# 第8章 密文集合运算

学习要求：了解密文集合运算的概念，重点掌握隐私集合求交、隐私交集求势问题的基本概念，了解隐私集合求并等概念；学习可交换加密、布隆过滤器、不经意伪随机函数等底层技术和密码原语；了解基于朴素哈希的PSI方案不安全的原因，掌握基于可交换加密的方案构造；掌握基于布隆过滤器进行通信优化的思想；掌握基于不经意伪随机函数的方案构造，了解布谷鸟哈希的概念和特征；完成应用实践部分的任务，复现所学方案；完成课后习题。

课时：2课时

建议授课进度：[8.1~8.2.2]、[8.2.3~8.3]

## 基本概念

### 8.1.1 定义及分类

#### 1. 密文集合交集运算

隐私集合求交（Private Set Intersection，PSI）允许持有各自集合的两方进行两个集合的交集计算，协议最后的结果是一方或是两方得到正确的交集内容，并且保证不会得到交集以外另一方集合中的任何信息。

PSI问题可以描述为：S拥有一个集合，R拥有一个集合，两方计算交集的元素，Alice不能获得集合的元素，Bob也不能获得集合的元素。 PSI分为两类场景：平衡PSI和非平衡PSI。其中，平衡PSI中两方集合数量大体相等，非平衡PSI中两方集合元素个数差异很大。

隐私交集求势（Private Set Intersection Cardinality，PSI-CA）允许两方求得交集集合元素的个数。

如图8-1所示，PSI的结果为，PSI-CA的结果为2。



图8-1 PSI和PSI-CA示意图

#### 2. 其他密文集合运算

隐私集合求并（Private Set Union，PSU）允许在不泄露任何信息的情况下，求解多方集合所有不相等元素的集合。

隐私集合求并的势（Private Set Union Cardinality，PSU-CA）允许在不泄露任何信息的情况下，多方集合所有不相等元素的个数，即并集的势。

### 8.1.2 可交换加密

#### 1. 定义

可交换加密是一种特殊的加密系统，不同的用户可以使用各自的加密密钥对数据进行多次加密，而解密时私钥的使用顺序可以和加密时公钥的使用顺序不同。即加密和解密的顺序不影响最终的运算结果。

**定义8-1（可交换加密）** 表示明文域，表示密钥域。可交换加密可用公式(8-1)表述：

(8-1)

其中表示一组双射。可交换加密性质可以用公式(8-2)表示：

(8-2)

其中属于明文域，属于密钥域。

#### 2.典型算法

一个常见的可交换加密算法为Pohlig–Hellman加密算法[[1]](#footnote-1)。下文对Pohlig–Hellman加密算法的原理进行具体介绍。

对于一个素数，根据费马小定理（Fermat's little theorem），有

因而，对于所有的整数，有：

根据上述等式，可以构建一个密码系统。令和表示明文信息、密钥和密文信息，其中

在实际应用中，明文通常会被定义为比特的整数：

因而保证了一定存在解密密钥满足：

其中满足：

在该加密算法中，加密运算为：

解密运算为：

至此，完成了Pohlig–Hellman密码系统的构建。根据上述密码算法的定义，该算法显然满足可交换加密的性质。对于两个不同的密钥和，满足

即加密的顺序不影响最终的结果。由于解密与加密使用的是同样的运算，因此解密操作同样不受加密顺序的影响，即：

加密运算和解密运算都只涉及幂指数运算，可以在不超过次乘法取模运算后计算出结果。通过加密密钥计算出解密密钥的运算只需执行一次并且可以离线执行。具体的运算需要执行扩展欧几里得算法。

一个需要注意的问题是，在通常的密码体系中，密钥为只需为到中的任意整数即可，但在该算法中，还应满足：

根据欧拉函数

，其中为所有质因子的集合，当取

，其中仍是一个质数，此时满足

而当很大时，根据欧拉函数，满足此条件的的比例为

另一方面，

，从而可以得出，当很大时，满足条件的的比例约为二分之一。因此一个满足上述限制条件的并不难选取。

### 8.1.3 布隆过滤器

布隆过滤器[[2]](#footnote-2)（Bloom Filter）是一个基于概率的用于元素包含检测的压缩数据结构。布隆过滤器使用一个包含个比特的数组来表示最多包含个元素的集合。布隆过滤器包含个相互独立的哈希函数每个哈希函数均将集合中的元素均匀的映射到从到的下标中。布隆过滤器中包含4个参数，在下文中，使用来表示一个数组长度为个比特位，拥有个哈希函数，包含个元素的集合的布隆过滤器。

在初始状态，布隆过滤器数组中的所有比特位的值均被置为0。在向布隆过滤器中插入集合中的一个元素时，首先使用个哈希函数对进行哈希，得到个下标。然后将数组中与个下标对应的个比特位的值全部置为。如图8-2所示，向布隆过滤器中插入和两个元素，分别将和每个元素对应的3个比特位的值置为。



图8-2 布隆过滤器插入示意图

要检验一个元素是否属于集合时，同样先使用个哈希函数对进行哈希，计算出所对应的个下标，然后检查数组中相应的个比特位的值。如果个比特位中任意一位的值为，则一定不属于集合，否则可能属于集合。如图8-3所示，对于元素，对应的3个比特位的值均为1，因此可以认为在集合中；而对于元素，由于其对应的3个比特位中有一个不为1，所以不在集合中。



图8-3 布隆过滤器元素检测示意图

由于布隆过滤器的哈希函数是确定的，所以如果元素被插入到了布隆过滤器中，那么对应的个比特位的值必然都为1，因此布隆过滤器不会出现假阴性。然而，假阳性在布隆过滤器中是可能出现的。即一个元素可能在某布隆过滤器所代表的集合中，但在此布隆过滤器中，该元素对应的个比特位的值每一个都为1。如图8-4所示，虽然元素不在集合中，但由于其对应的比特位都被元素和元素置为了1，因此会被误判为在集合中。



图8-4 布隆过滤器假阳性示意图

由于布隆过滤器中存在假阳性，因此在选取布隆过滤器参数时，通常首先确定可以接受的假阳率的上界，而后据此选择布隆过滤器的比特长度与哈希函数个数。下文中，将给出布隆过滤器的假阳率与参数的关系。

由于布隆过滤器中每一个哈希函数都将元素均匀的映射到0至中的某一个比特位中，因此在一次哈希后，布隆过滤器中某一比特位的值不为1的概率为

对于一个元素，将其插入布隆过滤器中需要经过次哈希，且个哈希函数是相互独立的，因此在插入一个元素后，布隆过滤器中某一比特位的值不为1的概率为

在将集合中个元素全部插入布隆过滤器中之后，布隆过滤器某一比特位的值仍不为1的概率为

那么某一比特位的值为1的概率则为

在对某个元素是否在集合中进行检测时，需要验证其对应的个比特位的值是否均为1。当这一元素不在布隆过滤器中时，其对应的个比特位的值却均位1的概率为：

当数据规模很大时，

可以看出，布隆过滤器的假阳率随着布隆过滤器的比特长度的增大而降低。通过对求导并使导数等于0，可以求出最优的哈希函数个数为：

### 8.1.4 不经意伪随机函数OPRF

不经意伪随机函数[[3]](#footnote-3)（Oblivious Pseudorandom Function，OPRF）是一种密码学原语，假设一方有输入，一方有密钥，OPRF允许输入方得到一个伪随机函数的输出。在这个过程中，输入方不知道密钥，另一方也不知道输入，如图8-5所示。



图8-5 OPRF功能示意图

通常来说，OPRF的底层实现一般为不经意传输OT技术，每个OPRF实例的摊销成本大约需要500比特的通信开销和一些对称密钥操作。

## 隐私集合求交运算

PSI作为应用广泛的特殊安全多方计算技术，近年来得到了长足的研究和发展，存在利用不同技术在不同场景和假设下实现的PSI协议，例如基于朴素哈希的、基于公钥密码的、基于同态加密的、基于混淆电路的等等。

### 8.2.1 基于朴素哈希的PSI

基于朴素哈希的PSI协议是在现实场景中有部署实例的，其优点在于简洁、高效，但缺点在于安全性不足。

利用哈希函数构造的PSI协议可以描述如下。S拥有集合，R拥有集合。他们分别对中的元素求哈希，对中的元素求哈希。R随后将哈希值发送给S。S对比两个集合的哈希值，并输出哈希值相等的元素，即交集元素，如图8-6所示。



图8-6 基于朴素哈希的PSI示意图

这个协议具有线性的计算性能和通信性能，因为协议只涉及到哈希值的计算，通信也只需发送严格线性数量的哈希值。但不幸的是，这个方案是不安全的，因为这个方案会泄露一方输入集合的隐私。为什么呢？如果属于比较小的域，例如为电话号码，只包含11个数字。S可以在离线阶段预先计算所有电话号码的哈希值，并将结果与从R收到的结果对比。这样，S就可以知道R的输入了。这也是此协议被称为朴素哈希的根本原因。因此，这是一个不安全的PSI协议。

### 8.2.2 基于可交换加密的PSI

本节介绍基于可交换加密的隐私集合求交协议。发送方S与接收方R分别持有数据集合与。双方希望计算出双方集合交集而不泄露双方集合的具体元素。基于可交换加密的隐私集合求交算法主要包换三个步骤：

* 设置阶段
* 数据加密和置换
* 交集计算

接下来，分别介绍方案中的各个步骤。令分别表示发送方S的可交换加密函数和哈希函数。同样的，使用分别表示接收方R的可交换加密函数和哈希函数。算法的流程如图8-7所示。



图8-7 基于可交换加密的隐私交集求交算法流程图

1）设置阶段

在设置阶段，发送方S和接收方R确定双方的密钥长度，并共同选取一个大素数。而后，发送方S与接收方R分别生成各自的加密密钥。

2）数据加密

在该阶段，发送方S和接收方R分别加密各自的集合和，而后，发送方S和接收方R分别将加密过的集合发送给对方。即发送方S将发送给接收方R，而接收方R将发送给发送方S。

随后，发送方S和接收方R分别将接收到的集合进行二次加密。对于发送方S，在接收到发送方R传输来的集合后，使用发送方的加密密钥对该集合再次进行加密，最终得到集合。与之相对的，对于接收方R，在其接受到发送方S传输来的集合后，使用接收方的加密密钥对再次进行加密，得到结果。

3）交集计算

由于加密算法是可交换的，即：

因此对于任何一个处于两方集合交集中的元素，其经过两次不同顺序的可交换加密后的结果是一样的。因此，两个经过两次加密的集合中相同的元素即为发送方集合与接收方集合交集中的元素。接收方R通过对比两个经过两次加密后的集合，统计其中相同元素的个数，即可计算出双方交集的大小。而由于两个集合中的元素均被发送方S加密过，而接收方R并不知道发送方S的加密密钥，因此接收方R只能计算出双方集合交集的大小，而无法计算出交集中具体元素的值。

扩展为隐私交集求势算法，可以在数据交换之前，进行随机的置乱，将不会暴露元素对应的明文。在上述算法里，因为顺序没有置乱过，所以，交集里相等的元素可以通过位置得到具体的明文数据。

### 8.2.3 低通信非平衡PSI

#### 1. 非平衡隐私交集求势算法

在本节中，将针对发送方集合大小与接收方集合大小非平衡的条件，对上一节中所述的算法进行改进，以期降低上一节中所述算法的通信复杂度与计算复杂度。考虑到协议的双方集合大小是非平衡的，发送方的集合大小远大于接收方的集合大小，因此应尽可能的将发送方的集合的传输与计算转换为传输接收方的集合的传输与计算，从而减少方案的通信时间与计算时间。同上一节相同，改进过的算法仍分为三个阶段：

* 设置阶段
* 数据加密和置换
* 交集大小计算

算法的流程如图8-8所示：



图8-8 基于可交换加密的隐私交集求势算法流程图

1）设置阶段

与上一节中相同，发送方S和接收方R首先选取一个安全系数，确定双方的密钥长度，并共同选取一个大素数。而后，发送方S与接收方R分别生成各自的加密密钥与。与上一节中不同的是，接收方R需要计算出与其加密密钥相对应的解密密钥。

2）数据加密和置乱

在该阶段中，发送方S和接收方R分别使用各自的加密密钥与加密各自的集合和，并对加密后的集合进行置乱。而后，发送方S与接收方R分别将各自加密并置乱过的集合发送给对方。即发送方S将集合发送给接收方R，而接收方R将集合发送给发送方S。

在发送方S接收到接收方R发送来的集合后，与上一节中相同，使用发送方的加密密钥对接收到的集合再次进行加密，即在其接收到发送方R传输来的数据后，用发送方的加密密钥对其再次进行加密并置乱，得到集合，然后将该集合发还给接收方R。而与上一节不同的是，在这一步中，接收方R无需进行任何操作。

3）交集大小计算

接收方在接收到发送方第二次发送来的集合后，通过设置阶段中计算出的解密密钥对该集合进行解密。由于使用的加密算法是可交换的，即：

加密的顺序不影响数据加密的最终结果，加密的顺序同样不影响解密，即：

因此接收方R可以顺利地在己方集合*Y*被发送方S再次加密后使用己方的解密密钥对收到的集合进行解密。接收方R使用对进行解密，最终得到仅经过发送方S的密钥加密并置乱过的集合接收方集合。同时，接收方R同样接收到了仅经过发送方密钥加密过的接收方集合。通过对比两个集合找出其中相同的元素，即可计算出发送方集合与接收方集合的交集大小。同时，由于接收方R并不知道发送方的密钥，因此接收方R只能计算出双方集合交集的大小，而无法通过解密得出具体元素的值。协议如图8-9所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 发送方S | | 接收方R | |
| 输入集合 | | 输入集合 | |
|  | *U V*  *T* | |  |
| 输出 | | | |

图8-9 非平衡隐私交集求势协议

相较于上一节中所述的算法，在发送方集合与接收方集合大小非平衡的条件下，由于将部分加密计算与数据传输转移到了规模较小的集合上，因此方案的通信量与计算量均有所降低。在上一节所述方案中，对于发送方集合，需要进行两次加密计算和两次传输。而在本节的方案中，对于发送方集合，只需要进行一次加密与一次传输，而对于接收方R的集合，共需要进行3次加解密计算和两次传输。在发送方集合与接收方集合大小非平衡的条件下，由于发送方集合大小远大于接收方集合大小，故对发送方集合的计算与传输的次数决定了算法所需的计算时间与通信时间。因此，将较于上一节所述算法，在发送方集合与接收方集合大小非平衡的条件下，本节算法的计算量与通信量均降低为上一节算法的一半。

#### 2. 通信优化的隐私交集求势算法

在上一节中，考虑到发送方集合与接收方集合规模是非平衡的，发送方S的集合大小远大于接收方R的集合大小，因而利用可交换加密的性质，将发送方集合的加密计算与传输转换为接收方集合的计算与传输，从而使得方案的计算量与通信量降为原先的一半。然而，对于一个规模较大的集合，一次传输的代价仍然很大。为了进一步提高协议的通讯效率，引入布隆过滤器来降低通信开销。

协议的参与者包括发送方和接收方。同样，协议仍分为三个阶段：

* 设置阶段
* 数据加密和置换
* 交集大小计算协议的

算法的流程图如图8-10所示：



图8-10 通信优化的隐私交集求势算法流程图

1）设置阶段

与上一节相同，发送方S和接收方R首先选取一个安全系数，确定双方的密钥长度，并共同选取一个大素数。而后，发送方S与接收方R分别生成各自的加密密钥与。而后接收方计算出与接收方加密密钥相对应的接收方解密密钥。除此之外，本届的算法中将会使用到布隆过滤器，因此发送方S与接收方R还需对布隆过滤器的参数进行约定，确定布隆过滤器的比特位数与哈希函数个数，并共同选取个哈希函数。

2）数据加密和置换

在该阶段，接收方R同之前一样，使用接收方的加密密钥加密接收方的集合，并对加密过后的集合进行置乱，而后将其传输给发送方S。即接收方R将发送给发送方S。发送方S则使用发送方的加密密钥加密发送方的集合。而与之前方案中不同的，在对其集合进行加密后，不直接将加密并置乱的结果发送给接收方R，而是根据加密结果构造布隆过滤器。对于集合中的每一个元素，发送方S计算该元素的个哈希，并将布隆过滤器中与个哈希结果对应的比特位的值置为1。最后，发送方S将构造好的布隆过滤器发送给接收方R。

在发送方S接收到接收方R发送来的集合后，与上一节中相同，使用发送方的加密密钥对接收到的集合再次进行加密。即在在其接受到发送方R传输来的数据后，用发送方的加密密钥对其再次进行加密。而后发送方对加密结果进行置乱，得到集合，然后将该集合发还给接收方R。同样，在这一步中，接收方R无需进行任何操作。

3）交集大小计算

接收方R在接收到发送方S发来的二次加密的集合后，通过设置阶段计算出的解密密钥对二次加密后的集合进行解密。由于加密算法是可交换的，即：

加密的顺序不影响数据加密的最终结果，加密的顺序同样不影响解密，即：

因此接收方R可以顺利地在己方集合*Y*被发送方S再次加密后使用己方的解密密钥对收到的集合进行解密。接收方R使用对进行解密，最终得到仅经过发送方S的密钥加密并置乱过的集合接收方集合。同时，在之前步骤中，接收方还接收到了由发送方S使用发送方密钥加密过的发送方集合构造的布隆过滤器。由于是由发送方集合经发送方密钥加密后结果构造的，显然，如果若接收方集合经发送方密钥加密后的集合中的某一个元素在该布隆过滤器中，该元素就在发送方集合与接收方集合的交集中。接收方通过检验由发送方加密的接收方数据中的每个元素是否在该布隆过滤器中，即可计算出发送方集合与接收方集合的交集大小。

对于中的每一个元素，接收方R分别计算经过个哈希函数后结果，并检验布隆过滤器中与哈希结果对应的个比特位的值是否均为1。若布隆过滤器中与个哈希结果对应的比特位的值均为1，那么可以认为该元素在双方的交集中。相反，若布隆过滤器中与哈希结果对应的个比特位的值至少有一个不为1，则该元素一定不双方集合的交集中。最终，经该集合中在布隆过滤器中的元素个数，就是发送方集合与接收方集合的交集大小。算法的伪代码如图8-11所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 发送方S | | 接收方R | |
| 输入集合 | | 输入集合 | |
| Constructs | *BF*(*U*) | |  |
| 输出 | | | |

图8-11 通信优化的隐私交集求势算法流程图

相较于上一节中所述的算法，在本节中，发送方S不再发送其集合的加密结果，取而代之的，发送由其集合的加密结果构造的布隆过滤器，从而达到进一步降低算法通信量的目的。相对的，算法增加了通过发送方集合的加密结果构造的布隆过滤器的步骤。在该步骤中，需要对发送方集合的加密结果中的每个元素进行次哈希。因此，在计算量方面，与上一节中的算法相比，本节的算法计算量有所增多。但相较于可交换加密算法，哈希算法的计算速度要远远快于可交换加密算法。对于算法的计算时间起决定性影响的，仍是可交换加密算法的计算次数。因此在计算复杂度上，仍可认为，该方案的计算复杂度为最初方案的一半。

在通信复杂度方面，本节的方案不再传输发送方集合的加密结果，取而代之的是，传输由发送方集合的加密结果构造的布隆过滤器，因此通信量的大小与布隆过滤器的比特长度有关。而布隆过滤器的比特长度则取决于所期望的布隆过滤器中假阳率的上界。布隆过滤器的假阳率与布隆过滤器比特长度及布隆过滤器中集合大小的关系如表8-1所示。在通常选取2048比特素数的情况下，根据所选用的布隆过滤器比特长度不同，本节中所述方案的通信量为上一节方案的十几至几十分之一不等。

表8-1 布隆过滤器假阳率与比特位数的关系

|  |  |
| --- | --- |
| **布隆过滤器比特位数与集合大小的比值** | **假阳率** |
| 16 | 4.59×10-4 |
| 20 | 6.71×10-5 |
| 24 | 9.87 ×10-6 |
| 32 | 2.10×10-7 |
| 48 | 9.65×10-11 |
| 64 | 4.43×10-14 |

#### 3. 带限制的隐私交集求势算法

在上一节中，通过引入布隆过滤器，将算法中发送方集合的传输转化为了传输发送方集合的布隆过滤器，进一步算法降低了通信量。然而，由于布隆过滤器的使用，也为方案的安全性带来了新的问题。在之前的方案中，均假设发送方S的集合大小与接收方集合的大小为公开的信息。因此在协议的执行过程中，双方可以通过验证加密过后的集合是否与最初的集合大小相等来验证方案中的某一方是否在协议执行过程中增加或删除了集合中的元素，从而使得最终无法计算出真实的双方集合交集的大小。然而由于布隆过滤器的使用，接收方R无法直接验证接收到的布隆过滤器是发送方根据协议生成的，或仅使用了发送方集合中的部分元素或向其中插入了本不在发送方集合中的元素。虽然在整个算法中，均假设发送方S与接收方R为半诚实的参与者，会遵守协议的执行，但仍希望通过进一步的改进以弥补新导致的问题。因此，在算法中对于额外做出如下规定。

在算法中，当发送方S将全部插入到布隆过滤器中之后，根据此时布隆过滤器中值为1的比特位的个数，决定将布隆过滤器中随机比特位的值置为0或1，直到布隆过滤器中值为1的比特位的个数满足：

其中为布隆过滤器中值为1的比特位的个数的期望。在布隆过滤器中，某一比特位的值为1的概率为：

因而布隆过滤器中值为1的比特位的个数的期望为：

通过限制布隆过滤器中值为1的比特位个数与向布隆过滤器中插入发送方集合规模的元素后值为1的比特位个数的期望相等，使得发送方很难向布隆过滤器中插入更多的元素以改变算法的结果。然而显然地，这也会对最终结果造成一定的误差。下文中将证明上述规定引起的误差进行分析。

首先，考虑由发送方集合加密结果构造的布隆过滤器中值为1的比特位的个数大于的情况。在发送方将集合中的所有的元素映射到布隆过滤器中后，若布隆过滤器中值为1的比特位个数大于，则需要发送方将随机的比特位置为0直至布隆过滤器中值为1的比特位个数等于。如第二章第二节中所述，在布隆过滤器中，每个比特位为1的概率是相互独立的，因此值为1的比特位的个数服从二项分布

当很大时，可以近似为一个正态分布。

将其转化为一个标准正态分布：

因此，发送方需要从将值从1转化为0的比特位个数不多于

，其中为标准正态分布的累积分布函数，为置信水平。

接下来，考虑将这些比特位从1变为0将会对计算出的发送方集合与接收方集合的大小带来多少影响。此时，布隆过滤器中值为1的比特位个数为：

同时布隆过滤器中共包含发送方集合大小个元素，因此平均每个比特位与之相关的元素个数是：

因此，在将布隆过滤器中随机的比特位置为0直至布隆过滤器中值为1的比特位个数*N*等于的过程中，可以近似地认为从布隆过滤器中删除的元素个数为：

而这些元素落在发送方集合与接收方集合的交集中的概率是：

最终，这些删去的元素占接收方集合比例为：

可以看出，在该情境下，协议的误差主要取决于发送方集合大小、双方交集占接收方集合比例和布隆过滤器比特位数。取置信水平为0.98，协议的误差率如表8-2，表8-3，表8-4所示：

从表8-2中可以看出，当发送方集合规模增大时，协议的误差率减小。当发送方的集合规模很大时，误差将减小一个可接受的范围。

表8-2 不同发送方集合大小下协议的误差率

|  |  |
| --- | --- |
| 发送方集合大小 | 误差率（‰） |
| =0.6,=20 |
| 500000 | 0.8542 |
| 1000000 | 0.6042 |
| 2000000 | 0.3823 |
| 5000000 | 0.2704 |
| 10000000 | 0.1912 |
| 20000000 | 0.1210 |

从表8-3中可以看出，双方交集占接收方集合比例与协议的误差率成正比。当双方交集的比例随着发送方集合与接收方集合的交集占接收方集合比例增大时，协议的误差率增大。

表8-3 不同双方交集占接收方集合比例下协议的误差率

|  |  |
| --- | --- |
| 双方交集占接收方集合比例 | 误差率（‰） |
| =10000000,=20 |
| 0.2 | 0.0403 |
| 0.3 | 0.0604 |
| 0.4 | 0.0806 |
| 0.5 | 0.1008 |
| 0.6 | 0.1210 |
| 0.7 | 0.1410 |
| 0.8 | 0.1612 |
| 0.9 | 0.1814 |

在表8-4中，随着布隆过滤器比特位数与发送方集合大小比值的增大，占接收方集合比例增大时，协议的误差率减小。即使用的布隆过滤器越大，算法的误差率越小，但与此同时，通信复杂度与使用的内存也会增大。

表8-4 不同布隆过滤器比特位数下协议的误差率

|  |  |
| --- | --- |
| 布隆过滤器比特位数 | 误差率（‰） |
| *|X|*=10000000,*r*=0.6 |
| 160000000 | 0.1369 |
| 200000000 | 0.1210 |
| 240000000 | 0.1142 |
| 320000000 | 0.0968 |
| 480000000 | 0.0791 |
| 640000000 | 0.0685 |

与之相对的，考虑发送方元素在布隆过滤器中所需比特数小于的情况。在发送方将集合中的所有的元素映射到布隆过滤器中后，若布隆过滤器中值为1的比特位个数小于，则需将随机的比特位值为1，直至布隆过滤器中值为1的比特位个数等于。与时相同，此时，发送方需要将置从0转化为1的比特位个数上限仍然为：

接下来，考虑这些比特位从1变为0将会对计算出的发送方集合与接收方集合的大小带来多少影响。

此时，布隆过滤器中值为1的比特位个数为：

同时布隆过滤器中共包含发送方集合大小个元素，因此平均每个比特位与之相关的元素个数是：

因此，将布隆过滤器中随机的比特位置为1直至布隆过滤器中值为1的比特位个数等于的过程中，可以近似的认为从布隆过滤器中新增的元素个数是：

这些元素落在发送方集合与接收方集合的交集中的概率是：

最终，这些删去的元素占接收方集合的比例为：

从公式中可以看出，在该情境下，协议的误差率主要取决于发送方集合大小、双方交集占接收方集合比例和布隆过滤器比特位数。取置信水平为0.98，不同情况下的误差如表8-5，表8-6，表8-7所示。

从表8-5中可以看出，与时的情况相反。随着发送方集合大小增大，协议的误差率增大。

表8-5 不同发送方集合大小下协议的误差率

| 发送方集合大小 | 误差率（‰） |
| --- | --- |
| =0.6,=20,=1010 |
| 500000 | 0.00034 |
| 1000000 | 0.00049 |
| 2000000 | 0.00077 |
| 5000000 | 0.00109 |
| 10000000 | 0.00156 |
| 20000000 | 0.00255 |

从表8-6中可以看出，双方非交集占接收方集合比例与协议的误差率成正比。当双方交集的比例随着发送方集合与接收方集合的交集占接收方集合比例增大时，协议的误差率减小。

表8-6 不同双方交集占接收方集合比例下协议的误差率

| 双方交集占接收方集合比例 | 误差率（‰） |
| --- | --- |
| =10000000,=20,=1010 |
| 0.2 | 0.00170 |
| 0.3 | 0.00223 |
| 0.4 | 0.00255 |
| 0.5 | 0.00265 |
| 0.6 | 0.00255 |
| 0.7 | 0.00223 |
| 0.8 | 0.00170 |
| 0.9 | 0.00096 |

在表8-7中，随着所选用的布隆过滤器比特位数与发送方集合大小比值的增大，协议的误差率随之减小。即使用的布隆过滤器越大，算法的误差率越小，但与此同时，算法的通信复杂度与使用的内存也会增大。

表8-7 不同布隆过滤器比特位数下协议的误差率

|  |  |
| --- | --- |
| 布隆过滤器比特位数 | 误差率（‰） |
| =10000000,=0.6,= |
| 160000000  200000000  240000000  320000000  480000000  640000000 | 0.00288  0.00255  0.00240  0.00204  0.00166  0.00144 |

比较与两种情况下的的误差率，可以看出情况下的误差率要远小于情况下的误差率。这是因为，在实际的场景中，发送方集合与接收方集合中元素的取值范围通常要远远大于发送方集合与接收方集合的大小，所以额外插入的值为1的比特位落入接收方集合中的概率很低。因此，在这种情况下，在根据协议的误差率上限来选取参数时，往往只需要考虑时的误差率即可。

### 8.2.4 基于OPRF的构造

现在已经有了很多种不同的方法来实现隐私集合求交，比如基于Diffie-Hellman密钥交换的方法、基于不经意传输的方法等等。而截至目前，最快速的隐私集合求交算法是基于不经意传输的。下面，我们介绍如何使用不经意传输来实现一个隐私集合求交算法。

#### 1. 从隐私比较到OPRF

我们假设有两个数据方（P1和P2）。P1拥有字符串，而P2拥有字符串，他们想在不泄露各自输入的前提下，比较是否等于。

基于OT的两方字符串比较算法如图8-12所示，它可以分为以下几个核心步骤：



图8-12 字符串比较示意图

（1）P2生成2长度为比特串，即个字符串对，每个字符串对包括两个比特串，分别对应0和1两位。这里的指的是双方待比较的字符串的长度，在图8-12的例子中，=3。是一个超参数，它的长度越大，该算法的安全性及正确性也就越高。

（2）P1对待比较字符串（）的每一位，P1和P2使用OT，使得P1能够获取P2每个字符串对中的一个比特串。具体而言，P1作为OT的接收者，P2作为OT的发送者；P2拥有两个长度为的比特串，P1输入0或1，返回P2该位字符（0或1）所对应的比特串。在这个过程中，由于采用OT，P1不知道P2另外一个比特串是什么，同时，P2不知道P1请求的是哪一个比特串。重复待比较字符串的每一位，P1便得到了1个长度为的比特串。然后P1对这个比特串做异或，得到一个字符串（记为）。

（3）P2对其待比较字符串（）的每一位，选择每个字符串对相应的比特串，这样P2也得到了1个长度为2的比特串。然后P2对这1个比特串做异或，得到一个字符串（记为），并将发送给P1。

（4）P1比较和，如果相等，那么就说明P1和P2的两个原待比较字符串也相等，反之说明它们不相等。

发送方P2持有一组二进制串，我们可以将这些二进制串整体当作一个随机种子，由P2方持有。从P1方的角度来看，隐私比较的过程，就是P1方输入数据，得到一个随机二进制串，这个二进制串由P2方持有的随机种子与输入来决定，同时P2方无法得知P1方的输入。这一过程，就可以看作是不经意伪随机函数（Oblivious Pseudorandom Function，OPRF）。

这样来看，我们就是使用不经意伪随机函数构建了一个隐私比较算法。接下来，我们要更进一步，看看如何使用不经意伪随机函数来构建隐私集合求交协议。

#### 2. 基于OPRF的PSI

1）基础构造

假设P1持有一组输入，P2持有一组输入。通过不经意伪随机函数，我们可以构造出一个非常朴素的隐私集合求交算法：

① P1构造个不经意伪随机函数的种子，。

② P2为中的每一个元素，执行一个对应不经意伪随机函数，得到集合。

③ P1为中的每一个元素，执行伪随机函数，得到集合。

④ P1将集合发送给P2，P2求交集，再将交集映射回，即可得到与的交集。

这种方法简单来讲，就是P2将中的每一个元素，都与P1的中的每一个元素，通过不经意伪随机函数进行隐私比较，进而得到与的交集。

如果直接用以上基于OT的字符串比较方法来做隐私求交，假设双方各有个元素，那么求交集过程中，通信量是：因为的大小是。对不同的对应的密钥，要对中每个元素都提供密文，才能保证整个比较过程是正确的。

2）优化方案

（1）布谷哈希

该协议要用到包含3个哈希函数的布谷鸟哈希[[4]](#footnote-4)（Cuckoo Hashing）。我们现在简要介绍布谷鸟哈希的基本原理。为应用布谷鸟哈希将个元素分配到个箱子中，首先选择3个随机哈希函数，并初始化个空箱子。为计算元素的哈希值，首先检查这三个箱子中是否有一个是空箱子。如果至少有一个箱子是空的，则将放置在其中一个空箱子内，并终止算法。否则，随机选择，将中的当前元素驱逐出箱子，将放置在此箱子中，并向其他箱子迭代插入被驱逐的元素。如果经过一定次数的迭代之后算法仍未终止，则将最后被驱逐出的元素放置在一个名为暂存区（Stash）的特殊箱子中。

谷鸟的学名为大杜鹃，本书采用了音译。布谷鸟的特点是把蛋下到别的鸟巢里。布谷鸟的幼鸟一般比别的鸟早出生，幼鸟出生后会把未出生的其他鸟蛋挤出鸟巢。布谷鸟哈希处理哈希碰撞的方法是驱逐出原来占用位置的元素，与布谷鸟的行为类似。因此，学者用布谷鸟的生物学典故借喻布谷鸟哈希的碰撞处理方法。

（2）应用布谷鸟哈希的PSI

首先，两个参与方为3-布谷鸟哈希选择3个随机哈希函数。假设P1的输入集合为，P2的输入集合为，且满足。P2应用布谷鸟哈希将集合中的元素放置在1.2个箱子和大小为的暂存区中。此时，P2的每个箱子中最多含有一个元素，暂存区中最多含有个元素。P2用虚拟元素填充箱子和暂存区，使每个箱子均包含一个元素，暂存区中包含个元素。

两个参与方随后执行1.2个OPRF协议，P2作为OPRF协议的接收方，分别将1.2个元素作为OPRF的输入。令表示第*i*个OPRF协议所对应的PRF。如果P2通过布谷鸟哈希将元素放置在第个箱子中，则P2得到;如果P2将元素放置在暂存区中，则P2得到。

另一方面，P1可以对任意计算。因此，P1计算得到下述两个候选 PRF的输出集合：

P1随机打乱集合和中元素的位置，并发送给P2。P2可按下述方法计算得到和的交集：如果P2有一个被映射到暂存区中的元素，则P2验证中是否含有所对应的OPRF输出。如果P2有一个被映射到哈希箱子中的元素，则P2验证中是否含有所对应的OPRF输出。

通过计算，我们可以发现，集合的大小为3，集合的大小为，是一个常数，因此P1需要传输的数据量为，是的。通过结合布谷鸟哈希，我们减少了协议所需要传输的数据量，加快了协议的执行速度。

直观上看，此协议可以抵御半诚实P2的攻击。这是因为元素所对应的PRF输出满足伪随机性。类似地，如果密钥具有关联性，但PRF的输出仍然满足伪随机性，则用OPRF实现关联密钥PRF也可以保证方案的安全性。

只要PRF的输出不发生碰撞（即对于），此协议的计算结果就是正确的。我们必须谨慎设置协议的参数，避免PRF的输出发生碰撞。

## 应用实践

本节基于公钥加密的原理，实现一个两方半诚实安全的PSI协议。其中使用的哈希单向函数选择布隆过滤器，判断出某个元素肯定不在或者可能在集合中，即不会漏报但可能会误报，通常应用在一些需要快速判断某个元素是否属于集合，但不严格要求正确的场合。

### 8.3.1 协议流程

（1）客户端和服务器端约定使用的RSA算法的模、公钥、比特位数以及布隆过滤器函数，服务器端数据集为，客户端数据集为；服务器端生成私钥；

（2）客户端离线准备，生成随机数（盲因子），并对自己的数据进行盲化处理，得到，并计算盲化因子对应的逆元；

（3）服务器计算；

（4）客户端发送盲化后的数据；

（5）服务器对客户端的数据进行盲签名，发送盲签名后得到的以及第三步中计算所得数据；

（6）客户端计算获得，并与从服务器端接收到的数据进行比较匹配获得最终结果。

### 8.3.2 实现方案

基于python语言实现方案。为了简化，通过range函数取range(0, 1024)作为服务器端的数据集合，取range(0, 1024, 249)（即0、249、498、747、996）作为客户端的数据集合。

基于gmpy2库实现RSA算法，主要使用其中的invert函数进行乘法逆元的计算，使用powmod函数进行公钥加密和私钥签名；基于pycryptodome库生成密钥。哈希单向函数使用的BF基于pybloom\_live库实现。

1）搭建运行环境

pip install gmpy2

pip install bitarray==1.7.1

pip install pycryptodome

pip install pybloom\_live

2）编写程序实现

1. import secrets
2. import gmpy2
3. import pybloom\_live
4. from Crypto.PublicKey import RSA
5. #设置RSA的密钥长度和指数
6. RSA\_BITS = 1024
7. RSA\_EXPONENT = 65537
8. RT\_COUNT = 0
9. #生成公私钥对
10. def generate\_private\_key(bits=RSA\_BITS, e=RSA\_EXPONENT):
11. private\_key = RSA.generate(bits=bits, e=e)
12. public\_key = private\_key.publickey()
13. return private\_key
14. #计算盲因子及其逆元，将数据序列化到本地文件，以便后续步骤读取使用
15. def generate\_random\_factors(public\_key):
16. random\_factors = []
17. rff = open('randomfactors.raw','w')
18. for \_ in range(RF\_COUNT):
19. r = secrets.randbelow(public\_key.n) #生成随机数
20. r\_inv = gmpy2.invert(r, public\_key.n) #求盲因子r的逆元
21. r\_encrypted = gmpy2.powmod(r, public\_key.e, public\_key.n) #对r进行公钥加密
22. random\_factors.append((r\_inv, r\_encrypted))
23. rff.writelines(f"{r\_inv.digits()}\n")
24. rff.writelines(f"{r\_encrypted.digits()}\n")
25. rff.close()
26. return random\_factors
27. #对数据进行盲化处理
28. def blind\_data(my\_data\_set, random\_factors, n):
29. A = []
30. bdf = open('blinddata.raw','w')
31. for p, rf in zip(my\_data\_set, random\_factors):
32. r\_encrypted = rf[1]
33. blind\_result = (p \* r\_encrypted) % n #盲化
34. A.append(blind\_result)
35. bdf.writelines(f"{blind\_result.digits()}\n")
36. bdf.close()
37. return A
38. #对数据使用私钥签名后添加到 BF
39. def setup\_bloom\_filter(private\_key, data\_set):
40. mode = pybloom\_live.ScalableBloomFilter.SMALL\_SET\_GROWTH
41. bf = pybloom\_live.ScalableBloomFilter(mode=mode)
42. for q in data\_set:
43. sign = gmpy2.powmod(q, private\_key.d, private\_key.n) #签名
44. bf.add(sign) #将签名加入布隆过滤器BF
45. bff = open('bloomfilter.raw','wb')
46. bf.tofile(bff)
47. bff.close()
48. return bf
49. #使用私钥进行盲签名
50. def sign\_blind\_data(private\_key, A):
51. B = []
52. sbdf = open('signedblinddata.raw','w')
53. for a in A:
54. sign = gmpy2.powmod(a, private\_key.d, private\_key.n) #盲签名
55. B.append(sign)
56. sbdf.writelines(f"{sign.digits()}\n")
57. sbdf.close()
58. return B
59. #判断交集
60. def intersect(my\_data\_set, signed\_blind\_data, random\_factors, bloom\_filter, public\_key):
61. n = public\_key.n
62. result = []
63. for p, b, rf in zip(my\_data\_set, signed\_blind\_data, random\_factors):
64. r\_inv = rf[0] #获取盲因子的逆元
65. to\_check = (b \* r\_inv) % n
66. if to\_check in bloom\_filter: #检查所得结果是否在BF中
67. result.append(p)
68. return result
69. if \_\_name\_\_ == '\_\_main\_\_':
70. #输入
71. client\_data\_set = list(range(0, 1024, 249))
72. server\_data\_set = list(range(0, 1024))
73. RF\_COUNT = len(client\_data\_set)
74. #服务器生成密钥
75. private\_key = generate\_private\_key()
76. public\_key = private\_key.public\_key()
77. # 客户端生成盲因子、对自己的数据进行盲化处理
78. random\_factors = generate\_random\_factors(public\_key)
79. A = blind\_data(client\_data\_set, random\_factors, public\_key.n)
80. #服务器使用私钥对拥有的数据进行签名并添加到BF
81. bf = setup\_bloom\_filter(private\_key, server\_data\_set)
82. #客户端将自己生成的盲化数据发送给服务器端；在实际应用中通过网络传输实现通信，但由于并非本书重点，通信相关的实现就不做介绍了
83. #服务器接收客户端盲化数据A并使用私钥进行盲签名，然后将盲签名后的数据B以及bf发送给客户端
84. B = sign\_blind\_data(private\_key, A)
85. #客户端将服务器端盲签名后的数据B与盲因子逆元相乘，若所得结果在BF种，则该元素在交集中，为两方共有数据
86. result = intersect(client\_data\_set, B, random\_factors, bf, public\_key)
87. print(result)

3）运行

python main.py

## 课后习题

1. 针对PSI和PSI-CA协议，下列说法错误的是（ ）

A. PSI协议的输出结果是交集

B. PSI-CA协议的输出结果是交集和势

C. 基于可交换加密的协议构造，其通信量和计算量是线性的

D. 基于布隆过滤器的协议，过滤器长度越长，出现假阳性的概率越低

2. 尝试基于可交换加密的两方PSI-CA协议，结合其他密码学原语构造两方隐私集合求并（PSU）协议。

# 第9章 安全多方计算

学习要求：掌握姚氏百万富翁问题的概念及其解决方案；掌握混淆电路的构造思想，能够生成并求解混淆电路；理解姚氏乱码电路协议的基本原理；掌握电路优化技术，包括行约减、FreeXOR技术和半门技术；掌握ABY框架及其在安全双方计算中的应用，理解数值比较算子的实现；了解安全多方计算的发展历史、现状和研究进展，

课时：2课时

建议授课进度：[9.1~9.2]、[9.3~9.4]

## 布尔电路

### 姚氏百万富翁问题

姚期智（Andrew Chi-Chih Yao）是中国第一个也是目前为止唯一的图灵奖获得者。他为现代密码学打开了一道新的大门。其中一项重要贡献，就是安全多方计算。他提出了著名的姚氏百万富翁问题：假如有两个百万富翁Alice和Bob各有钱和，他们想知道谁的钱多，但是又不想把自己的钱数量告诉对方，怎么样才能比大小呢？

下面介绍姚期智先生的解：

（1）先把问题简化，假如和是1到10之间的数。

（2）首先Bob挑选一个大整数，然后用Alice的公钥加密得到。然后把数发给Alice。

（3）Alice拿到这个数之后，计算以下这些数：

然后除以一个素数取余得到：

因为Alice的钱是，那么Alice做一下这个事情：

也就是将第及后面的数都加1，然后把这一串数字发给Bob。

（4）Bob只需要看第个数字，如果等于，说明第个数字没有改动过，处于左侧，即；否则，说明。然后，Bob把结果返给Alice。

看着很复杂，实际上这个协议很简单。重点我们要去理解姚期智是怎么想的？

Bob选择了一个非常大的数，然后加密得到。然后把发给了Alice。这个时候Bob的信息已经藏在这个数据里面了，但是因为很大，所以Alice没办法从之中推导出，对Alice来说就是个随机数。

然后，Alice计算了10个数，。对Alice来说都是没有意义的随机数。但是，其中一个数是有意义的，那就是，为什么呢？因为，而是什么呢？那么理所当然。所以说这十个数对Alice没有意义，但是对Bob是有意义的，因为他知道第个数就是。

接下来，Alice在这十个数从第个数开始都+1。再给Bob，会发生什么？因为Bob知道第个数是，如果Bob收到的是，说明在没有改动过的左侧区域，所以。

是不是Alice实际上巧妙地把自己的数的信息放进这十个数当中，但是因为她知道这十个数除非是第个数，否则其它的数对Bob没有任何意义。所以她不用担心自己的数泄露（操作也是随机操作），但是又可以比较大小。这就是这个协议的中心思想。

### 混淆电路构造思想

Yao的上述百万富翁求解的协议仅限于比较大小，而其他的运算还没有支持。当然后来一个在此基础上更加伟大的密码学算法被发明出来了，那就姚氏乱码电路（Garbled Circuit，GC），也叫姚氏混淆电路，是最著名、最广为人知的MPC协议。

混淆电路的思想很简单，它源于一个事实：可以通过设计电路来对目标问题进行求解。电路可以通过与或非门实现任意一个函数，而多方计算的目标就是在保护各方输入信息的情况下进行目标函数的计算。

对参与运算的双方，从参与者的视角可以分为电路产生者（circuit generator）与电路执行者（circuit evaluator）。混淆电路有两个阶段：

* 电路产生阶段：电路产生者将待计算的函数转化为电路。参与运算的双方先就需要安全计算的目的依靠专有编程语言（DSL）或相关编程语言扩展等进行编程，然后针对实现计算的程序进行编译，生成电路文件；
* 电路执行阶段：电路执行者安全地计算电路。利用不经意传输（Oblivious Transfer，OT）、加密等密码学原语执行电路，在不泄露电路的输入和中间结果的前提下，完成电路的计算。

#### 1. 基于查找表的构造思想

回顾一下，为了要求解函数的值，参与方持有，参与方持有。这里的和分别为和的输入域。

1）将函数表示为查找表

首先考虑一个输入域很小的函数。由于的输入域很小，我们可以很快地枚举出所有可能的输入对。可以把函数表示为一个包含行的**查找表***，*每行的条目为，即该表只有1列，共有行。只要能定位，就可以获得的输出。

可以按照下述方式生成查找表（如图9-1所示）：

1. 通过为每一个可能的输入和随机指定一个强密钥来加密。也就是说，对于每一个和每一个，将选择和；
2. 随后，同时使用两个密钥加密中相应的数据项；
3. 最后，将加密且经过随机置换的查找表发送给。



图9-1 生成查找表的过程

2）利用茫然传输隐藏行内访问的元素

我们现在的任务是让（只能）解密与参与方输入相关联的数据项。具体实现方式是让向（茫然的）发送密钥和：

1. 已知自己的输入，因此只需要将密钥直接发送给即可（因为只是发送了一次性选的密钥，这个过程不会泄露）；
2. 已知自己的输入，它可以使用选1-OT协议从获得。

一旦收到和，就可以使用这些密钥解密，得到输出。

最重要的是，在此过程中无法获得任何其他信息。这是因为只拥有一对密钥，只能打开（解密）查找表中的一个数据项。需要特别强调的是，单独使用或都不允许部分解密密文，甚至不能单独使用或判断某个密文是否是用或加密得到的。

3）安全求解函数

由于置换表是随机置换后发给，仅仅得到了解密密钥，那么如何知道对应的数据项呢？

解决这个问题最简单的方法是在的加密条目中编码一些附加信息。例如，可以在的每一行字符串的末尾附加个0。如果解密了错误的行，则解密结果的末尾仅有很低的概率包含个0，这样就可以知道解密结果是错误的。虽然这个方法是可行的，但是它对于来说效率很低，因为平均要解密查找表中至少一半的条目。

4）利用标识置换实现行的定位

1990年，Beaver等人提出了标识置换（Point-and-Permute）技术[[5]](#footnote-5)，可以确定特定输入对应的密文数据项在置乱后的查找表中的位置。此方法的基本思想是将密钥的一部分（即第一个密钥的后比特和第二个密钥的后比特）作为查找表的置换标识，标识密钥将加密哪行（或者哪个位置）密文，并根据置换标识对加密后的查找表进行置换。在具体协议中，可以设计为先确定查找表的置换标识，然后根据这个置换标识完成置换。

为了避免查找表的各行在分配过程中产生冲突，必须保证置换标识不会在的密钥空间和的密钥空间中出现冲突，可以通过多种方式实现这一点。严格来说，密钥长度必须要达到相应的安全等级。因此，参与方并不会直接把密钥中的一部分作为置换标识，而是将置换标识附加在密钥之后，使密钥满足所需的长度要求。在后续讨论中，我们假定求值方已知要解密查找表中的哪一行。在描述协议时，我们会根据上下文决定是否有必要明确指出把标识置换技术作为协议的一个组成部分。

5）降低查找表的大小

显然，上述方案的效率较低，因为查找表的大小与的定义域大小呈线性关系。但是，对于布尔电路门这样的小型函数，因为其定义域的大小仅为4，用查找表表示此类小型函数是比较高效的。

因此，可以进一步将表示为布尔电路，并用定义域大小为4的查找表求解每一个门的输出，如图9-2所示。对于布尔电路而言，电路实现与或非即可实现完备，可以模拟任意的函数。



图9-2 基于布尔电路的求解示意

#### 2. 混淆电路的生成与求解

如图9-3所示，门可以是与门、或门等，它接收两个输入，输出一个结果。



图9-3 混淆电路基础结构

1）生成混淆电路

首先以与门为例，一个常见的与门及其**真值表（Truth Table）**如图9-4所示，将该与门的输入线记为，输出线记为。



图9-4 与门电路真值表

随机生成6个密钥，分别表示这三条线为0和1时的两种情况。如分别代表为0和为1，分别代表为0和为1。我们称这些密钥为**导线标签**（Wire Label），称导线的明文值为**导线值**（Wire Value）。

接着该门利用对称加密算法生成4个密文。表示用作为加密密钥、使用加密算法来加密，即。查找表加密过程如图9-5所示。



图9-5 查找表加密过程

2）随机置换查找表

就是构建的**初始查找表**, 查找表中的每一行条目都是门输出值所对应导线标签的密文。考虑安全性，需要对查找表的元素进行随机置乱。在查找表只包含4行密文的情况下，标识置换技术非常简单和高效。

（1）只需要2个比特长的置换标识，可以在为每条导线选择对应的随机密钥的时候，随机为其产生对应的1位置换比特（图9-6中红色部分）即可；

（2）任意两个输入的密文存储到两个密钥关联的置换比特所标识的位置。比如，如果为随机选择的置换比特为1，则的置换比特则被设置为0，同样，如果为随机选择的置换比特为0，则的置换比特则被设置为1，在这个情况下，产生的最后的乱码表就是。



图9-6 随机置换查找表

通常把置换后的查找表称为**乱码表**（Garbled Table）或乱码门（Garbled Gate），并将所有乱码表发送给。**所有的乱码门就构成了混淆电路**。

3）对混淆电路求值

在收到乱码表后，电路求值方开始对电路求值。对于中属于的输入导线，直接将对应的明文相关的密钥（又称为**激活标签**）发送给。对于中属于的输入导线，通过2选1-OT协议从得到对应的激活标签。激活标签对应的明文值称为激活值。假设门上两条线的输入的值对为(0,1)，那么输入线对应的电路计算值为。是直接发送给的，而是通过2选1-OT协议获得的。

因为两个密钥各自蕴含了置换比特，所以，可以找到对应的要计算的密文。

很重要的一点是，对一个乱码门的求解，允许求值方得到输出对应的激活标签，并在不知道中间激活标签所对应的激活值的条件下，利用中间激活标签继续做下一个电路门的输入，直到完成的安全求值。

最终，完成乱码电路的求值，并得到与电路输出导线关联的密钥。把得到的密钥发送给解密，即可完成的安全求值。一个混淆电路的求值过程如图9-7所示。

图9-7 混淆电路求值过程实例

4）解码表

我们注意到，可以不用将导线标签发送给解密，这样可以节省一轮通信过程。具体方法是让在发送乱码电路的同时发送输出导线的解码表。解码表只是将输出导线的每个导线标签映射为对应的导线值（即相应的明文值）。此时，得到输出导线标签的可以在解码表中直接查找导线标签所对应的导线值，得到明文输出。

#### 3. 百万富翁的电路求解示例

1）逻辑电路

混淆电路要求计算的函数能被逻辑电路表示，所以如何将函数转化为一个逻辑电路是关键的一步。混淆电路的构造从门开始先加密一个门再延伸到加密整个电路。

我们以著名的姚氏百万富翁问题为例，尝试将这一大小比较的函数转化为电路。不妨将两个人的财富用二进制表示为，其中。我们可以逐位比较，并用归纳法来判断它们的大小。

我们定义变量以及其初始值：

在已知的情况下，可以做如下推导。

(9-1)

公式(9-1)描述的逻辑也很直接，即的充分必要条件是 ，或 且。通过这个方法，我们可以依次获得。对应到逻辑电路，由于，可以表示为，可以表示为，其中表示取反，公式(9-1)可以转化成如图9-8所示的逻辑电路（图中圆圈表示取反）。



图9-8 公式(9-1)逻辑电路示意图

我们将上述电路封装成一个三个输入、一个输出的模块。我们将个这样的模块串联起来，就完成了判断的电路，如图9-9所示。



图9-9 混淆电路串联示意图

该电路中，为整个电路的输出。当输出是1时，成立。

上文提到的电路用到了多处取反，并不是最优的。通过观察，我们不难发现。电路的真值表如图9-10所示。



图9-10 电路的真值表

电路可以由此优化为如图9-11所示的电路：



图9-11 优化电路示意图

2）电路生成和求解

首先说明一下，在这个例子中，我们将隐藏掉置换标识的处理，也不严格按照混淆电路协议执行过程，我们更关注于解释整个求解过程。

**Step 1: Alice生成混淆电路**

（1）Alice基于上述电路生成对应的混淆电路，生成过程主要分四步。

① Alice对电路中的每一线路（Wire）进行标注。如图9-12所示，Alice一共标注了七条线路，包括模块的输入输出，和模块内的中间结果。

② 对于每一条线路，Alice生成两个长度为的字符串。

③ 这两个字符串分别对应逻辑上的0和1。

④ 这些生成的标注会在Step 2有选择性地发给Bob，但Bob并不知道对应的逻辑值。



图9-12 电路求值步骤示意图

（2）Alice对电路中的每一个逻辑门的真值表用进行替换，由替换0，由替换1。比如电路图中左上方的XOR门的输入是，输出是，对应的真值表可以做如下转换，如图9-13所示。



图9-13 真值表替换过程

（3）Alice对每一个替换后的真值表的输出进行两次对称加密，加密的密钥是真值表对应行的两个输入。比如真值表的第一行是，我们就用来加密，生成。对于中间的与门，Alice（电路生成方）将以前面的两个门输出的和的导线标签参与门的生成，即用、、、参与对称加密。

**Step 2: Alice和Bob通信**

（1）Alice将她的输入对应的字符串发送给Bob。比如，那Alice会发送给Bob。由于Bob不知道对应的逻辑值，也就无从知晓Alice的秘密了。

（2）Bob通过不经意传输协议从Alice获得他的输入对应的字符串。不经意传输保证了Bob在中获得一个，且Alice不知道Bob获得了哪一个。所以Alice也就无从知晓Bob的了。

（3）Alice将所有逻辑门的乱码表都发给Bob。在这个例子中，一共有四个乱码表。

**Step 3: Bob求解生成的混淆电路**

Alice和Bob通信完成后，Bob开始沿着电路进行解密。此时Bob拥有所有输入的标签和所有乱码表，他可以逐一对每个逻辑门的输出进行解密。假设当前轮迭代中，Bob拥有的输入标签为，他可以：

（1）对于电路图左上方的XOR，用解密获得；

（2）对于电路图左下方的XOR，用解密获得；

（3）对于电路图中间的AND，用解密获得；

（4）对于电路图右侧的XOR，用解密。

值得注意的是，由于乱码表每一行的密钥都不同，所以Bob只能解密其中一行。而且Bob并不知道解密出来的对应的逻辑值，也就无从获得更多信息了。而Alice全程不参与Bob的解密过程，所以也无法获得更多信息。

**Step 4: 共享结果**

最后Alice和Bob共享结果，Alice分享或者Bob分享，双方就能获得电路输出的逻辑值了。

### 姚氏乱码电路协议

图9-14形式化描述了姚氏乱码电路协议的生成过程，图9-15总结了姚氏乱码电路协议的执行过程。为了简化协议的描述，我们给出的是安全性依赖于随机预言机模型的变种协议，但应用较弱的伪随机函数存在性假设也足以完成姚氏乱码电路协议的构造。在加密乱码表中的各个条目时，协议使用了表示的随机预言机。

图9-14 姚氏乱码电路协议：生成过程

**参数**：

·参与方和，其输入分别为和。

·实现函数下的布尔电路。

**生成乱码电路**：

1. 的角色为乱码电路生成方，执行图9-14所示的步骤。随后，将得到的乱码电路(包括输出解码表)发送给。
2. 对于需要提供输入值的导线，向发送输入值所对应的激活标签。
3. 对于需要提供输入值的导线，作为发送方，作为接收方，两方执行OT协议:
   1. 的秘密值为导线上的两个导线标签，的选择比特输入值为在此条导线上的输入值。
   2. OT协议执行完毕后，得到导线的激活标签。
4. 从输人导线的激活标签开始，按照拓扑顺序逐门对收到的求值。

对于乱码表为，输入激活标签为和的门，计算输出激活标签：

1. 应用输出解码表得到输出。对的所有门完成求值后，把第二个密钥设置为“out”，解码最终的输出门，得到明文计算结果。将得到的计算结果发送给，双方均将计算结果作为协议的输出。

**参数**：

·实现函数下的布尔电路。

·安全参数

**生成乱码电路**：

1. **生成导线标签**。对于的每一条导线，随机选择导线标签:
   1. 。
   2. 使得。
2. **构造乱码电路**。对于的每一个门，按照拓扑顺序执行下述步骤：
   1. 假设是一个实现函数的2-输入布尔门，其中输入导线标签为，，，，输出导线标签为，。
   2. 构造的乱码表。的输入值为，输入值共有22种可能的组合。对于每一种组合，设，根据输入导线标签上的置换标识对乱码表中的条目排序，将条目放置在位置上。
3. **输出解码表**。对于每一条输出导线（此导线也是门的输出导线），假设其对应的导线标签为，，要为两个可能的导线值0,1}创建解码表。设（因为是逐比特执行异或计算，所以只需要使用输出的最低位生成）。根据导线标签上的置换标识对解码表中的条目排序，将条目放置在位置上（因为），所以放置的位置不会发生冲突）。

图9-15 姚氏乱码电路协议：求值过程

## 电路优化

### 行约减

1999年，Naor等人[[6]](#footnote-6)考虑了这样一个技巧：不再随机选取输出线路的加密标签，而是选取合适的输出线路加密标签使得混淆表的第一个密文（取决于颜色比特和门函数）总为特殊值，而另一个加密标签仍然随机选取。例如，在图9-16中，混淆表的第一行为，则直接取，即可使得混淆表的第一行密文为零串。这样Garbler就不再需要发送第一个密文了，从而将Garbler在每个门的密文发送的数量从4个减少为3个，Evaluator可以“重构”出第一个密文，然后正常解密即可得到正确的标签。通常把这种技术记为GRR3。第一个密文取特殊值并不会影响混淆电路的安全性，这是因为Evaluator无法有效区分剩下的三个密文到底是针对真值0的加密标签还是1的加密标签。



图9-16 行约减

### FreeXOR技术

Kolesnikov[[7]](#footnote-7)观察到，在GESS导线秘密分享方案中，XOR门的秘密份额数量不随电路深度的增加而增加。Kolesnikov给出了秘密份额数量的下界，证明独立秘密份额的数量成指数级增长是不可避免的。然而，这个下界对XOR门不成立（或者更广泛地讲，对“偶数”门不成立，即真值表输出包含两个0和两个1的门）。

#### 1. GESS方案1

GESS方案中为XOR门生成秘密份额的方法非常简单，令表示输出导线秘密值，选择，并将秘密份额设置为。当输入为时，直接输出。很容易验证电路求值方可以通过这种方式重建出正确的输出导线秘密值。举例来说，。令，我们可以观察到，每一条导线的导线标签偏移量均相同：。

由于所有门导线的秘密份额都拥有相同的偏移量，因此无法将上述XOR门构造方法直接插入到姚氏乱码电路中，这是因为标准姚氏乱码电路要求导线标签都是独立随机选取的。如果在姚氏乱码电路中也这样设置，则会破坏姚氏乱码电路协议的正确性和安全性。之所以会破坏正确性，是因为GESS方案XOR门的输出导线秘密值是在秘密分享的过程中才生成的，而姚氏乱码电路要为预先生成好的输出导线秘密值生成乱码表。之所以会破坏安全性，是因为GESS方案XOR门所生成的秘密值具有相关性，但在姚氏乱码电路中导线标签将作为加密密钥使用，不能具有相关性。从安全性角度看，更大的问题是密钥和加密消息之间具有相关性，引入了循环（Circular）依赖性。

#### 2. FreeXOR方案[[8]](#footnote-8)

FreeXOR方案将GESS方案的XOR门技术引入到乱码电路中。FreeXOR是Kolesnikov 和Schneider提出的乱码电路优化技术。GESS方案XOR门的求值过程不引入任何开销（不需要乱码表，秘密份额数量不会增加），但姚氏乱码电路要为XOR门生成完整的乱码表，求值过程也需要解密乱码表。FreeXOR技术将GESS方案的XOR门生成技术引入到乱码电路中，通过调整乱码电路秘密值的生成过程来解决直接引入此技术引发的正确性问题。FreeXOR 技术提出，增强生成乱码电路乱码表时所使用加密方案的安全性假设，就可以让新的方案满足安全性要求。FreeXOR要求生成乱码电路所有导线标签时，都要使用相同的偏移量，这样就可以将GESS方案的XOR门构造方法引入到乱码电路中。也就是说，对于乱码电路每一条导线的每一对导线标签，我们要求，其中是预先随机选取的。引入导线标签相关性即可让XOR门正确地重建输出导线标签。

图9-17完整地描述了FreeXOR乱码电路协议。为保证安全性，FreeXOR不能像标准姚氏乱码电路那样使用较弱的、基于伪随机数生成器（Pseudo-Random Generator，PRG）的加密方案，而是要使用随机预言机来加密门的输出导线标签。PRG的标准安全性定义不能保证的输出是伪随机的，但换成随机预言机就可以了。Kolesnikov和Schneider 在论文中提到，使用比随机预言机模型稍弱的关联健壮性假设就足以让方案满足安全性要求。Choi等人[[9]](#footnote-9)在理论层面详细论述了FreeXOR所需的密码学假设。他们指出，标准关联健壮性假设实际上不足以使FreeXOR满足安全性要求，证明FreeXOR的安全性要使用关联健壮性的一个特定变种假设。

**参数：**

·实现功能函数的布尔电路；安全参数。

·令表示一个可看成随机预言机的哈希函数。

**协议：**

1. 随机选择全局密钥偏移量：。
2. 对于中的每一条输入导线，随机选择0所对应的输入导线标签：

将另一条输入导线标签设置为。

1. 按照拓扑顺序对的每个门执行下述步骤：
2. 如果是个输入导线标签为的门:

I. 将0所对应的输出导线标签设置为；

II. 将1所对应的输出导线标签设置为。

1. 如果是个输入导线标签为的2输入门：

I. 随机选择0所对应的输出导线标签。

II. 将1所对应的输出导线标签设置为。

III. 创建的乱码表。的输入值共有种可能的组合。对于每一种

可能的组合，设置：

按照输入标识对条目排序，将条放置在位置<>上。

1. 按照图9-14的方法计算输出乱码表。

图9-17 FreeXOR乱码电路协议

FreeXOR乱码电路协议的执行过程与标准姚氏乱码电路协议完全一样，唯一的区别是在图9-14的步骤4中，不需要为XOR门进行任何加解密处理：对于输入导线标签为的XOR门，可以直接计算得到输出导线标签。Kolesnikov等人[[10]](#footnote-10)对 FreeXOR进行了进一步的扩展，提出了fleXOR。在fleXOR中，可以用0个、1个或2个密文构造XOR门的乱码表，具体选择哪种构造方式取决于电路结构和电路中各个门的组合关系。可以让fleXOR与应用在AND门上的GRR2兼容，从而支持2密文AND门。下一节介绍的半门技术比fleXOR更加简单，AND门的密文数量为2，且与FreeXOR完全兼容。

### 半门技术

Zahur等人[[11]](#footnote-11)提出了一种高效的乱码电路构造技术，每个AND门只包含两个密文，且与FreeXOR完全兼容。此技术的关键思想是将AND门表示为两个半门（Half Gates）XOR结果。每个半门都是一个AND门，但其中一个参与方已知此AND门的一个输入。半门的乱码表只包含两个条目，再进一步用GRR3技术将乱码表的密文数量降低到一个。应用半门实现一个AND门需要构造一个电路生成方半门（电路生成方已知此半门的其中一个输入）和一个电路求值方半门（电路求值方已知此半门的其中一个输入）。我们接下来描述半门的构造方法，并讲解如何将半门组合起来实现一个AND门。

电路生成方半门。首先，考虑一个输入导线为和、输出导线为的AND门。电路生成方AND半门要计算，其中电路生成方已知。如果，电路生成方不需要考虑即可知道必为0；如果1，则。我们用，和分别表示导线、和中0所对应的导线标签。应用FreeXOR，可知的导线标签为或者为。电路生成方计算得到两个密文：

应用标识置换优化技术，根据的标识比特在乱码表中设置好两个密文的位置。

为了对半门求值并得到，电路求值方计算（等于或）的哈希值，解密对应的密文。如果电路求值方拥有，则计算并将结果与第一个密文求异或，得到（导线值0对应的正确导线标签）。如果电路求值方拥有，则计算并得到。如果，得到的就是；如果，得到的就是。直观来看，电路求值方永远无法同时得到和，因此非激活标签对电路求值方来说是完全随机的。FreeXOR论文中构造通用电路的编程组件时也无意中用到了这一思想。

可以用乱码行缩减技术，通过适当设置将两个密文进一步缩减为一个密文，这将大幅降低协议的通信开销。

1）电路求值方半门

电路求值方半门要计算的是，其中电路求值方已知，电路生成方不知道此半门的任何一个输入。因此，电路求值方根据导线的明文值应用不同的方法对半门求值。电路生成方提供两个密文：

由于电路求值方已知，因此不需要（也不能）在乱码表中打乱密文的顺序。如果，电路求值方知道其拥有的是，因此计算并得到输出导线标签。如果，电路求值方知道其拥有的是，因此计算并得到。电路求值方随后再对结果和导线标签求异或，从而在不知道或明文值的条件下得到（当）或（当）。与电路生成方半门类似，乱码行缩减技术可将两个密文进一步缩减为一个密文。在这种情况下，电路求值方只需要令（让第一个密文为全0），向电路生成方发送第二个密文。

2）合并半门

剩下要做的就是在乱码电路中把两个半门组合起来，使两个参与方都不知道明文值的条件下对门求值。这里要用到的技巧是电路生成方生成一个完全随机的比特；并用将原始AND门拆分为两个半门：

由于上式可用乘法分配律合并为，因此上式等价于。可以用电路生成方半门构造出第一个AND门。可以用电路求值方半门构造出第二个AND门，但前提是电路生成方要向电路求值方披露。由于是完全随机的，且电路求值方不知道的值，因此向电路求值方披露不会泄漏任何敏感信息。电路生成方不知道，但可以通过下述方法在不引入任何额外开销的条件下将传递给电路求值方。电路生成方将设置为导线中0所对应导线标签的标识比特。由于标识比特本身就是随机选择得到的，因此电路求值方可以在不知道的条件下直接从导线激活标签上的标识比特中得到。

由于FreeXOR不需要为XOR门生成或发送任何乱码表，因此我们只需要两个密文、调用两次、执行两次“免费”的XOR操作，即可完成对AND门的求值。Zahur等人证明，只要生成导线标签时所用的满足关联健壮性假设，则半门技术就是可证明安全的。在包括低延时局域网在内的任何场景下，网络带宽引入的性能开销远远大于加密引入的性能开销，因此半门技术优于任何已知的乱码电路方案。此外还证明，在任何“线性”类乱码电路方案中，门所需的密文数量都不可能少于两个。因此在“线性”假设下，对于任何由2-输入布尔门组成的电路，半门方案都是网络带宽最优方案。

## 算术电路

在工程和计算机科学中，函数被经典地表示为布尔电路，布尔电路基本上由逻辑门（与门、或门、非门等）和连接它们的导线组成。如前面姚氏混淆电路描述的一样，一根导线是1个比特位，逻辑门是基本单元，它可以处理两个比特位的与或非等基本运算。对于比特串上的与或运算，需要通过定义多个逐位的基本逻辑门进行并联。

算术电路函数的更紧凑表示是算术电路，不像布尔电路的导线选择空间是，算术电路导线的值从中选择。门操作为模加法或模乘法，图9-18表示了一个简单的算术电路。



图9-18 简单的算术电路

可以将任何布尔电路表示为上的算术电路，但是，如果的模足够大，那么得到的函数的算术电路表示可能比其他布尔电路表示的大小要小得多，因为对于每个整数的加法或乘法来说，一次操作就足够了。

### 9.3.1 BGW协议[[12]](#footnote-12)

由Ben-Or等人在1988年提出的BGW协议是首批支持多个参与方计算的 MPC（Secure Multi-party Computation）协议之一。我们这里直接介绍个参与方的BGW协议，这个协议描述起来会相对简单一些。

#### 直观思想

**BGW协议可以用于对域上包含加法、乘法、常数乘法门的算术电路求值。**

BGW协议则强依赖于Shamir秘密分享方案[[13]](#footnote-13)，巧妙地利用了Shamir秘密分享方案的同态特性对各个秘密份额进行适当的处理，就可以在秘密值上实现安全计算。

给定，我们令[]表示各个参与方持有的Shamir秘密份额。具体来说，某一参与方选择一个阶最高为的随机多项式，并令。每个参与方把作为的秘密份额。我们称为秘密分享的阈值，即任意个秘密份额都不会泄漏与相关的任何信息。

BGW协议的固定范式为：对于算术电路的每一条导线，各个参与方都持有导线值所对应的秘密份额；每个电路门进行计算的时候，通过拥有的秘密份额（在乘法操作需要交互）完成输出值的秘密份额的计算。

#### 具体构造

1）秘密共享

假设BGW协议的参与者一共有方：，假设参与者需要输入秘密，则参与者首先利用门限秘密共享机制将秘密共享给其他所有参与者，阈值的选择根据具体使用情景下的安全性要求决定。

当所有参与者的输入都通过门限秘密共享机制分享后，每个参与者都掌握了协议输入的子秘密。

假设一个门的输入分别为和，秘密和已经分别由秘密分配函数

分配完成，，，参与者掌握和的子秘密和。

2）输入导线

对于属于参与方的输入导线，知道明文导线值。参与方将秘密份额分发给其他所有参与方。

3）加法门

考虑输入导线为和、输出导线为的加法门。各个参与方共同持有输入导线秘密份额和。参与方的目标是获得输入导线值和求和的秘密份额。假设输入导线值和所对应的多项式分别为和。如果每个参与方在本地对秘密份额求和，得到*+*，则各个参与方将共同持有多项式上的一个点。由于的阶最高也为，因此各参与方所持有的*+*构成了的有效秘密份额。

注意，由于Shamir具有加法同态性，加法门的求值过程不需要参与方之间进行交互。所有计算过程都是在本地完成的。可以利用相同的方法在秘密值上乘以一个公开常数——每个参与方在本地计算秘密份额乘以常数的结果即可。

4）乘法门

考虑输入导线为和、输出导线为的乘法门。各个参与方共同持有输入导线值和的秘密份额[]和[]。参与方的目标是获得输入导线值和乘积的秘密份额[]。如上所述，参与方可以在本地对秘密份额相乘，这使得各个参与方共同持有多项式上的一个点。然而，得到的多项式阶数最高可达到，超过了秘密分享的阈值。

为了解决秘密分享阈值溢出的问题，各个参与方需要一起完成多项式的降阶步骤。每个参与方持有的秘密份额是，其中是一个阶最高可达到的多项式。参与方的目标是得到的有效秘密份额，且对应多项式的阶不超过阈值。

这里要利用的核心结论是，可以用各个参与方秘密份额的线性函数表示。

其中项表示对应的拉格朗日系数。因此，降阶步骤执行过程如下所述：

（1）每个参与方生成的阶秘密分享，并将秘密份额[]分发给其他参与方。为了简化符号表示，我们没有为秘密份额所对应的多项式命名。需要记住的是，每个参与方选择了最高为阶的多项式，且此多项式的常系数为。

（2）各个参与方在本地计算。请注意，该表达式仅涉及秘密份额的加法和常数乘法运算。

由于[]的秘密分享阈值为，因此的秘密分享阈值也为，这就满足了固定范式的要求。

请注意，参与方对BGW协议中的乘法门求值时需要进行交互，即各个参与方需要发送秘密份额[]。还需要注意的是，BGW协议要求，否则由于的阶可能会达到，个参与方没有足够的信息确定的值，因此当时，BGW协议在个参与方被攻陷的条件下是安全的（即BGW协议的安全性依赖于多数诚实假设）。

5）输出导线

电路完成求值后，参与方最终会持有输出导线的秘密份额[]。每个参与方将秘密份额广播给其他参与方，使得所有参与方都能得到。

前述整体过程为算术电路上的运算方法，布尔电路中的运算与前述构造一致。在布尔电路上，可将异或门和与门分别看成在有限域上的加法和乘法。将异或用模为2的加法进行计算，与用模为2的乘法进行计算。

### 9.3.2 Beaver三元组[[14]](#footnote-14)

将MPC协议划分为（参与方输入未知时的）预处理阶段和（参与方选择好输入时的）在线阶段是一种很受欢迎的MPC协议构造范式。预处理阶段为各个参与方生成一些相互之间具有一定关联性的随机量。参与方于在线阶段可以消耗这些随机量。一些主流恶意安全MPC协议也应用了这一范式。

#### 直观思想

为了理解如何将协议中的一部分操作转移到预处理阶段，我们需要回顾一下BGW协议。BGW协议中唯一的实际开销为对每个乘法门求值时的通信开销。然而，由于这一步骤是在对秘密份额进行操作，而参与方只能于在线阶段得到秘密份额（也就是说，秘密份额的取值依赖于电路输入），因此将这部分操作转移到预处理阶段好像并不是那么简单。尽管如此，Beaver提出了一种非常聪明的方法，可以将大部分通信量都转移到预处理阶段。

Beaver三元组（或称乘法三元组）指的是秘密份额三元组，其中和是从某个适当的域中选择出的随机数，而。可以用很多种方法在离线阶段生成Beaver三元组，例如以随机数作为输入直接执行BGW乘法子协议。在线阶段中，每对一个乘法门求值都需要“消耗”一个Beaver三元组。

考虑一个输入导线为和的乘法门。各个参与方持有秘密份额[]和[]。为应用Beaver三元组，计算，参与方执行下述步骤：

（1）各个参与方在本地计算，并打开（即所有参与方均向其他参与方告知自己持有的秘密份额，就可以得到）。虽然的取值依赖于秘密值，但由于秘密值被随机值所掩盖，因此打开不会泄漏与相关的任何信息。

（2）各个参与方在本地计算，并打开。

（3）观察下述等式：

由于和已被打开，而各个参与方持有秘密份额，因此各个参与方在本地即可通过下述公式计算秘密份额[：

[

应用这一技术，只需要公开两个参数即可通过本地计算完成乘法门的求值。总的来说，对每个乘法门求值时，每个参与方需要对外广播两个域元素。而在普通BGW协议中，每个参与方需要（通过安全通信信道）发送个域元素。不过，用这种方式比较性能开销实际上忽略了生成Beaver三元组所引入的计算和通信开销。但需要注意，可以通过一些方法批量生成Beaver三元组，使生成每个Beaver三元组的平均开销仅为每个参与方发送常数个域元素[[15]](#footnote-15)。

#### 抽象

虽然BGW协议（更准确地说是BGW协议的降阶步骤）依赖于Shamir秘密分享方案，但Beaver三元组方法恰当地对BGW协议进行了抽象。实际上，只要“抽象秘密分享方案”的秘密份额满足下述性质，就可以使用Beaver三元组方法：

（1）加同态性：给定和公开值，参与方不需交互即可计算得到以及。

（2）可打开性：给定，参与方可以选择向所有其他参与方披露。

（3）隐私性：攻击者（无论是何种攻击者）都无法从中得到与相关的任何信息。

（4）Beaver三元组：各个参与方可以为每一个乘法门构造满足的随机三元组。

（5）随机输入工具：对于属于参与方的输入导线，各个参与方可以得到一个随机秘密份额，秘密份额对于除的所有参与方来说都是随机的，只有已知。在协议执行过程中，当为此条输入导线选择好输入值后，可以向所有其他参与方公开（但这不会泄漏的任何相关信息），参与方可以利用加同态性于本地计算得到。

只要抽象秘密分享方案满足上述所有性质，Beaver三元组方法就是安全的。进一步，只要抽象秘密分享方案在恶意攻击者的攻击下仍然满足可打开性和隐私性，则Beaver三元组方法也可以抵御恶意攻击者的攻击。如果Beaver三元组方法在恶意攻击者的攻击下是安全的，则恶意攻击者无法伪造出未被打开的秘密值。

## ABY框架及应用实践

本节使用ABY（Arithmetic sharing, Boolean sharing, and Yao’s garbled circuits）[[16]](#footnote-16)框架简要给出数值比较算子的实现。ABY是一个安全双方计算框架，允许双方在敏感数据上评估函数，同时保护数据的隐私。支持三种不同的共享类型：算术共享、布尔共享和姚氏共享，并允许三者之间的有效转换。

数值的比较过程可以简化为：

假设有两个正整数。转换为二进制后，可分别表示为、。如果最高位，则可判定；若，则比较下一位和，循环此过程直至最低位。

定义变量，可得，其对应的逻辑电路图如图9-19所示：



图9-19 逻辑电路示意图

对于正整数来说，将个图中的逻辑电路串联，即可组成完整的数值比较逻辑电路，其中。

#### 搭建ABY框架（在ubuntu系统下进行安装）

① 安装g++、make、cmake、libgmp-dev、libssl-dev、libboost-all-dev、git、doxygen（可选）、graphviz（可选）

sudo apt-get install g++

sudo apt-get install make

sudo apt-get install cmake

sudo apt-get install libgmp-dev

sudo apt-get install libssl-dev

sudo apt-get install libboost-all-dev

sudo apt-get install git

sudo apt-get install doxygen

sudo apt-get install graphviz

② 克隆ABY git库

git clone <https://github.com/encryptogroup/ABY.git>

③ 进入ABY框架目录

cd ABY/

④ 创建并进入build目录

mkdir build && cd build

⑤ 使用CMake进行配置

cmake ..

⑥ 在build目录中调用make

make

#### 程序实现

1. #include <ENCRYPTO\_utils/crypto/crypto.h>
2. #include <ENCRYPTO\_utils/parse\_options.h>
3. #include "ABY/src/abycore/aby/abyparty.h"
4. #include "ABY/src/abycore/circuit/booleancircuits.h"
5. #include "ABY/src/abycore/circuit/arithmeticcircuits.h"
6. #include "ABY/src/abycore/circuit/circuit.h"
7. #include "ABY/src/abycore/sharing/sharing.h"
8. #include "ABY/src/abycore/aby/abyparty.h"
9. #include <math.h>
10. #include <cassert>
11. #define ALICE "ALICE" //客户端
12. #define BOB "BOB" //服务器端
13. /\*命令行输入的相关参数\*/
14. int32\_t read\_test\_options(int32\_t\* argcp, char\*\*\* argvp, e\_role\* role,  
    uint32\_t\* bitlen, uint32\_t\* nvals, uint32\_t\* secparam,  
    std::string\* address, uint16\_t\* port, int32\_t\* test\_op) {
15. uint32\_t int\_role = 0, int\_port = 0;
16. parsing\_ctx options[] = { {(void\*) &int\_role, T\_NUM, "r",   
     "Role: 0/1", true, false},  
     {(void\*) nvals, T\_NUM, "n",   
     "Number of parallel operation elements", false, false},

{(void\*) bitlen, T\_NUM, "b", "Bit-length, default 32",  
 false, false },

{(void\*) secparam, T\_NUM, "s",   
 "Symmetric Security Bits, default: 128", false, false},

{(void\*) address, T\_STR, "a", "IP-address,  
 default: localhost", false, false },   
 {(void\*) &int\_port, T\_NUM, "p", "Port, default: 7766",  
 false, false },   
 {(void\*) test\_op, T\_NUM, "t", "Single test (  
 leave out for all operations), default: off",   
 false, false } };

1. if (!parse\_options(argcp, argvp, options,   
    sizeof(options) / sizeof(parsing\_ctx))) {
2. print\_usage(\*argvp[0], options,  
    sizeof(options)/sizeof(parsing\_ctx));
3. std::cout << "Exiting" << std::endl;
4. exit(0);
5. }
6. assert(int\_role < 2);
7. \*role = (e\_role) int\_role;
8. if (int\_port != 0) {
9. assert(int\_port < 1 << (sizeof(uint16\_t) \* 8));
10. \*port = (uint16\_t) int\_port;
11. }
12. return 1;
13. }
14. /\*构建数值比较电路\*/
15. /\* s\_alice是alice的共享对象
16. s\_bob是bob的共享对象
17. bc是布尔电路对象\*/
18. share\* BuildGTCircuit(share\* s\_alice, share\* s\_bob,   
     BooleanCircuit \*cir){
19. std::vector<uint32\_t> a, b;
20. a = s\_alice->get\_wires();
21. b = s\_bob->get\_wires();
22. /\*将两方输入填充到相同长度\*/
23. uint32\_t maxlen = std::max(a.size(), b.size());
24. if(a.size() != b.size()) {
25. uint32\_t zerogate = cir->PutConstantGate(0, 31);
26. a.resize(maxlen, zerogate);
27. b.resize(maxlen, zerogate);
28. }
29. uint32\_t i, rem;
30. uint32\_t inputbitlen = std::min(a.size(), b.size());
31. std::vector<uint32\_t> agtb(inputbitlen);
32. std::vector<uint32\_t> eq(inputbitlen);
33. for (i = 0; i < inputbitlen; i++) {
34. agtb[i]=cir->PutANDGate(a[i],cir->PutINVGate(b[i]));   
     //比较xi ?> yi
35. }
36. for (i = 1; i < inputbitlen; i++) {
37. eq[i] = cir->PutINVGate(cir->PutXORGate(a[i], b[i]));  
     //判断xi ?= yi
38. }
39. rem = inputbitlen;
40. while (rem > 1) {
41. uint32\_t j = 0;
42. for (i = 0; i < rem;) {
43. if (i + 1 >= rem) {
44. agtb[j] = agtb[i];
45. eq[j] = eq[i];
46. i++;
47. j++;
48. } else {
49. cir->PutANDGate(eq[i+1], agtb[i]));
50. If (j > 0) {
51. eq[j] = cir->PutANDGate(eq[i], eq[i+1]);
52. }
53. i += 2;
54. j++;
55. }
56. }
57. rem = j;
58. }
59. share\* shr = new boolshare(1, cir);
60. shr->set\_wire\_id(0, agtb[0]); //输出结果
61. return shr;
62. }
63. /\* role 程序的执行方：客户端或服务器端
64. address IP地址
65. seclvl 安全级别
66. bitlen 输入的比特长
67. nthreads 线程数量
68. mt\_alg 生成乘法三元组的算法
69. sharing 共享类型对象，比如Yao's共享、布尔共享、算术共享 \*/
70. /\*测试电路\*/
71. int32\_t test\_GT\_circuit(e\_role role, const std::string& address, uint16\_t port, seclvl seclvl, uint32\_t bitlen, uint32\_t nthreads, e\_mt\_gen\_alg mt\_alg, e\_sharing sharing) {
72. /\* 创建ABYParty对象。ABYParty对象定义了所有要进行的操作的基础，基于此对象扮演的角色来执行相应的操作 \*/
73. ABYParty\* party = new ABYParty(role, address, port, seclvl,  
     bitlen, nthreads, mt\_alg);
74. /\* 获得程序中所有可以使用的共享类型 \*/
75. std::vector<Sharing\*>& sharings = party->GetSharings();
76. /\* 基于选择的共享类型创建电路对象 \*/
77. Circuit\* circ = sharings[sharing]->GetCircuitBuildRoutine();
78. /\* 初始化alice和bob的输入，实际应用中，每一方仅提供一个输入值 \*/
79. uint32\_t alice\_input, bob\_input, output;
80. srand(time(NULL));
81. alice\_input = rand();
82. bob\_input = rand();
83. /\* 创建alice和bob原始数据的共享对象s\_alice和s\_bob作为电路的输入，使用PutINGate()将自己的数据输入电路，使用PutDummyINGate()将对方的数据输入电路; s\_out存储电路的输出 \*/
84. share \*s\_alice\_input, \*s\_bob\_input, \*s\_out;
85. if(role == SERVER) {
86. s\_alice\_input = circ->PutDummyINGate(bitlen);
87. s\_bob\_input = circ->PutINGate(bob\_input, bitlen, SERVER);
88. } else { //role == CLIENT
89. s\_alice\_input = circ->PutINGate(alice\_input, bitlen, CLIENT);
90. s\_bob\_input = circ->PutDummyINGate(bitlen);
91. }
92. /\* 传递共享对象和电路对象构建数值比较电路 \*/
93. s\_out = BuildGTCircuit(s\_alice\_input, s\_bob\_input,  
     (BooleanCircuit\*) circ);
94. /\* 使用PutOUTGate将电路输出写入输出共享对象s\_out，可以通过修改参数调整接收输出的对象 \*/
95. s\_out = circ->PutOUTGate(s\_out, ALL);
96. /\* 使用ABYParty对象执行电路 \*/
97. party->ExecCircuit();
98. /\* 将输出共享对象s\_out转换为整数输出 \*/
99. output = s\_out->get\_clear\_value<uint32\_t>();
100. std::cout<<"Testing Greater\_than operator in " <<  
      get\_sharing\_name(sharing) << " sharing: " << std::endl;
101. if(role == CLIENT)
102. std::cout << "\nAlice Input:\t" << alice\_input;
103. if(role == SERVER)
104. std::cout << "\nBob Input:\t" << bob\_input;
105. std::cout << "\nCircuit Result:\t" << (output ? "ALICE greater than BOB" : "BOB greater than ALICE")<<std::endl;
106. /\*输出各个阶段的运行时间  
     std::cout <<"set up time: "<< party->GetTiming(P\_SETUP) << "\n" <<"online time: "<<party->GetTiming(P\_ONLINE)<<"\n"<<"totaltime:"  
     <<party->GetTiming(P\_TOTAL)<<"\n"<<"base ot time: "<<party->GetTiming(P\_BASE\_OT)<<std::endl;\*/
107. delete party;
108. return 0;
109. }
110. int main(int argc, char\*\* argv) {
111. e\_role role;
112. uint32\_t bitlen = 32, nvals = 31, secparam = 128, nthreads = 1;
113. uint16\_t port = 7766;
114. std::string address = "127.0.0.1";
115. int32\_t test\_op = -1;
116. e\_mt\_gen\_alg mt\_alg = MT\_OT;
117. read\_test\_options(&argc, &argv, &role, &bitlen, &nvals,   
      &secparam, &address, &port, &test\_op);
118. seclvl seclvl = get\_sec\_lvl(secparam);
119. //选用GMW执行电路
120. test\_GT\_circuit(role, address, port, seclvl, 32,   
      nthreads, mt\_alg, S\_BOOL);
121. return 0;
122. }

#### 编译并运行

（1）目录结构：

GT\_test

├── ABY

├── CMakeLists.txt

└── GT\_test.cpp

（2）CMakeList.txt如下:

1. find\_package(ABY QUIET)
2. if(ABY\_FOUND)
3. message(STATUS "Found ABY")
4. elseif (NOT ABY\_FOUND AND NOT TARGET ABY::aby)
5. message("ABY was not found: add ABY subdirectory")
6. add\_subdirectory(ABY)
7. endif()
8. add\_executable(GT\_test GT\_test.cpp)
9. target\_link\_libraries(GT\_test ABY::aby)

（3）在GT\_test目录下执行命令

/GT\_test$ cmake .

/GT\_test$ make

（4）在GT\_test目录下打开两个终端，分别输入如下代码，即可运行

./GT\_test -r 0

./GT\_test -r 1

## 课后习题

1. 在姚氏混淆电路的求值方的主要功能是（ ）

A. 对混淆电路的输出进行解密

B. 逐个门计算混淆电路的输出

C. 生成随机的激活标签

D. 控制混淆电路的运行状态

2. 在混淆电路的生成过程中，可能会采用的技术包括（ ）

A. 一次性密码本

B. 标识置换

C. 静态密钥交换

D. 基于公钥密码的加密

1. S Pohlig, M Hellman. An improved algorithm for computing logarithms over GF (p) and its cryptographic significance (corresp.). In IEEE Transactions on information Theory, 1978, 24(1): 106-110. [↑](#footnote-ref-1)
2. B H Bloom. Space/time trade-offs in hash coding with allowable errors. In Communications of the ACM, 1970, 13(7): 422-426. [↑](#footnote-ref-2)
3. M J Freedman, Y Ishai, B Pinkas, and O Reingold. Keyword search and oblivious pseudorandom functions. In Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005: 303-324. [↑](#footnote-ref-3)
4. R Pagh, F F Rodler. Cuckoo hashing. In Journal of Algorithms, 2004, 51(2): 122-144. [↑](#footnote-ref-4)
5. D. Beaver, “Correlated pseudorandomness and the complexity of private computations,” in the Proceedings of the twenty-second annual ACM symposium on Theory of computing (STOC '90), pp. 479-488. [↑](#footnote-ref-5)
6. M. Naor, B. Pinkas, and R. Sumner, “Privacy preserving auctions and mechanism design,” in 1999 ACM Conference on Electronic Commerce, 1999, pp.129–139. [↑](#footnote-ref-6)
7. V. Kolesnikov, “Gate evaluation secret sharing and secure one-round two-party computation,” in 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security, pp. 136-155. [↑](#footnote-ref-7)
8. V. Kolesnikov, and T. Schneider, “Improved garbled circuit: FreeXORgates and applications,” in 2008 International Colloquium on Automata, Languages and Programming - ICALP, pp. 486-498. [↑](#footnote-ref-8)
9. S. G. Choi, J. Katz, R. Kumaresan, and H. S. Zhou, “On the security of the “free-XOR” technique,” in 2012 Theory of Cryptography Conference, 2012, pp. 39-53. [↑](#footnote-ref-9)
10. V. Kolesnikov, P. Mohassel, and M. Rosulek, “FleXOR: Flexible garbling for XOR gates that beats free-XOR,” in 2014 CRYPTO Conference on Cryptology, pp. 440-457. [↑](#footnote-ref-10)
11. S. Zahur, M. Rosulek, and D. Evans, “Two halves make a whole: Reducing data transfer in garbled circuits using half gates,”in 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 220-250. [↑](#footnote-ref-11)
12. M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in 1988 Proceedings of the twentieth annual ACM symposium on Theory of computing, 1988, pp. 1-10. [↑](#footnote-ref-12)
13. A. Shamir, “How to share a secret,” in 1979 Communications of the ACM, 1979, pp. 612-613. [↑](#footnote-ref-13)
14. D. Beaver, “Efficient multiparty protocols using circuit randomization”, in 1992 advances in Cryptology —CRYPTO’91, 1992, pp. 420-432. [↑](#footnote-ref-14)
15. Z. Beerliová-Trubíniová, M. Hirt, “Simple and efficient perfectly-secure asynchronous MPC,” in 2007 international Conference on the Theory and Application of Cryptology and Information Security, pp. 376-392. [↑](#footnote-ref-15)
16. D. Demmler, T. Schneider, and M. Zohner, “ABY-A framework for efficient mixed-protocol secure two-party computation,” in 2015 Proceedings of the Network and Distributed System Security Symposium. [↑](#footnote-ref-16)