Fast Database Joins and PSI for Secret Shared Data

> - 📜【MRR 2020】*Fast Database Joins and PSI for Secret Shared Data*
>   - *Author(s)*：Payman Mohassel，Peter Rindal，Mike Rosulek
>   - *Venue*：CCS
>   - *Materials*：PDF，Video，Code

Overview

论文贡献：提出了一个协议**Join Protocol**——能够在秘密份额的形式下实现**快速**的数据库连接(Database Joins)

协议效率：协议是常数轮的，计算和通信的开销与表的大小成线性关系即 $O(n)$

支持的数据库连接(*Database Joins*)类型：

- Inner Join (PSI)
- Full Join(Union)
- Left Join(PSI)
- Outer Join(Union)

> 连接协议还支持使用任意的谓词过滤选择 where 子句. 比如：
>
> $$select\ X_2\ from\ X\ inner\ join\ Y\ on\ X_1 = Y_1\ where\ X_2 > 3$$

除了这些不同的连接操作之外，框架还支持对表的两大类操作：

- 通用的 **SQL select** 语句，它可以对每一行执行计算（例如计算两列的最大值）并使用 where 子句谓词过滤结果。
- 聚合，它对表的所有行执行操作——例如，计算给定列的总和、计数或最大值。

> **协议只能对唯一主键进行连接**
>
> “Our core protocol requires each table to contain unique values in the column defining the join (i.e., we can only join on “unique primary keys”）”

Setting

协议的基本设定

敌手模型	敌手数量	参与方	外包计算
Semi-honest	Honest majority	3 party	支持

协议的秘密共享框架

> 论文作者采用了 ABY^3 的**复制秘密共享**，他们本身也是 ABY^3 的作者（Peter Rindal，Mike Rosulek）

复制秘密共享

将秘密 x 分成三个随机数，满足：$x = x_1 + x_2 + x_3$

共享表示为：$[\![x]\!] := (x_1, x_2, x_3)$

其中三个计算方拥有的共享值为：

Server 1：$[\![x]\!]_1 = (x_1, x_2)$

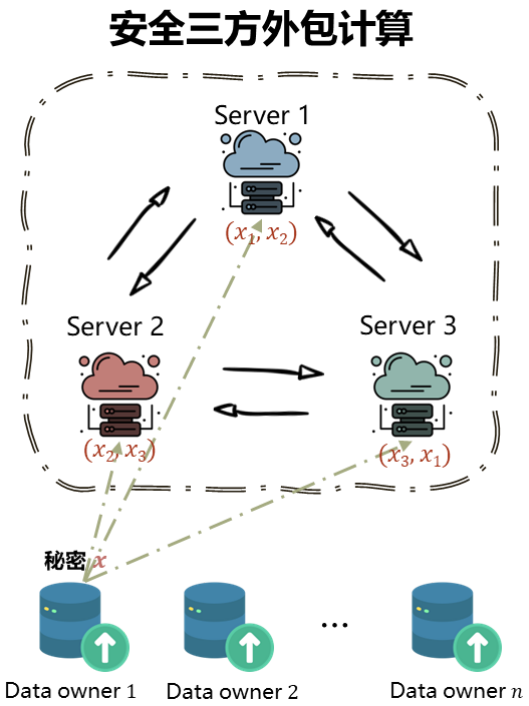
Server 2：$[\![x]\!]_2 = (x_2, x_3)$

Server 3：$[\![x]\!]_3 = (x_1, x_3)$

在本文中复制秘密共享可以在本地转化为加性秘密共享即 $《x》 := (x_1, x_2)$ 满足 $x = x_1 + x_2$，同样加性秘密共享也可以转化为复制秘密共享，但需要一轮通信。

安全三方外包计算

三个非共谋(non-colluding)服务器作为计算方，其输入可以由服务器本身提供，也可以由外部的多个数据拥有者通过复制秘密共享的形式提供

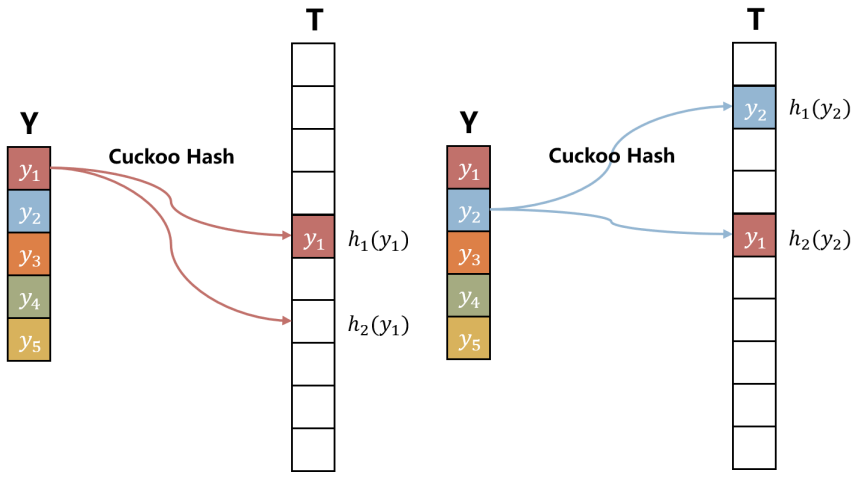


Join Protocol(without any privacy)

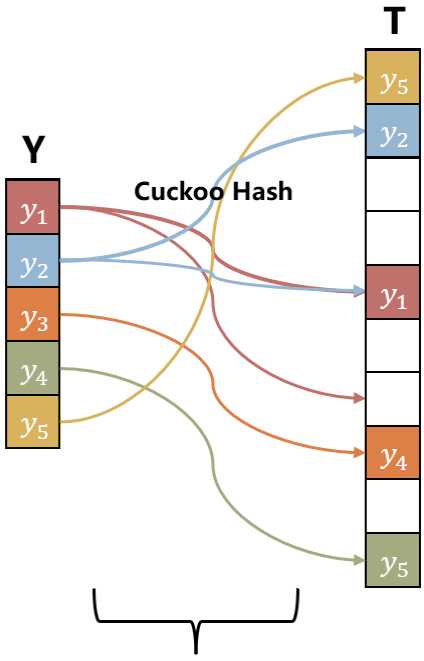
我们先从明文状态下的数据库介绍连接协议的框架，可以分为四个阶段：

假设现在有两张表，X 和 Y 分别是两张表的连接键，颜色相同表示数值相同，如本节最后一图中所示即 $y_1 = x_1, y_2 = x_4$

1. 利用布谷鸟哈希将表 **Y** 映射到哈希表 **T** 中



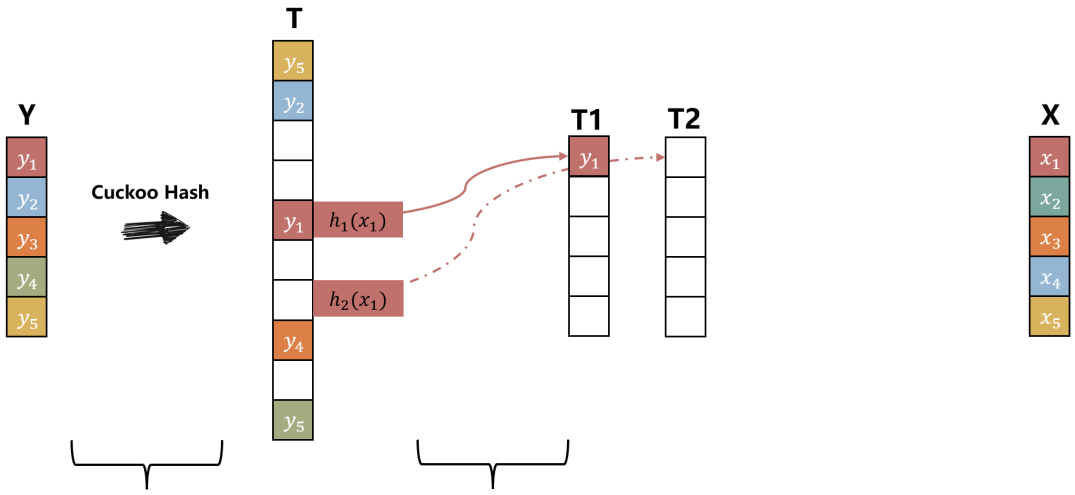
> **布谷鸟哈希（Cuckoo Hash）：**
>
> 1.当两个哈希任意位置为空，则选择一个位置插入
>
> 2.当两个哈希有位置为空时，则插入到空位置
>
> 3.当两个哈希位置均不为空时，随机选择两者之一的位置上key踢出，计算踢出的key 另一个哈希对应的位置进行插入



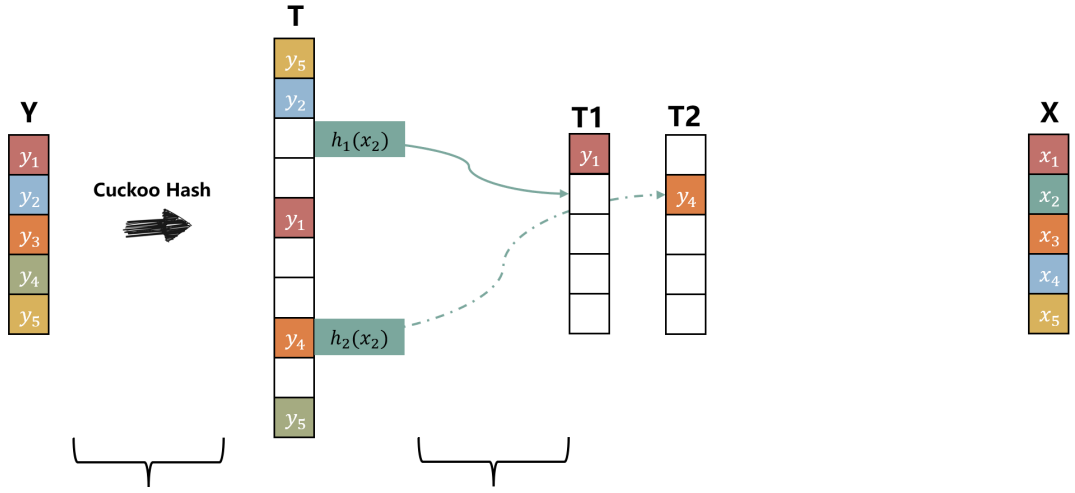
1. 利用布谷鸟哈希将表 **Y** 映射到哈希表 **T** 中

---

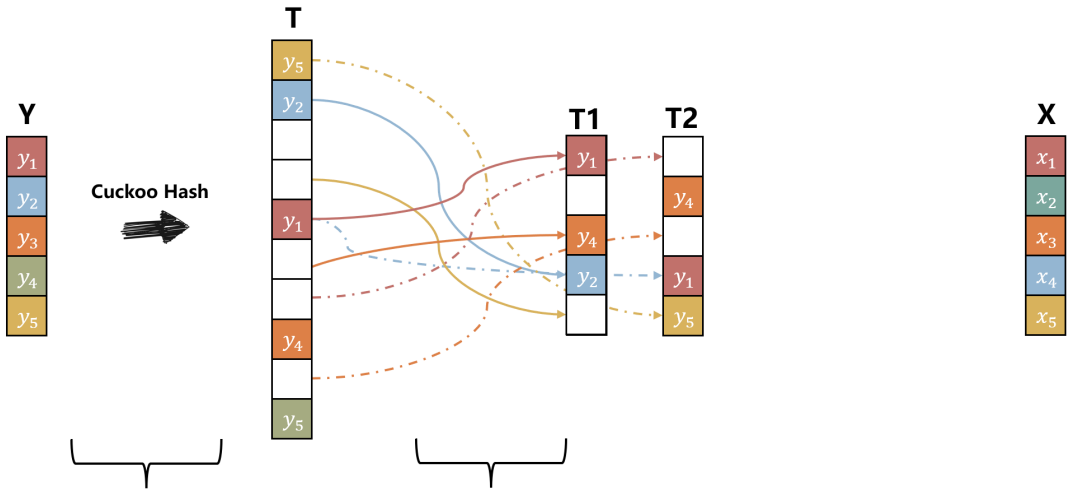
2. 对表 **X** 进行布谷鸟哈希后与表 **T** 进行对比，如果是 h_1 匹配则放入新表 **T1** 中， h_2 匹配则放入新表 **T2** 中



1. 利用布谷鸟哈希将表 **Y** 映射到哈希表 **T** 中
2. 对表 **X** 进行布谷鸟哈希后与表 **T** 进行对比，如果是 h_1 匹配则放入新表 **T1** 中，h_2 匹配放入 **T2**

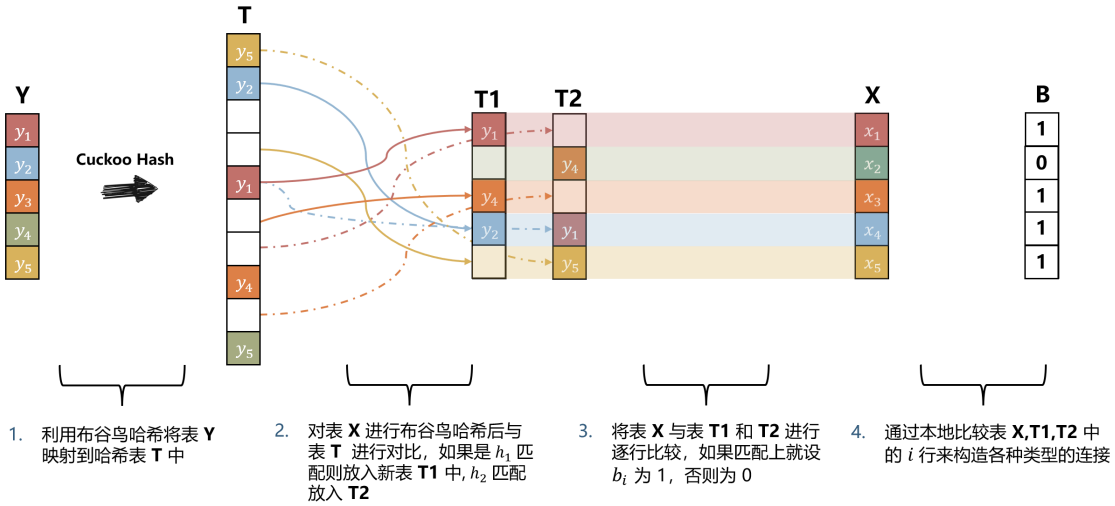


1. 利用布谷鸟哈希将表 **Y** 映射到哈希表 **T** 中
2. 对表 **X** 进行布谷鸟哈希后与表 **T** 进行对比，如果是 h_1 匹配则放入新表 **T1** 中，h_2 匹配放入 **T2**



1. 利用布谷鸟哈希将表 **Y** 映射到哈希表 **T** 中
2. 对表 **X** 进行布谷鸟哈希后与表 **T** 进行对比，如果是 h_1 匹配则放入新表 **T1** 中，h_2 匹配放入 **T2**

3. 将表 **X** 与表 **T1** 和 **T2** 进行逐行比较，如果匹配上就设 b_i 为 1，否则为 0
4. 通过本地比较表 **X,T1,T2** 中的 i 行来构造各种类型的连接



Join Protocol(translates to the secret shared setting)

现在我们的任务是要将对连接框架进行修改，使其在秘密共享下进行。

那么这里就有两个最主要的挑战：

1. 如果在不泄露 Y 的前提下构建布谷鸟哈希表
2. 如何从布谷鸟哈希表中选取值

论文对其 ***Join Protocol*** 添加了一些协议，具体变化如下：

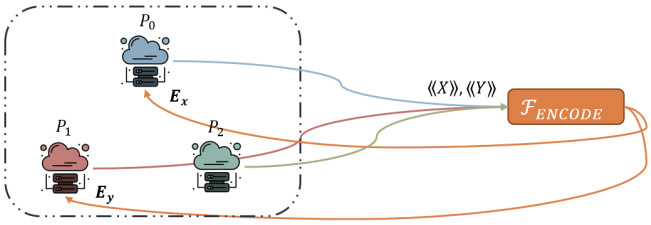
1. 计算连接键的 ***randomized encoding***
2. 参与方 P_1 根据表 Y 构建布谷鸟哈希表 T ，之后利用 ***permutation protocols*** 重排秘密份额
3. 对于表 X 的每行 x , P_0 使用 ***oblivious switching network*** 将其映射到布谷鸟哈希表相应的位置 i_1, i_2 根据秘密共享三元组 $(x, T[i_1], T[i_2])$
4. x 连接键与 $T[i_1], T[i_2]$ 进行比较，只要其中有一个匹配上了，则填充相应的 $Y^{'}$ 行，否则相应的 $Y^{'}$ 行设置为 *NULL*
5. 然后可以通过比较 X 和 $Y^{'}$ 的第 i 行来构造各种类型的连接。

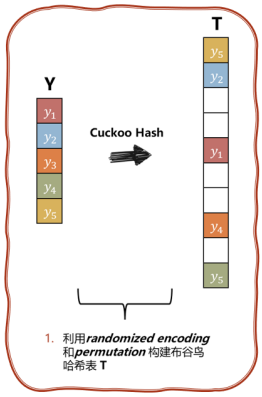
1. 如何构建布谷鸟哈希表

作者提出了一个$\mathcal{F}_{ENCODE}$ 协议， 我们可以将其想象成一个黑盒子，其功能如下：

> $\mathcal{F}_{ENCODE}$:
>
> ***input :*** 所有参与方对应的 X 和 Y 的秘密份额
>
> ***output :*** 返回 Y 的加密值 E_y 给 P_1, X 的加密值 E_x 给 P_0

- Randomized Encodings
  - 所有参与方将其 X 和 Y 的秘密份额发送给 $\mathcal{F}_{ENCODE}$
  - $\mathcal{F}_{ENCODE}$返回 Y 的加密值 E_y 给 P_1, X 的加密值 E_x 给 P_0



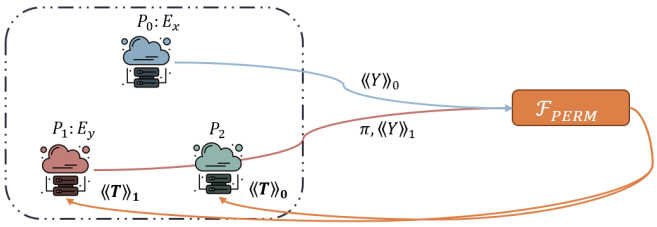


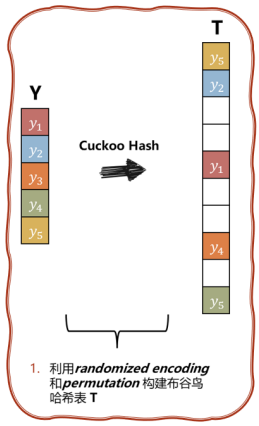
首先，对连接键进行 ***randomized encoding***

接下来应用$\mathcal{F}_{PERM}$ 协议构建布谷鸟哈希表，其功能如下：

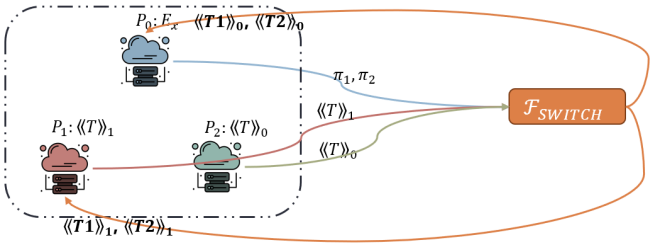
> $\mathcal{F}_{PERM}$：
>
> ***input：*** P_1 将发送的 $\langle\langle Y\rangle\rangle_1$和 π， P_0 发送 $\langle\langle Y\rangle\rangle_0$
>
> ***output：*** 返回 $\langle\langle T\rangle\rangle_1$ 给 P_1 ,返回 $\langle\langle T\rangle\rangle_0$ 给 P_2

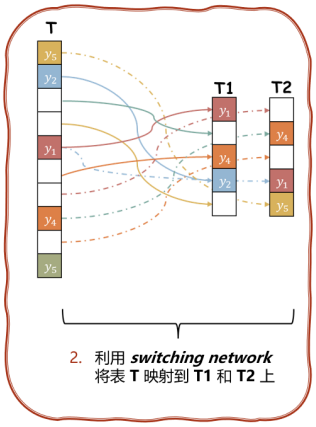
- Constructing the Cuckoo Table
  - P_1 本地对 E_y 进行布谷鸟哈希，并记录下映射关系 π
  - P_0 和 P_1 本地将 $[\![Y]\!]$ 转换为 $\langle\langle Y\rangle\rangle$
  - P_1 将 $\langle\langle Y\rangle\rangle_1$和 π 发送给 $\mathcal{F}_{PERM}$， P_0 将 $\langle\langle Y\rangle\rangle_0$ 发送给 $\mathcal{F}_{PERM}$
  - $\mathcal{F}_{PERM}$返回 $\langle\langle T\rangle\rangle_1$ 给 P_1 ,返回 $\langle\langle T\rangle\rangle_0$ 给 P_2





- Selecting from the Cuckoo Table
  - P_0 构造映射函数 π_1 , π_2分别对应 E_x 的两个哈谷鸟哈希映射， 即$\pi_l(i) = j$ 满足 $h_l(E_x[i]) = j$
  - P_0 将 π_1 , π_2 和 发送给 $\mathcal{F}_{SWITCH}$, P_1 将 $\langle\langle T\rangle\rangle_1$ 发送给 $\mathcal{F}_{SWITCH}$, P_2 将 $\langle\langle T\rangle\rangle_0$ 发送给 $\mathcal{F}_{SWITCH}$
  - $\mathcal{F}_{SWITCH}$ 返回 $\langle\langle T1\rangle\rangle_0$, $\langle\langle T2\rangle\rangle_0$ 给 P_0 ,返回 $\langle\langle T1\rangle\rangle_1$, $\langle\langle T2\rangle\rangle_1$ 给 P_1



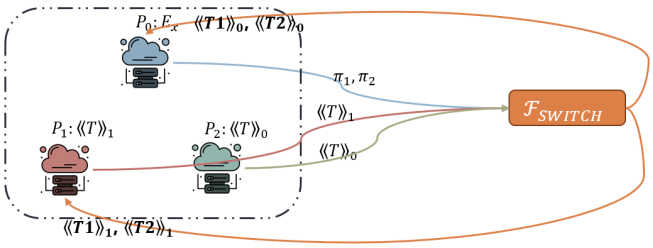


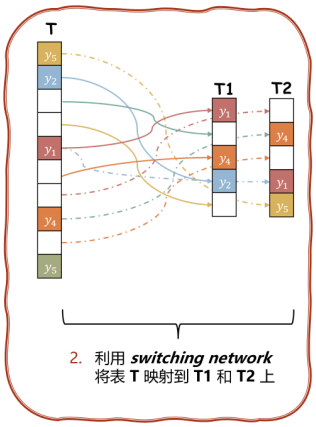
2. 如何从布谷鸟哈希表中选取值

应用$\mathcal{F}_{SWITCH}$ 协议构建 **T1** 和 **T2**，其功能如下：

> $\mathcal{F}_{SWITCH}$：
>
> ***input：*** P_0将发送 π_1, π_2 ， P_1 将发送的 $\langle\!\langle T \rangle\!\rangle_1$ ， P_2 发送的 $\langle\!\langle T \rangle\!\rangle_0$
>
> ***output：*** 返回 $\langle\!\langle T1 \rangle\!\rangle_0, \langle\!\langle T2 \rangle\!\rangle_0$ 给 P_0 ,返回 $\langle\!\langle T1 \rangle\!\rangle_1, \langle\!\langle T2 \rangle\!\rangle_1$ 给 P_1

- Selecting from the Cuckoo Table
  - P_0 构造映射函数 π_1 , π_2分别对应 E_x 的两个哈谷鸟哈希映射， 即$\pi_l(i) = j$ 满足 $h_l(E_x[i]) = j$
  - P_0 将 π_1 , π_2 和 发送给 $\mathcal{F}_{SWITCH}$, P_1 将 $\langle\!\langle T \rangle\!\rangle_1$ 发送给 $\mathcal{F}_{SWITCH}$, P_2 将 $\langle\!\langle T \rangle\!\rangle_0$ 发送给 $\mathcal{F}_{SWITCH}$
  - $\mathcal{F}_{SWITCH}$ 返回 $\langle\!\langle T1 \rangle\!\rangle_0, \langle\!\langle T2 \rangle\!\rangle_0$ 给 P_0 ,返回 $\langle\!\langle T1 \rangle\!\rangle_1, \langle\!\langle T2 \rangle\!\rangle_1$ 给 P_1





The Protocols Description

Randomized encodings $\mathcal{F}_{ENCODE}$

Oblivious Encodings

- Define cuckoo hash function as `PRF` output

  $h_1(x) = F_k(x, 1)$

  $h_2(x) = F_k(x, 2)$

- Within MPC, compute

  $[k] \leftarrow \{0,1\}^\kappa$

  for $i = 1, \ldots, n$:

  $[h_1(x_i)] = F_{[k]}([x_i], 1)$

  $[h_2(x_i)] = F_{[k]}([x_i], 2)$

  Reveal $h_1(x_i), h_2(x_i)$ to party P_1

- Similarly, reveal $h_1(y_i), h_2(y_i)$ to party P_1
- Since all $x_i (resp. y_i)$ are unique, revealing PRF outputs does not leak information.

takes as input several tuples $([\![B_i]\!], [\![X_i]\!], P_i,)$ $$

- $[\![B_i]\!]$: 其中 $B_i \in \{0,1\}^d$， 表示 d 比特的数组
- $[\![X_i]\!]$: 其中 $X_i \in (\{0,1\}^d)^\sigma$， 表示 d 个长度为 σ 的字符串
  - P_i : 表示 P_i 输出加密后的三元组

Oblivious switching network $\mathcal{F}_{ENCODE}$

- inputs:
  - P_1 inputs an arbitrary function $\pi(i) : [n] \to [m]$
  - P_1, P_2 inputs $[A] = [a_1], \ldots, [a_n]$
  - P_3 has no input
- Output:
  - P_1, P_3 output $[D] = [d_1], \ldots, [d_m]$

    where $[d_i] = [a_{\pi(i)}]$

3pc Oblivious Permutation $\mathcal{F}_{PERM}$

- Restriction, $\pi(i) : [n] \to [n]$ is a permutaion.
- P_1 samples random oermutations π_2, π_3 st.

$$\pi(i) = \pi_3(\pi_2(i))$$

Oblivious switching network $\mathcal{F}_{Switch}$

1. If a_i appears k times in output then a_i should have $k-1$ dummies after it...
2. If a_i appears k times in output then duplicate a_i into the next $k-1$ dummies after it...
3. Reorder the array as desired.

Duplication Layer $\prod_{dup}$

- P_1 has $s \in \{0,1\}$ and P_2 has $a_0, a_1 \in \{0,1\}^\sigma$
- P_2 samples $r, w_0, w_1 \leftarrow \{0,1\}^\sigma$ and $\phi \leftarrow \{0,1\}$
- P_2 sends m_0, m_1, ϕ to P_1 where
  - $m_1 = a_2 \oplus r \oplus w_\phi$
  - $m_2 = a_2 \oplus r \oplus w_{\phi \oplus 1}$
- P_2 sends w_0, w_1 to P_3
- P_1 sends $\rho = \phi \oplus s$ to P_3
- P_3 send w_ρ to P_1
- P_2 outputs $[d_1]_2 = r$ and P_1 outs $[d_1]_1 = m_s \oplus w_p$