Waldo: A Private Time-Series Database from Function Secret Sharing

> 📜【DRP 2022】*Waldo: A Private Time-Series Database from Function Secret Sharing* [1]
>
> - *Author(s)*：Emma Dauterman，Mayank Rathee，Raluca Ada Popa，Ion Stoica
> - *Venue*：2022 IEEE Symposium on Security and Privacy (SP)
> - *Materials*：PDF，Slides，Video1，Video2，Video3，Code

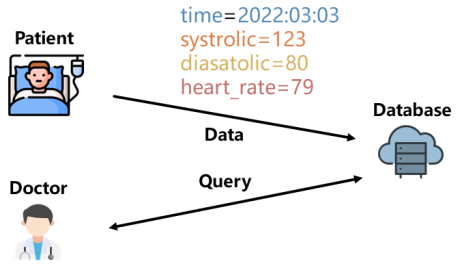
Overview

#1 文章针对什么问题?

今天的许多应用程序依赖云数据库来存储和查询时间序列数据。虽然外包存储很方便，但这些数据通常很敏感，这使得数据泄露成为一个严重的问题。下图举了一个远程病人监护系统的例子。

对于病人而言，即数据的提供方，其通过身上的传感器上传数据至云端，但由于存在数据泄露的危险，病人可能不想将这些数据存储在云端，

对于医生而言，即查询者，其可能希望运行图中所示的查询语句来评估充血性心力衰竭风险，但不愿意向服务器透露过滤值**(systrolic > 110等)** 或查询结果 **(AVG(heart_rate))**



MedicalHistory

time	systrolic	diasatolic	heart_rate
2022:03:00	110	70	80
2022:03:01	112	88	72
2022:03:02	108	72	76

```
SELECT AVG(heart_rate) FROM MedicalHistory
WHERE (time BETWEEN 2022:03:01 and 2022:03:02)
AND (systrolic > 110 OR diasatolic > 80)
```

#2 文章采用了什么方法？针对相关问题，前人已经提出了什么方法？有哪些不足？本文所提出的方法有何不同？有哪些创新性?

上述问题的一种解决方案是对加密的时间序列数据执行查询，比如 **Timecrypt** [2] 和 **Zeph** [3]。但这些系统有两个严重的局限性

- 它们仅支持按时间对数据流进行聚合（例如，一周内的平均心率），然而许多现代的时间序列数据库支持多维数据，并允许用户根据未预定义的不同谓词进行过滤。
- 这种方案会向服务器泄露查询时间间隔，这对某些应用程序来说可能是个问题。例如，如果医生正在查询患者的心率，则查询的时间段可能会显示患者心脏病发作或开始服用新药的时间。

通用**MPC** 是解决这一个问题的一个工具，然而现有工具需要大量通信。在分布式信任环境中，服务器可能部署在不同的云中以最大限度地减少多台服务器受到威胁的可能性，许多的通信往返和大带宽意味着高延迟和高金钱成本。

文章介绍了 ***Waldo***，一个功能丰富且安全性强的时序数据库，支持**多谓词过滤**，保护**数据内容**以及**查询过滤值和搜索访问模式**，并在 3 方诚实多数中提供恶意安全环境。

多谓词过滤：

Waldo 提供了两种类型的索引：

- 基于多个谓词的加法聚合（例如总和、计数、均值、方差、标准差）
- 在一段时间间隔上的任意聚合（例如最大 、最小 、top-k）。

之前的工作 (即 **Timecrypt** 和 **Zeph**) 只支持随时间的聚合。

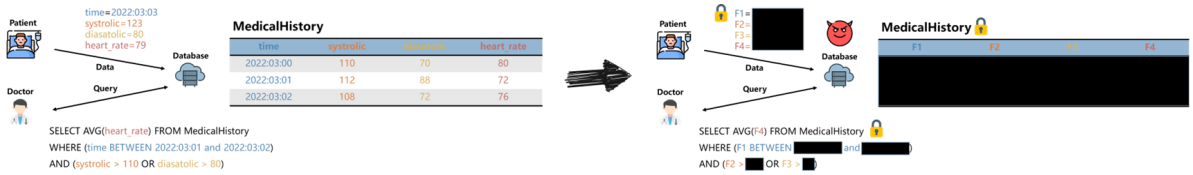
提供恶意安全环境：

Waldo 使用三台服务器并在至少两台服务器是诚实的情况下提供恶意安全性，这意味着如果最多有一个服务器是恶意的，它会提供安全中止。

保护数据内容以及查询过滤值和搜索访问模式：

对于攻击者来说,可见的信息为：

1. 数据库模式（即每个表的记录数、特征数和每个特征的大小）；
2. 查询的结构（即谓词的数量、谓词的类型(包括谓词是相等还是范围谓词以及谓词对应的特征）、连词和否定的结构、被聚合的特征）；
3. 何时执行查询以及哪个客户端执行了查询。



#3 取得了什么效果？ 本文所提出的方法有什么效果？特别是在可以量化的性能指标上，提升了多少？

Waldo 建立在函数秘密共享（Function Secret Sharing，FSS）上。ORAM 和通用 MPC 不太适合时间序列数据库设置，因为它们需要多轮交互 (ORAM) 和大量通信开销 (MPC)。

为了克服这些缺点，并在服务器位于不同的信任域时提高效率， 需要减少对有限且昂贵的通信的依赖 ，而是利用计算资源的代价来代替通信的代价，因为这些资源便宜得多且易于增加。考虑到这个目标，文章转向函数秘密共享 (FSS)，这是一种较新的密码学原语，它允许客户端生成一个函数的秘密共享，然后不同的服务器可以使用它来评估相等谓词（如 heart_rate = 82）或范围谓词（如 systolic > 110 ）。更重要的是，服务器可以在没有交互的情况下评估它们在谓词中的份额（**相比之下，其他最先进的 MPC 技术需要与比较次数成比例的交互**）

在 32 核机器上运行 ***Waldo***。特征的大小为 2^8，***Waldo*** 在 0.22 秒内针对 210 条记录运行具有 8 个范围谓词的查询，而 MPC 为 1.75 秒，ORAM 为 9.60 秒，并且 220 条记录用时 11.82 秒，而 MPC 为 45.72 秒，ORAM 为 16.70 秒。对于 210 到 220 条记录，MPC 在服务器之间使用的带宽比 Waldo 多 9-82 倍，对于 1-10 个谓词，ORAM 在客户端和服务器之间使用的带宽比 Waldo 多 20-152 倍。 Waldo 也是高度可并行的。

#4 文章进行了什么实验？ 设计了什么实验方案？搭建了什么实验平台？得到了什么实验结果？如何对比/分析实验结果。

Experiment setup:

• Three 32-core servers

• 3 Gbps connection with 20ms RTT

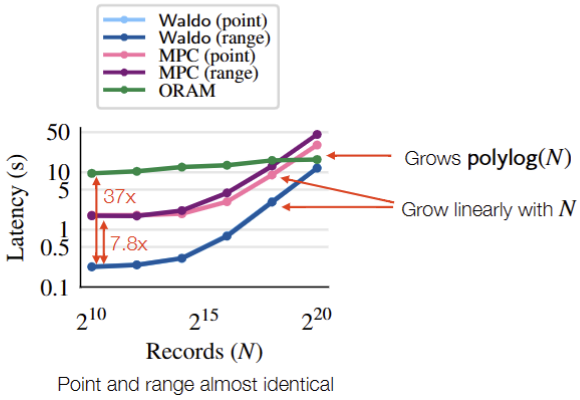
• 8-predicate queries

Baselines:

• Oblivious multidimensional tree (R-tree with PathORAM)

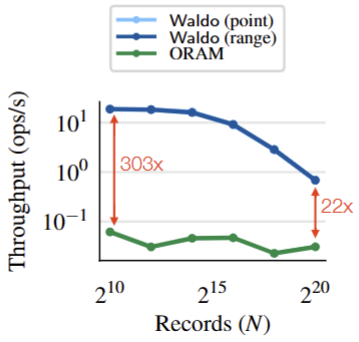
• Generic MPC: MP-SPDZ (3PC honest-majority)

Latency



Point and range almost identical

Throughput



。。。

#5 对我们有什么启发？ 和我们要研究的问题/课题有何借鉴/批判作用?

。。。

Setting and Tools

协议的基本设定

敌手模型	敌手数量	参与方
Malicious	Honest majority	3 party

所用基本工具

复制秘密共享（RSS）

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

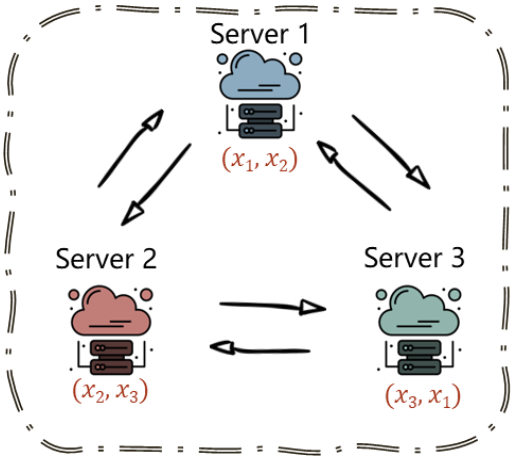
共享表示为：$[\![x]\!] := (x_1, x_2, x_3)$

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：$[\![x]\!]_3 = (x_1, x_3)$



函数秘密共享（FSS）

常见的函数共享有 3 种：

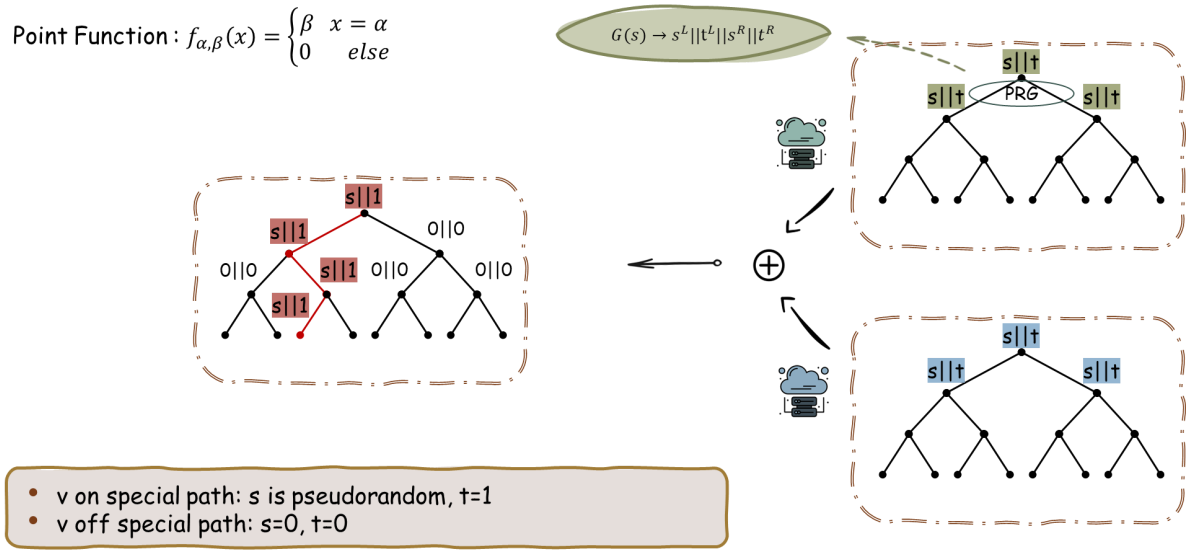
- 点函数秘密共享
- 比较函数秘密共享
- 区间函数秘密共享

具体构造参考[4][5][6]，下面简单介绍一下点函数秘密共享。

点函数秘密共享（PFSS）主要由两部分组成：

- $Gen(\alpha, \beta) \to K_1, K_2$
- $Eval(K_i, x) \to f(x)_i$

简单来说就密钥生成方利用 $Gen(\alpha, \beta)$ 将点函数进行拆分生成两个密钥，之后将其发送给两个服务器分别计算 $Eval(K_i, x)$，其计算值分别是 $f(x)_i$，两方重构后便是最后的结果 $f(x)$。对于两个服务器来说，其各自计算结果对它们是不可区分的，它们无法分辨其计算结果是 0 还是 β 的秘密份额，对计算方来说只是没有意义的比特串。

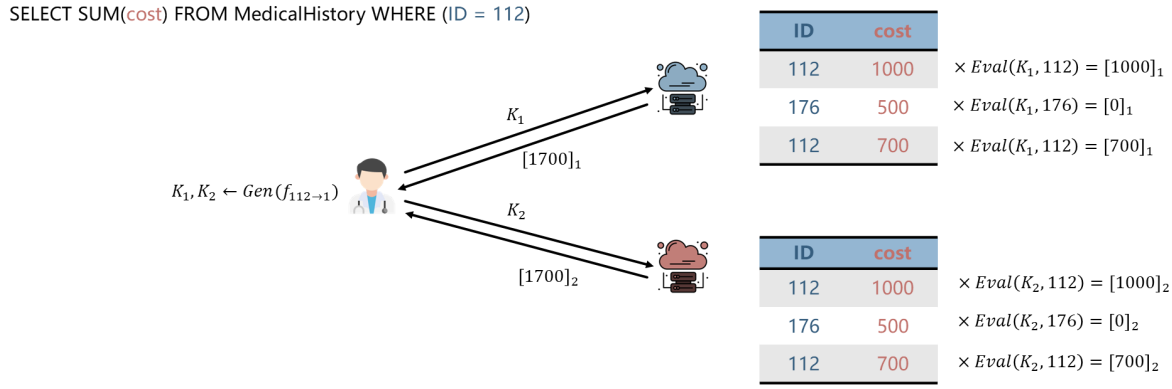


Multi-predicate queries

1. Single predicate with Semihonest security

FSS for public data

将 **FSS** 应用到公共数据很简单，查询方只需将 FSS 密钥分发给服务器，服务器对需要查询的特征的每一行都进行 **FSS** 计算，最后将其结果之和返回给查询方即可，查询方将两方计算结果相和即为最终查询结果。



FSS for private data

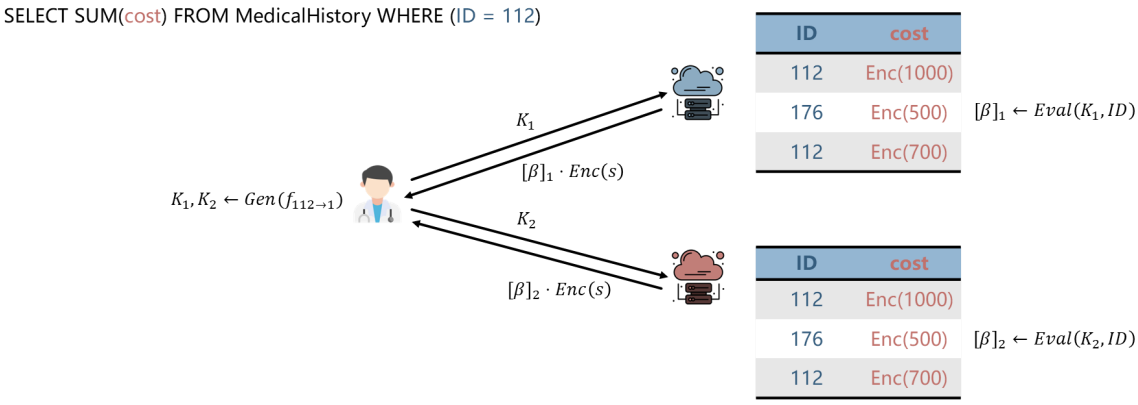
使用 **FSS** 评估私有数据存在一个问题，因为为了正确性，需要为计算函数秘密共享的服务器提供相同的输入以产生正确的输出。然而服务器不应该知道数据库中的明文值，这对服务器来说应该是不可见的，以明文形式提供输入显然是不可行的。

之前关于使用 FSS 进行安全计算的工作 [5] [6] 通过确保服务器持有附加掩码版本$(x + r)$的秘密来保持值不被泄露。在数据库设置中，每个条目 x 必须使用独立采样的 r 进行屏蔽。因此，即使服务器只需要评估单个函数，我们也需要为每个数据库条目 x 提供不同的 r。这实际上意味着函数共享的大小将与数据库的大小相匹配，从而违背了使用 FSS 来最小化通信的目的。

为了保证数据对服务器是不可见的，有两种可能的解决方法：

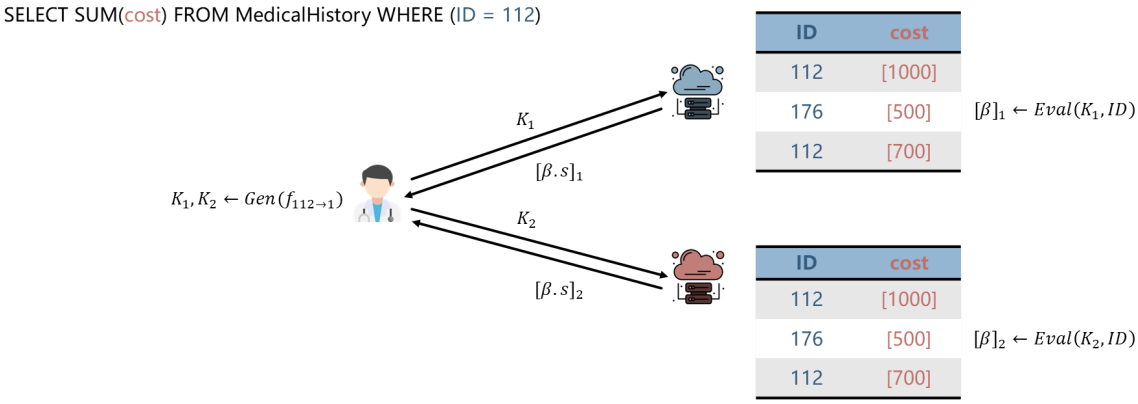
1. **Additively homomorphic encryption**

一种方法是使用加法同态方案对数据库进行加密。然而这需要昂贵的公钥操作，明文下的数据加密后可能达到上千位。



2. **Secret shares**

另一种方法是使用秘密共享的方式将数据库拆分为两部分，然而这其中每一行乘法计算都需要乘法三元组，代价昂贵且需要双方进行通信。



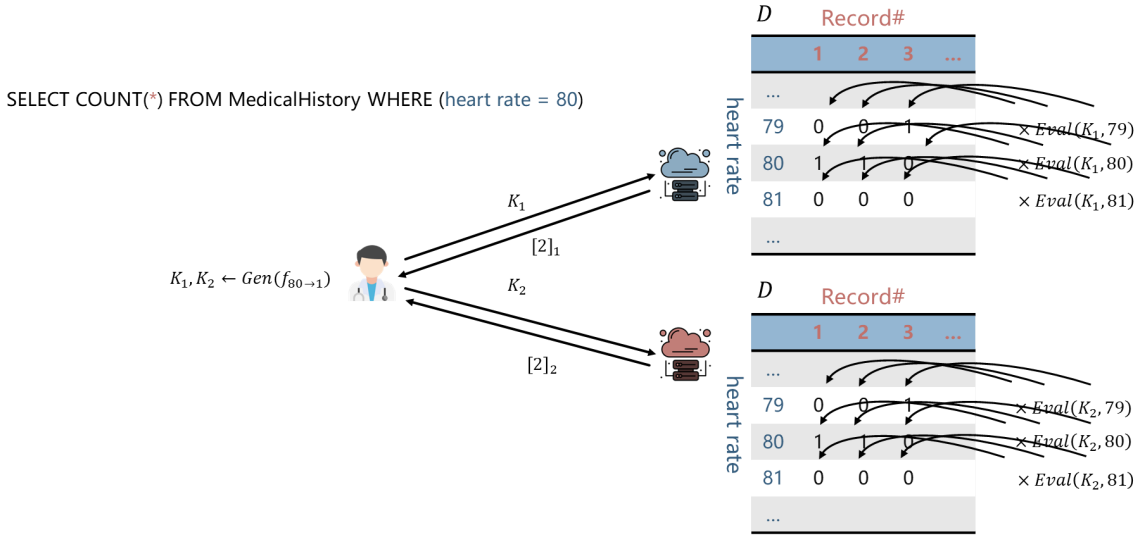
文章对这个问题的解决方案受到 **Dory** [7] 的启发。对于每个特征，其构建一个大小为 $N \times 2^l$ 的表，其中 N 是可以查询的记录数，2^l 是特征大小（即该特征的可能值的数量)。对于每条记录，在表中对应值的位置设置为 “1”，在其他地方设置为 “0”。这种结构为 **one-hot** 索引，因为它是一个 **“one-hot”** 向量表。

现在我们可以利用搜索索引的结构而不是内容来利用 FSS。我们要计算匹配谓词的记录数。

假设服务器需要计算匹配谓语的记录数（e.g. SELECT COUNT(*) FROM MedicalHistory WHERE (heart rate = 80))

对于表 D 中的每个条目 $d_{i,j}$，其中记录索引 $i \in \{N\}$，特征值 $j \in \{2^l\}$，服务器利用其 FSS 密钥对每个值进行评估，将结果乘以表条目 $d_{i,j}$，然后计算

$$EvalPred(s, K, D) \leftarrow \sum_{j=0}^{2^l-1} (Eval(K, j) \cdot \sum_{i=0}^{N} d_{i,j}) \tag{1}$$



这里还有两个问题。首先，需要考虑如何使用较小的特征尺寸 2^l 对不同类型的记录值进行编码，因为所需的计算量为 $O(N \cdot 2^l)$。其次，但如果 $d_{i,j}$ 的值被加密，则求和将不会产生正确的结果。我们在下面解决这两个问题。

对于第一个问题，文章指出所有带有谓词计算的隐私特征已经来自一个小的域（大小为2^8）。文章还总结了三种使用小特征尺寸在大域中编码值的技术。

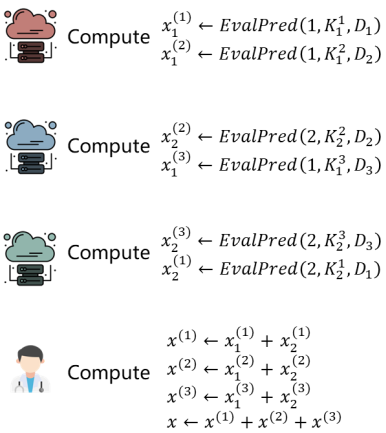
对于第二个问题（在计算正确结果的同时保护数据库内容）。如之前提到的一样，如果采用 2-out-of-2 secret sharing，即表 D 也以相同的方式共享，则方程 (1) 中的每条记录都需要进行 2^l 次乘法，则总共需要执行 $N \cdot 2^l$ 次乘法来评估单个谓词，如果这些乘法必须使用 Beaver 三元组，这是不切实际的。

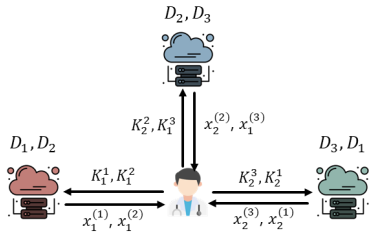
为了解决这个问题，文章引入了复制秘密共享，它隐藏了数据内容，且只需要本地乘法。复制秘密共享需要从两台服务器切换到三台服务器，但是这第三台服务器能显着提高性能。RSS 与 FSS 的组合使得单深度乘法基本上免费（无需通信）

具体构造如下：

服务器以 RSS 的方式拥有数据库表 D 的 D_1, D_2, D_3 的其中两部分，查询方生成三对 FSS 密钥 $(K_1^1, K_2^1,), (K_1^2, K_2^2), (K_1^3, K_2^3)$ 并分发给服务器。

则





通过这种方式，RSS 允许我们对服务器隐藏数据内容并评估单个谓词而无需服务器之间的通信。整个查询过程中服务器之间都不需要进行交互。

同时数据拥有者对服务器追加记录也很简单：只需为其特征值构造 one-hot 向量 并以 RSS 的方式发送给服务器即可。

2. Multiple predicates with semihonest security

3. Multiple predicates with malicious security

4. Putting it together

References

1. Dauterman E, Rathee M, Popa R A, et al. Waldo: A private time-series database from function secret sharing[C]//2022 IEEE Symposium on Security and Privacy (SP). IEEE, 2022: 2450-2468. ↩

2. Lukas Burkhalter, Anwar Hithnawi, Alexander Viand, Hossein Shafagh, and Sylvia Ratnasamy. Timecrypt: Encrypted data stream processing at scale with cryptographic access control. In USENIX NSDI, pages 835–850, 2020. ↩

3. Lukas Burkhalter, Nicolas Küchler, Alexander Viand, Hossein Shafagh, and Anwar Hithnawi. Zeph: Cryptographic enforcement of end-to-end data privacy. In USENIX OSDI, pages 387–404, 2021. ↩

4. Boyle E, Gilboa N, Ishai Y. Function secret sharing: Improvements and extensions[C]//Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016: 1292-1303. ↩

5. Boyle E, Gilboa N, Ishai Y. Secure computation with preprocessing via function secret sharing[C]//Theory of Cryptography Conference. Springer, Cham, 2019: 341-371. ↩ ↩

6. Boyle E, Chandran N, Gilboa N, et al. Function secret sharing for mixed-mode and fixed-point secure computation[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2021: 871-900. ↩ ↩

7. Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica. DORY: An Encrypted Search System with Distributed Trust. In USENIX OSDI, pages 1101–1119, 2020. ↩