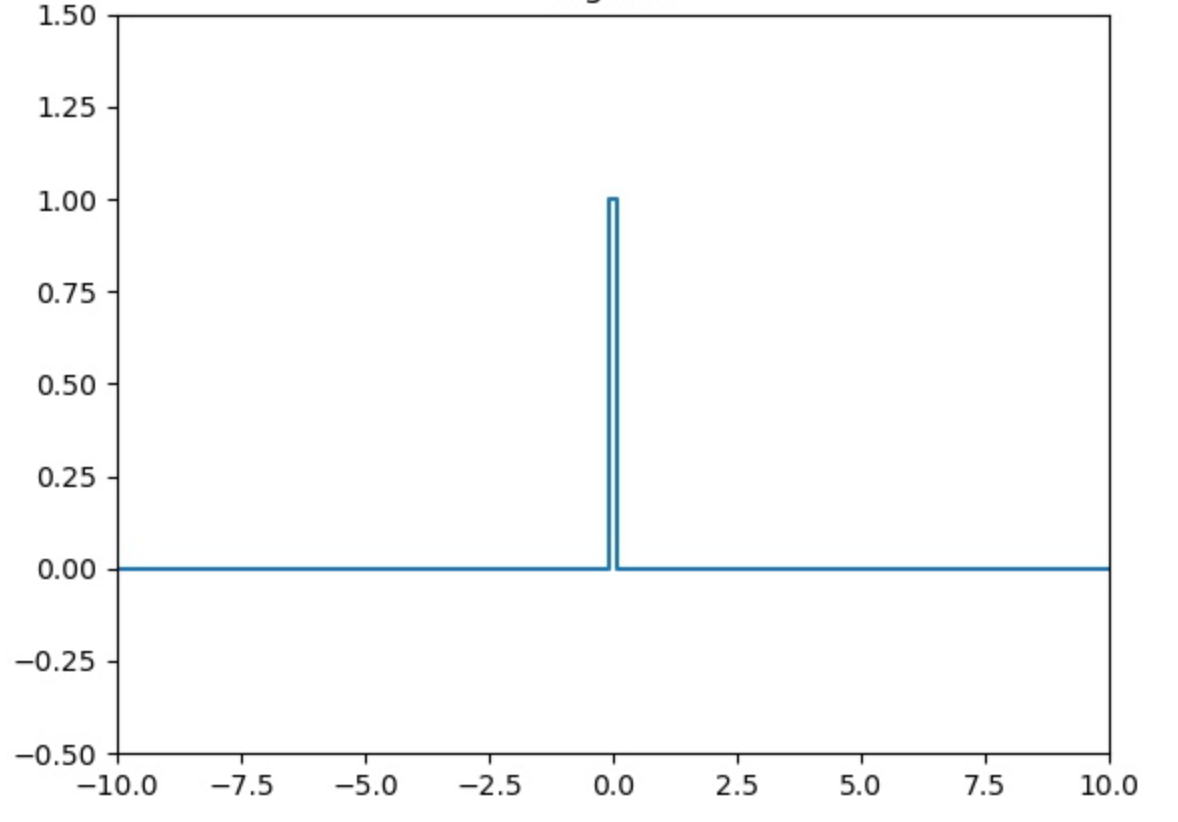
上面这种情况的编码可以完成原则1，但是不能完成原则2、3，举个例子：

* A 和 A 比较，res = 1 - 1 = 0，snp + 0 还是 snp；N 和 N 比较同理；
* T 和 A 比较，res = 4 - 1 = 3，snp + 3 会引发计算错误；
* N 和 A 比较，res = 13 - 1 = 12，snp + 12 ，但这里 res 应该为 0；

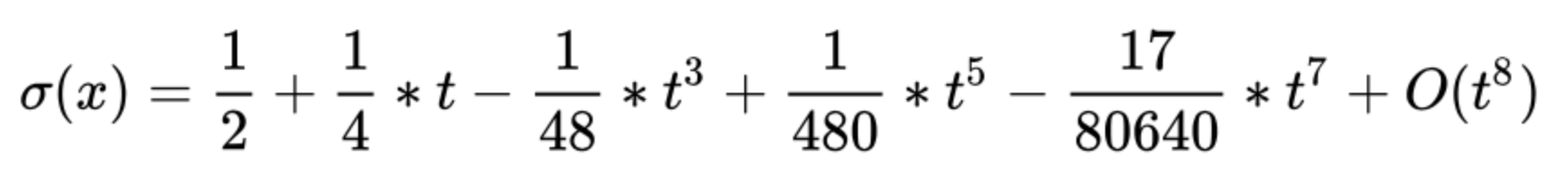
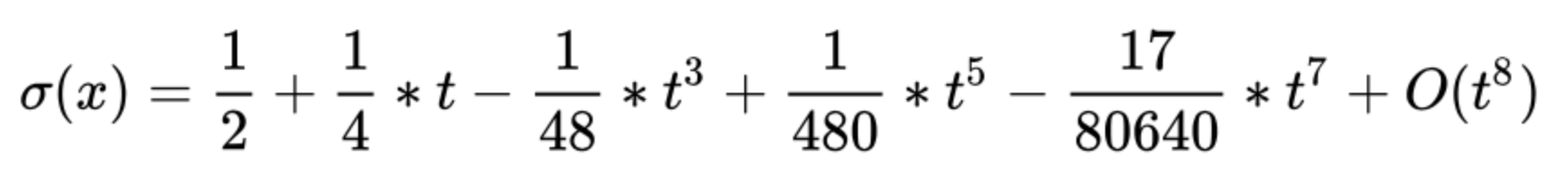
## 分析需要解决的问题

对于原则1，目前已经可以通过减法运算解决；

对于原则2，我们理想情况应该是当 res != 0 时候，res 都变成 1 ，计入 snp 的计算中，即只要 res!= 0 的时候， snp += 1，面对这个情况，我们有两个解决方案，一是**使用多个Sigmoid函数**（存在泰勒展开，可以展开为乘法和有限次加法的表达式）进行组合，模拟如下图的函数效果：



Sigmoid函数的泰勒展开如下图所示：



二是可以选择为了减少计算量，我们可以将每个位置的碱基的相减结果，**在解密之后进行逻辑判断**，同时为了防止数据泄漏，可以将两个被比较的基因序列中，每个位置碱基相减之后的结果的顺序**打乱**，从而无法做到反推，不泄漏乙方原始数据。

对于原则3，如果两个序列有任何一个碱基位置是缺失，则 snp 不累加；乙在运算时候，是知道自己哪些数据是N（缺失），直接对甲方的当前的数据乘0即可。现在剩下一种情况，甲方已经加密的缺失值N = 13，在减法的情况下，乙是不能做预先判断的，在对应位置不为N时候，N – A = 12，按照原则2解密之后仍然会让snp + 1，但是此种情况下，snp不应该 +1。考虑到前面的思路，我们可以使用Sigmoid或者解密之后进行逻辑判断，而 N 的编码值是可以定义的，我们可以把N的编码值根据需求变更，例如： N = 1012，在 N = 1012 的时候，对方是任何基因，那么相减的数值（或绝对值）都大于1000，引入第三个 Sigmoid 函数或或者也可以采取逻辑判断的方式，将snp需要增加的值削减为0即可完成所需逻辑，如下图所示：

一張含有 文字 的圖片

自動產生的描述

在真正的代码实现中，由于在L3中发现了其他的碱基，我们最终将N的编码定为了 1014。

## 技术路线的选择

至此，我们已经基本勾勒出了一个可以使用同态加密技术实现基因距离度量的方案，我们面临了两个技术选择：

* 使用全同态加密技术，用模拟Sigmoid的方式完成基因距离度量运算，此时可以得到加密的snp结果，加密结果可以用于后续PG2条件中NJ树的计算；
* 可以将上述需要Sigmoid函数的部分使用解密 + 逻辑运算的方式完成，由此可以减少计算量，同时满足PG1所需安全需求，在此文档中，我们使用了这种方法。

经过梳理之后，所需的同态运算为一**次密文与明文的减法（sub\_plain）操作**，而不需要密文与密文之间的加法、乘法操作，所以在算法选择上，我们选择**半同态加密算法（paillier）**，而非全同态加密算法，用以节省计算量，同时保证运算精度，**不产生精度损失**，**计算结果和明文计算一致**。

对于全同态 + PG2的实现方案，由于时间原因，我们并没有完成。

# 其他工程改进

我们注意到，由于碱基的数量是有限的，而所有基因序列，都是十几个碱基的组合，所以我们对加密流程进行了一些优化：从直接加密整个序列，改进为加密碱基，在加密过程中，用碱基加密后的密文替代碱基的明文。

请注意，此处每个碱基如果只对应一个密文，那么整个加密机制则类似“单表替代密码体制”，容易遭受到密码分析攻击，在工程实施上，对于每个碱基，我们会生成each\_cnt个密文（工程实现中赋值为42），形成一个哈希表（程序中命名为HashMap），这个哈希表的键是碱基的编码数值，键对应的值是多个密文组成的列表，在执行加密替换操作中，每次以正态分布随机选择密文的序号进行替换，而非每个密文机会均等，该操作破坏了单表替代密码体制存在的特征，使分析攻击难以进行。

# 安全性分析

我们在本章节回答四个问题：

## 解密碱基的相减结果是否会引起信息泄漏？

对于碱基相减的结果，例如x = Base\_A – Base\_B，在已知Base\_A与x 的情况下，是可以反推出Base\_B的，但是在我们的方案中，会把碱基相减的列表进行顺序打乱，当顺序打乱时，不影响 snp 的计算，但会让甲方无法辨每个碱基对应的乙所持有的敏感数据。故不会引起信息泄漏。

如果还需增加安全措施，可针对每次碱基相减结果乘一随机数，使得碱基相减的结果不为整数，完全无法推断内容，同时会对上述原则3的解决方法略有影响（所乘随机数范围在3.14-10时候，判断绝对值则应该为小于1000 \* 3.14，依此类推）。该增强方案在编码中也进行了实现，但默认不开启。可通过取消注释部分代码开启。

## N = 1014 的编码方式是否会引起信息泄漏？

首先乙只能看到加密数据，所以甲的基因序列所有敏感信息均不会泄漏。

当甲乙碱基均为为N（缺失）时候，或者乙的碱基为N时，结果为0，与双方碱基都为AGCT或其他碱基时的相减结果相同；

当乙的碱基不为N，而甲的碱基为N时候，甲在解密时候会得到大于1000的绝对值，甲推断出该打乱后位置中，甲的碱基缺失，作为甲来说无意义（因为甲知道自己基因序列的明文）。

甲可以计算自己缺失碱基的数量（N的数量）与解密大于1000的结果数量的差，该差是甲乙两方同时为N的碱基数量，但不能用于反推任何敏感信息。由于当前赛道是半诚实假设，甲不能恶意更改数据用来探测乙的敏感信息，该变量在当前假设下，无法泄漏更多信息，也无法反推乙的明文基因序列。

此外，由于减少了加密操作，全量数据集的加密操作的时间得以大幅下降。

## 密钥安全性

我们提供3072和4096两种密钥长度，均符合128-bit安全条件，提交版本中使用3072位，可以通过更改参数实现4096位（甚至更多位）密钥。在工程实现中，密钥的设定位于solution/utils/crypto.py 12行的 generate\_key 方法，密钥长度作为方法参数传入，并最终作为 python 的同态加密库 phe 中的 paillier.generate\_paillier\_keypair 方法的输入参数，如下图所示：

文本

描述已自动生成

而generate\_key函数在 proc.py 中15行，属于A方的运行函数，在 key\_gen 函数中首先被调用，此处key\_gen方法的参数 key\_size 会直接传入 generate\_key方法，如下图所示：

图形用户界面, 文本, 应用程序

描述已自动生成

在程序入口文件 start.py 第42行，可以看到以3072作为密钥长度传入参数，如下图所示：

文本

描述已自动生成

## 工程优化是否降低了安全性？

对于每个碱基，我们生成了each\_cnt个不同的加密碱基，并在替换过程中进行随机选择，提交程序中，each\_cnt变量赋值为42，即15个碱基对应了630个秘文，暴力破解情况下计算量为 630 ^ 15，符合128-bit安全条件。

# 性能测试

经大赛组委会工作人员提示，目前在程序执行过程中新增了计时选项，可以在程序运行过程中，打印密钥生成时间、加密时间、密文上的计算时间、解密时间时间，由于对数据没有预处理，所以没有打印预处理时间，打印情况如下图红框中所示：

一張含有 文字 的圖片

自動產生的描述

另外，针对通信量的统计也在程序中进行打印，如下图所示：

一張含有 文字 的圖片

自動產生的描述

使用阿里云ecs的4核心实例进行测试，结果如下，请注意加密操作指的是对全量数据完成加密的所需时间，单位为秒；加密计算与解密的单位不同，分别为**条/秒**和**秒/条**，也就是加密计算速度较快，一秒钟可以执行若干条基因序列的加密比较运算，解密操作速度较快;一条基因距离数据的解密时间大概为若干秒，具体结果如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集** | **加密操作（全量）** | **加密计算（条/秒）** | **解密操作(秒/条)** |
| L1 | 25秒 | 15.80条/秒 | 5.42秒/条 |
| L2 | 30秒 | 1.77条/秒 | 15.09秒/条 |
| L3 | 7分23秒 | 0.34条/秒 | 126.93秒/条 |

在内存管理中，涉及加密的变量，会在完成加密操作后持久化到硬盘中，解密时从硬盘读入再解密，密文则会在解密之后释放。在L1数据集执行中，全程内存占用在140M左右。

# 结束语

我们使用半同态加密（paillier），实现了**T1 + PG1 + PP1**场景下的，不存在精度损失的新冠病毒进化树构建任务。在本次比赛中，对基因数据处理、同态加密、进化树构建等内容有了新的认识，期望本次竞赛的所学能在未来对某一领域有所贡献，我们感谢大赛组委会提供的数据和宝贵机会！