PSI的抽象过程，从OTE到OPRF

> 从*OTE*到*Multi-Point OPRF*的发展，基于*OT*以及*ROM(random oracle model)*的*PSI*逐渐在应用层面变得实际，在理论层面，研究也在不断从精妙的*protocol*设计中抽象出来，衍生出利用*OPRF*实现*PSI*的重要概念。

PSI全称为private set intersection，是密码学中多方安全计算MPC领域下，一个非常重要的应用分支，其协议的主要内容是，双方各自输入一个集合，如何在不揭露任何其他信息的情况下，计算并输出双方集合的交集。近年来，利用基于OTE构造OPRF的PSI成为了最有效的PSI协议种类，无论是Single-Point OPRF亦或者Multi-Point OPRF[1]，这些协议在计算量和通讯量上的优化使得千万级甚至亿级别的PSI在公网环境下的应用成为可能，但是从OTE到PSI的过程也是历经了十年之久，其中诞生了一些重要的协议和对OTE的优化方法，本文将从这些方面来对较为重要的结论和协议进行介绍，希望能够较完整的概括整个发展过程。
下文中的讨论局限于***Semi-Honest***假设。

1. PRF、OPRF

PRF全称为Pseudo-Random Function，是密码学中非常重要的概念，密码学一门非常著名的入门级别公开课Stanford的cryptography I中就详细介绍过PRF的概念。PRF的定义为$F:K\times X\rightarrow Y$，其中K是实现决定好的key的空间，一个好的PRF需要满足两个条件，首先PRF的是一个计算efficient的函数，其次该函数与一个从$X\rightarrow Y$ 的随机函数无法分辨。

OPRF的概念完全基于PRF，但是引入了部分OT的概念，所以OPRF协议具有两个参与方，Sender和Receiver（这里与OT协议中的表示保持一致），Sender拥有一个PRF $F:K\times X\rightarrow Y$ 以及一个保密的$k\in K$，Sender在OPRF协议中的输入为PRF与k，Receiver的输入为$x\in X$。协议执行的结果为Receiver获得$F_k(x)$，同时对k一无所知，而Sender则不会获得任何信息，包括$x,F_k(x)$。

以上是PRF和OPRF的基本抽象概念，为了让这些概念看起来不那么枯燥，这里浅谈一下为什么PSI需要OPRF。最直接的原因是，OPRF本身就是PSI这个过程高度抽象后的数学表现形式之一，可能并不是唯一的抽象形式（学识有限难以就这个问题继续探讨）。当然，对于集合A和集合B，在执行PSI协议的过程时，如果集合A,B所处的空间的熵非常大的情况下，则PSI协议将会从OPRF这个抽象形式退化到一种更简单的形式，即Sender和Receiver可以用哈希函数的形式。

1.1 哈希函数形式

集合所处的空间熵大的情况意味着$|X|$非常大，$|X|$是空间的大小，则对于Sender 和Receiver来说，对集合中的所有元素使用一个哈希函数 $H:X\rightarrow Z$，即使双方都拥有对方的元素的$H(x)$，穷举所有的元素来反推对方的集合元素是计算上不可行的，同时H的性质使其无法从结果进行反推，所以双方只需要输出$H(x)$的交集部分即可。

1.2 OPRF形式

对于取值空间有限的两个元素集合，穷举是计算可以执行的情况下，哈希函数下的攻击是异常简单的，只要穷举所有的元素计算哈希值，则该协议不具任何安全性。重新考虑PSI的过程，PSI是一个求交过程，为了使得原始数据不暴露，所以要将原始数据X通过某个函数F映射到“看似”随机的输出Y上，然后在“看似”随机的输出Y上进行求交，如果映射过程F是不可逆的，随机输出的空间$|Y|$足够大的时候，加上双方无法进行用穷举的方法来对所有元素映射到输出空间的时候，协议就是安全的。

以上的过程的F完全可以使用PRF，那么问题是如何确保双方无法穷举的手段计算所有元素的输出并进行攻击？无论是基于盲签名还是OPRF实现的思想是一样的，即Sender一方计算PRF，Receiver一方进行求交，Receiver没有k，则无法穷举，同时OPRF协议中Sender无法获取任何信息，所以即使穷举也不具有可以参照的$F_k(x)$。当然盲签名中利用的是基于公钥加密实现的OPRF，而近几年的几篇高效PSI实现方式则依赖于OTE。

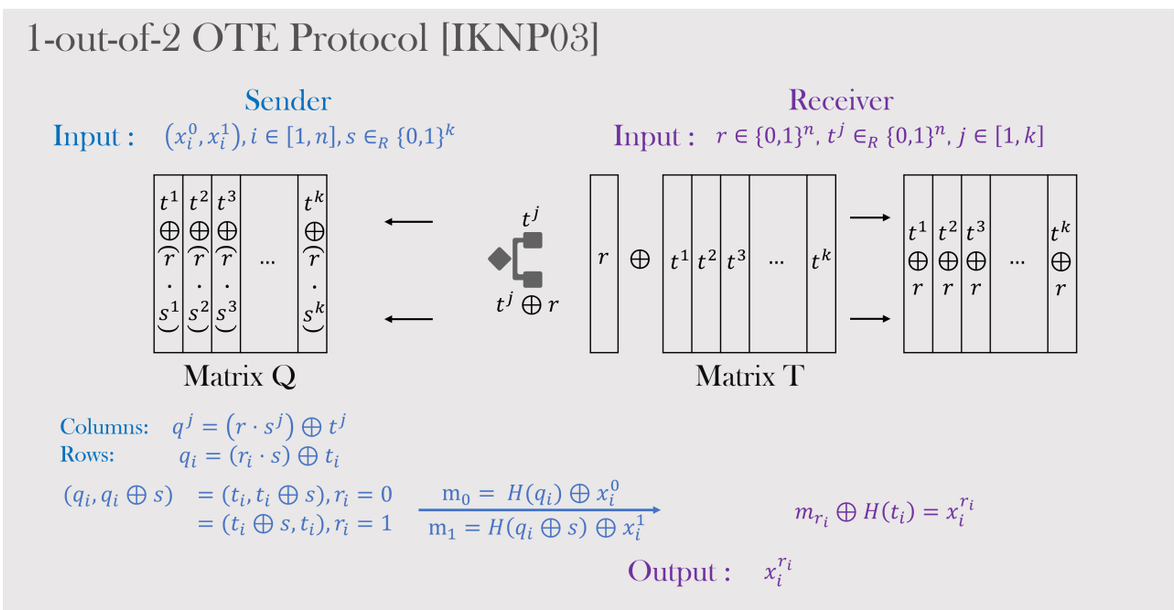
所以顺着上述思路，PSI协议的设计问题则转换为如何实现最有效的OPRF？（一个更有意思和更难的问题是在不依赖与ROM假设的情况下，如何实现OPRF甚至是效率很高的OPRF？引用2[2]利用Paillier这种同态加密手段构造了一种不依赖ROM假设的OPRF，非常有意思）

2. OTE：INKP03，KK13

在理解如何用OTE构造OPRF前必须要了解的是OTE的构造方式。OTE的研究和构造在过去有了不小的突破，从INKP03[3]到KK13[4]，包括在下一部分介绍的Random OTE，都在多方安全计算的领域上发挥了不小的作用。OTE最开始是为了解决运行大量基于公钥非堆成加密体系的OT的计算量、传输量巨大的问题，提出利用少量非对称加密OT + 大量对称加密来降低计算量和传输量。

2.1 INKP03协议

首先，INKP03是1-out-of-2 OTE 协议，下图是流程图，图中矩阵t, q的上标表示列，下标表示行



协议输入：

- Sender 和 Receiver先确定安全参数k，同时k也是基于非对称加密的OT运行的次数；

- Sender的输入是长度为n的消息对，每个消息对中包含两条消息(x_i^0, x_i^1)，同时随机生成一个二进制数组$s \in_R \{0,1\}^k$；
- Receiver的输入是一个长度为n的选择数组r，$r \in \{0,1\}^n$，对于任意的$r_i = 0$，则Receiver想获取Sender第i个消息对中的x_i^0，如果$r_i = 1$，则想要获取x_i^1，同时Receiver随机生成k个长度为n的随机数组t^j，如图中的Matrix T；

协议执行过程和输出：

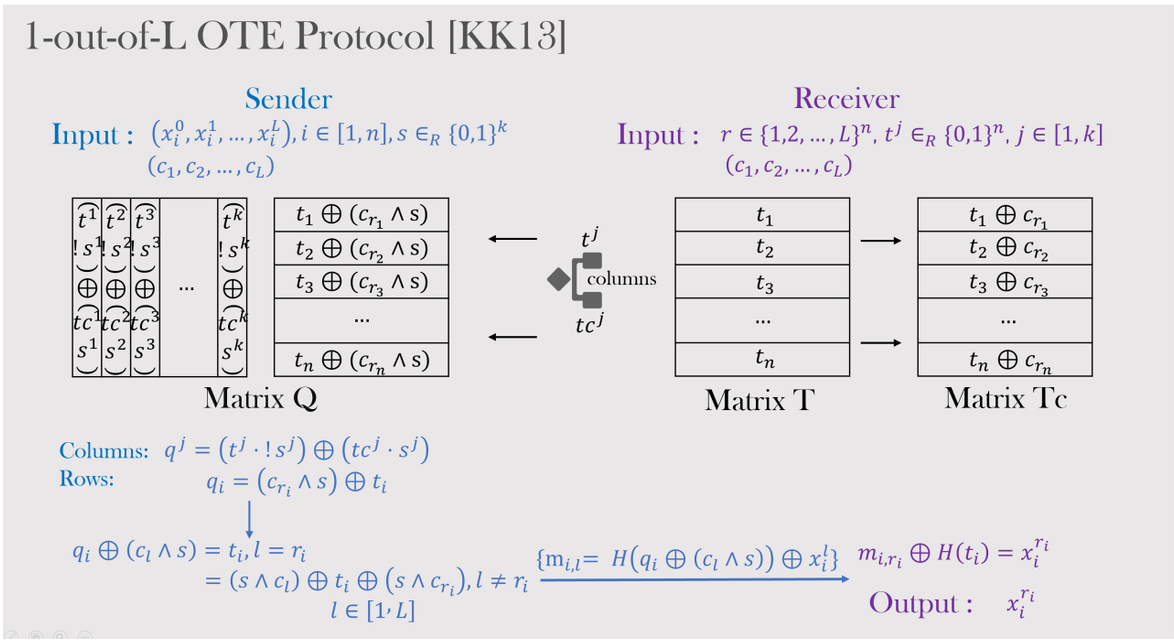
- Receiver如图中所示，由于r与任意的t^j是等长的二进制数组，所以使用$r \oplus T$可以得到一个新的Matrix，对于新的Matrix 和 Matrix T中的每一列，即$(t^j, t^j \oplus r)$，可以发现形成一个新的消息对，这时Receiver作为非对称加密下OT的sender，输入是消息对$(t^j, t^j \oplus r), j \in [1,k]$，而原本的Sender则作为非对称加密下OT的receiver，输入是之前随机生成的数组$s \in_R \{0,1\}^k$；
- 上面的OT协议运行后Sender得到一个Matrix Q，首先Matrix Q中的每一列，当$s^j = 0, q^j = t^j, s^j = 1; q^j = t^j \oplus r$，即$q^j = (r \cdot s^j) \oplus t^j$，则对于Matrix Q的每一行，容易验证$q_i = (r_i \cdot s) \oplus t_i$，因为当$r_i = 0$时，无论$s$的取值，$q_i^j = t_i^j$，当$r_i = 1$时，则$q_i^j = t_i^j \oplus s^j$，所以对于$(q_i, q_i \oplus s)$这个消息对来说，其等于

$$\begin{aligned}(q_i, q_i \oplus s) &= (t_i, t_i \oplus s) \text{ 当 } r_i = 0 \\ &= (t_i \oplus s, t_i) \text{ 当 } r_i = 1\end{aligned}$$

- 由于$s \in \{0,1\}^k$，当k较大时，$H(t_i \oplus s)$对于Receiver来说无法计算的，H是一个哈希函数，所以Sender 发送$(H(q_i) \oplus x_i^0, H(q_i \oplus s) \oplus x_i^1)$，Receiver只能解开其中一个，即$x_i^{r_i}$，以上就是协议的全部内容。

2.2 KK13协议

在下图中是KK13协议的流程图，Receiver初始构造Matrix T是通过每一行来构造，看似与INKP03协议非常不同，实际上后面我们会发现KK13协议是INKP03协议的直接扩展。图中矩阵t, q的上标表示列，下标表示行。



协议输入：

- Sender 和 Receiver先确定安全参数k，同时k也是基于非对称加密的OT运行的次数，其次要确定一套公用的code $(c_1, c_2, \dots, c_L)$，$c_L \in \{0,1\}^k$ L是Receiver选择的范围，也是Sender的每个消息“对”包含的消息个数，其用于混淆Receiver的选择，同时保证安全性；

- Sender的输入是长度为n的消息“对”，每个消息对中包含L条消息$(x_i^1, x_i^2, \ldots, x_i^L)$，同时随机生成一个二进制数组$s \in_R \{0,1\}^k$；
- Receiver的输入是一个长度为n的选择数组r，$r \in \{1,2,\ldots,L\}^n$，对于任意的$r_i = l$，则Receiver想获取Sender第i个消息对中的x_i^l，同时Receiver随机生成n个长度为k的随机数组t_i，如图中的Matrix T；

协议执行过程和输出：

- Receiver如图中所示，根据Matrix T，生成另一个Matrix Tc，Tc中的每一行$tc_i = t_i \oplus c_{r_i}$，然后将两个矩阵根据列划分成$k$列，则得到$(t^j, tc^j), k$ 个消息对，这些消息对可以用作非对称加密下OT的输入，这时Receiver成为sender，Sender成为receiver，输入是之前随机生成的数组$s \in_R \{0,1\}^k$；
- 上面的OT协议运行后Sender得到一个Matrix Q，首先Matrix Q中的每一列，当$s^j = 0, q^j = t^j, s^j = 1, q^j = tc^j$，然后分析Matrix Q的每一行发现

$$\begin{aligned} q_i &= t_i \oplus (c_{r_i} \wedge s), \wedge : AND \\ (q_i \oplus (c_l \wedge s)) &= t_i, l = r_i, \\ &= t_i \oplus (c_{r_i} \wedge s) \oplus (c_l \wedge s), l \neq r_i \end{aligned}$$

- 如果$(c_{r_i} \wedge s) \oplus (c_l \wedge s)$是随机的，则对于Receiver来说$H(t_i \oplus (c_{r_i} \wedge s) \oplus (c_l \wedge s))$是无法计算的，而这要求$c_{l_1}, c_{l_2}, l_1 \neq l_2$之间的距离足够远，否则可能得到很多个0，极大程度上降低了安全参数k。如果存在上述code，则Sender可以发送$\{m_{i,l} = H(q_i \oplus (c_l \oplus s)) \oplus x_i^l, i \in [1,n], l \in [1,L]\}$，其中对于每个$i$，只有Receiver选择的那条才可以进行解密得到。

2.3 其他的一些问题

首先，我们前文中提到KK13协议是INKP03协议的直接扩展，原因是虽然INKP03协议中构建Matrix T是通过列构建的，但是同样也可以通过KK13这种行来构造，只要令$(c_0, c_1) = (\vec{0}, \vec{1})$即可，而INKP03协议无需讨论code安全性的原因是c_0, c_1的距离是最远的，所以$(c_0 \wedge s) \oplus (s \wedge c_1) = s$，而$s$本身就是随机选取的。所以KK13协议是INKP03协议的直接扩展。

Sender发送的消息很长的情况下的处理方式，当$x_i^1, \ldots, x_i^L$的长度很长时，上面的协议可以转换为对对称加密密钥Key的OTE，Receiver拿到Key，然后Sender用对称加密对消息进行加密，Receiver同样只能拿到一个Key。

当n特别大的时候，非对称加密下OT消耗的问题，由于当n特别大的时候，直接进行OT不现实，因为拿RSA体系来说，n越大意味着密钥长度越长，消耗同样巨大，而密钥长度带来的还是不必要的安全性提升（从128提升到512在计算上都是不可能的，这里的128、512都是对称加密下的安全参数，非对称加密要达到128需要远长于128的大数分解）。所以和上面一个小问题类似，初始的OT协议可以传输对称加密的密钥Key，然后Receiver将加密后的两个Matrix发送给Sender，Sender进行解密拼接即可。

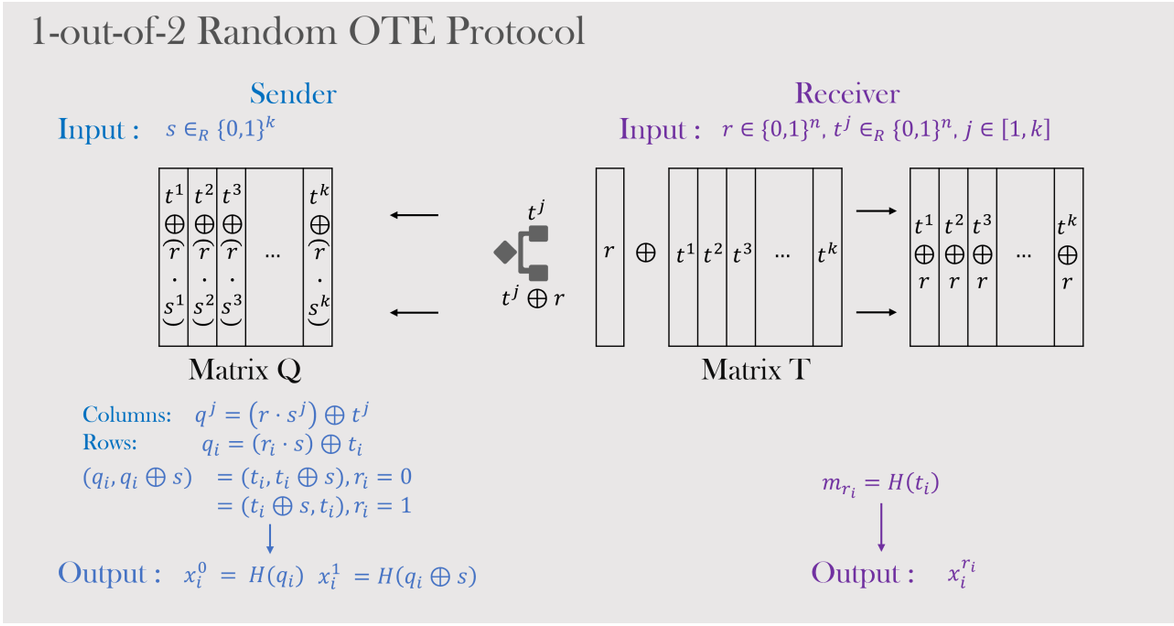
3. Random OTE 随机OTE

在一些OT、OTE的应用环境下，有时候OT协议中，Sender的输入以及Receiver的输出是随机的，在这种情况下Gilad Asharov[5]等人提出了Random OTE的想法，即将信息内禀在Receiver构造Matrix的时候，而由Sender获取Matrix的随机OT来保证随机性，从而节省最后一步中，由Sender向Receiver发送消息。

与传统OTE不同的是，由于Random OTE的信息是内禀在Receiver的Matrix中的，所以Random OTE中Sender的输入并不包含消息对，同时在协议执行结束的时候Sender会获得消息对。这是与传统OTE最大的不同。

2.1 1-2 Random OTE

首先先来介绍1-out-of-2 Random OTE，下图是流程图，图中矩阵t, q的上标表示列，下标表示行



协议输入：

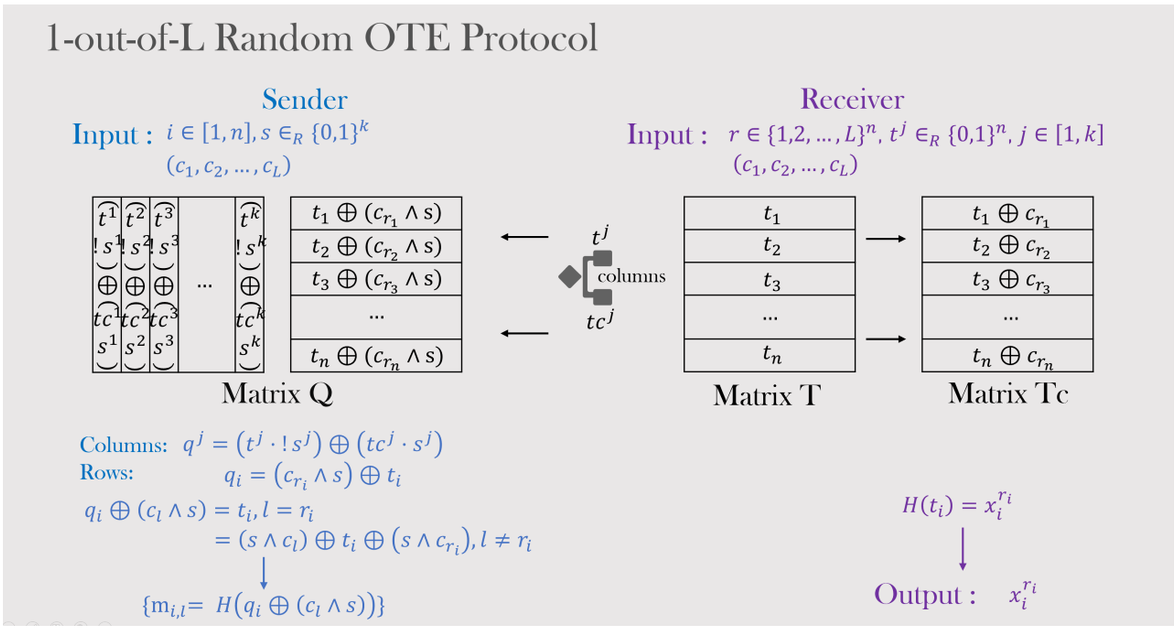
- Sender和Receiver先确认安全参数k;
- Sender的输入是一个随机生成的$s \in_R \{0,1\}^k$;
- Receiver的输入是一个长度为n的选择数组，同时Receiver生成k个长度为n的随机数组t^j组成Matrix T;

协议执行过程和输出

- Receiver对Matrix T中的每一列异或选择数组r，则得到另一个Matrix，其中每一列是$t^j \oplus r$;
- Sender使用k次传统OT获取$(t^j, t^j \oplus r)$中的一条消息，此时如前文中OTE中一样，取决于s^j，然后Sender用k列消息构建Matrix Q。Matrix Q中的每一列满足$q^j = t^j \oplus (r \cdot s^j)$，所以其中的每一行满足条件$q_i = t_i \oplus (r_i \cdot s)$;
- 对于Sender来说，此时需要构建输出，由于$(x_i^0 = H(q_i), x_i^1 = H(q_i \oplus s))$中Receiver仅仅能获知一条，即$x_i^{r_i}$，所以Sender可以构建随机消息对$(x_i^0, x_i^1)$;
- 对于Receiver来说，则可以直接输出$x_i^{r_i} = H(t_i)$，无需等待Sender发送任何消息，前文中在INKP OTE中讨论过为什么是$H(t_i)$;

2.2 1-L Random OTE

L选1 OTE协议与上面2选1 OTE十分类似，Sender需要Receiver的构造矩阵来生成随机消息对，



协议输入：

- Sender 和 Receiver先确定安全参数k，同时k也是基于非对称加密的OT运行的次数，其次要确定一套公用的code $(c_1, c_2, \dots, c_L)$，$c_L \in \{0,1\}^k$ L是Receiver选择的范围，也是Sender的每个消息“对”包含的消息个数，其用于混淆Receiver的选择，同时保证安全性；
- Sender的输入是长度为n的随机生成的二进制数组$s \in_R \{0,1\}^k$；
- Receiver的输入是一个长度为n的选择数组r，$r \in \{1,2,\dots,L\}^n$，对于任意的$r_i = l$，则Receiver想获取Sender第i个消息对中的x_i^l，同时Receiver随机生成n个长度为k的随机数组t_i，如图中的Matrix T；

协议执行过程和输出：

- Receiver如图中所示，根据Matrix T，生成另一个Matrix Tc，Tc中的每一行$tc_i = t_i \oplus c_{r_i}$，然后将两个矩阵根据列划分成$k$列，则得到$(t^j, tc^j), k$ 个消息对，这些消息对可以用作非对称加密下OT的输入，这时Receiver成为sender，Sender成为receiver，输入是之前随机生成的数组$s \in_R \{0,1\}^k$；
- 上面的OT协议运行后Sender得到一个Matrix Q，首先Matrix Q中的每一列，当$s^j = 0, q^j = t^j, s^j = 1, q^j = tc^j$，然后分析Matrix Q的每一行发现

$$\begin{aligned} q_i &= t_i \oplus (c_{r_i} \wedge s), \wedge : AND \\ (q_i \oplus (c_l \wedge s)) &= t_i, l = r_i, \\ &= t_i \oplus (c_{r_i} \wedge s) \oplus (c_l \wedge s), l \neq r_i \end{aligned}$$

- 上面对$\{c_l\}$的作用进行过简要的介绍，这里直接介绍一下Sender如何构建随机消息，由于$q_i \oplus (c_l \wedge s), l \in [1, L]$中Receiver只能知晓其中一条，所以Sender可以构建消息“对”$\{x_i^l\}, x_i^l = H(q_i \oplus (c_l \wedge s))$；
- Receiver则也可以直接获取$x_i^{r_i} = H(t_i)$，无需收到Sender发送的任何消息，省去最后一轮通讯；

2.3 Random OTE和OPRF的联系

在前文中我们对OTE、Random OTE进行了较为详细的描述和介绍，现在终于可以回到OPRF的主题上。首先可以观察的是Random OTE的传统OTE的一种变体，省去了输入，为Sender增加了输出，同时减少了最后一轮的通信，这在L选1中可能会节省大量的通信量。接下来的问题是，Random OTE和OPRF之间的关系是什么？其实仔细观察可以发现Random OTE就是一种OPRF的构造形式。

重新回顾OPRF的定义，在OPRF中，Sender需要拥有k, F，Receiver输入x，最终Receiver拿到$F_k(x)$，而Sender对于$F_k(x), x$一无所知。首先对于每个x，$x \in X$，X是Receiver的输入的空间。则对于2选1的Random OTE来说，$X = \{0, 1\}$，也就是说Receiver的选择输入空间是$0, 1$，而Sender虽然可以穷举输入空间，但是不能获知Receiver输入和输出的是哪条信息。对于广义的其他OPRF也是如此，Sender永远都可以穷举输入空间，得到输出空间，但是Sender却无法在其中对Receiver的输入产生映射。

对于L选1的Random OTE同样也是如此，如果$L = 2^\sigma$，则Receiver的输入空间是$\{0, 1\}^\sigma$，同样构造了OPRF。

4. 基于**OTE**的**OPRF**到**PSI**

我们知道，Random OTE是一种OPRF的构造方式，接下来我们对如何构建这种OPRF进行更为详细介绍，同时对两个比较重要的PSI协议进行较为详细的描述，分别是PSZ18[6]协议和CM18[7]协议。

4.1 **OPRF**的构建过程

上文中对如何Random OTE和OPRF的关系进行了简单介绍，下面详细来描述OPRF的构建过程。假设Receiver的输入x的所处空间为$X = \{0, 1\}^\sigma$，当$\sigma = 1$时，OPRF直接将退化为2选1 Random OTE。当$\sigma > 1$时，则情况略有不同，我们知道对于L选1的Random OTE来说，Receiver和Sender最开始要约定好一套code，$c_l, l \in [1, L]$，对于Receiver的每个选择，Receiver在生成Matrix的时候都会使用相应的$c_{r_i} \oplus t_i$生成另一个矩阵。

我们知道，如果$L = 2^\sigma$，则我们有对于X中的每一个元素，都有$x_l \to c_l$的双射，满足关系$x_l \to c_{x_l}$。则Receiver在OPRF中的输入是x等价于Receiver在Random OTE中挑选第x个消息，所以成功构建了OPRF，最终Receiver拿到的第x消息，就是$F_k(x)$，因为我们知道$H(q_i \oplus (c_l \wedge s))$对于每个$c_l$都不同，则对于每个$x_l$都不同。

而当OPRF用在PSI中时，不仅要Receiver拿到输出，Sender也要对自己的元素拿到输出，由于Sender的输入空间也一定是X，所以我们一定有，对于每一个Sender的输出x_s，我们都可以找到一个c_s与之相对应，为c_{x_s}，所以可以计算

$$F_k(x_s) = H(q_i \oplus (c_{x_s} \wedge s))$$

根据前文的讨论可知，$k = s$，是Sender作为初始OT的receiver方时的选择数组。

4.2 **PSZ18**协议

PSZ18协议中，用到了在不少PSI算法中都使用到的Cuckoo Hash的方法，Cuckoo Hash的想法在于尽可能多的将元素放在一个Hash Table中，为了避免碰撞，Cuckoo Hash选择增加Hash的个数，如果发生碰撞，就将之前放在其中的元素拿出来，然后选取新的Hash函数，直到元素被放进去或者某个元素的所有Hash函数都已经尝试过，对于那些没有最终放入的元素，放在一个特殊的列表中。

当然，只有一方可以使用Cuckoo Hash的方法，另一方则需要用全部的Hash函数对全部的元素都做一边Hash Table，这样才能确保交集中元素不会被漏掉。

原文中花了大量的篇幅来讨论Cuckoo Hash中Hash Table的大小、Hash函数个数的选择等等问题。我们这里不再进行赘述，主要集中在如何做PSI协议层面。

另一个值得一提的Hash技巧是permutation-based Hash，即将元素放入Hash Table的时候，在位置中隐藏部分元素的信息，进而起到缩短放入元素的长度的作用。具体做法是将x分为$x_l|x_r$两个部分，假设$x_l \in \{0,1\}^\mu$，找一个函数f使得$f(x_r) \to \{0,1\}^\mu$，其中2^μ是Hash Table的大小，然后计算$f(x_r) \oplus x_l$就是放入元素的Hash Table中Bin的编号。所以我们可以看到x_r所处的空间应小于$f(x_r)$即x_l也就是Hash Table的大小，否则就会产生n对1映射，更有可能产生多个元素放入同一个bin中的现象。

Sender's View: Normal Hash

Elements_D	...	Elements_1	Index
$x^1_{r,D}$	...	$x^1_{r,1}$	1
$x^2_{r,D}$	...	$x^2_{r,1}$	2
...	...	...	...
$x^i_{r,D}$	...	$x^i_{r,1}$	$x^i_l \oplus f(x^i_r)$
$x^{i+1}_{r,D}$	...	$x^{i+1}_{r,1}$	$x^{i+1}_l \oplus f(x^{i+1}_r)$
...	...	...	...
$x^{2^\mu}_{r,D}$	...	$x^{2^\mu}_{r,1}$	$x^{2^\mu}_l \oplus f(x^{2^\mu}_r)$

Receiver's View: Cuckoo Hash

Index	Elements
1	x^1_r
2	x^2_r
...	...
$x^i_l \oplus f(x^i_r)$	x^i_r
$x^{i+1}_l \oplus f(x^{i+1}_r)$	x^{i+1}_r
...	...
$x^{2^\mu}_l \oplus f(x^{2^\mu}_r)$	$x^{2^\mu}_r$

上图是Hash后的Sender和Receiver拿到的Hash Table，其中Receiver的每个bin中只有一个元素，而Sender的bin中可能有多个元素，最多可能是D个（概率决定）。所以整个PSI协议的过程就是双方先生成Hash Table，然后Receiver根据每个Hash bin中计算OPRF函数，同时Sender也对自己集合中的元素计算OPRF函数值，然后Sender将元素对应的OPRF打乱后发送给Receiver，Receiver求得最终的交集并输出。

协议输入：

- Sender输入集合$X_S = \{x_{s,1}, x_{s,2}, \ldots, x_{s,n_s}\}$；
- Receiver输入集合$X_R = \{x_{R,1}, x_{R,2}, \ldots, x_{R,n_R}\}$；
- Sender和Receiver约定安全参数k，以及$x_l|x_r$中x_r的空间大小L，即1-Out-Of-L OTE中的L，同时约定Hash Table 的大小2^b，即x_l的空间大小。同时约定一套code $\{c_l\}$，每个code长度为b，有L个，即$c_l, l \in [1, L], c_l \in \{0,1\}^b$；

协议执行过程与输出：

- 首先像前图中一样，Sender和Receiver将自己的集合元素全部放入Hash Table中，其中Receiver放入Cuckoo Hash Table中，Sender放入普通Hash Table中，其中可能会有多个Hash函数使用；
- 可知Hash Table的大小为2^b，所以我们需要执行一个Random OTE协议，在这个协议中，Receiver的输入是一个选择数组，选择$[1, L]$中的一个随机元素，而前文中我们知道Hash Table中放置的元素即x_r的取值范围也是$[0, L-1]$，所以可以用x_r的真实值作为选择数组中的值，所以Receiver具有2^b长度的一个选择数组，为

$$r = (x^1_r, x^2_r, \cdots, x^{2^b}_r)_R$$

- Sender的输入是普通的Hash Table，用到了多个Hash函数，一个bin中可能包含多个值。而在Random OTE中，Sender则需要再输入一个随机的选择数组，也就是在Random OTE协议中的s，根据实现决定的安全参数k，则$s \in \{0,1\}^k$；
- 然后如同Random OTE中一样，Receiver构建Matrix T，Matrix Tc，然后Sender接受Matrix根据选择数组s，构建Matrix Q，其中每一行满足$q_i = t_i \oplus (s \wedge c_{x^i_{r,R}})$。此时Receiver无需接受Sender返回任何消息，可以直接构建对应于自己输入的OPRF，即$H(t_i)$。同时Sender可以对自己第i个bin中元素$x^{i,1}_{r,S}, x^{i,2}_{r,S}, \ldots, x^{i,D}_{r,S}$分别计算OPRF

$$F_k(x_{r,S}^{i,j}) = H(q_i \oplus (s \wedge c_{x_{r,s}^{i,j}})), j \in [1, D]$$

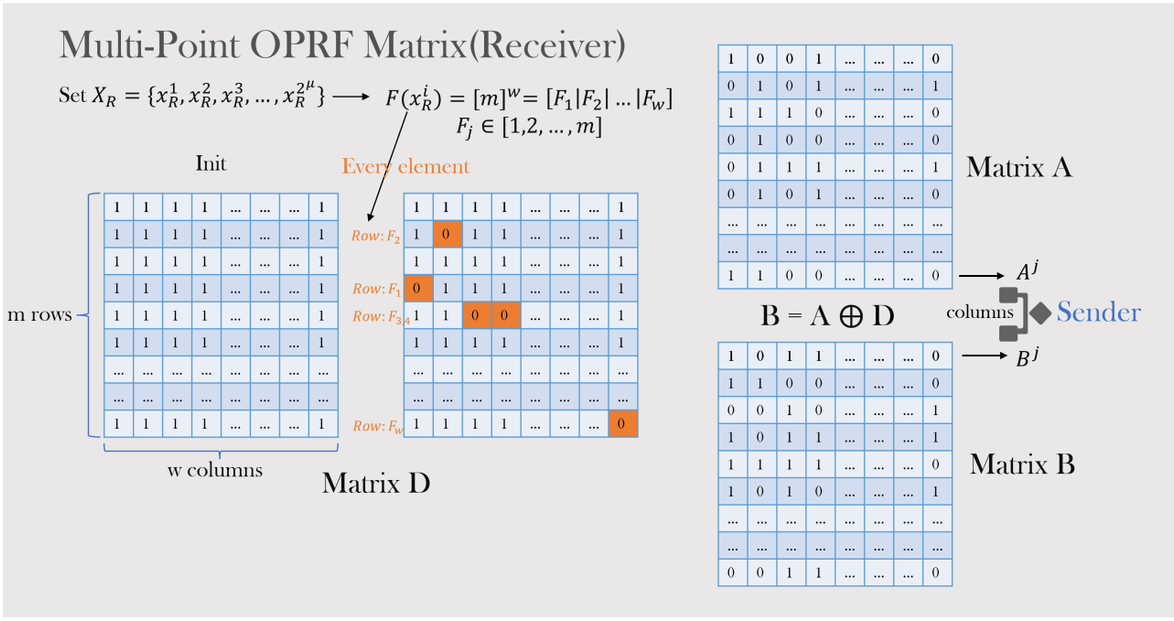
- Sender在对自己的所有元素计算完OPRF之后，将得到的所有OPRF值打乱后发送给Receiver，Receiver在其中查找是否有与自己得到的OPRF重合的值，如果有，则自己的某个元素在交集中。可以将OPRF直接打乱发送的原因是$q_i = t_i \oplus (s \wedge c_{x_{r,R}^i})$，其中$t_i$是随机的，所以即使$x_{r,R}^{i_1} = x_{r,R}^{i_2}$，但是$t_{i_1} \neq t_{i_2}$，所以$q_{i_1} \neq q_{i_2}$，最终拿到的Hash函数则不同。

4.3 CM18协议

与上面PSZ18协议不同的是，CM18协议是一种multi-point OPRF PSI，而PSZ18被称为single-point OPRF PSI，之所以用multi-point可以看到在PSI协议的构建中，某个元素是否在集合中取决于在矩阵中的多个点的位置，而并非像PSZ18协议中，用单个OPRF描述。

与PSZ18相同的是，CM18协议也依赖于ROM假设、Random OTE协议。所以Receiver同样需要构建Matrix，而该协议核心的部分就是Matrix的构建部分，首先我们将介绍如何去构建Matrix。

下图是Receiver生成Matrix的过程。



Receiver先生成一个$m \times k$的全1矩阵，即m行k列，记为Matrix D，然后Receiver需要一个PRF函数，将集合中的元素映射到$\{0,1,\ldots,m\}^w$，也就是$F_k(x_R^i) = [F_1|F_2|\ldots|F_w]$，其中$F_j \in [1,2,\ldots,m]$。然后对于每个元素都有上述PRF函数的映射，然后将映射到的值，对应的Matrix D中的第j列、第F_j行置为0。

然后Receiver生成一个随机Matrix A，同时异或上面生成的Matrix D，得到Matrix B。Matrix A 和 Matrix B的每一列将作为OT中的输入。这里我们可知，如果一个元素$x^* \in X_R$，也就是Receiver的集合中，那么所有$H(x^*)$所对应的位置中，Matrix D的值全为0，所以Matirx B中这些位置的值与Matrix A中一致（异或0等于自身）。

以上就是整个Matrix生成的逻辑和步骤，下面来介绍整个PSI协议流程。

协议输入：

- Sender输入集合$X_S = \{x_{s,1}, x_{s,2}, \ldots, x_{s,n_s}\}$；
- Receiver输入集合$X_R = \{x_{R,1}, x_{R,2}, \ldots, x_{R,n_R}\}$；

- Sender和Receiver约定安全参数k，也就是这里的w，同时Receiver输入一个PRF的key k，Sender输入一个关于Matrix的选择数组$s \in \{0,1\}^w$；

协议执行过程与输出：

- Receiver先按照上面图中所示构建Matrix D、A、B，然后将Matrix A、B中的w列作为2选1 OT的输入，Sender则输入s来选择，如果$s^j = 0$，则选择A^j列，如果$s^j = 1$，则选择B^j列；
- Sender在拿到所有选择后的列后构建Matrix C，其中我们知道Matrix C中的每一列满足$C^j = A^j \oplus (s^j \cdot D^j)$，同时我们也有每一行满足

  $$C_i = A_i \oplus (s \wedge D_i), i \in [1, m]$$

  此时Receiver发送PRF的key给Sender，Sender可以计算自己集合中所有元素的PRF值；
- Sender对于每个自己集合中的元素，都可以计算$F_k(x_S^i) = [F_1^i | F_2^i | \dots | F_w^i]$，然后找到Matrix C中对应的$w$个位置，我们知道对于这些位置而言，如果$x_S^i \in X_R$，也就是在交集中，则一定有对应的位置$D_i^j = 0$，所以一定有Matrix C的位置与Matrix A的位置的值一样。所以Sender此时把这些对应位置的值抽出来，计算

  $$H(C_1^{F_1^i} | C_2^{F_2^i} | \dots | C_w^{F_w^i}), i \in [1, |X_s|]$$

  发送给Receiver，注意是对每一个元素都做这样的计算；
- Receiver在接受到这些值后，所作的操作就比较简单了，我们知道对于在交集里的元素，一定有

  $$H(C_1^{F_1^i} | C_2^{F_2^i} | \dots | C_w^{F_w^i}) = H(A_1^{F_1^i} | A_2^{F_2^i} | \dots | A_w^{F_w^i})$$

  所以Receiver可以将自己集合中的所有元素的对应的Matrix A中的值抽离出来，计算上述Hash 值，然后查找是否在Receiver发送过来的集合中有重合，若有，则该元素在交集里。