面向隐私集合计算的数据结构综述

张响[F]

日期：2022 年 8 月 15 日

摘　要

隐私集合计算（Private Set Operation，PSO）属于安全多方计算领域的热点问题，允许两个参与方在没有额外信息泄露的情况下对各自私有集合进行安全计算。在 PSO 协议的设计中，往往运用了一些优化技巧降低协议的渐进复杂度，其中各类高级的数据结构是重要工具之一。本文梳理和总结了 PSO 协议设计中应用的三类数据结构包括哈希表、过滤器和不经意键值对存储。通过梳理和总结各类数据结构发挥的作用并为其提供性能对比分析与基准测试，可供 PSO 协议选择更高效更合适的优化技术，对隐私保护场景下的数据共享问题具有重要意义。

1　引言

大数据时代，海量数据的交叉计算可以为科研、医疗、金融等领域提供更好的支持。然而现实中数据作为机构或个人的核心资产，出于隐私保护及利益的考虑内部数据通常不对外开放，数据孤岛现象普遍存在，导致数据的价值无法体现。既要应用数据，又要保护数据安全。如何在分布式环境下保障数据和隐私安全的同时发挥数据价值，是当前亟待解决的问题。自 1982 年姚期智 [49] 借“百万富翁”问题引入安全多方计算（Secure Mutiparty Computation，MPC）概念以来，其已成为解决多个参与者在协同计算过程中隐私保护问题的关键技术之一。

隐私集合计算（Private Set Operation，PSO）属于安全多方计算领域的热点问题，允许两个参与方在没有额外信息泄露的情况下对各自私有集合进行安全计算，包括隐私集合求交（Private Set Intersection，PSI）和隐私集合求并（Private Set Union，PSU）等。以 PSI 协议为例，发送方输入集合 X，接收方输入集合 Y，两方运行 PSI 协议之后，接收方除获得 $X \cap Y$ 的信息外不能获得 $X \backslash Y$ 中的任何信息，而发送方不获得任何信息。在隐私保护的场景中，PSO 协议具有重要意义，如 PSI 的应用场景包括联系人匹配 [25]、计算广告转化率 [23]、DNA 检测与模式匹配 [46] 和接触者追踪 [15] 等；PSU 的应用场景包括利用 IP 黑名单联合查询进行网络风险评估 [22] 与隐私保护数据聚合 [7] 等。

在 PSO 协议的设计中，往往运用了一些优化技巧降低协议的渐进复杂度，其中各类高级的数据结构是重要工具之一。各类高级数据结构在 PSO 方案中广泛应用，但存在选择过多、具体作用不清晰的问题。通过梳理这些高级数据结构发挥的作用、总结它们在不同使用场景的应用，可以为设计 PSO 协议、优化 PSO 方案提供很大的帮助。现将不同数据结构在具体协议中发挥的作用总结为以下三类，如表1所示。

（1）**数据对齐**：哈希分桶技术在 PSO 协议的设计中得到了大量的应用。发送方和接收方将各自集合中的元素分别映射到两个哈希表中，每个哈希表有 B 个桶，并向每个桶中填充哑元以防止额外信息泄露。两方相同的元素会被映射到相同索引的桶中。哈希分桶相当于将双方的输入集合分别划分成了 B 个不相交的子集，出现在子集的交集中的元素一定属于原始集合的交集。对于每个桶中的元素，直接逐桶执行 PSI 或 PSU 子协议。不同的协议中选用了不同的哈希函数构造哈希表以达到最佳的效率，包括简单哈希表（Simple Hashing Table）、平衡分配哈希表（Balanced Allocation Hashing Table[3] 和布谷鸟哈希表（Cuckoo Hashing Table）[36]。

（2）**成员测试**：PSO 协议的设计中可以利用布隆过滤器（Bloom Filter）[5]，布谷鸟过滤器（Cuckoo Filter）[16] 和真空布隆过滤器（Vacuum Filter）[36] 进行成员测试。具体做法为，发送方将元素集合 X 插入过滤器并将过滤器发送给接收方，接收方通过查询操作可获知其查询元素 y 是否属于集合 X。利用各类过滤器将发送的元素集合进行压缩实现了通信开销的优化。

（3）**数据编码**：一些 PSO 协议中利用了多项式或乱码布隆过滤器（Gabled Bloom Filter）[14] 对集合中的元素进行编码。Garimella 等人将上述技术抽象成不经意键值对存储（Oblivious Key-Value Store，OKVS）[19]。通过构造效率更高的 OKVS 实例化 3H-GCT 可以直接得到更高效的 PSO 协议。接收方将其集合中的元素 y 作为键并选取随机值 r，利用键值对 $\{y, r\}$ 进行编码构造一个 OKVS，发送方同样以其集合中的元素 x 作为键进行解码得到对应值。若 $x \in Y$ 则接收方解码可以得到发送方选择的随机值。

表 1: 数据结构分类及其在隐私集合计算中发挥的作用

数据结构	类型	作用
哈希表	简单哈希表（Simple Hashing Table）	数据对齐
	平衡分配哈希表（Balanced Allocation Hashing Table）[3]	
	布谷鸟哈希表（Cuckoo Hashing Table）[36]	
过滤器	布隆过滤器（Bloom Filter）[5]	成员测试
	布谷鸟过滤器（Cuckoo Filter）[36]	
	真空过滤器（Vacuum Filter）[36]	
不经意键值对存储	乱码布隆过滤器（Gabled Bloom Filter）[14]	数据编码
	探测及异或字符串（Probe-and-XOR of Strings）[41]	
	3-哈希乱码布谷鸟哈希表（3-Hash Garbled Cuckoo Table）[19]	

综上所述，隐私集合计算是安全多方计算的一个重要研究分支，是近几年来国内外的研究热点。各类数据结构的应用使得 PSO 协议的效率得到了极大的提高：a. 利用哈希表进行数据对齐；b. 利用过滤器进行成员测试；c. 利用 OKVS 进行数据编码。通过梳理和总结各类数据结构发挥的作用并为其提供性能对比分析与基准测试，可供 PSO 协议选择更高效更合适的优化技术，对隐私保护场景下的数据共享问题具有重要意义。

2 预备知识

2.1 符号说明

本文中出现的符号和其描述说明如表2所示。

表 2: 符号说明

符号	描述
S，R	分别表示发送方和接收方
X，Y	分别表示发送方和接收方的集合
x_i，y_j	分别表示发送方和接收方集合中的第 i 和第 j 个元素
n_x，n_y	分别表示发送方和接收方集合的大小，大多数情况 $n_x = n_y = n$
m	数据结构中桶的个数
$[m]$	表示集合 $\{1, 2, \cdots, m\}$
b	每个桶中槽的个数
λ	统计安全参数
γ	负载因子（$\gamma = n/m$ 且 $0 \leq \gamma \leq 1$）

2.2 PSO 协议

本文中涉及的 PSO 协议包括隐私集合求交（Private Set Intersection，PSI），隐私集合求并（Private Set Union，PSU），隐私集合交集权值求和（Private Set Intersection-Sum，PSI-Sum）和隐私集合交集/并集求势（Private Set Intersection/Union Cardinality，PSI/PSU-CA）协议。

定义 2.1. PSO 协议: 两个参与方分别是发送方 S 和接收方 R, 其中 S 输入集合 $X = \{x_1, \cdots, x_n\}$, R 输入集合 $Y = \{y_1, \cdots, y_n\}$（在 PSI-Sum 协议中, R 的输入还包括权值的集合 $W = \{w_1, \cdots, w_n\}$）。大多数情况下两方集合大小是公开的。

- PSI：S 不获得任何信息，R 获得 $X \cap Y$；
- PSU：S 不获得任何信息，R 获得 $X \cup Y$；
- PSI-Sum：S 获得 $|X \cap Y|$，R 获得 $\sum_{i:x_i \in Y} w_i$；
- PSI/PSU-CA：S 不获得任何信息，R 获得 $|X \cap Y|$ 或 $|X \cup Y|$。

2.3 不经意伪随机函数

不经意伪随机函数（Oblivious Pseudorandom Function，OPRF）是一个两方协议，其中发送方拥有伪随机函数 F 的密钥 k，接收方拥有输入 x；执行协议之后接收方获得 $F_k(x)$ 而发送方不获得任何信息。OPRF 协议的功能函数可以描述为 $\mathcal{F}_{OPRF} : (k, x) \rightarrow (\perp, f_k(x))$。Freedman 等人 [18] 基于 Naor-Reingold 伪随机函数构造了 OPRF，该构造需要幂指操作且需要的 OT 实例个数与 PRF 输入元素的长度线性相关。Camenisch 等人 [8] 基于盲签名构造了 OPRF 协议。Kolesnikov 等人 [27] 仅基于 OT 扩展构造了批处理 OPRF 协议使得计算效率得到了极大的优化。

2.4 不经意传输

不经意传输协议（Oblivious Transfer，OT）是安全多方计算中各种安全计算协议的密码原语，包括姚氏乱码电路和 GMW 协议。在标准 1-out-of-2 OT 协议中，发送方输入两个字符串 (m_0, m_1)，接收方输入选择比特 $r \in \{0,1\}$；OT 协议允许接收方获得选择比特对应的字符串 m_r，不允许其获得关于另一个字符串 m_{1-r} 的任何信息，而发送方不允许获得任何信息。OT 协议的构造需要基于公钥密码操作，无法满足 MPC 中大量 OT 协议需求的问题。得益于高效的 OT 扩展协议 [2, 24, 26]，仅需要 $O(\kappa)$ 次公钥加密操作和 $O(n)$ 次快速的对称加密操作即可获得 $n \gg \kappa$ 个 OT 实例。

2.5 不经意多项式求值

不经意多项式求值（Oblivious Polynomial Evaluation，OPE）[33] 是 MPC 中一个重要的基础协议。协议中有两个参与方，发送方持有一个定义在域 $\mathbb{F}$ 上且度为 d 的多项式 $P(\cdot)$，接收方输入 $x \in \mathbb{F}$；协议的目的是接收方针对其输入仅能获得 $P(x)$ 无法得知任何关于多项式 P 的信息，发送方无法得知关于 x 的信息。OPE 协议是一个重要的基础组件并能用于解决大量的密码学问题，包括 RSA 密钥生成 [20]，不经意关键词搜索 [18]，隐私集合求交 [10, 11, 17, 21] 等。

2.6 安全模型

- 半诚实模型（(Semi-honest Model)：半诚实模型也称为被动安全模型。在半诚实模型中，各参与方会遵循协议规则诚实地执行协议，但是会尝试从其他参与方的输入或协议的中间计算结果中获得额外信息。
- 恶意模型（Malicious Model)：恶意模型也称为主动安全模型。在恶意模型中，参与方会任意地偏离协议规则以破坏协议的安全性，包括恶意篡改输入信息、拒绝参与协议、提前终止协议等。

3 隐私集合计算中的数据结构

3.1 哈希表

简单哈希表（Simple Hashing Table）在简单哈希构建哈希表的方案中，哈希表由 m 个桶 $B_1, B_2, \cdots, B_m$ 构成，利用一个随机的哈希函数 $h : \{0,1\}^* \mapsto [m]$ 将元素映射到表中。元素 x 始终会被存放到桶 $B_{h(x)}$ 中，不管该桶中是否已经存放了其他元素。文献 [37] 指出，若将 m 个元素映射到 m 个桶的哈希表中，其中拥有最多元素的桶中元素个数 $max_b = O(\frac{\ln m}{\ln \ln m})$。

平衡分配哈希表（Balanced Allocation Hashing Table）利用“两个选择的力量”[31]，Azar 等人在 [3] 中提出了用两个哈希函数 $h_1, h_2 : \{0,1\}^* \mapsto [m]$ 构造哈希表的平衡分配哈希方案。插入元素 x 时，始终将 x 放置在 $B_{h_1(x)}$ 和 $B_{h_2(x)}$ 其中存储元素更少的一个桶中。若将 m 个元素映射到 m 个桶的哈希表中，其中拥有最多元素的桶中元素个数 $max_b = O(\frac{\ln \ln m}{\ln 2})$[37]。相比于简单哈希，平衡分配哈希的优势在于元素更加均匀地映射到了哈希表中。

布谷鸟哈希表（Cuckoo Hashing Table） Pagh 和 Rodler 在 [36] 中提出了布谷鸟哈希，使用 k 个哈希函数 $h_k : \{0,1\}^* \mapsto [m]$ 将元素 x 映射到表中，且每个桶中最多存放一个元素。计算 $h_1(x), h_2(x), \cdots, h_k(x)$，若存在空桶则元素 x 随机放入空桶中；否则随机选取桶 $B_{h_i(x)}$ 逐出其中的元素，对逐出的元素 x' 同样执行上述操作。若逐出操作达到了阈值，将无法插入的元素放入额外空间 stash 中。通过调整哈希函数个数和桶个数参数，可以实现无 stash 的布谷鸟哈希表。具体参数分析见文献 [38] 中的 3.2 节。

3.2 过滤器

过滤器是一种空间效率高的近似成员测试（Approximate Membership Querie，AMQ）数据结构，用于检查元素是否属于一个集合。过滤器利用微小的查询误判率换取空间效率。查询误判率即过滤器对于集合元素进行查询操作时做出错误判断的数量占总判断数量的比率。以下介绍了三种典型的过滤器，分别是布隆过滤器、布谷鸟过滤器和真空过滤器。表3描述了查询误判率为 $2^{-\lambda}$ 时三种过滤器的性能。

表 3: 查询误判率为 $2^{-\lambda}$ 时，不同过滤器的性能

过滤器	平均空间开销	哈希函数个数	负载因子	是否支持删除	是否支持并行
布隆过滤器	1.44λ	λ	$1/(1.44\lambda)$	否	是
布谷鸟过滤器	$(\lambda+3)/\gamma$	2	γ	是	否
真空过滤器	$(\lambda+3+\log_2(\gamma))/\gamma$	2	γ	是	否

[1] 布谷鸟过滤器和真空过滤器每个桶中槽的数量 $b = 4$。

布隆过滤器（Bloom Filter） Bloom 在 1970 年提出了布隆过滤器 [5]。其主要构造为用 k 个不同的哈希函数 $h_1, h_2, \cdots, h_k$ 将 n 个元素 $x_1, x_2, \cdots, x_n$ 映射到 m 个桶 $B_1, B_2, \cdots, B_m$ 中，每个桶的初始值置 0。插入元素 x 时其映射的 k 个位置对应的桶 $B_{h_i(x)}$ 置 1。查询元素 x 时，算法通过检查 k 个桶 $B_{h_i(x)}$ 中的值是否全 1 输出 “True” 或 “False”。输出 “True” 表示查询的元素以 $1-2^{-\lambda}$ 的概率在集合中；输出 “False” 表示查询的元素以绝对的概率不在集合中。由于存在哈希碰撞使得布隆过滤器中多个元素用同一比特信息表示的情况，标准布隆过滤器无法删除元素，若需要删除某一个元素只能重新构建整个过滤器。

布谷鸟过滤器（Cuckoo Filter） Fan 等人在 [16] 中提出了支持删除操作的布谷鸟过滤器。与布隆过滤器不同，布谷鸟过滤器利用两个不同的哈希函数 h_1 和 h_2 将 n 个元素映射到 m 个桶中，且每个桶中有多个大小为 ℓ 比特的槽。布谷鸟过滤器将关于元素 x 的指纹 f_x 存放在 B_{i_1} 或 B_{i_2} 中，其中：

$$\begin{aligned} i_1 &= h_1(x) \\ i_2 &= i_1 \oplus h_2(f_x) \end{aligned} \tag{1}$$

容易证明在只知道元素的指纹和其中一个位置索引的情况下，可以通过异或操作得到另一个位置的索引。布谷鸟过滤器利用 “逐出” 操作并设置逐出次数上限处理碰撞。若元素映射到的两个桶都不存在空槽，则随机的选择其中一个桶如 B_{i_1} 的非空槽，将该位置存放的指纹 f' 替换为

f_x 并将 f' 逐出到备用位置 $B_{i_1 \oplus h_2(f')}$，若备用位置非空则将备用位置存放的指纹逐出，在达到逐出次数上限之前递归地执行此操作，直到没有元素被逐出。布谷鸟过滤器通过检查元素 x 映射的两个可选桶 B_{i_1} 和 B_{i_2} 中是否存在指纹 f_x 判断其是否在集合中。删除元素 x 通过删除布谷鸟过滤器中对应的指纹 f_x 实现。

真空过滤器（Vacuum Filter）在 [48] 中，为了进一步提高空间效率，Wang 等人提出了真空过滤器。由于布谷鸟过滤器的构造基于桶个数 m 必须为 2 的幂次方这一假设，在一定程度上会造成空间浪费。例如，构造布谷鸟过滤器时若实际上需要的桶的个数为 1025，m 需要设置为 2048，造成将近 50% 的空间浪费。为了解决这一问题，真空过滤器将整个表划分成大小相同的块，每个块中桶的个数 L 为 2 的幂次方，并保证插入元素的两个可选桶 B_{i_1} 和 B_{i_2} 出现在同一个块中。为了平衡负载因子（load factor）和空间访问效率（locality），Wang 等人另外提出了将表划分成大小不同块的算法。真空过滤器的插入策略和查询策略与布谷鸟过滤器相同，且同样支持删除操作。

3.3 不经意键值对存储

3.3.1 定义

不经意键值对存储（Oblivious Key-Value Store，OKVS）是一种在一系列键和值之间建立映射关系的数据结构。它由编码算法和解码算法构成，且满足正确性和不经意性两个性质。

定义 3.1. 不经意键值对存储 [19]：令 $\mathcal{K}$ 表示键（key）的集合，$\mathcal{V}$ 表示值（value）的集合。具体的编码算法和解码算法如下所示。

- $Encode(\{(x_1, y_1), \ldots, (x_n, y_n)\})$：通过键值对构造不经意键值对存储数据结构。算法输入一组键值对 $\{(x_i, y_i)\}_{i \in [n]} \subseteq \mathcal{K} \times \mathcal{V}$，输出一个对象 D（或者以可忽略的概率输出错误指示符 $\perp$）。
- $Decode(D, x)$：通过不经意键值对存储数据结构和 key 查询 value。算法输入对象 D 和需查询的键 x，输出 x 的映射值 $y \in \mathcal{V}$。

正确性（Correctness）：对任意 $A \subseteq \mathcal{K} \times \mathcal{V}$ 都有：a. 编码算法输出 $\perp$ 的概率是可忽略的；b. 若 $Encode(A) = D$ 且 $D \neq \perp$，则对于任意 $(x, y) \in A$，$Decode(D, x) = y$。

不经意性（Obliviousness）：以 $\mathcal{K}_1 = \{x_1^0, \ldots, x_n^0\}$ 和 $\mathcal{K}_2 = \{x_1^1, \ldots, x_n^1\}$ 为编码算法输入，并均匀随机地选择 $y_i \leftarrow \mathcal{V}$，若 $D \neq \perp$，则分布 $\{D \mid y_i \leftarrow \mathcal{V}, i \in [n], Encode((x_1^0, y_1), \ldots, (x_n^0, y_n))\}$ 与分布 $\{D \mid y_i \leftarrow \mathcal{V}, i \in [n], Encode((x_1^1, y_1), \ldots, (x_n^1, y_n))\}$ 计算不可区分。

3.3.2 OKVS 实例化

下面描述了几种 OKVS 实例化，并在表4中给出了性能总结。

多项式（Polynomials）多项式可看作一个最简单的 OKVS 实例化，其中编码算法为利用 FFT 插值算法将键值对 $\{(x_i, y_i)\}_{i \in [n]}$ 构造为多项式 P，其中 $P(x_i) = y_i$；对应的解码算法为 $y = P(x)$。利用多项式实例化 OKVS 数据结构的优势在于其负载因子达到了最优为 1，即空间开销最小。然而，FFT 插值算法带来了较大的计算开销，导致其编码开销和解码开销较大，分别为 $O(n \log^2 n)$

表 4: 键值对个数为 n，错误率为 $2^{-\lambda}$ 时，不同 OKVS 实例化的性能

OKVS	哈希函数的个数	负载因子	编码开销	解码开销	是否处理环
多项式	-	1	$O(n\log^2 n)$	$O(\log n)$	否
GBF	λ	$O(1/\lambda)$	$O(\lambda n)$	$O(\lambda)$	否
PaXoS	2	$0.4-o(1)$	$O(\lambda n)$	$O(\lambda)$	是
3H-GCT	3	$0.81-o(1)$	$O(\lambda n)$	$O(\lambda)$	是

和 $O(\log n)$。构造一个高效的 OKVS 实例化关键在于提升编码与解码效率的同时，仅仅牺牲小部分的空间效率。

乱码布隆过滤器（Gabled Bloom Filter，GBF）乱码布隆过滤器本质上与布隆过滤器相同，最早由 Dong 等人 [14] 在 2013 年提出。乱码布隆过滤器由大小为 m 的数组构成，利用 k 个不同的哈希函数 $h_1, h_2, \cdots, h_k : \{0,1\}^* \mapsto [m]$ 计算元素的索引值。用 $D[j]$ 表示数组 D 中的第 j 个位置的值。与布隆过滤器不同，GBF 中每个位置存放 λ 比特的随机值而非 1 比特。构造 GBF 时哈希函数个数 $k=\lambda$ 保证了构造算法失败即遇到环的概率为 $2^{-\lambda}$，因此不需要额外处理环的操作。

$GBF.Encode(\{(x_1,y_1),\ldots,(x_n,y_n)\})$ 算法描述：

1. 初始化：将数组 D 中每个位置初始值设置为 $\perp$。
2. 赋值：对于输入的键值对 (x_i,y_i)，令 $J=\{h_j(x_i)|D[h_j(x_i)]=\perp\}$ 为数组中未赋值位置的索引集合。若 $J=\emptyset$ 则 GBF 构造失败算法终止，否则选取使得 $y_i=\bigoplus_{j=1}^{k} D\left[h_j(x_i)\right]$ 成立的随机值并将其赋值给 $\{D[j], j\in J\}$。
3. 随机值填充：所有键值对插入完成后，对于数组中满足 $D[j]=\perp$ 的位置 j，选取随机值进行填充。

$GBF.Decode(D,x)$ 算法描述：

1. 对于任意输入键 x，计算 $y=\bigoplus_{j=1}^{k} D\left[h_j(x)\right]$ 得到映射值。

3-哈希乱码布谷鸟哈希表（3-Hash Gabled Cuckoo Table，3H-GCT）为了得到一个更加高效的 OKVS，Pinkas 等人 [41] 在 2020 年提出了 PaXoS 数据结构。它的构造结合了布谷鸟哈希和乱码布隆过滤器的思想，即利用 2 个哈希函数而非 λ 个哈希函数构造乱码布隆过滤器。基于 3 个哈希函数构造的布谷鸟哈希表效率高于 2 个哈希函数构造的布谷鸟哈希表的思想，Garimella 等人 [19] 提出了 3H-GCT，为目前最高效的 OKVS 实例化。与 GBF 能以任意顺序插入元素不同，PaXoS 和 3H-GCT 的构造中元素均只能以特定的顺序插入。PaXoS 和 3H-GCT 的构造方法类似，以 3H-GCT 为例，其利用 3 个哈希函数 $h_1, h_2, h_3 : \{0,1\}^* \mapsto [m]$ 计算元素的索引值。

$3HGCT.Encode(\{(x_1,y_1),\ldots,(x_n,y_n)\})$ 算法描述：

1. 初始化：令数组 $D=L||R$，其中 $|L|=m$ 且 $|R|=O(\log n)+\lambda$。将数组中每个位置初始值设置为 $\perp$。
2. 形成无向超图 $\mathcal{G}_{3,m,n}$：利用哈希函数 h_1, h_2, h_3 将键 x_i 映射到边 $(h_1(x_i), h_2(x_i), h_3(x_i))$。由 n 个键可构成一个节点数为 m 边数为 n 的无向超图 $\mathcal{G}_{3,m,n}$，图中每条边连接 3 个节点。
3. 赋值：为数组 D 赋值，使满足对于任意 (x_i,y_i)，

$$y_i=\langle l(x_i)||r(x_i), L||R\rangle \tag{2}$$

其中 $l(\cdot):\{0,1\}^* \mapsto \{0,1\}^m$ 输出除 $h_1(\cdot)$，$h_2(\cdot)$ 和 $h_3(\cdot)$ 三个位置为 1 其余位置为 0 的向量；$r(\cdot):\{0,1\}^* \mapsto \{0,1\}^{O(\log n)+\lambda}$ 输出随机向量。

- 依次选择超图 $\mathcal{G}_{3,m,n}$ 中度为 1 的节点并将其对应的键压入栈 S 中；即若节点 $j \in [m]$ 满足 $\{x_i \notin S | j \in \{h_1(x_i), h_2(x_i), h_3(x_i)\}\}$ 其中边 $(h_1(x_i), h_2(x_i), h_3(x_i))$ 的度为 1，则将 x_i 压入栈 S 中并将该边从 $\mathcal{G}_{3,m,n}$ 中移除。
- 当超图中不存在度为 1 的边后，对于 $x_i \in S$，$h_1(x_i)$，$h_2(x_i)$ 和 $h_3(x_i)$ 三个位置中至少有一个未赋值，选取使得公式2成立的随机值并将其赋值给 D。
- 超图中剩余的节点形成了环且环中节点数为 $O(\log n)$。对于所有 $x_i \notin S$，利用高斯消元法求解公式2构成的方程组。初始化时将 GCT 数组扩展了 $O(\log n)+\lambda$ 比特保证了方程组有解，将求得的解赋值给 D。

4. 随机值填充：所有键值对插入完成后，对于数组中满足 $D[j] = \perp$ 的位置 j，选取随机值进行填充。

$3HGCT.Decode(D, x)$ 算法描述：

1. 将数组 D 拆分为 $D = L||R$，其中 $|L| = m$ 且 $|R| = O(\log n)+\lambda$。
2. 对于任意输入键 x，计算公式2得到映射值。

4 数据结构在隐私集合计算中的应用

本节描述了不同数据结构在各类 PSO 协议中的应用，具体内容见表5。

表 5: 不同数据结构在 PSO 协议中的应用

协议	数据结构	类型	协议类型	组件	安全模型
[17]	哈希表	平衡分配哈希表	PSI	OPE	半诚实模型/恶意模型
[27, 37]		简单哈希表 + 布谷鸟哈希表	PSI	OPRF	半诚实模型
[40]*		简单哈希表 + 布谷鸟哈希表	PSI	OPRF	半诚实模型
[10, 11]*		简单哈希表 + 布谷鸟哈希表	非平衡 PSI	OPE	半诚实模型
[28]		简单哈希表	PSU	RPMT+OT	半诚实模型
[14]	过滤器	布隆过滤器 + 乱码布隆过滤器	PSI	OT	半诚实模型
[44]		布隆过滤器 + 乱码布隆过滤器	PSI	OT	恶意模型
[25]		布隆过滤器	非平衡 PSI	OPRF	半诚实模型
[43]		布谷鸟过滤器	非平衡 PSI	OPRF	半诚实模型
[13]		布隆过滤器	PSI/PSU/PSI-CA/PSU-CA	AHE	半诚实模型
[23]		布隆过滤器	PSI-Sum	AHE	半诚实模型
[41]	OKVS	探测及异或字符串	PSI	OT	恶意模型
[45]		探测及异或字符串	PSI	OPRF	半诚实模型/恶意模型
[50]		3-哈希乱码布谷鸟哈希表	PSU	mqRPMT+OT	半诚实模型

[1] 表中“*”表示使用了置换哈希优化哈希表的构造。

4.1 哈希表在 PSO 协议中的应用

2004 年，Freedman[17] 首先给出了基于不经意多项式求值（Oblivious Polynomial Evaluation，OPE）的 PSI 协议。基于 OPE 的 PSI 协议的设计思想是：发送方或接收方将其集合元素表示成多项式的根，利用多项式的性质并结合密码学工具进行交集求解。在该协议中，接收方以其集合中的元素作为根生成多项式 $P(z)$。接收方用 Paillier 或 ElGamal 半同态加密算法将多项式的系数加密发送给发送方。根据同态加密的性质，发送方对其输入集合中的所有元素进行密态求值并利用随机数 r 盲化后将结果 $Enc(r \cdot P(x_i) + x_i)$ 返回给接收方比对。接收方解密，若元素 y_i 在交集中，则对任意 r 都有 $r \cdot P(y_i) + y_i = y_i$；否则 $r \cdot P(y_i) + y_i$ 是一个随机值。该方案的通信开销为线性的复杂度但其计算开销非常高，主要的计算开销在于产生一个次数为 $|Y|-1$ 的高次多项式以及对高次多项式进行 $|X|$ 次密态计算。因此论文指出利用平衡分配哈希 [3]，发送方和接收方将各自集合中的元素均匀地分配到 B 个桶中，每个桶中元素个数最多为 M，从而达到降低多项式次数的目的。

Pinkas 等人 [37] 在 2014 年提出了基于 OT/OPRF 的 PSI 协议。Freedman 等人在 [18] 中指出了 OPRF 与 PSI 协议之间的联系。关键思想在于将发送方和接收方输入集合中的元素替换为伪随机函数值即随机盲化，其中发送方拥有密钥 k 可任意地计算其输入元素的 PRF 值 $X' = \{F_k(x) : x \in X\}$，接收方无法自行计算只能得到 OPRF 协议中输出的 PRF 值 $Y' = \{F_k(y) : y \in Y\}$。接收方通过对比 X' 和 Y' 可以得到交集。方案 [37] 的主要思想为将 1-out-of-2 OT 看作接收方输入元素定义域为 $r \in \{0,1\}$ 的单点 OPRF，用函数表示为 $F((m_0, m_1), r) = m_r$，其中 (m_0, m_1) 作为 OPRF 的密钥。若接收方需要计算 n 个点的 OPRF 值则需要调用 n 次单点 OPRF 协议，发送方则需生成 n 个不同的密钥。得益于高效的 OT 扩展协议 [2, 24, 26]，仅需要少量的公钥加密操作和快速的对称加密操作即可获得大量 OT 实例，该方案有着显著的计算开销优势。然而由于该方案中构造的是单点 OPRF，其通信开销为 $O(n^2)$。为了提高通信效率，论文指出可以借鉴哈希分桶的思想，将元素映射到哈希表中并逐桶执行 PSI 子协议得到交集。其中一方构造简单哈希表，另一方构造布谷鸟哈希表保证每个桶中至多只有一个元素。通过合理选取参数如哈希函数的个数和哈希表中桶的个数，该方案的通信开销可以减小到 $O(n \log n)$ 的复杂度。观察到上述方案需要的 OT 实例个数不仅与集合中元素的个数相关还与元素的长度相关，2015 年 Pinkas 等人 [40] 提出使用置换哈希构造哈希表减小每个桶中放入元素的长度实现了优化。2016 年，Kolesnikov 等人 [27] 对该方案进一步优化使得 OT 实例的个数只与集合中元素的个数相关。他们提出用 1-out-of-N OT 替换 1-out-of-2 OT 对每个元素进行逐 N 比特比较。该方案为目前计算开销最小的 PSI 方案。

Chen 等人在 2017 年 [10] 和 2018 年 [11] 基于不经意多项式求值提出了适用于在两方集合大小相差较大场景下的非平衡 PSI 方案。假设接收方为拥有小集合的一方。主要构造为，发送方而非接收方以其集合中的元素作为根产生度为 d 多项式 $Q(z) = \sum_{i=0}^{d} a_i z^i$；对于 $y_j \in Y$ 接收方利用 FHE 算法将其加密得到 $\{Enc(y_j)\}_{j \in [n_y]}$ 并发送给发送方进行密态求值。对于每个 $Enc(y_j)$ 发送方选择随机数 r_j 并计算 $d_j = r_j \prod_{i=0}^{d} Enc(a_i (y_j)^i)$，将 $\{d_j\}_{i \in [n_y]}$ 返回给接收方；接收方解密后根据 d_j 是否为 0 判断交集元素。为了减小计算多项式带来的开销，方案中同样使用了简单哈希表和布谷鸟哈希表并利用置换哈希优化技巧减小哈希表中存放元素的长度。此外，结合分窗、划分和模转换技术，该方案的通信开销只与小集合大小线性相关与大集合大小亚线性相关。

2019 年 Kolesnikov 等人 [28] 首次提出了仅利用对称加密技术实现的 PSU 协议。该协议的核心为利用 OPRF 和多项式构造反向私有成员测试 (Reverse Private Membership Test，RPMT) 子协议，其可以测试发送方的元素是否属于接收方的集合，并让接收方获得结果。两方执行 n 次 RPMT 子协议之后，执行 n 次 OT 协议获得并集。其中发送方将 $\{x, \perp\}$ 作为 OT 协议的输入，接收方将 RPMT 子协议的输出结果 $r \in \{0,1\}$ 作为 OT 协议的输入并获得 $\{x\} \cup Y$。为了降低多项式构造 RPMR 造成的计算和通信开销，该协议利用两方构造简单哈希表使计算开销达到了 $O(n \log n \log\log n)$，通信开销达到了 $O(n \log n)$。

4.2 过滤器在 PSO 协议中的应用

Dong 等人 [14] 在 2013 年提出了利用布隆过滤器和乱码布隆过滤器构造的半诚实模型下的 PSI 协议。与布隆过滤器不同，乱码布隆过滤器中每个桶存放的值不再是 1 比特信息而是 λ 比特关于插入元素的秘密分享值。发送方和接收方根据其各自集合中的元素分别生成一个乱码布隆过滤器和布隆过滤器，并逐桶执行消息长度为 λ 的 OT 协议。该 PSI 协议能处理的集合数量首次突破了亿级别。2017 年 Rindal 和 Rosulek[44] 通过生成比所需的布隆过滤器比特数略多的 1-out-of-2 OT 构建 Cut-and-Choose 技术，以此实现恶意模型下的 PSI 协议。

在 Kiss 等人 [25] 提出利用布隆过滤器编码发送方数据集减小通信开销的基础上，2018 年 Resende 和 Aranha[43] 首次将布谷鸟过滤器应用到了 PSI 方案中。他们指出将布谷鸟过滤器与 Baldi 等人 [4] 在 2011 年提出的基于 OPRF 的 PSI 方案结合，可以得到一个通信开销更小的优化方案。文献 [4] 中的 OPRF 协议用函数可表示为 $F_\alpha(x) = H'(H(x)^\alpha)$，其中 H 表示随机喻言机（Random Oracle），H' 表示将群元素映射到字符串的哈希函数。具体来说，（1）发送方输入集合为 X 并选择密钥 $\alpha \in Z_q^*$，接收方输入集合为 Y；（2）对于 $y_j \in Y$ 接收方随机选择 $\beta_j \in Z_q^*$ 并将 $H(y_j)^{\beta_j}$ 发送给发送方，发送方返回 $(H(y_j)^{\beta_j})^\alpha$，接收方将指数 β_j 移除并输出 $H'(H(y_j)^\alpha) = H'((H(y_j)^{\beta_j\alpha})^{1/\beta_j})$；（3）对于 $x_i \in X$ 发送方计算 $H'(H(x_i)^\alpha)$ 并发送给接收方计算交集。在优化的协议中，发送方不直接发送盲化后的元素而是向接收方发送一个布谷鸟过滤器，其中插入盲化后的元素 $H'(H(x_i)^\alpha)$；接收方依次查询元素 $H'(H(y_j)^\alpha)$ 是否在过滤器中。利用布谷鸟过滤器将发送的元素集合进行压缩实现了通信开销的优化，相比于 Baldi 等人 [4] 的方案，该优化方案传输的数据量减少到原来的十分之三，相应地运行速度快了 3.3 倍。

2020 年，Ion 等人 [23] 提出了一个基于布隆过滤器和同态加密的 PSI-Sum 方案。该协议中，发送方和接收方分别生成加法同态加密方案的密钥对 (pk_1, sk_1) 和 (pk_2, sk_2) 并交换公钥 pk_1 和 pk_2。发送方利用 k 个哈希函数将其集合 X 中的元素插入布隆过滤器 BF 并利用 pk_1 对布隆过滤器中的每一个比特 $BF[i]$ 进行加密。发送方将加密后的布隆过滤器 $Enc(pk_1, BF)$ 发送给接收方。接收方对其集合 Y 中的元素进行成员测试且得到的结果为密文形式。具体做法为，对于 $y_i \in Y$ 接收方选择随机数 r_i 并计算 $ct_i = r_i \cdot (\sum_{j=0}^{k} Enc(pk_1, BF(h_j(y_i))) - Enc(pk_1, k))$，其中 $\{h_j(y_i)\}_{j\in[k]}$ 表示元素 y_i 映射在布隆过滤器中索引值集合；若 $y_i \in X$ 则 ct_i 为对明文 0 的加密，否则为随机值。同时接收方利用 pk_2 将集合元素 y_i 对应的权值 w_i 进行加密，并将 $\{ct_i, Enc(pk_2, w_i)\}$ 随机置换后发送给发送方，发送方只能解密 ct_i 获得交集大小。发送方将 $\sum_{i:Dec(sk_1,ct_i)=0} Enc(pk_2, w_i)$ 发送给接收方，接收方解密后可获得交集元素权值的和。与上述方案类似，Davidson 和 Cid[13] 利用布隆过滤器与 AHE 方案展示了一个 PSO 协议的设计框架，

包括 PSI 协议，PSU 协议和 PSI/PSU-CA 协议。

4.3 OKVS 在 PSO 协议中的应用

Pinkas 等人 [41] 在 2020 年首次将布谷鸟哈希表应用到了恶意模型下的 PSI 协议。通过改进布谷鸟哈希算法构造了一个新的数据结构即探测及异或字符串（Probe-and-XOR of Strings，PaXoS），并结合具有同态性质的 OT 扩展协议 [35] 得到了一个高效的恶意模型下的 PSI 协议。该协议的计算效率几乎与已知最快的半诚实模型下的 PSI 协议 [27] 一样高。PaXoS 利用 2 个哈希函数将集合元素映射到对应桶并放入该元素的秘密分享值，以消除布谷鸟哈希构建恶意 PSI 时泄露发送方未在集合交集中的集合信息问题 [53]。

2021 年，Rindal 和 Schoppmann[45] 基于不经意向量线性求值（Vector Oblivious Linear Evaluation，Vector-OLE）构造了一个新的 OPRF，并且他们观察到用更高效的 OKVS 实例化 PaXoS 替换多项式与 OPRF 相结合构造可编程的 OPRF，可以得到比 [39] 中通信开销更小的 PSI 方案。文献 [45] 中的方案通信开销和计算开销均为 $O(n)$ 并且该方案为目前通信开销最小的 PSI 方案，在集合大小为 2^{20} 时，该方案的通信开销为 53MB 比基于 Diffie-Hellman 的 PSI 协议 [30] 的通信开销更低。

在方案 [28] 的基础上，2022 年 Zhang 等人 [50] 提出了多点 RPMT（Multi-Query Reverse Private Membership Test，mqRPMT）子协议的构造，将 OKVS 应用到了 PSU 协议使得 [28] 中的方案不再依赖于多项式编码和哈希分桶技术并得到了一个通信开销为 $O(n)$ 的 PSU 协议。其中多点 RPMT 子协议的构造为，（1）接收方选择一个随机值 r 并加密 n 次得到密文 $\{r_1, r_2, \cdots, r_n\}$；（2）对于 $y_i \in Y$，接收方利用键值对 $\{y_i, r_i\}$ 构造 OKVS 并将该 OKVS 发送给发送方；（3）发送方以 $x_i \in X$ 作为键解码，将得到的密文 $\{r_1^*, r_2^*, \cdots, r_n^*\}$ 重随机化后发送给接收方；（4）接收方解密 $\{r_i^*\}_{i\in[n]}$ 根据结果是否为 r 判断发送方的元素 x_i 是否属于 Y。

5 性能分析与比较

本节对过滤器和不经意键值对存储数据结构的功能性进行了测试。基于已有的开源代码在表6中展示了不同数据结构实现细节与接口支持情况。

表 6: 不同数据结构实现细节与接口支持（●表示支持，○表示不支持）

数据结构/开源代码	哈希函数类型	数据类型			操作			
		整型	字符串	数组	插入	查询	删除	序列化
BloomFilter[6]	MurmurHash	●	●	●	●	●	○	○
CuckooFilter[12]	MurmurHash/TwoIndependentMultiplyShift	●	●	○	●	●	●	○
VacuumFilter[47]	MurmurHash/TwoIndependentMultiplyShift	●	●	●	●	●	●	○
GBF[29, 32]	AES-Hash	●	○	○	●	●	○	○
PaXos[34]	xxHash	●	○	○	●	●	○	●
3H-GCT[34]	xxHash	●	○	○	●	●	○	●

哈希函数类型： 6 种类型的数据结构在构造时均使用了不同数量的哈希函数。因此哈希函数

的性能包括散列是否均匀、计算哈希值的效率等与数据结构的插入、查询和删除操作的性能息息相关。

数据类型：丰富而全面的数据类型支持可极大地提升代码的可用性。布隆过滤器、布谷鸟过滤器和真空过滤器的实现均采用范型编程，支持对 C++ 中基本数据类型如不同字节长度的整型数据类型和字符串数据类型进行操作。其中除布谷鸟过滤器外，其他两种类型的过滤器还实现了对数组类型的数据进行操作。而 PaXos 和 3H-GCT 均只支持对 8 字节整型数据的操作，GBF 只支持对 16 字节整型数据的操作。

操作：6 种数据结构的实现均支持插入和查询两个基本操作。在隐私集合计算的应用场景中存在集合元素需要进行周期性更新的情况。若数据结构的实现支持删除操作，则避免了集合元素更新时需要重新构建一个新的数据结构造成的计算开销问题。布谷鸟过滤器和真空过滤器的删除操作实现比较直接，直接删除元素对应指纹即可；而其他类型的数据结构实现删除操作需要在构造方法上进行优化。在 PSO 协议的设计中，通常一个数据结构需要从一方发送给另外一方。序列化接口即将数据结构对象转换成字符串便于网络传输的实现。只有 PaXos 和 3H-GCT 实现了序列化接口。

5.1 试验环境

本文的实验在 DELL OptiPlex 3060 上运行，使用的是 Ubuntu 20.04.3 LTS 64 位操作系统，CPU 型号为 Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz，内存大小为 16GB。本文的实验采用 C++ 编程语言实现，系统的 GCC 版本为 9.4.0（Ubuntu 9.4.0-5ubuntu1）。

5.2 测试结果

测试时除乱码布隆过滤器的构造使用 16 字节的整形数据外其余数据结构的构造均使用 8 字节的整型数据。布谷鸟过滤器和真空过滤器中使用指纹长度为 16 比特。三种过滤器的查询误判率均保证小于 0.01%。下面两个图中横坐标均表示元素个数的对数。

图1展示了元素个数与平均空间开销的对比结果。三种过滤器均有较好的压缩性能。其中布谷鸟过滤器由于需要满足桶个数 m 为 2 的幂次方，在元素个数取某些值时其负载因子只能达到 0.5 左右而其他大多数情况下负载因子可以到达 0.9 以上，因此其平均空间开销随元素个数变化的曲线存在较多波折。当元素个数大于 2^{16} 时真空过滤器的平均空间开销最小。图2展示了元素个数与插入元素所用时间的对比结果。其中开源库 [34] 中关于 PaXos 部分的实现存在问题，因此暂未给出 PaXos 的插入元素时间性能结果。

6 总结

随着安全多方技术研究的逐步深入，隐私集合计算作为安全多方计算的一种重要应用近些年来效率得到了极大的提升。其中各类数据结构的正确应用和高效实现发挥了重要作用。本文总结了三类数据结构在隐私集合计算中的应用，其中哈希表用于数据对齐，过滤器用于成员测试，不经意键值对存储用于数据编码。另外对于过滤器和不经意键值对存储两类数据结构提供了性能对比分析和基准测试。

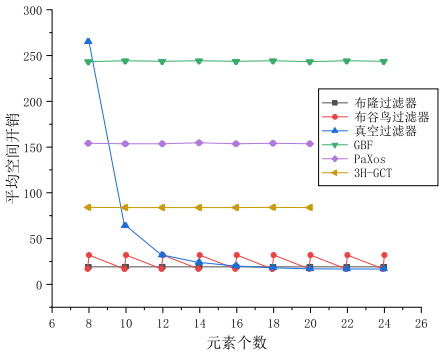
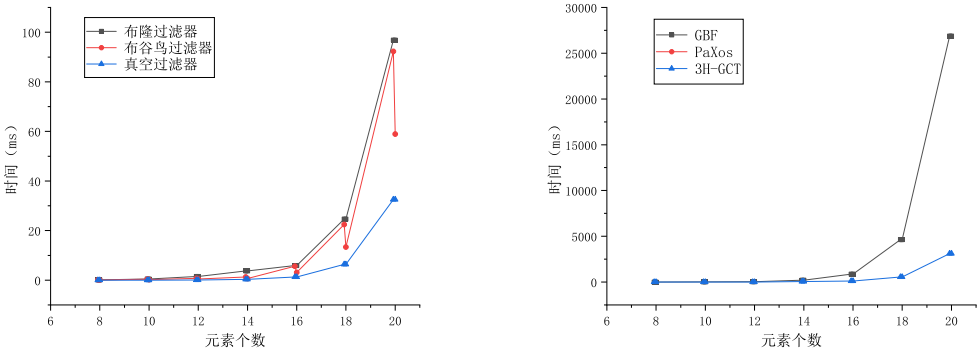


图 1: 元素个数与平均空间开销对比



(a) (b)

图 2: 元素个数与插入元素所用时间对比

参考文献

[1] Yuriy Arbitman, Moni Naor, and Gil Segev. "Backyard cuckoo hashing: Constant worst-case operations with a succinct representation". In: *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE. 2010, pp. 787–796.

[2] Gilad Asharov et al. "More efficient oblivious transfer and extensions for faster secure computation". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 535–548.

[3] Yossi Azar et al. "Balanced allocations". In: *Proceedings of the twenty-sixth annual ACM symposium on theory of computing*. 1994, pp. 593–602.

[4] Pierre Baldi et al. "Countering gattaca: efficient and secure testing of fully-sequenced human genomes". In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 691–702.

[5] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[6] *BloomFilter*. https://github.com/ArashPartow/bloom.

[7] Martin Burkhart et al. "{SEPIA}:{Privacy-Preserving} Aggregation of {Multi-Domain} Network Events and Statistics". In: *19th USENIX Security Symposium (USENIX Security 10)*. 2010.

[8] Jan Camenisch, Gregory Neven, et al. "Simulatable adaptive oblivious transfer". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2007, pp. 573–590.

[9] Melissa Chase and Peihan Miao. "Private set intersection in the internet setting from lightweight oblivious PRF". In: *Annual International Cryptology Conference*. Springer. 2020, pp. 34–63.

[10] Hao Chen, Kim Laine, and Peter Rindal. "Fast private set intersection from homomorphic encryption". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 1243–1255.

[11] Hao Chen et al. "Labeled PSI from fully homomorphic encryption with malicious security". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1223–1237.

[12] *CuckooFilter*. https://github.com/efficient/cuckoofilter.

[13] Alex Davidson and Carlos Cid. "An efficient toolkit for computing private set operations". In: *Australasian Conference on Information Security and Privacy*. Springer. 2017, pp. 261–278.

[14] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 2013, pp. 789–800.

[15] Thai Duong, Duong Hieu Phan, and Ni Trieu. "Catalic: Delegated PSI cardinality with applications to contact tracing". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2020, pp. 870–899.

[16] Bin Fan et al. "Cuckoo filter: Practically better than bloom". In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, pp. 75–88.

[17] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. "Efficient private matching and set intersection". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2004, pp. 1–19.

[18] Michael J Freedman et al. "Keyword search and oblivious pseudorandom functions". In: *Theory of Cryptography Conference*. Springer. 2005, pp. 303–324.

[19] Gayathri Garimella et al. "Oblivious key-value stores and amplification for private set intersection". In: *Annual International Cryptology Conference*. Springer. 2021, pp. 395–425.

[20] Niv Gilboa. "Two party RSA key generation". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 116–129.

[21] Carmit Hazay. "Oblivious polynomial evaluation and secure set-intersection from algebraic PRFs". In: *Journal of Cryptology* 31.2 (2018), pp. 537–586.

[22] Kyle Hogan et al. "Secure multiparty computation for cooperative cyber risk assessment". In: *2016 IEEE Cybersecurity Development (SecDev)*. IEEE. 2016, pp. 75–76.

[23] Mihaela Ion et al. "On deploying secure computing: Private intersection-sum-with-cardinality". In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2020, pp. 370–389.

[24] Yuval Ishai et al. "Extending oblivious transfers efficiently". In: *Annual International Cryptology Conference*. Springer. 2003, pp. 145–161.

[25] Ágnes Kiss et al. "Private Set Intersection for Unequal Set Sizes with Mobile Applications." In: *Proc. Priv. Enhancing Technol.* 2017.4 (2017), pp. 177–197.

[26] Vladimir Kolesnikov and Ranjit Kumaresan. "Improved OT extension for transferring short secrets". In: *Annual Cryptology Conference*. Springer. 2013, pp. 54–70.

[27] Vladimir Kolesnikov et al. "Efficient batched oblivious PRF with applications to private set intersection". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 818–829.

[28] Vladimir Kolesnikov et al. "Scalable private set union from symmetric-key techniques". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2019, pp. 636–666.

[29] *libPSI*. https://github.com/osu-crypto/libPSI.

[30] Catherine Meadows. "A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party". In: *1986 IEEE Symposium on Security and Privacy*. IEEE. 1986, pp. 134–134.

[31] Michael Mitzenmacher. "The power of two choices in randomized load balancing". In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (2001), pp. 1094–1104.

[32] *MultipartyPSI*. https://github.com/osu-crypto/MultipartyPSI.

[33] Moni Naor and Benny Pinkas. "Oblivious transfer and polynomial evaluation". In: *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 1999, pp. 245–254.

[34] *ObliviousDictionary*. https://github.com/cryptobiu/ObliviousDictionary.

[35] Michele Orrù, Emmanuela Orsini, and Peter Scholl. "Actively secure 1-out-of-N OT extension with application to private set intersection". In: *Cryptographers' Track at the RSA Conference*. Springer. 2017, pp. 381–396.

[36] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo hashing". In: *Journal of Algorithms* 51.2 (2004), pp. 122–144.

[37] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Faster private set intersection based on {OT} extension". In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 797–812.

[38] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Scalable private set intersection based on OT extension". In: *ACM Transactions on Privacy and Security (TOPS)* 21.2 (2018), pp. 1–35.

[39] Benny Pinkas et al. "Efficient circuit-based PSI with linear communication". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 122–153.

[40] Benny Pinkas et al. "Phasing: Private set intersection using permutation-based hashing". In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 515–530.

[41] Benny Pinkas et al. "PSI from PaXoS: fast, malicious private set intersection". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2020, pp. 739–767.

[42] Benny Pinkas et al. "SpOT-light: lightweight private set intersection from sparse OT extension". In: *Annual International Cryptology Conference*. Springer. 2019, pp. 401–431.

[43] Amanda C Davi Resende and Diego F Aranha. "Faster unbalanced private set intersection". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2018, pp. 203–221.

[44] Peter Rindal and Mike Rosulek. "Improved private set intersection against malicious adversaries". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2017, pp. 235–259.

[45] Peter Rindal and Phillipp Schoppmann. "VOLE-PSI: fast OPRF and circuit-psi from vector-ole". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2021, pp. 901–930.

[46] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. "Privacy preserving error resilient DNA searching through oblivious automata". In: *Proceedings of the 14th ACM conference on Computer and communications security*. 2007, pp. 519–528.

[47] *VacuumFilter*. https://github.com/wuwuz/Vacuum-Filter.

[48] Minmei Wang and Mingxun Zhou. "Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters". In: *Proceedings of the VLDB Endowment* (2019).

[49] Andrew C Yao. "Protocols for secure computations". In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, pp. 160–164.

[50] Cong Zhang et al. "Optimal Private Set Union from Multi-Query Reverse Private Membership Test". In: *Cryptology ePrint Archive* (2022).

[51] 宋祥福 et al. "面向集合计算的隐私保护统计协议". In: 计算机研究与发展 57.10 (2020), p. 2221.

[52] 申立艳 et al. “隐私保护集合交集计算技术研究综述”. In: 计算机研究与发展 54.10 (2017), p. 2153.

[53] 魏立斐 et al. “面向隐私保护的集合交集计算综述”. In: 计算机研究与发展 (2021), pp. 1–18.