软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(9):2753−2777 [doi: 10.13328/j.cnki.jos.005591]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

不经意随机访问机研究综述*

吴鹏飞[1,3], 沈晴霓[1,3], 秦 嘉[2,3], 钱文君[1,3], 李 聪[1,3], 吴中海[1,3]

[1](北京大学 软件与微电子学院,北京 102600)

[2](北京大学 前沿交叉学科研究院,北京 100871)

[3](北京大学 软件工程国家工程研究中心,北京 100871)

通讯作者: 吴中海,沈晴霓, E-mail: {wuzh,qingnishen}@ss.pku.edu.cn

摘 要: 随着云计算与大数据技术的发展,隐私保护越来越受到人们的关注.加密是一种常见的保护数据隐私的方法,但是单纯地利用加密手段并不能抵抗所有类型的攻击.攻击者可以通过观察用户对数据的访问模式来推断隐私信息,其中包括数据的重要程度、数据的关联性,甚至是加密数据的内容等.不经意随机访问机是一种重要的保护访问模式的手段,它通过混淆每一次访问过程,使其与随机访问不可区分,从而保护真实访问中的访问操作、访问位置等信息.不经意随机访问机在安全云存储系统以及安全计算领域有着非常重要的作用.利用不经意随机访问机可以降低攻击者通过访问模式推测隐私信息的可能性,减小系统受到的攻击面,从而提供更安全更完整的服务.对不经意随机访问机的研究与应用进行综述,主要介绍了不经意随机访问机的相关概念以及设计方法,重点分析并总结了目前学术界研究的性能优化的常见策略及其优劣性,主要包括针对客户端与服务器的平均带宽与最坏情况带宽优化、存储开销优化以及交互轮数优化等方面.同时讨论了将不经意随机访问机应用于安全存储系统的一般性问题,如数据完整性保护以及支持多用户并发访问等,也讨论了将其应用于安全计算领域的问题,如安全计算协议设计以及不经意数据结构的设计等;最后,对不经意随机访问机未来的研究方向进行了展望.

关键词: 不经意随机访问机;访问模式;隐私保护;安全计算;安全存储

中图法分类号: TP311

中文引用格式: 吴鹏飞,沈晴霓,秦嘉,钱文君,李聪,吴中海.不经意随机访问机研究综述.软件学报,2018,29(9):2753−2777. http://www.jos.org.cn/1000-9825/5591.htm

英文引用格式: Wu PF, Shen QN, Qin J, Qian WJ, Li C, Wu ZH. Survey of oblivious RAM. Ruan Jian Xue Bao/Journal of Software, 2018,29(9):2753−2777 (in Chinese). http://www.jos.org.cn/1000-9825/5591.htm

Survey of Oblivious RAM

WU Peng-Fei[1,3], SHEN Qing-Ni[1,3], QIN Jia[2,3], QIAN Wen-Jun[1,3], LI Cong[1,3], WU Zhong-Hai[1,3]

[1](School of Software and Microelectronics, Peking University, Beijing 102600, China)

[2](Academy for Advanced Interdisciplinary Studies, Peking University, Beijing 100871, China)

[3](National Engineering Research Center for Software Engineering, Peking University, Beijing 100871, China)

Abstract: With the development of cloud computing and big data technology, privacy protection draws more people's attention. Data encryption is a common way to protect data privacy, but solely using encryption cannot resist all types of attacks. Adversary can observe the access pattern on how users access to the data, to infer the private information including the importance of the data, the relevance between the data and even the plaintext of encrypted data. Oblivious RAM (ORAM) is an important method to protect the access pattern, including access operations and access locations, by obscuring an actual access, which makes adversary unable to distinguish it from a random one. ORAM makes an important role in designing secure cloud storage systems and secure computation. ORAM can reduce the

* 基金项目: 国家自然科学基金(61672062, 61232005)
Foundation item: National Natural Science Foundation of China (61672062, 61232005)
收稿时间: 2018-01-03; 修改时间: 2018-03-16; 采用时间: 2018-04-21

possibility of the adversary inferring the private information through the access pattern and reduce the attack surface of the system, so as to provide a safer and more complete service. This paper summarizes the researches and application settings of the ORAM, mainly introducing the relevant concepts of the model as well as design methods with focus placed on analyzing and summarizing common strategies to optimize the model and their advantages and disadvantages, as well as optimizations for amortized and worst-case bandwidth between client and server, storage overheads reduction and round-trips reduction. Moreover, this paper discusses general issues of the ORAM used for secure storage system designing including data integrity and concurrent accesses for multi-clients. The paper also discusses some issues of the ORAM used for secure computation, including secure computation protocols designing and oblivious data structure designing, and finally makes a conclusion for the future research directions of the ORAM.

Key words: oblivious RAM; access pattern; privacy protection; secure computation; secure storage

随着大数据与云计算技术的发展,越来越多的数据可以在云端进行存储、计算以及共享.但是这也带来了一系列安全问题,例如:个人用户如果直接将敏感数据以明文形式存储在云服务器,那么对于不可信的云服务提供商或者其他攻击者可以直接获取用户的隐私信息,并通过数据挖掘的方式推测用户的其他个人信息.因此,为了保证用户的隐私数据不被恶意的窃取,传统的手段是对数据内容进行加密,密钥由用户个人持有.用户通过上传、下载密文数据来保护个人的隐私信息不被泄露[1,2].

但是事实上,加密只能保证数据内容的机密性,却不能保证其他的隐私信息不被泄露[3].在云存储场景中,用户需要查询某一个数据块的内容.为了保证数据隐私性,通常会对数据库中的数据进行加密.但是在执行查询请求的过程中,数据块的索引并不能加密,这就泄露了用户访问数据库元素的位置.这使得攻击者可以通过用户的访问模式(access pattern)来推断存储数据的重要性,例如统计每一个数据块的访问频率;同时,攻击者还可以通过匹配前后两个连续的访问模式来推断数据查询之间的关联关系[3],甚至是加密数据的内容[4].在实际场景中,这些信息潜在性地暴露了用户的行为特征、兴趣爱好、社交范围等,例如,一个人查询某一类药物,可以推断他患有什么疾病.另一方面,在安全计算领域,当用户将隐私数据和任务以加密方式存储在内存中,如果暴露了处理器对内存的访问模式,就可能泄露数据和任务本身的信息[6,7].因此,在大数据与云计算等应用场景中,仅通过对数据本身的内容加密并不能完全的保护用户的隐私.对用户的访问模式的保护,也是目前重点的研究目标.

不经意随机访问机是目前保护访问模式的一种重要手段.这一技术的目的是隐藏对真实数据块的访问,使得攻击者不能区分每一次访问是真实的还是随机的.研究表明:不经意随机访问机应用于云存储系统可以有效的防止攻击者利用访问模式获取隐私信息,减小数据存储系统的攻击面,打破了单纯使用传统加密方式来保护数据隐私的系统框架,为用户提供更完善的安全存储服务.同时,利用不经意随机访问机来模拟安全多方计算的执行过程已被证实比传统用电路模拟的方法更高效[8],它极大地提升了对于海量输入数据的安全计算性能.因此,不经意随机访问机为多方计算应用于大数据场景提供了可能[9],这对于设计更安全高效的分布式计算框架有着重要的意义.但是,不经意随机访问机也会带来额外的开销,比如:为了隐藏访问模式,需要对多个数据块进行访问,这增大了客户端与服务器之间的带宽,客户端需要更大的缓存空间来存储从服务器端返回的额外数据块.所以,不经意随机访问机的实用性还面临很大的挑战.针对这个问题,学术界的学者已经提出了多种设计方案与优化策略.本文通过整理并归纳了近 10 年关于不经意随机访问机的研究工作,介绍了模型常见的设计方案以及性能优化策略,并且结合云存储以及安全计算场景分析不经意随机访问机在实际应用中的问题与挑战,为在该领域的研究人员提供方法总结与研究思路,并对今后的研究方向进行探讨.

本文第 1 节给出在暴露访问模式场景下推测加密信息的威胁场景,并对不经意随机访问机的定义以及相关概念进行概述.第 2 节对不经意随机访问机研究领域关注的问题以及研究现状进行总结.第 3 节对已有的不经意随机访问机设计方案以及优化策略进行总结,并比较不同方案在性能上的差异.第 4 节分析不经意随机访问机在云存储场景下的问题以及已有的解决方案.第 5 节总结不经意随机访问机结合安全计算场景的问题以及已有的解决方案.本文最后对全文进行总结,阐述不经意随机访问机仍然需要解决的问题并展望未来的研究方向.

1 不经意随机访问机概述

1.1 加密环境下隐私泄露的威胁场景

一些研究工作已经证实,仅通过加密方式是不能完全保护数据隐私的.在一些环境下,攻击者依然可以通过观察数据的访问模式推测加密数据的隐私信息.例如在可搜索加密模式(searchable encryption scheme)下推测用户的查询内容[4]、加密数据的顺序[5]以及在分布式计算框架,如 MapReduce 或者 Spark 的洗牌(shuffle)过程中推测用户的隐私信息[10−12]等.

以文献[4]中的场景为例,用户 Alice 将一个文本集 D 用可搜索加密模式进行加密,并提交至半诚实服务器 Bob,D 中一共包含 m 个不同的单词.Alice 通过提交请求 Q,查询某一个单词 K 在 D 中的哪些文本中出现.为了保证查询的隐私性,Alice 首先将 K 转化成可识别的陷门单词 $Trapdoor_K$,查询结果 R=(doc_1,doc_2,…,doc_n)以明文形式返回(暴露了访问模式).当 Bob 搜集到从 Alice 提交的 l 个查询请求 $\tilde{Q}=(Q_1,Q_2,...,Q_l)$ 以及返回结果集合 $\tilde{R}=(R_1,R_2,\ldots,R_l)$后,可以推断出 Alice 每一次查询的单词 K 是什么,即,确定查询单词的顺序集合 $K_A=\langle K_{a_1},K_{a_2},...,K_{a_l}\rangle$,使得 $\forall i\in[1,l],Trapdoor_{K_{a_i}}=Q_i$,如图 1 所示.

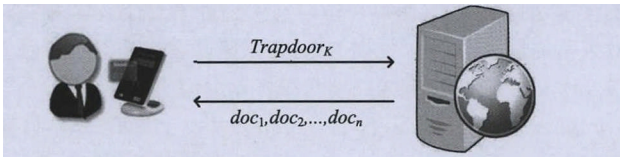


Fig.1 Threat model[4]

图 1 威胁场景[4]

在这一攻击场景中,Bob 可以监听 Alice 与其之间的通讯信道,并且拥有以下两种背景知识.

- Bob 知道 Alice 提交 l 个查询请求中,k 个请求具体对应什么单词.即,$K_Q=\{\langle x,y\rangle\,|\,(x\in K_A)\wedge(y\in\tilde{Q})\wedge(y=Trapdoor_x)\}$,并且 k=$|K_Q|$.那么,这 k 个请求可表示为 $\tilde{S}=\{y\,|\,\exists x,\langle x,y\rangle\in K_Q\},\tilde{S}\subset\tilde{Q}$;
- Bob 知道关于 D 中单词的分布 M,M 是一个 $m\times m$ 的矩阵,M 中的每一个元素 M_{ij} 表示第 i 个单词和第 j 个单词在均匀采样的文本 $d\in D$ 中同时出现的概率,即 M_{ij}=Pr[($K_i\in d$)∧($K_j\in d$)].

因为 Bob 已经知道了 l 个查询请求中 k 个请求所对应的单词,他只需要通过观察查询结果来确定剩下的 $\tilde{Q}-\tilde{S}$ 个查询.实际上,根据 Bob 拥有的第 2 个背景知识,对于任意查询 $Q'\in\tilde{Q}-\tilde{S}$,其返回结果 R'应与先验知识 M 相一致.因此,可以将攻击过程转化成求解一个优化问题:

$$\arg\min_{\langle a_1,\ldots,a_l\rangle}\sum_{Q_i,Q_j\in\tilde{Q}}\left(\frac{R_{Q_i}\cdot R_{Q_j}^T}{n}-(K_{a_i}\cdot M\cdot K_{a_j}^T)\right)^2 \tag{1}$$

$$\text{s.t.}\begin{cases}\forall j,Q_j\in\tilde{S};\forall a_j=x_j,\langle K_{x_j},Q_j\rangle\in K_Q\\ \forall j,\|Q_j\|=1\end{cases} \tag{2}$$

对于公式(1)中两个查询 Q_i 和 Q_j,可以根据它们的查询结果计算出两个单词在文本中联合出现的频率,即 $(R_{Q_i}\cdot R_{Q_j}^T)/n$.然后,Bob 通过背景知识计算出两个单词联合出现的先验概率,即 $K_{a_i}\cdot M\cdot K_{a_j}^T$.要使推测查询 Q_i 对应的单词尽可能准确,即令先验概率和后验概率之间的距离最短即可.通过实验验证,这一方法可以推测出大约 80%的查询请求.

1.2 相关概念

不经意随机访问机(oblivious RAM,简称 ORAM)的概念最早起源于 RAM(random access machine)模型,RAM 是一种重要的计算仿真手段.在这个模型中,处理器通过对存储器的读写来实现程序的执行.上个世纪 80

年代,为了隐藏程序对内存的访问模式来避免软件的逆向工程,Goldriche 等人在此基础上提出了 ORAM[13].所谓的访问模式,指的是处理器访问内存的操作序列和地址序列[14].ORAM 保证了在存储器中的任意数据块不会永久驻留在某一个物理地址中,这确保了任意两次访问不会产生关联.同时,ORAM 将每一次读写访问(access)细化成一次读取加一次写回的原子操作(operation),其中读访问转化成读取内容再写回相同内容,写访问转化成读取内容再写回更新后的内容,使得攻击者不能够区分具体的访问方式,如图 2 所示.

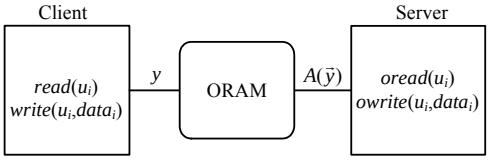


Fig.2 An overview of oblivious RAM[10]

图 2 不经意随机访问机框架图[10]

因此,ORAM 可以很好地保护以下 4 种属性:(1) 访问数据块的位置;(2) 数据块请求的顺序;(3) 对相同数据块的访问频率;(4) 具体的读写访问方式.这使得在访问结束之后,攻击者不能根据访问模式来区分任意两个相同长度的访问序列.在第 1.1 节的场景中,可以使用 ORAM 来隐藏客户端对服务器的访问模式,避免攻击者直接获取查询结果 R.通常,ORAM 的定义是基于客户端与服务器之间有状态的交互式两方协议[15].我们用$((c_out, c_state),(s_out,s_state))\leftarrow protocol((c_in,c_state),(s_in,s_state))$来表示客户端与服务器之间的执行协议,其中,$c\_in$ 与 c_out 表示客户端的输入与输出,s_in 与 s_state 表示服务器的输入与输出,c_state 与 s_state 表示协议执行前后客户端与服务器的状态.

定义 1(oblivious RAM (ORAM)[15]). ORAM 通常包含以下客户端与服务器之间的交互行为:

- $((\perp,C),(\perp,D))\leftarrow Setup(1^{\lambda},(Input,\perp),(\perp,\perp))$:表示客户端与服务器构造 ORAM 的交互协议,其中,λ表示安全参数,$Input$ 表示客户端提交的初始数据集,服务器的输入为空($\perp$),同时,客户端与服务器的初始状态也为空.在 Setup 协议执行完之后,客户端达到状态 C,服务器达到状态 D,同时,两方均没有输出;
- $((data,C'),(\perp,D'))\leftarrow Access((op,C),(\perp,D))$:表示客户端访问服务器的执行协议,其中,$op$ 表示一个读操作 $read(addr)$或者是一个写操作 $write(addr,data)$.$addr$ 表示被读取或者写入的数据块的地址,$data$ 表示被写入的数据.C 与 D 分别表示客户端与服务器执行访问协议之前的初始状态.当协议执行完之后,如果是读操作,客户端获得对应地址 $addr$ 的数据,如果是写操作,客户端获得更新之前对应地址 $addr$ 的数据.C'与 D'分别表示客户端与服务器执行访问协议之后的状态.

定义 2(oblivious RAM 的安全性[16]). 用 $\vec{y}=((op_1,addr_1,data_1),(op_2,addr_2,data_2),...,(op_M,addr_M,data_M))$ 表示一个长度为 M 的数据请求序列,其中,每一个操作 op_i 表示一个读操作 $read(addr_i)$或者是一个写操作 $write(addr_i, data)$.用 $A(\vec{y})$ 表示对于给定的数据请求序列 $\vec{y}$ 所产生的访问序列.如果一个 ORAM 是安全的,那么对于两个长度相同的数据请求序列 $\vec{y}$ 和 $\vec{z}$,它们的访问模式 $A(\vec{y})$ 和 $A(\vec{z})$ 是计算不可区分的.

定义 3(oblivious RAM 的正确性[17]). 如果一个 ORAM 的设计是正确的,那么以数据请求序列 $\vec{y}$ 为输入得到的数据返回结果与直接通过 $\vec{y}$ 得到的数据返回结果有超过 $1-negl(|\vec{y}|)$ 的概率是一致的,其中 $negl(|\vec{y}|)$ 表示以 $\vec{y}$ 为输入的可忽略函数.

在 ORAM 的设计中,通常考虑两个重要的组成部分:一个是 ORAM 的初始化,另一个是利用 ORAM 进行数据访问.在初始化阶段[8,18],服务器为客户端创建 ORAM 空间,客户端通过安全参数获得用来加密数据库的密钥.然后,利用语义安全(semantic secure)加密的方式加密数据库中的每一个数据块.通常,数据块中不但包含数据内容,还包括数据块索引以及数据块的位置、是否被访问过等属性信息.最后,客户端将加密好的数据块上传至 ORAM.在有些 ORAM 设计以及应用[9,19,20]中也提到:在执行正式访问之前,还包含一个预热过程(warm-up),这是通过预先访问 $O(N)$个数据块,使得 ORAM 达到一个稳定状态,其中,N 指的是服务器存储数据块的数量.在通用的 ORAM 研究中,一般不考虑它的初始化开销;但是在结合安全计算场景中,由于初始化需要结合协议来完

成,这一部分必不可少,而且开销很高[21].

在利用 ORAM 对服务器进行访问时,通常包含两种主要操作:读(read),写(write).读操作和写操作是 ORAM 的两种原子操作.读操作是将访问的数据块取回客户端本地;写操作是将取回本地后的数据块更新后再写回服务器.客户端对服务器的任意读写访问都可以通过这两种原子操作来表示.为了维护访问的隐私性与 ORAM 系统稳定性,不同设计模型还需要设计其他的离线操作.对于平方根模型和层次模型,在一个访问周期(epoch)结束后,需要将所有的数据块位置重新混淆,这称为混洗操作(reshuffle).对于树状模型,为了保证每一个数据块集合(bucket)不溢出,需要不断的将数据块从一个物理地址移动到更大空间的物理地址,这称为驱逐操作(eviction).因此可以看出,不经意随机访问机主要有以下特性.

(1) 低效性:相比于正常的访问,不经意随机访问机需要执行额外的操作来保护访问模式的隐私性,这种访问模式往往带来昂贵的开销,包括带宽以及本地存储等,这严重的限制了 ORAM 的实用性;

(2) 安全性:不经意随机访问机提供了保护访问模式的可证明安全性,相比于传统的加密手段,该技术可以很大程度上减少攻击者利用访问模式推断隐私信息的可能性;

(3) 用途广:不经意随机访问机可以广泛地应用于安全存储以及安全计算领域,对于存在数据访问的应用,都可以利用 ORAM 提供访问模式的保护.

2 不经意随机访问机研究的现状分析

不经意随机访问机面临的最大问题是性能开销大.纵观不经意随机访问机的发展,其设计模型大致可以分为 5 类:简单模型、平方根模型、层次模型、分区模型和树状模型.不同的设计模型表示服务器存储数据块的数据结构不同,用户通过对服务器ORAM的访问来获取所需的数据块.这些模型设计的目标主要还是提高性能.

- 简单模型:服务器以数组的方式连续存储客户端的数据块.为了隐藏客户端访问了哪一个数据块,客户端每一次访问需要遍历所有数据块.对于不需要的数据块,客户端读取之后再写回相同的数据块;对于目标数据块,客户端读取后在本地更新,再写回更新后的数据块.同时,即使客户端找到目标数据块,依然继续访问;
- 平方根模型:将服务器划分成两部分:排列数组(permuted array)与缓冲区(shelter).排列数组中包含 N 个真实数据块(real block)和 $\sqrt{N}$ 个无效数据块(dummy block).客户端每次访问之前需要将排列数组中的所有的数据块混洗.在每一次访问中,客户端先查找目标数据块是否在缓冲区中:如果在缓冲区中,就从排列数组中读取一个无效数据块;如果没有在缓冲区中,就从排列数组中读取目标数据块.为了混淆数据块的访问位置,每一个访问周期($\sqrt{N}$ 次访问)需要重新混洗排列数组;
- 层次模型:将服务器的存储划分为层,第 i 层包含 2^i 个数据块集合(bucket),对于每一层而言,当一个访问周期(2^i 次访问)结束后,需要将当前层的数据块和下一层的数据块合并并且混洗后放入下一层.每一层实际上都是一个 hash 表,并且包含一个 hash 函数.每一层的访问周期结束后,需要更换 hash 函数.当客户端访问一个数据块时,从最顶层到最底层依次查找,通过计算 hash 函数来判断目标数据块是否在当前层:如果在,则根据 hash 函数计算所得的偏移量获得对应数据块;如果不在,则继续查找下一层.与简单模型类似,为了保护访问模式的隐私,即使找到了目标数据块也会继续查找下一层,之后,每一层会随机访问一个无效数据块,直到所有层都被访问.当客户端更新完数据块后,将其写入服务器最顶层.由于顶层的访问周期短,其中的数据块会频繁地混洗到下一层,因此不用担心顶层数据块溢出的问题;
- 分区模型:将数据存储在 $\sqrt{N}$ 个服务器(partitions)中,每一个服务器利用平方根模型或者层次模型构建,同时,客户端存储数据块索引到数据块在服务器位置之间的映射表以及每一个服务器的缓存(cache slots).对于客户端每一次访问,先根据映射表查找数据块所在的服务器,然后使用对应 ORAM 模型的访问方式获取数据块.当获得数据块之后,将其重新分配一个新的服务器,并写入对应服务器的缓存中.缓存满之前执行将数据写回对应的服务器;
- 树状模型:在层次模型上进行改进,将每一个数据块集合分配到树的节点上,客户端本地存储每一个数

据对应树的叶子节点的映射关系(position map).以 Tree ORAM[27]为例,每一次访问先查询数据块所在的叶子节点,将从根节点至这个叶子节点上所有的数据块集合取回本地.更新完之后,将目标数据块写入根节点.每一次访问结束之后,在每一层随机选择 v 个数据块集合执行驱逐操作,将一个真实数据块写入它对应叶子节点那条路径上的子节点数据块集合中.同时,再选出一个无效数据块写入它另一个子节点的数据块集合中.

表 1 归纳了这 5 种设计模型的优缺点.

Table 1 Pros & cons of oblivious RAM designing models

表 1 ORAM 设计模型优缺点

ORAM 分类	优点	缺点
简单模型	(1) 模型便于理解与实现; (2) 可以以简单模型为基础,来衡量其他 ORAM 的性能	(1) 开销太大,客户端需要存储所有的数据块; (2) 客户端与服务器之间的平均通信带宽和最坏情况带宽都很高,不适用于很多具体的应用场景
平方根模型	只需要访问缓冲区中的数据,相比于简单模型降低了带宽	(1) 需要复杂的混洗(不经意排序)操作,利用排序网络[22−24]实现会带来额外的 $O(N\log N)$的开销; (2) 平均带宽依然很高,不适用于很多具体的应用
层次模型	(1) 相比于平方根模型,层次模型考虑了每一个数据块被访问的频率[13]; (2) 在平方根模型的基础上进一步优化了带宽	(1) 依然需要复杂的混洗操作; (2) 每一次混洗后,需要依据 hash 函数计算每一个数据块对应的数据块集合,容易导致 hash 值冲突; (3) 平均带宽依然很高,不适用于很多具体的应用
分区模型	(1) 分布式 ORAM 可以适用于云存储场景; (2) 在层次模型上进一步优化了带宽	(1) 需要更大的客户端存储空间; (2) 每一个服务器依然需要复杂的混洗操作,同时,驱逐操作也相对困难
树状模型	(1) 每一次驱逐操作只与当前节点和其父子节点产生关联,降低了驱逐操作的复杂度; (2) 相比于平方根模型和层次模型,降低了最坏情况的复杂度	(1) 驱逐操作的复杂度很高; (2) 客户端需要存储一个映射表,增大了客户端的存储空间

我们总结了近 10 年在信息安全顶级会议(CCS,S&P,USENIX Security 和 NDSS)上关于 ORAM 的研究工作.其主要研究方向大致分为以下 3 大类.

- 通用 ORAM 性能优化.这一类研究方向主要的目标是优化 ORAM 本身的性能,减小与一般访问之间的开销差距.衡量的指标主要包括:(1) 客户端与服务器之间的平均带宽(amortized bandwidth);(2) 最坏情况的带宽(worst-case bandwidth);(3) 带宽加速比(bandwidth blowup);(4) 交互轮数(round-trips); (5) 客户端存储空间(client storage)等.也有一些其他的指标,例如在线带宽(online bandwidth)、离线带宽(offline bandwidth)、数据块大小(block size)、是否具有服务器计算能力(server-computation capability)等,表 2 总结了常见 ORAM 的性能.在绝大多数 ORAM 研究中[13,16,17,25−28],一般考虑存储加密数据的服务器是不可信的,提交访问请求的客户端是可信的;也有一些论文研究对于恶意客户端的场景[29,30];
- ORAM 安全存储系统设计.这一类方向主要考虑设计与实现利用 ORAM 保护访问模式的安全存储系统.除了优化系统性能之外,还需要解决 ORAM 结合存储场景的实用性问题,例如如何保证数据访问的完整性[31−33]以及如何支持多用户在异步网络下并发访问的问题[20,34,35]等;
- 结合 ORAM 的安全计算.一些研究工作证实[8,37]:在海量数据输入的场景下,利用 RAM 模型模拟计算过程要比单纯利用传统电路实现安全计算要更高效.将 ORAM 与安全计算协议相结合,既可以保证计算过程的安全性(基于 Yao 的混淆电路),也可以保证在计算过程中数据访问的隐私性(基于 ORAM).通常,这一类方向主要包含两个研究点:(1) 设计基于 ORAM 的安全两方或者多方计算协议[8,37];(2) 不经意数据结构的设计[38,39]等.

还有一些其他结合 ORAM 的研究方向,例如安全处理器的设计[40]、利用 ORAM 加密磁盘文件[41]等.

Table 2 Common oblivious RAMs and performance comparison

表 2 常见的 ORAM 与性能比较

ORAM 分类	ORAM 代表	客户端存储大小	服务器存储大小	平均带宽	最坏情况带宽	数据块大小	交互轮数
简单模型	Trivial ORAM[13]	$O(1)$	$O(N)$	$O(N)$	$O(N)$	$\Omega(1)$	$O(N)$
平方根模型	Square-Root ORAM[13]	$O(1)$	$O(N)$	$O(\sqrt{N}\log N)$	$O(N\log N)$	$\Omega(1)$	$O(\log N)$
层次模型	Hierarchical ORAM[13]	$O(1)$	$O(N\log N)$	$O(\log^3 N)$	$O(N\log^2 N)$	$\Omega(1)$	$O(\log N)$
分区模型	Partition ORAM[16] (SSS ORAM)	$O(\sqrt{N})$	$O(N)$	$O(\log^2 N)$	$O(\sqrt{N})$	$\Omega(\log N)$	$O(\log N)$
树状模型	Tree ORAM[27]	$O(\log^2 N)$	$O(N\log N)$	$O(\log^3 N)$	$O(\log^3 N)$	$\Omega(\log N)$	$O(\log N)$
	Path ORAM[17]	$O(\log N)$	$O(N)$	$O(\log N)$	$O(\log N)$	$\Omega(\log^2 N)$	$O(\log N)$
	Ring ORAM[25]	$O(\log N)$	$O(N)$	$O(\log N)$	$O(\log N)$	$\Omega(\log^2 N)$	$O(\log N)$
	Onion ORAM[26]	$O(1)$	$O(N)$	$O(1)$	$O(1)$	$\Omega(\log^5 N\cdot\log^2\log N))$	$O(\log N)$
	S3ORAM[28]	$O(1)$	$O(N)$	$O(1)$	$O(1)$	$\Omega(\log N)$	$O(\log N)$

3 通用 ORAM 性能优化方案分析

对 ORAM 进行性能优化,是目前对 ORAM 研究的核心问题.这一类方向通常不考虑具体的应用场景,主要是研究如何设计高效合理的 ORAM 结构,或者是应用一系列优化策略提升 ORAM 的性能.用来衡量 ORAM 性能的指标有很多,目前,这一类问题最关注的指标有 5 种:客户端与服务器的平均带宽、客户端与服务器最坏情况的带宽、客户端的存储开销、服务器端的存储开销和客户端与服务器的交互轮数.

3.1 客户端与服务器平均带宽优化

客户端与服务器之间的平均带宽是考量 ORAM 性能的重要指标之一.在现有的 C-S 架构中,通常客户端与服务器的内存空间可以达到几个 GB 甚至几十个 GB,磁盘空间也可以高达几百个 GB,甚至 TB 级.但是客户端与服务器之间的通信带宽通常只有 MB 级.因此,带宽资源要比存储资源更珍贵是这一类问题的出发点.在 ORAM 中,平均带宽指的是每一次访问客户端与服务器之间所需要传输的平均比特数.特别地,在平方根模型和层次模型中,由于混洗过程产生的离线带宽,通常按一个访问周期中总共传输的比特数与周期内访问次数的比值作为平均带宽.对平均带宽的优化策略有很多,研究比较多的有基于布谷鸟哈希(cuckoo hashing)的优化策略、基于服务器计算的优化策略、基于映射表的优化策略以及基于 k 叉树的优化策略等.

3.1.1 基于布谷鸟哈希的优化策略

在传统的层次结构中,第 i 层包含 2^i 个数据块集合,并且每一个数据块集合包含 $O(\log N)$个数据块.在查询数据块时,先根据哈希函数计算数据块所在的数据块集合,然后在数据块集合中线性查找对应的数据块.而在利用布谷鸟哈希[3]时,可以直接将数据块存在每一层中,不需要依赖于数据块集合.因此在每一次查询中,可以减少 $O(\log N)$的线性搜索时间.文献[14,42]在布谷鸟哈希的基础上结合了 Hadoop MapReduce,设计出 MapReduce 布谷鸟哈希(MapReduce cuckoo hashing),通过并行的在两个哈希表构造出的二分图上执行宽度优先搜索算法,可以更快地判断是否发生冲突.

在层次模型中,两层数据块混洗后计算哈希值,重新确定数据块的位置容易导致哈希值冲突,进而产生数据溢出的问题.一种场景的解决策略是利用布谷鸟哈希[43]的方法,将每一层的哈希表替换成两个,每一个哈希表都有一个哈希函数.如果其中有一个哈希函数计算的哈希值出现冲突,就利用另一个哈希函数进行计算,并将数据块放入对应哈希表中.当然也会存在两个哈希函数都冲突的情况,那么就将第 1 个哈希表中对应的数据块取出,用其第 2 个哈希函数重新计算哈希值,如果对应位置为空,那么就将其放入那一个哈希表的对应位置.但是布谷鸟哈希并不能完全抵抗哈希值的冲突,如果两个哈希表依然存在冲突问题,需要重新选择两个哈希函数,并计算每一个数据块的哈希值.

3.1.2 基于服务器计算的优化策略

早期的 ORAM 研究中,通常将服务器理解为只能提供存储功能的对象.在文献[13]中,Goldreich 等人提到:在 $O(1)$的客户端存储空间下,不具备服务器计算能力的 ORAM 有着$\Omega(\log N)$的带宽下界.在近几年的研究中,一

些工作[25,26]为了进一步提升ORAM性能,允许服务器有计算能力,虽然其中有一些计算是不安全的.通常将没有计算能力的 ORAM 称为标准模型(standard model),有计算能力的 ORAM 称为非标准模型(non-standard model).

在传统的基于树状模型的 ORAM 中,当客户端提交一个数据请求后,服务器需要将目标数据块所在的从根节点至叶节点路径上的所有数据块返回给客户端,以保证数据访问的隐私性.但是这将导致至少 $O(\log N)$数量级的开销.文献[25,44,45]通过允许服务器执行异或计算,将路径上的所有数据块执行异或.这样做使得每一次读取的返回结果只有一个数据块,可以将在线带宽降为 $O(1)$.由于路径上除了目标数据块是有效的之外,其余数据块均为无效数据块.因此,客户端可以很容易地通过层数、偏移量等信息构造无效数据块,进而恢复目标数据块[45].但是由于在一般 ORAM 研究的威胁模型中通常考虑服务器是不可信的,所以虽然这种方法降低了客户端与服务器之间的带宽,但是允许服务器执行异或操作是不安全的.

还有一些工作结合隐私信息检索(PIR),允许服务器执行一些安全的计算,例如同态加密[15,19,26,46,48,49].由于隐私信息检索本身就可以很大程度上避免服务器推测用户的隐私,因此这一类工作也可以适用于 ORAM.客户端通过提交一个加密的选择向量(select vector),服务器利用其执行同态加密计算,得到一个加密的目标数据块返回给客户端,客户端通过解密得到结果.但是利用同态加密依然存在不足:(1) 计算开销很大,虽然有一些工作[26,46]使用了半同态或者是部分同态,但是依然很难用于实际;(2) 迭代的利用同态加密会导致密文大小持续性增长,带来额外的存储开销,文献[26]利用了 Damgård-Jurik 加密系统[50]缓解了密文膨胀,但是这个问题依然存在.除了同态加密之外,允许服务器支持秘密共享[28]也可以用来实现ORAM,而且这种方法要比利用同态加密高效的多.客户端将数据以秘密共享的形式存放在多个服务器中,通过将选择向量秘密共享发送给服务器,每一个服务器执行隐私信息检索之后,将目标数据块的秘密分块返回给客户端,客户端根据秘密重构恢复数据块.但是由于秘密共享所基于的求解多项式方程组的数学原理,使得其依赖于 t-隐私(t-privacy)[51],很难应用于基于不可信云管理员的云场景中.

允许服务器执行隐私信息检索除了可以降低平均带宽,还可以减少客户端的存储(见第 3.3 节).因此,利用隐私信息检索的服务器计算来设计 ORAM,这一研究方法近年来受到了广泛的关注[52].

3.1.3 基于客户端存储的优化策略

在传统的层次模型中,可以通过计算每一层的哈希值确定数据块在层中的位置.但是如果数据块不在当前层,需要继续查找下一层,再一次计算哈希值.这会带来额外 $O(\log N)$的计算开销.一种解决方案[53]是在客户端本地存储一个数据块与层数之间的映射关系,当需要查找数据块时,可以先通过定位数据块所在的层,然后执行哈希函数确定所在的数据块集合.这样可以减少 $O(\log N)$的查询开销.

这种基于映射表的优化策略在树状模型中广为使用[17,25−28],客户端本地记录树中每一个数据块对应的叶子节点,当需要查询数据块时,可以直接定位到从根节点至叶子节点所在的路径.这样避免了复杂的遍历过程,提升了平均带宽.明显地,这种基于映射表的策略也可以降低交互轮数(见第 3.5.1 节),避免复杂的客户端比较验证的过程,因此,这种方法在很多种 ORAM 中都有应用[17,25−28].但是在客户端存储映射表也会带来额外的存储开销,一种常见的解决方案是利用迭代构造的思想(见第 3.3.1 节).

还有一种客户端存储的优化是客户端存储一个缓冲区(stash),当每一次数据写回时,不直接将其写回服务器,而是先保存在缓冲区中.当缓冲区满之前,将其中数据块再全部写回服务器.这样的好处在于:每一次查找数据块时,先在本地查找,如果存在可以直接获取,降低了与服务器之间的通信带宽.例如在 Partition ORAM[16]中,客户端有着对应于每一个服务器的本地缓存,大小为 $O(\sqrt{N})$.因此在这种情况下,平均带宽仅为 $O(\log N)$(迭代构造时为 $O(\log^2 N)$).还有文献[41]提出将树状模型的前 $\log\sqrt{N}$ 层存储在客户端,称为 tree top caching.这样避免了每一次都需要访问整个服务器带来的开销.这种设计也被用于优化 Ring ORAM[25].

3.1.4 基于 k 叉树的优化策略

对于传统的树状模型,通常使用的是二叉树结构,每一个节点只允许有两个子节点.由于在查询过程中,树状模型是按树的层数返回数据块,例如 Tree ORAM[27]中,当查找到目标数据块后还要继续查找,直到叶子节点,以保证访问的隐私性;Ring ORAM[25]中,需要将 $O(\log N)$个数据块返回客户端,其中一个为目标数据块,其余的均

为无效数据块(如果不考虑 XOR 计算).因此,查询的次数与返回的数据块与树的层数有很大关系.一种常见的解决方案是使二叉树扩展为 k 叉树.这样可以保证在不增加存储容量的前提下,降低树的层数.对于二叉树而言,访问的层数为 $O(\log_2 N)$,而 k 叉树仅有 $O(\log_k N)$,因此可以降低 $O(\log k)$的平均带宽.但是这种方法并不是没有缺点,降低了层数导致每一层数据块集合增加,使得树状模型的驱逐操作会更复杂.

3.1.5 方案比较与讨论

以上 4 种优化策略是常见的提升 ORAM 平均带宽的方法.为了更直观地表达和对比每一种具体方案的优缺点与性能,我们做了如下总结(见表 3).

Table 3 Comparisons of commom technologies to improve amortized bandwidth of ORAM

表 3 提升 ORAM 平均带宽常见技术比较

分类	技术	安全性	使用环境	性能提升	缺陷分析
基于布谷鸟哈希的优化策略	布谷鸟哈希	是	层次模型	$O(\log N)$	(1) 服务器需要额外的存储哈希表; (2) 不能完全抵抗哈希值冲突,如果发生冲突需要额外的计算
	MapReduce 布谷鸟哈希	是	层次模型	$O(\log N)$	同上
基于服务器计算的优化策略	XOR 计算	否	均可	$O(\log N)$	计算不安全,需要采取额外的措施抵抗恶意攻击者
	同态加密	是	均可	$O(\log N)$	(1) 计算开销大,很难用于实际; (2) 密文膨胀导致服务器需要额外的存储开销
	秘密共享	是	均可	$O(\log N)$	(1) 对同一个数据块要多个服务器共享秘密份额,存储开销较高; (2) 安全性依赖于 t-隐私,使用场景受限
基于客户端存储的优化策略	映射表	是	均可	$O(\log N)$	客户端需要额外的存储
	缓冲区	是	层次模型 树状模型	$O(\log N)$	(1) 客户端需要额外的存储; (2) 对于多用户场景会导致数据冲突
基于 k 叉树的优化策略	k 叉树构建	是	树状模型	$O(\log k)$	树状模型的驱逐操作将很复杂

通过分析这 4 种优化策略的方案与优劣性,实际上可以将这 4 种方案的出发点归为 3 个角度.

- 从存储角度以空间换时间.基于布谷鸟哈希和基于客户端存储的优化策略都是通过在服务器或者是在客户端添加额外的存储来保证在访问过程中减少带宽,从而降低访问的响应时间,但是这会导致存储开销的增大;
- 从计算角度减少访问的传输量.对于基于服务器计算的优化策略而言,它将访问过程中所需要传输的数据块定位到一个数据块,减少不必要的传输;同时,在访问过程中不会暴露数据块本身的信息,客户端也可以在访问结束之后重构目标数据块;
- 从优化设计模型角度降低平均带宽.由于树状模型每一次访问所需传输的数据块来源于同一条路径上的每一层,因此这一类方案尝试通过改变树结构来提高效率.基于 k 叉树的优化策略从设计结构上降低了树的高度,从而减少传输带宽.与前两者不同,这种方法没有采用额外的技术,但是这也会带来驱逐操作的复杂性.

3.2 客户端与服务器最坏情况带宽优化

在很多场景下,我们不能期望 ORAM 每一次访问都可以达到平均带宽的开销.当一次访问伴随混洗和驱逐操作时,复杂的后台计算将制约当前访问的响应时间.这种情况有时需要几倍甚至几十倍的平均带宽.因此,只追求平均带宽的最优是很难应用于实际场景的.最坏情况带宽考虑了这一问题,通过定量计算在一个访问周期或者是多次访问中最坏情况时的带宽,可以帮助我们更好地衡量 ORAM 的性能.这一类指标的优化方案一般包含以下 4 种:基于 de-amortization 的优化策略、基于树状模型的优化策略、基于改进混洗操作的优化策略和基于改进驱逐操作的优化策略等.

3.2.1 基于 de-amortization 的优化策略

分析最坏情况发生的原因,往往是因为一个访问周期中的一次访问与复杂的混洗和驱逐操作并发执行.例

如在层次模型中,如果一个访问的目标数据块此时恰好在执行混洗,需要合并写入下一层,当前的访问必须等混洗操作执行完之后才能执行.一种常见的解决策略是不用等复杂的混洗和驱逐操作最后执行,而是将其均摊到每一次访问,或者是每一个较短的时间段内执行,这种优化策略称为 de-amortization.例如在 Partition ORAM[16]中设计了均衡器(amortizer)模块,每一个时间周期内写回 $O(\log N)$的数据,这样就将原来复杂的 $O(\sqrt{N})$ 的最坏情况降低为 $O(\log N)$.De-amortization 优化策略也被应用于文献[25,26]中,对于每 A 次访问执行一次的驱逐操作,可以将其平均到每次访问中执行,这样,每一次访问只需要驱逐 $1/A$ 的节点,缓解了最坏情况下的带宽问题.

3.2.2 基于树状模型的优化策略

树状模型是一种经典的 ORAM 设计模型,也是目前最常用、变种最多的模型.最初,树状模型提出的初衷就是为了解决平方根模型和层次模型在最坏情况下的带宽问题.这两者都需要复杂的混洗操作,其中:平方根模型需要利用不经意排序将缓冲区的数据写回排列数组,需要 $O(n\log n)$的复杂度;而层次模型也需要将写满数据块的层与下一层的数据块执行不经意排序后写入下一层.在最坏情况下,层次模型的不经意排序要涉及到所有的层.树状模型继承了层次模型的思想,并做了创新:它将每一层的多个数据块集合划分到树的每一个节点中.这样的优势在于:当每一次执行驱逐操作,每一个数据块集合只与其父子节点有关,而与其兄弟节点无关,这样可以将驱逐操作的复杂度降到 $O(\log N)$,如图 3 所示.很大程度上提升了 ORAM 在最坏情况下的性能.如今,几乎所有的通用 ORAM 性能改进所衍生的 ORAM 都是基于树状模型[17,25−28],这也继承了树状模型在最坏情况下带宽的优势.

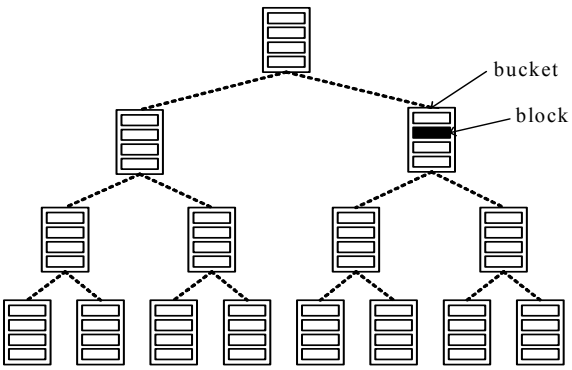


Fig.3 An overview of tree model[27]

图 3 树状模型结构图[27]

3.2.3 基于改进混洗操作的优化策略**

在平方根模型和层次模型中,复杂的混洗操作是提升性能的最大阻碍.为了保证访问的隐私性,需要将数据块在一个访问周期结束的时候更新位置,并且在同一个访问周期内,每一个无效数据块都只能被访问一次.一些工作尝试改进复杂的混洗过程来间接地提升 ORAM 性能.

对于混洗过程而言,为了保证数据在排序过程中的隐私性,通常采用不经意排序(oblivious sorting)[3],它要求排序过程中的访问模式不依赖于具体数据.冒泡排序是一种不经意排序,但是需要 $O(n^2)$的排序复杂度.一些其他工作在不经意排序算法上提出了改进,如 Batcher 提出了一种排序网络方案[22],复杂度为 $O(n\log^2 n)$;Atjai 等人[23]提出了更为优化的改进 AKS 排序网络,复杂度为 $O(n\log n)$,但是有着很大的常数项(大约是 6 100),因此实用性较差.2010 年,Goodrich 等人[24]利用随机希尔排序构造出一种复杂度同样为 $O(n\log n)$、但是常数项很小(大约是 7~8)的不经意排序算法.平方根模型和层次模型默认采用复杂度为 $O(n\log n)$的不经意排序算法.文献[51]中也给出了一种基于归并排序的不经意排序方法,复杂度也为 $O(n\log n)$.还有一些工作改进了混洗操作,例如文

**由于平均带宽需要将混洗操作的带宽平均到每一次访问中,因此基于改进混洗操作的优化策略实际上也可以优化平均带宽,这里我们考虑最直接的影响.

献[54]中利用客户端存储,通过本地计算两个数据块集合的位置,然后将其发送给服务器.将每一层的混洗转化成只涉及4个数据块集合(两个在客户端,两个在服务器)的归并操作.因此,这种方法被称为四联归并(quadruplet merge).

3.2.4 基于改进驱逐操作的优化策略***

与混洗操作类似,在树状模型中,驱逐操作是影响性能的最大因素.由于在树状模型中,每一次访问结束需要将目标数据块放入数的根节点中,为了保证每一个节点数据块集合不溢出,需要周期性的将数据块不断的写入其子节点中.

在Tree ORAM[27]中,对于每一次访问后的数据写回,需要先在树中每一层随机选择v个数据块集合,并从每一个数据块集合中选出一个真实数据块写入它对应叶子节点那条路径上的子节点数据块集合中.同时,再选出一个无效数据块写入它另一个子节点的数据块集合中,以保证访问的隐私性.但是这种驱逐操作需要在每一次访问之后都执行.为了避免这个问题,文献[46]提出了一种基于逆字典序(reverse lexicographical)的驱逐策略.在这种策略中,每一次驱逐操作并不是像Tree ORAM那样是随机的,而是固定的(通过模运算确定);并且是在根节点至叶节点的一条路径上执行驱逐操作.具体对应哪一个路径,可以通过将模运算的结果以二进制形式表示,再逆序来确定.这样做的目的是使任意连续两次驱逐操作重叠的路径尽可能的少.同时,这种驱逐策略有以下两个优点:(1) 不需要每一次访问后都执行,仅需要给定一个驱逐频率的参数,用来表示一个访问周期中访问的次数,当一个访问周期之后再执行;(2) 这种驱逐策略可以减少服务器的计算量,驱逐路径仅需要通过一个模运算确定,而且驱逐的节点仅与路径中的节点有关.这种驱逐策略被广泛应用在ORAM设计中,例如Onion ORAM[26],S3ORAM[28]等.

还有一些工作改进了树状模型的驱逐操作,例如,文献[55]中设计了一种基于贪婪方法的驱逐策略,当客户端执行访问之后,将本地缓冲区中当前访问的数据块尽可能地写入从根节点到与其分配叶节点相一致路径的数据块集合中.这种基于贪婪方法的驱逐策略也被用于文献[17,56]中.

3.2.5 方案比较与讨论

通过比较以上4种优化最坏情况带宽的方法,其实可以总结归纳为一点:都是通过优化在ORAM中最复杂的混洗和驱逐操作来降低带宽.稍有不同的是:基于树状模型的方案是从模型层角度进行优化,包括基于树状模型的变种[17,25,26,28]也都继承了树状模型在最坏情况下的带宽优势;而其他3种都是从方法角度优化,通过分析混洗和驱逐模型本身在设计与实现上的缺点,找出可能提升性能的方法,因此这一类方法往往可以适用于一大类的ORAM模型.通过分析以上4种方法的特点,我们可以归为3类.

- 平均化的思想.由于混洗和驱逐在ORAM中是非常复杂的执行过程,因此这种方法将这个执行平均到每一次或者每一段时间中,可以保证不会出现最坏情况下响应时间太长的情况.但是这也导致了每一次访问时间要比原来的访问时间略有提升;
- 优化混洗操作.树状模型的提出,实际上也可以理解为优化平方根模型和层次模型在混洗过程中的缺点,但是带来了驱逐操作.由于混洗操作通常依赖于不经意排序,因此这一类优化工作主要是设计或者选择一个最优的排序算法;
- 优化驱逐操作.在树状模型的节点中可能会出现数据溢出的问题,因此对于驱逐操作的优化通常只针对树状模型.并且在不同场景下驱逐操作的优化也略有不同,例如在云存储场景中,需要结合多用户,考虑驱逐操作时客户端数据的备份问题,在安全计算场景中需要考虑驱逐操作电路实现的复杂度等.

值得一提的是,也有一些工作,例如Onion ORAM,通过结合树状模型和优化驱逐操作的策略(3节点的驱逐(triplet eviction))来提升带宽,从而达到更好的效果.

***与基于改进混洗操作的优化策略类似,由于平均带宽需要将驱逐操作的带宽平均到每一次访问中,因此基于改进驱逐操作的优化策略实际上也可以优化平均带宽,这里我们考虑最直接的影响.

3.3 客户端存储开销优化

对客户端的存储优化也是目前通用 ORAM 性能提升的主流指标之一,由于客户端主要负责对数据的更新和计算,因此尽可能少地占用客户端的存储资源,是这个问题的出发点.对于优化客户端的存储,常用的优化策略包括以下 3 种:基于迭代构造的优化策略、基于隐私信息检索的优化策略和基于布隆过滤器的优化策略.

3.3.1 基于迭代构造的优化策略

为了降低服务器与客户端的平均带宽,一种常见的策略是在客户端本地存储一个映射表,以避免线性查询的需要(见第 3.1.3 节).但是这种方法无疑会增加客户端的存储开销.针对这个问题,文献[26]提出一种基于迭代构造的优化策略,将存储在客户端的映射表以另一个 ORAM 的形式存储在服务器中.当需要查询一个数据块的位置时,先在这个小 ORAM 中查询数据块在大 ORAM 中的位置.通过迭代构造,这个小 ORAM 在本地的映射表也可以用一个更小的 ORAM 存储在服务器.以 Tree ORAM[27]为例,假定树的高度为 H,那么服务器一共可以存储 2^H-1 个数据块集合(由于通常服务器存储量很大,因此可以近似看成 2^H 个),其中,叶子节点为 2^{H-1} 个,所以服务器的存储是叶子节点的常数倍,将这个 ORAM 记为 $ORAM_0$.此时,客户端存储的映射表记录 $ORAM_0$ 中每一个数据块与 2^{H-1} 个叶子节点的映射关系.通过迭代,可以将这 2^{H-1} 个叶子节点以另一个高度为 $H-1$ 的小 ORAM 存储在服务器,记为 $ORAM_1$.这样,本地只需要存储 $ORAM_1$ 中每一个数据块与其 2^{H-2} 个叶子节点的对应关系,分区模型的迭代构造方法如图 4 所示.通过对数级的迭代构造,最终客户端本地的映射表存储可以降低为 $O(1)$.这种迭代构造在通用 ORAM 性能提升中很常见[16,17,25−27],但是迭代构造会增加客户端与服务器的交互轮数.

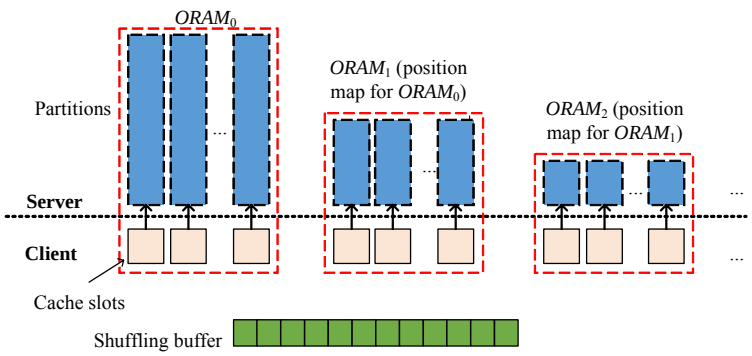


Fig.4 Recursive construction of partition ORAM[27]

图 4 分区 ORAM 的迭代构造[27]

3.3.2 基于隐私信息检索的优化策略

基于隐私信息检索的技术这几年受到研究人员的关注,利用 PIR 设计 ORAM 不仅可以很大程度上降低客户端与服务器之间的平均带宽(见第 3.1.2 节),还可以减小客户端的存储.对于传统的树状模型,每一次访问服务器都需要遍历从根节点到叶子节点的完整路径,有时还需要将其中每一个节点的数据块返回给客户端以保证访问的隐私性.因此,客户端需要用一个缓冲区存放从服务器返回的所有数据块(见第 3.1.3 节).但是利用隐私信息检索技术不需要依赖于遍历多个节点来保证隐私性,相反,客户端将选择向量发送给服务器,服务器只需要通过少量的计算可以直接获得对应的数据块结果,而计算过程的本身并不泄露任何隐私.因此,客户端不需要设计复杂的存储结构来记录服务器返回的数据,只需要用一个存储单元记录目标数据块即可.这样,客户端的存储可以降为 $O(1)$.这种技术被应用于 Onion ORAM[26],S3ORAM[28]中.

3.3.3 基于布隆过滤器(Bloom filter)的优化策略

基于布隆过滤器的技术也是一种常用的降低客户端存储的优化策略.通常,为了降低客户端与服务器的平均带宽,客户端可以存储一个数据块与逻辑地址之间的映射表(见第 3.1.3 节)来降低在逻辑地址内线性搜索的开销,但是这无疑会增加客户端的开销.一种基于布隆过滤器的解决方案是不需要本地存储映射表,而是将查找

的方法封装在一个加密的布隆过滤器中,并且存放在服务器ORAM中的每一层.当客户端提交查询时,会利用每一层加密的布隆过滤器来判断数据块是否在当前层:如果存在,就返回对应的结果;如果不存在,就随机查找无效数据块以保证访问的隐私性.这种技术被应用于文献[26,31,54]中.

3.3.4 方案比较与讨论

以上 3 种优化方法都是针对客户端存储,但是 3 种方法优化目标不同,并且各有优缺点.针对这些方法的不同特征,表 4 对这 3 种方法进行了总结:

Table 4 Comparisons of commom technologies to optimize the client size of ORAM

表 4 优化 ORAM 客户端大小的常见技术比较

方法	优化目标	适用环境	缺陷分析
基于迭代构造的优化策略	映射表	分区模型 树状模型	增加客户端与服务器之间的交互轮数
基于隐私信息检索的优化策略	缓冲区	均可	需要客户端与服务器进行额外的计算
基于布隆过滤器的优化策略	映射表	层次模型	(1) 增加服务器存储开销; (2) 增加客户端与服务器之间的交互轮数

通过比较分析以上 3 种优化方法,我们发现这 3 种方法主要是从两个角度优化客户端存储.

- 用服务器存储空间替换客户端存储.相比客户端,服务器存储要更低廉.基于迭代构造的优化策略和基于布隆过滤器的优化策略都是尝试在服务器端存储额外的数据,使得客户端需要通过执行更多的查找来获取相应的检索信息,而这些信息本可以在客户端本地存储后直接获得.因此,这些方法的缺点就在于增加了交互轮数;
- 从计算角度来减少客户端存储.结合第 3.1.2 节利用隐私信息检索可以降低平均带宽,这种方法使得客户端不需要更多的存储空间来缓存从服务器返回的数据.因此,这种基于隐私信息检索的 ORAM 优化策略在很多 ORAM 设计中经常使用.

3.4 服务器存储开销优化

在大多数ORAM设计中,为了保证访问的隐私性,服务器中不仅存储对用户有用的真实数据块,还存储一部分无效数据块.对于每一种具体的 ORAM,存储无效数据块的数量有很大差异,有时为了提升查询的性能,服务器也存储一些额外的工具(例如布隆过滤器等).这些都会给服务器带来额外的存储开销.针对这些问题,也有一些工作尝试对服务器的存储进行优化来提升服务器的效率.这一类方法主要是针对 ORAM 中数据块集合大小,通过设计可变大小的数据块集合或者是增加(减少)部分数据块集合大小等.

在传统的树状模型中,每一个节点可以包含 $O(k)$个数据块,但是实际上只有 $O(1)$个数据块是真实数据块,因此带来了 $O(k)$的存储开销.文献[46]提出了一种改进策略,允许每一个节点多存放 m 个真实数据块,那么每一个节点容量变成 $O(m+k)$,但是存储开销降为 $O(m/k)$.当 $m=k$ 时,开销仅为 $O(1)$.由于只增加了真实数据块的数量,此时依然可以保证节点溢出概率不变,为 e^{-k}.Sánchez-Artigas 等人[57]也发现:除了数据块集合的大小对服务器存储有很大的影响之外,还会影响驱逐操作.因此,他们提出两种改进策略:首先,根据每一次在同一路径上数据块的访问对数据块执行分组;同时,将 Tree ORAM 中的驱逐操作优化为对父节点和两个子节点数据块集合的驱逐.通过实验验证:相比于 Tree ORAM,在服务器端存储 2^{15} 个数据块时,这样的优化可以降低大约 80%服务器端的存储开销,同时降低大约 8 倍的通信开销.Moataz 等人[58]发现:传统树状结构的 ORAM 如果想扩展树的大小,只有通过添加叶子节点,但是这有可能破坏数据结构的概率完整性.因此,他们通过设计一种节点数据块集合大小可变的 ORAM,通过实验验证,这种优化方法可以降低 87%的服务器存储开销和 7%的通信开销.

对于服务器存储开销的优化策略主要出现在 ORAM 早期研究的一些论文中,近几年,针对这一问题的改进策略相对较少.但是这一问题依然不容忽视,例如,Path ORAM[17]需要 28 倍的存储空间,Ring ORAM[25]需要 6~8 倍的存储空间.也就是说,对于 1GB 的真实数据,服务器需要存储几倍甚至几十倍的额外数据,这还不考虑数据备份的问题,因此,这种开销在实际应用中是非常昂贵的.提供一种更好的解决服务器端的存储开销问题的方

案,是未来的研究重点之一.

3.5 客户端与服务器交互轮数优化

客户端与服务器之间的交互轮数也是通用 ORAM 性能提升的考虑指标之一,因为复杂的交互轮数会导致一次访问的响应时间延长,如果再考虑网络延迟的影响,使得 ORAM 很难应用于实际.因此,也有一些技术尝试通过优化交互轮数使得 ORAM 获得性能上的提升,这一类问题的解决方案通常有两种:一种是基于映射表的优化策略,另一种是基于混淆分支函数的设计.

3.5.1 基于映射表的优化策略

在第 3.1.3 节中,我们给出了允许客户端存储数据块索引与 ORAM 逻辑地址之间的映射表可以很大程度上减少客户端与服务器之间的平均带宽.其实从另一角度来说,映射表也是通过快速准确地定位目标数据块的位置,减少不必要的访问来降低带宽.因此,基于映射表的优化策略也可以用来降低客户端与服务器之间的交互轮数.Boneh 等人[53]将平方根模型和层次模型相结合,设计出一种可以降低服务器与客户端访问轮数的优化策略.他们保留了平方根模型中的缓冲区,并将其结合层次模型,设计出层次结构的缓冲区.同时,客户端存储每一个数据块索引与缓冲区层数之间的映射关系.当客户端访问数据块时,可以直接根据索引找到所在的层,进而执行访问,使得仅通过一次访问就可以得到数据块,避免了传统层次模型中需要遍历所有层的开销.同时,这在树状模型中使用也很广泛[17,25−28],如果没有映射表,服务器需要通过深度优先搜索或者是广度优先搜索的方法去先查找数据块的位置,这种方法的交互轮数将是很高的.

3.5.2 基于混淆分支函数的优化策略

在第 3.3.3 节中,我们提到了利用布隆过滤器可以降低客户端的存储,但是在层次模型中,每一层都需要有一个布隆过滤器,因此依然需要计算 $O(\log N)$次才能完成一次访问,因此,这也会带来 $O(\log N)$的交互轮数.文献[54]将整个查找过程定义为一个分支函数,并将其在客户端编译成混淆程序后,外包给服务器进行计算.当服务器计算完成后,将结果返回给客户端,不需要每一层执行完之后返回结果.因此,这种方法也可以将整个查询过程的交互轮数降为 $O(1)$.

3.5.3 方案比较与讨论

以上两种优化交互轮数的方法有很大的不同,并且各有优缺点.

- 第 1 种基于映射表的优化是通过及时地定位目标数据块,避免复杂的查找过程,从而降低查询次数.这种方法直接并且高效,因此被广泛的使用.但是这种方法也会带来客户端存储的开销;
- 另一种基于混淆分支函数的优化策略主要是针对利用布隆过滤器优化客户端存储后,导致交互轮数增加的问题,因此这种方法往往是应用在有布隆过滤器的场景下,因此使用较少.但是这种外包计算的思想完全可以借鉴.利用类似于 Yao 电路计算的协议,使得客户端和服务器之间执行一个安全计算,从而让客户端获得目标数据块.但是需要添加额外的应对恶意服务器场景的措施,提供相应的检测方法,以应对服务器返回错误的计算信息.

3.6 小 结

性能提升是目前ORAM研究的主流方向,除了在通用ORAM设计方向,在结合云存储和安全计算场景中也都有所涉及.通过分析与讨论已有工作的优化策略,我们大致可以对现今 ORAM 设计与性能提升的工作总结为以下 4 点.

- 每一种方案都有优缺点.在设计一种优化策略时,很难达到在改进一种性能指标之后保持其他性能的不变.以优化平均带宽为例,其中,基于布谷鸟哈希和基于客户端存储的优化策略都是通过在服务器存储部分数据来实现带宽提升,但是这同样会增加服务器的存储开销.因此,很多类似的优化方法都是通过针对目标问题,在不影响全局性能的情况下,尽可能地做出优化;
- 一种方案有时可以解决多个问题,不一定只局限于某一种特定的性能.以基于隐私信息检索的优化策略为例,它可以通过定位ORAM服务器中某一个具体的数据块来实现 $O(1)$的平均带宽;相应的,客户端

不需要很大的缓冲区来存储多余的数据块.这种方法在一定程度上实现了 ORAM 两个主要性能的提升,因此受到了广泛的关注.除了隐私信息检索,基于映射表的优化可以实现平均带宽和交互轮数的优化等;

- 不同方案可以叠加,在设计 ORAM 时,可以尝试利用多种优化策略,有时可以起到更好的效果.例如,树状模型在设计时就针对最坏情况下的带宽,因此其本身就具备一定的优化最坏情况带宽的特点.所有树状模型的变种也都继承了这个特征,但是例如 Onion ORAM 就在树状模型的基础上使用 3 节点的驱逐来进一步优化最坏情况的带宽.因此,从多个角度来优化可以更好地解决问题;
- ORAM 的性能优化设计往往是多种方案的权衡.在设计一种 ORAM 时需要全局考虑,考量每一种指标的重要性,因为每一种方案都有优缺点,一种性能提升,往往会导致其他性能的降低.例如在利用布谷鸟哈希优化平均带宽的时候,会导致服务器开销增加,可以利用针对服务器存储的优化策略,但是这也可能导致新的问题.同时,我们还要考虑不同方案叠加的效率问题.因此,要获得全局最优解,往往需要从多个角度共同考量.

4 结合云安全存储场景的 ORAM

结合云存储场景的ORAM安全存储系统设计,也是目前ORAM重点研究方向之一.利用传统的加密手段保护存储系统,只能够保护数据本身的机密性,利用 ORAM 设计的安全存储系统可以抵御攻击者通过访问模式推断用户的隐私信息.而这种存储系统应用于云场景,除了本身的性能问题之外,也会有很多实用性的问题.因此,这一类研究方向重点考虑如何解决 ORAM 安全存储系统在实用场景下带来的挑战,例如,如何保证数据在存储与访问过程中的完整性以及如何支持多用户并发访问的问题等.

4.1 ORAM的数据完整性保护

数据完整性保护是利用 ORAM 设计存储系统考虑的目标之一.对于恶意的云存储服务提供商来说,他可以通过随意地篡改数据、违反用户的执行协议来推测隐私信息.但是这会导致用户在访问过程中获得错误的数据,进而影响计算的准确性.因此,有一些工作通过在 ORAM 访问数据中添加少量的验证信息,来检测攻击者是否存在恶意篡改数据的行为.

一种经典的方法是对于存储在服务器中的每一个数据块都添加一个消息认证码(MAC),对于每一次访问,客户端通过随机检测从服务器中返回数据块中的 *MAC* 值是否正确:如果存在错误,就可以证实服务器存在恶意的数据篡改.同时,为了抵抗重放攻击,每一次客户端验证完 *MAC* 值后,会利用一个随机数重新计算 *MAC* 值,使得恶意的云存储提供商不能够自主地计算验证码,并且利用以往的返回信息来混淆用户.由于这种基于 *MAC* 值的保护数据完整性的方法有着相对少量的存储开销,并且容易被用户验证,因此被广泛使用[32].除了这些基本的利用 *MAC* 值保护访问完整性的方法之外,Williams 等人[31]也尝试在 ORAM 中的每一层添加一个利用抗冲突的增量哈希(incremental hashing)计算的校验和来避免恶意的服务器在混洗过程中隐藏或者复制数据块.通过增量哈希,用户不需要在每一次混洗之后重新计算哈希值,也便于用户维护服务器的存储信息.在 PrivateFS[33]中,Williams 等人利用 sha256 算法验证数据块的完整性.

部分完整性保护工作在ORAM设计的时候也考虑到了,例如在Partition ORAM[16]中,每一次混洗之后,利用了一个随机值来保证当前层的完整性,使得当前层在下一次混洗之前不能被攻击者篡改.Path ORAM[17]将整个树状 ORAM 看成是一棵 Merkle 树,其中每一个节点都存储一个哈希值 $H(b_1||b_2||\ldots||b_Z||h_1||h_2)$,其中,$b_i, i\in\{1,2,\ldots,Z\}$表示当前节点中的每一个数据块,$h_1,h_2$表示其子节点的哈希值.通过这种方法,也可以来验证数据完整性.

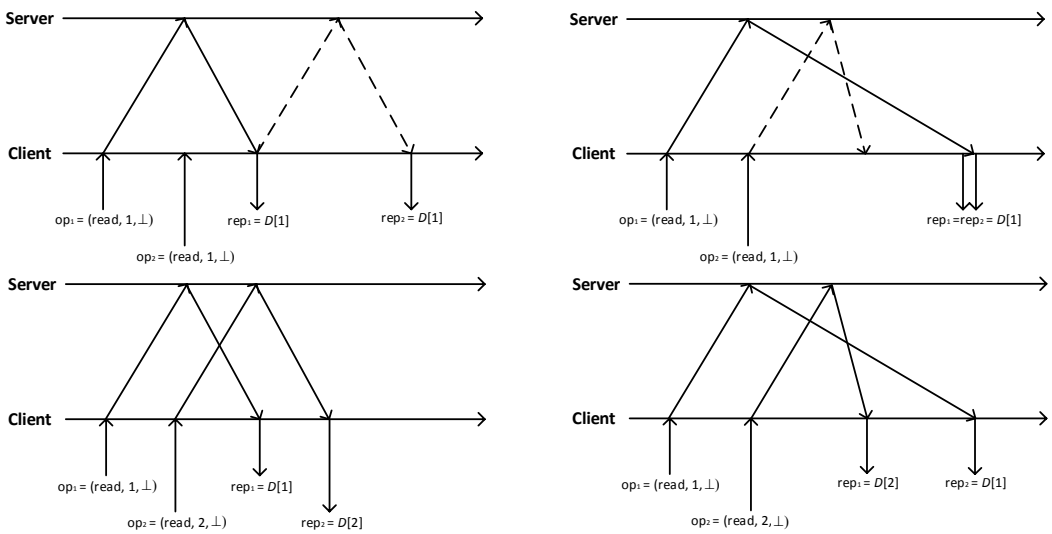
4.2 支持多用户的并发访问

绝大多数 ORAM 设计[13,16,17,25–28]通常只考虑一个客户端的场景,然而在云场景中,用户往往是很多个,因此在设计结合ORAM的安全存储系统的时候,需要考虑多用户并发访问的问题.尤其是在异步通信的网络中,这种问题通常很难处理,需要考虑的问题包括:(1) 数据同步问题,多用户访问同一个数据的时候数据更新会很复杂,

需要对每一个客户端的数据进行统一的管理,如果处理不当会导致数据丢失,甚至出现安全问题;(2) 当服务器在执行混洗或者驱逐操作的时候,数据块位置发生变化,此时如果客户端还保留着更新完的数据,需要添加一些额外的模块来保证驱逐完成以后数据能够准确地写回.

考虑多个客户端同时访问一个服务器的场景,为了保护每一个客户端访问数据时的访问模式,并且维护数据的一致性.最简单的一种思路是每一个客户端轮流访问,服务器端存储关于客户端的状态信息(state),以判定当前客户端能否执行访问操作.一个客户端必须等前一个客户端执行完之后才能继续访问.但是这种方法严重制约了访问的响应时间,并且不利于并发访问的设计.2012 年,Goodrich 等人[59]创新性地提出了一种无状态(stateless)ORAM,它利用伪随机 hash 函数以及布谷鸟哈希的方法实现了对数据库的不经意访问的仿真.无状态 ORAM 服务器端不需要保存任何关于客户端的状态信息,因此可以用来实现对多用户的并发访问.Williams 等人[33]利用无状态 ORAM 的构想,结合 Hierarchical ORAM 实现了一种不经意文件存储系统 PrivateFS.其中利用了一个日志模块(remote results log)来记录数据块的访问顺序以及数据块中的值.当用户需要请求一个数据块时,服务器先将这个日志模块发送给客户端,用来判定当前是否有相同的数据块被访问:如果有,就执行一个虚假访问(fake query),以保证数据访问的隐私性;如果没有,就执行真实访问(real query).但是对于 PrivateFS 而言,用户的并发访问并不是完全的并发,因为在用户获得访问结果、将其写入日志模块的时候,必须等之前的访问写完才能继续写入.因此,PrivateFS 依然存在访问性能的瓶颈.

为了解决 PrivateFS[33]中依然存在的线性访问的性能瓶颈,Stefanov 等人利用 Partition ORAM 实现了一种分布式安全存储系统 ObliviStore[20],并给出了分布式 ORAM 的完整定义.ObliviStore 首次将存储系统定义在异步网络(asynchronous network)中,使得任意用户可以在任意时间提交请求,并且不会泄露访问模式的隐私.同时,ObliviStore 还实现了以下几个优化策略:(1) 利用块混淆(batch shuffling)的策略提高了每一个分区内 ORAM 的混淆性能;(2) 为了避免并发访问以及延迟对客户端存储的影响,利用信号量对客户端存储进行定量的表示.但是 ObliviStore 并没有完全意义上实现并发访问的安全性.文献[34,35]都发现了:在 ObliviStore 中,如果一个客户端同时对一个数据块访问两次,与对不同数据块访问两次,服务器返回的时间不同,如图 5(a)所示(注:实线表示真实访问,虚线表示虚假访问).



(a) 在 ObliviStore 中对相同数据块访问的响应时间(上图)
对不同数据块访问的响应时间(下图)

(b) 在 CURIOUS 中对相同数据块访问的响应时间(上图)
对不同数据块访问的响应时间(下图)

Fig.5 Attacks against ObliviStore and CURIOUS[35]

图 5 对 ObliviStore 和 CURIOUS 的攻击场景[35]

这样会导致攻击者可以根据服务器的响应时间来判断客户端是否访问了同一个数据块,从而泄露了用户的隐私.这是因为 ObliviStore 对于处理同一数据块的访问请求采用了序列调度机制.为了解决这个问题,文献[34]提出了一种同样基于 Partition ORAM 的新的分布式文件存储系统 CURIOUS,取消了 ObliviStore 中的信号量,同时,对于一个客户端对同一数据块的连续两次访问请求,CURIOUS 只允许第一次访问请求执行真实访问,之后,所有请求都执行虚假访问,直到真实访问将数据写回.这样,ORAM 可以直接处理连续的异步请求,而不需要等待前一个访问结束.但是在异步通信中,真实访问有可能比虚假访问有更长的响应时间,对于 CURIOUS 来说,如果发生这种情况,在客户端获得目标数据块之前,虚假访问会等待真实访问结束后,再将数据返回给客户端.因此,文献[35]中证明,这种处理方式依然不安全.在真实的部署中,很难向攻击者隐藏响应时间的信息.如果响应时间暴露,攻击者依然可以推断出客户端是否访问同一个数据块的隐私信息,如图 5(b)所示.

Sahin 等人[35]针对 ObliviStore[20]和 CURIOUS[34]存在的攻击者对相同数据块的连续访问会依据响应时间推测是否访问同一数据块的攻击,提出了针对异步网络的新的安全性定义,称为 aaob-security(adaptive asynchronrous obliviousness),并基于 Path ORAM 实现了支持多用户并发访问的文件存储系统 TaoStore. TaoStore 提供了两个额外的模块:一个是处理器(processor),用来处理客户端与服务器之间的请求;另一个是序列器(sequencer),用来序列化服务器的请求,使请求结果的顺序与请求提交的顺序相一致.

4.3 小 结

为了更直观、明确地分析和对比不同的 ORAM 安全存储系统,我们从攻击者模型、基于的 ORAM 设计模型、完整性保护、是否支持多用户并发以及是否考虑异步网络等角度进行比较,见表 5.

Table 5 Comparisons of secure storage systems based on ORAM

表 5 ORAM 安全存储系统比较

ORAM 安全存储系统	攻击者模型	ORAM 设计模型	完整性保护	是否为分布式	是否支持多用户并发	是否考虑异步网络
Ref.[31]	半诚实或者恶意服务器	层次模型	MAC、增量哈希、随机值	×	×	×
Ref.[60]	半诚实服务器	层次模型	---	×	×	×
SR-ORAM[61]	恶意服务器	层次模型	MAC、增量哈希、随机值	×	×	×
PrivateFS[33]	半诚实或者恶意服务器	层次模型	MAC、哈希树	×	√	×
ObliviStore[20]	恶意服务器	分区模型	认证消息	√	√	√
Ref.[62]	半诚实或者恶意服务器	分区模型	校验和加密、可验证数据块	√	×	×
Burst ORAM[45]	恶意服务器	分区模型	认证消息	√	√	√
GORAM[29]	不可信服务器、恶意客户端	树状模型	访问控制、完整性证明	×	√	×
CURIOUS[34]	半诚实服务器	层次模型	–	×	√	√
Ref.[63]	不可信服务器	树状模型	–	×	√	×
TaoStore[35]	半诚实服务器	树状模型	–	×	√	√

综上所述,ORAM 在结合云场景下的安全存储系统研究取得了一定的发展,但是总体来说,与真正可以应用在云环境依然有很长的距离,主要体现在:

- 能够支持分布式的存储系统还很少.从 ORAM 本身的模型角度来说,只有分区模型是真正意义上的分布式系统,其他的模型都以单服务器为主,因此很难应用于目前的云环境.考虑用多个树状模型组合设计 ORAM 存储集群,是未来的研究方向之一;
- 效率依然很低,尤其是访问时间.以 ObliviStore[20]为例,对于一般的不需要保护访问模式的数据访问,完成一个访问任务需要 38.5s,而 ObliviStore 需要 181s;同时,ObliviStore 需要产生 17 倍的流量[34].因此在真实的云场景中,这种开销是非常巨大的;
- 满足真正安全的 ORAM 存储系统几乎没有.真正实现异步网络下多用户并发访问的安全 ORAM 存储系统只有 TaoStore[35],但是 TaoStore 也不是很完美,例如它是一个单服务器的存储系统,并不适用于云场景.那么是否存在多服务器下的异步网络访问的攻击场景,还有待进一步研究;
- 安全存储系统的功能还不够完善.存在的 ORAM 存储系统只是从完整性和安全性角度考虑,但是实际上,一个存储系统的功能远不止这些,例如数据备份等问题.

5 结合安全计算场景的 ORAM

不经意随机访问机还有一个重要的用途就是可以用于安全计算.因为 ORAM 来源于 RAM 模型,与图灵机(turing machine)、布尔电路(boolean circuit)和分支程序(branching programs)一样,RAM 是一种重要的可以模拟计算过程的模型[37],因此,ORAM 最初被用来保护程序的执行过程,不被恶意的攻击者逆向.利用 ORAM 应用于安全计算场景,通常是与安全多方计算(secure multi-party computation,简称 SMC)相结合.传统的实现安全多方计算的方式是构造电路,并在其上设计相应的协议,例如 Yao 协议[64]、GMW 协议[65]等.但是这种基于电路实现安全计算有一个局限性,电路大小与输入数据成正比[8].考虑一个场景:客户端想对服务器执行一次隐私信息检索 f,传统的方法是构造一个客户端与服务器之间查询的电路,客户端输入要查询哪一个数据的比特串 x,而服务器输入的是整个数据库 D,因此,设计一个隐私信息检索的电路是非常庞大的.但是利用 RAM 可以更高效地实现这种海量输入数据的计算,通过实验证实,利用 RAM 实现计算所需要的时间是与输入数据大小呈亚线性(sublinear)关系[8].因此,利用 RAM 来实现安全计算是一种重要的途径.

Ostrovsky 等人[36]指出:利用 ORAM 可以安全地执行任何 RAM 程序,其中的计算部分可以利用 Yao 的混淆电路(garbled circuit)[64]实现,对存储的访问部分可以利用基于 ORAM 的安全计算协议(SC-ORAM[66])来实现.并且,文献[36]也给出了从一般性的 ORAM 转化成 SC-ORAM 的方法,称为 Ostrovsky-Shoup 编译器[37].因此,如今绝大多数结合安全计算场景的 ORAM 研究都是将 SC-ORAM 与电路相结合,并且主要分为两类:结合 ORAM 的安全计算协议设计和不经意数据结构(oblivious data structure)的设计.

5.1 结合ORAM的安全计算协议设计

与通用 ORAM 性能提升的角度不同的是:在安全计算领域,ORAM 中的客户端与服务器通常指的是 CPU 和内存单元.而利用总线通信的 CPU 和内存之间的传输速度要比一般基于网络通信的客户端与服务器之间的速度快得多.因此在安全计算中,一般不以带宽作为衡量 ORAM 性能的指标.相反,服务器和客户端的计算复杂度在这其中起到了决定性的作用,这一部分的研究工作通常以优化协议中服务器与客户端的计算复杂度和电路复杂度作为目标,设计更为高效的 ORAM 安全计算协议.

5.1.1 基于优化计算复杂度的协议设计

Gordon 等人[8]最早将 ORAM 应用于安全两方计算,首次利用 Tree ORAM 实现了一种可以在亚线性时间内实现安全两方计算的协议.他们的工作主要基于两个发现:(1) 利用 RAM 实现的很多计算,其执行时间与输入数据的大小呈亚线性关系,这比电路要高效得多;(2) 如果计算方多次执行非平凡函数(non-trivial function)的计算,那么有可能实现其性能的进一步优化.文献[8]将 Tree ORAM[27]与 Yao 的混淆电路相结合,并利用了异或计算与不经意传输扩展(OT extension)等优化策略.假定一个利用 RAM 实现的函数 f 可以在时间大小为 t 和空间大小为 s 中被执行,那么利用文献[8]中的方案将 f 转化成一个客户端与服务器之间的安全两方计算协议,执行的平均时间为 $O(t)\cdot poly\log(s)$,客户端需要 $O(\log s)$的空间,服务器需要 $s\cdot poly\log(s)$的空间.

Lu 等人[37]利用 Hierarchical ORAM[13]设计了一种可以用于安全两方计算的 ORAM 计算协议.他们将 Hierarchical ORAM 按层次的不同划分在两台服务器上,并且在协议的执行过程中两台服务器始终没有交互.由于 Hierarchical ORAM 性能低效的原因在于复杂的混洗操作,文献[37]利用对数据块的标记(tagging),客户端每一次访问之前都需要通过计算目标数据块的标记获得访问地址,而不用依赖于数据块本身.同时,对数据块的标记可以简单地利用伪随机函数进行维护,不需要利用针对所有数据块的混洗操作.如果一个用 RAM 实现的函数 f,其输入与函数本身大小为 Λ,函数运行时间为 T,那么应用文献[37]中方案将 f 转化成一个安全两方计算协议,那么协议的通信和计算复杂度为 $O((\Lambda+T)\log(\Lambda+T))$.如果允许对数据集执行预处理的话,那么在线的通信和计算复杂度可以降为 $O((\varepsilon+T)\log(\Lambda+T))$,其中,$\varepsilon$表示在线输入的大小.

第 5.1.2 节中的文献[8]和文献[9]都提出了在 SC-ORAM 中,当数据的总量 N 小于一个阈值时,利用线性搜索所需要的时间要比设计更复杂 ORAM 的时间更短.Zahur 等人[21]利用这个特征分析了通用 ORAM 研究与基于安全计算 ORAM 研究的区别,同时结合了 Square-root ORAM[13],进一步降低了阈值.在传统 Square-root

ORAM 设计中,客户端利用一个 hash 函数来计算映射表,文献[21]将其替换为利用共享数组直接存储.这样的好处在于:当初始化以及混洗的时候,只需要利用简单的混洗网络,而不用对数据块执行复杂的不经意排序.同时,文献[21]将 Square-root ORAM 结合迭代技术,降低了客户端的存储.

Doerner 等人[67]发现,文献[8,21,68]的设计其实都有缺陷,例如都使用了迭代构造的方法,这种方法虽然可以优化客户端存储的大小,但是会增大通信轮数,同时,他们的方法都有着很高的存储开销,都需要存储每一比特数据的线路标记(wire label);其次,这些 ORAM 的初始化也需要很大的开销.针对以上 ORAM 存在的问题,Doerner 等人提出一种新的 ORAM 设计,称为 flORAM.与传统 SC-ORAM 设计不同的是:flORAM 将读写隔离,设计成只写存储(write-only memory,简称 WOM)和只读存储(read-only memory,简称 ROM)两部分,并将它们看成是分布式 ORAM 的两个服务器,加快了读取数据的速度;其次,flORAM 通过存储数据的密文来代替传统存储线路标记,降低了存储开销;同时,flORAM 创新性地利用基于函数秘密共享(function secret sharing)[40]的隐私信息检索,实现了对 WOM 数据写入和对 ROM 的数据读取不暴露隐私信息.

考虑到文献[37]中每一次访问都需要 $O(\log N)$轮次的交互,同时,基于层次模型需要周期性的混洗操作,Wang 等人[69]结合 Tree ORAM[27]与隐私路径检索技术提出了一种基于两服务器的高效 ORAM 计算协议.在协议中,客户端通过两服务器的隐私信息检索来读取目标数据块所在的路径.这样的优势在于避免了传统 Tree ORAM 每一次访问之后更新映射表,进而避免了迭代构造,并使得每一次访问只需要与服务器交互一次.同时,Wang 等人对隐私路径检索进行了优化,减少并行读取路径中数据块的次数.在文献[69]中,Wang 等人还创新性地提出了在 ORAM 初始化时随机生成映射表的方法,降低了安全计算中 ORAM 初始化的开销.

5.1.2 基于优化电路复杂度的协议设计

与计算复杂度相类似,由于基于 ORAM 的安全计算协议在实现中需要编译成电路,所有的计算过程将以电路形式表达.一种直观的想法就是利用电路的复杂程度来衡量协议的计算性能.从这个角度出发,wang 等人[68]创新性地将电路复杂度(circuit complexity)引入基于 ORAM 的安全计算协议设计的评估中,通过计算不同 ORAM 在实现后电路中的与门电路的数量,来比较 ORAM 的性能.

Wang 等人还发现,文献[37]的方案存在两个缺点:(1) 由于使用层次模型,当查找一个数据块时,需要迭代地计算每一层的 hash 函数以确定数据块的位置,这会导致安全协议不能高效地执行;(2) 为了解决层次模型中数据块哈希值冲突的问题,文献[37]也给出了应用布谷鸟哈希[26]的方法,但是基于布谷鸟哈希的层次模型需要更大的存储空间.针对这两个问题,Wang 等人[68]提出了一种基于 Path ORAM[17]设计的适用于安全计算的 ORAM,称为 Path SC-ORAM.其中应用了 3 个不经意排序,使 Path ORAM 的电路复杂度从理论上性能降低了 $O(\log n)$的数量级,同时还提出了一种启发式的最高效 ORAM 的设计方式.由于 Path ORAM 中最复杂的部分在于驱逐操作,文献[37]中应用不经意排序将驱逐部分的电路优化至 $O(D\log N\log\log N)\omega(1)$大小.因为有更大的常数级开销,这在实用性上依然低于传统的 $O(D\log^2 N)$大小的电路.

为了避免设计复杂的驱逐电路、优化整体的 ORAM 性能,Wang 等人[9]进一步优化了在 Path ORAM 上的电路复杂度,设计出了 Circuit ORAM.这种 ORAM 将每一次查找数据块的路径与驱逐路径合并.为了达到这个目标,在原有数据块的基础上增添了两种元数据,使得一次查找就可以提前知道数据块驱逐所需要的信息,包括数据块放置的位置等.通过这种方法,Circuit ORAM 的性能比文献[37]中启发式渐进最优的 ORAM 性能提升了 5.1 倍.

5.2 不经意数据结构的设计

在结合 ORAM 的安全计算领域,还有一类研究方向是设计不经意数据结构.与第 5.2 节研究的角度不同的是,这一个方向主要是从实现角度设计或者优化 ORAM 性能,而之前的方向是从算法、协议层进行设计优化.不同的数据结构的设计通常需要结合具体的使用场景,例如在图搜索计算,还是共享数据存储等.在多方计算的 SC-ORAM 的实现中,通常需要客户端与服务器的状态进行共享,因此,设计和利用不经意数据结构存储数据可以保证访问的隐私性和高效性.

基于文献[36]中 ORAM 应用于安全计算的思想与文献[8]中利用 ORAM 构造平均亚线性时间下的安全两

方计算协议,Keller 等人[38]首次设计了多种在多方计算中常用的数据结构,并利用不经意技术进行实现,其中包括数组、字典、优先队列等.利用文献[38]中的优先队列实现 Dijkstra 算法可以在 $O(|E|\log^5|E|+|V|\log^4|V|)$的时间内实现单源点最短路径的计算,并且可以保证源点与汇点的隐私性,其中,E 和 V 分别表示图中边和点的数量.同时,文献[38]还证明了利用提出的不经意数据结构对结合安全计算的 ORAM 进行块初始化(batch initialization),要比简单地按序初始化提升 $O(\log^3 N)$的数量级.

Wang 等人[39]在文献[38]的基础上一般化不经意数据结构的设计原则,将其归纳为两点:一种是基于指针(pointer)的技术,另一种是基于位置(locality)的技术.基于指针的技术通常用于树状 ORAM 构建中,通过父节点存储一个指向子节点的指针,可以在查找父节点到子节点的路径中直接获取子节点的位置,避免重复地查找映射表,这样可以降低 $O(\log n)$数量级的复杂度.基于位置的技术通常应用于有着二倍维度(doubling dimension)的访问模式图中.图中每一个点被划分至一个簇(cluster)中.每一个簇包含 $O(\log N)$个点,通过图中每一个点对其他簇的访问,可以追踪整个访问过程.每一个簇可以像 ORAM 的数据块一样被访问,这样,当需要读取或者更新点的信息时候,先将点所在的簇与其临近的所有簇读入缓冲区,对于其临近的簇执行虚假访问,目标点所在的簇执行真实访问,这样保证了访问的隐私性.

当 ORAM 直接用于密文数据库系统时,存在很大的局限性.Hoang 等人[70]对存在的局限性进行分析.

1) 对于面向行的数据库,在利用 ORAM 进行改进时,会将数据库中的每一行作为 ORAM 中的数据块,例如 SEAL-ORAM[71].但是这种方案在执行插入、删除或者更新时,需要移动 ORAM 中的所有数据块;同时,如果在数据库中执行类似统计或者条件查询等列相关操作时,需要下载 ORAM 中的所有数据块,这将导致很大的带宽问题;
2) 另一种方法是将数据库中的每一个数据单元作为 ORAM 中的数据块,但是这会增加映射表的大小.Hoang 等人[70]分析:可以利用文献[39]中的思想,将 $O(\log N)$个数据单元存入一个 ORAM 的数据块中来避免映射表的问题.但是这会增加按行或列查询所有数据单元的请求次数,不适用于大型数据库.

针对这两个局限性,Hoang 等人[70]提出了两种用于密文数据库系统的不经意数据结构:oblivious matrix structure(OMAT)与 oblivious tree structure(OTREE).前者为数据库中每一个表构造不经意的矩阵结构,可以支持多种查询操作而不需要下载整个 ORAM 数据库;后者为基于树的数据库实例提供不经意访问操作.对于不经意条件查询,利用文献[70]中的 OTREE 要比利用文献[39]中的方案更高效.

5.3 小 结

结合安全计算场景的 ORAM 与传统意义上客户端-服务器 ORAM 有较大的不同,具体总结为以下几点.

(1) 保护对象不同:在安全计算中,存储的数据由多方共享,任意一方的访问模式都不能泄露,而一般 ORAM 中,访问模式的保护只针对客户端,是单向的;
(2) 实现方式不同:在研究结合安全计算 ORAM 的协议设计时,需要考虑初始化过程的复杂度,这一部分往往是通过协议实现的.而一般意义上的 ORAM 数据库初始化是默认存在的;
(3) 优化策略不同:传统研究的 ORAM 主要针对客户端与服务器的带宽以及客户端本地的存储,而结合安全计算场景的 ORAM 更多地考虑计算的复杂度.

通过分析总结上述的研究工作,并结合以上与传统 ORAM 研究的不同,我们分析总结了结合安全计算 ORAM 协议目前的研究方向.

- 以优化协议性能为主要目标.与通用的 ORAM 性能优化一样,结合安全计算场景的 ORAM 也存在着复杂度太高的问题,但是优化角度不同.这里更倾向于降低 ORAM 在安全计算上的实现复杂度;
- 推广 ORAM 的使用以实现在更多场景下的安全计算.由于 ORAM 本身扎实的安全理论基础及其广泛的适用性,将 ORAM 迁移到其他安全领域实现多种类的安全计算也是研究热点之一,例如图计算、社交网络挖掘等.

6 不经意随机访问机研究展望

由于不经意随机访问机本身低效性的特点,导致其应用于实际的问题更加突出,例如设计安全存储系统与安全计算协议等.所以,这项技术目前的研究重点依然在于性能的提升.同时还有一个不容忽视的挑战就是不经意随机访问机应用于实际所产生的额外问题,例如在安全存储场景中的数据备份问题等.所以,未来不经意随机访问机研究需要兼顾性能、功能、应用等多方面的因素,具体表现为:

- 性能的进一步提升.不只局限于传统的讨论客户端与服务器之间的带宽以及客户端本地的存储问题,还会涉及进一步优化服务器的存储、交互轮数等其他之前很少考虑的指标.在结合安全存储场景下,也会着重考虑多用户并发访问下系统的响应时间;在安全计算场景下的计算复杂度等.因此,这一类性能优化问题将一直是 ORAM 研究的主流,这也推动 ORAM 本身不断的发展;
- 功能的进一步完善.未来的 ORAM 将会支持更多的操作,不只支持简单的读写访问,甚至可以在其上完成较为复杂的数据结构操作,例如,可以用读写进行组合实现查询删除等[27].其他场景下的功能也可以利用不经意数据结构实现(见第 5.2 节).因此,这一类方向将促进 ORAM 应用于实际;
- 实用型 ORAM 应用系统的出现.对于存储系统而言,将不仅仅像 ObliviStore[20]和 TaoStore[35]那样的原型系统,真正可以应用于存储海量数据,支持多用户并发访问以及快速响应的系统会逐步推出.同时,对于安全计算,也会出现基于 ORAM 的安全计算系统,或者是安全计算模块,可以兼容目前大多数分布式计算系统,例如 MapReduce,Spark 等.这使得云计算与大数据的技术与安全计算领域相结合.

我们认为,未来不经意随机访问机研究应该重点包含以下几个方面.

(1) 基于隐私信息检索技术的多指标性能优化的 ORAM 设计

隐私信息检索技术可以用来优化 ORAM 的平均带宽(见第 3.1.2 节)和客户端存储(见第 3.3.2 节),相比于一般的优化方案,隐私信息检索具有两种优点:(1) 可以同时解决通用 ORAM 性能提升中的两个重要指标,这是其他优化策略所不具有的;(2) 这种技术的缺陷很少,因此今年来受到广泛的关注.以隐私信息检索技术为出发点,在保证平均带宽和客户端存储最优的情况下优化别的性能指标将是未来性能提升的重点.从第 2 节的表 2 可以看出,如今在性能优化上平均带宽和客户端存储这两种性能已经达到最优的 $O(1)$.但是其他的指标依然存在相对的劣势,例如数据块的大小、交互轮数等.能否在保证前面两种性能依然是最优的基础上优化其他的性能指标,是一个非常重要的问题,继续以隐私信息检索技术为研究对象是未来的一种重要研究趋势.

(2) 不经意并行随机访问机(OPRAM)的设计与实现

OPRAM 最初是由 Boyle 等人提出[72],用来解决多主体对 ORAM 的并行访问问题.与第 4 节中多用户并发访问存储系统不同的是,OPRAM 是从模型层实现,不参考任何应用场景;而第 4 节是 ORAM 结合安全存储场景设计系统,需要添加一系列的模块才能从应用层实现.可以将 OPRAM 理解为支持多用户并发访问的 ORAM 在模型层的抽象,因此,OPRAM 将更适用于云平台以及多核处理器结构.针对 OPRAM 的设计依然以性能改进作为驱动力.其衡量指标与传统 ORAM 类似,主要是针对通信带宽以及客户端的存储.当 OPRAM 用于多核计算上时,也需要考虑计算复杂度的问题.因此,OPRAM 作为 ORAM 在并行性上的扩展,在保留 ORAM 所有的能力之外,也针对一些特殊的应用场景添加额外的功能.这一类设计将 ORAM 推向更广的应用,也是未来 ORAM 研究的重要途径之一.

(3) 基于 ORAM 的安全计算系统的设计与实现

结合 ORAM 的安全计算(见第 5 节)依然以优化计算协议的复杂度作为目前的主流工作.相比于安全多方计算,这一类工作在保证计算协议的安全性上还保证了每一个计算方访问数据的隐私性.之前已经有一些工作实现了基于安全多方计算的安全计算系统,例如基于电路技术的安全两方计算系统 FairPlay[73]、多方计算系统 FairPlayMP[74]、基于秘密共享技术的多方计算系统 Sharemind[75]等.但是目前,结合 ORAM 的安全计算依然只停留于协议的设计,还没有出现基于 ORAM 的安全计算系统.因此,未来的一个重要研究方向是将结合 ORAM 的安全计算应用于实际,设计出实用型的系统.

同时,针对已有计算系统的改进也是未来的一种研究方向.例如设计实现基于 MapReduce,Spark 的安全计

算模块,其上使用 ORAM 来保护计算节点对文件系统或者是内存单元的读写访问.也可以基于流式计算系统 Storm 或者是 Spark Streaming,在实现快速的流水计算同时,保护对内存的访问模式,这种场景对 ORAM 的响应时间要求会很高.因此,未来将 ORAM 结合系统来实现是研究的一种重要的趋势.

7 结束语

不经意随机访问机是当前保护访问模式隐私的重要手段,其研究受到了学术界的广泛关注,近几年也有较大的进展,主要集中于模型在平均带宽与客户端存储上的性能优化以及与安全存储、安全计算领域相结合的应用.但是从目前来看,ORAM 在实用性上依然面临严峻的挑战,未来还有很多的工作值得去做.

本文对不经意随机访问机的研究进行综述,包括 ORAM 的相关概念、设计方法以及优化模型性能的常见策略及其优劣性,讨论了将 ORAM 用于安全存储系统设计以及安全计算领域的一般性问题,并对未来不经意随机访问机的依然存在的问题、挑战与趋势进行了分析.期望通过我们的工作,能给以后的研究者提供有益的借鉴与参考,为不经意随机访问机的进一步发展做出贡献.

致谢 在此,我们向对本文的工作给予支持和宝贵建议的评审老师和同行表示衷心的感谢!

References:

[1] Wu X, Xu L, Zhang X. Poster: A certificateless proxy re-encryption scheme for cloud-based data sharing. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 869−872. [doi: 10.1145/2046707.2093514]

[2] Jung T, Li XY, Wan Z, Wan M. Control cloud data access privilege and anonymity with fully anonymous attribute-based encryption. IEEE Trans. on Information Forensics and Security, 2015,10(1):190−199. [doi: 10.1109/tifs.2014.2368352]

[3] Pinkas B, Reinman T. Oblivious ram revisited. In: Proc. of the 30th Annual Cryptology Conf., Vol.6223. Berlin, Heidelberg: Springer-Verlag, 2010. 502−519. [doi: 10.1007/978-3-642-14623-7_27]

[4] Islam MS, Kuzu M, Kantarcioglu M. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In: Proc. of the 18th Network and Distributed System Security Symp., Vol.20. 2012. 1−15. [doi: 10.1.1.673.8809]

[5] Dautrich Jr JL, Ravishankar CV. Compromising privacy in precise query protocols. In: Proc. of the 16th Int'l Conf. on Extending Database Technology. ACM Press, 2013. 155−166. [doi: 10.1145/2452376.2452397]

[6] Zheng W, Dave A, Beekman JG, *et al*. Opaque: An oblivious and encrypted distributed analytics platform. In: Proc. of the 14th USENIX Symp. on Networked Systems Design and Implementation. USENIX Association, 2017. 283−298.

[7] Shaon F, Kantarcioglu M, *et al*. SGX-BigMatrix: A practical encrypted data analytic framework with trusted processors. In: Proc. of the 24st ACM Conf. on Computer and Communications Security. ACM Press, 2017. 1211−1228. [doi: 10.1145/3133956.3134095]

[8] Gordon SD, Katz J, Kolesnikov V, *et al*. Secure two-party computation in sublinear (amortized) time. In: Proc. of the 19th ACM Conf. on Computer and Communications Security. ACM Press, 2012. 513−524. [doi: 10.1145/2382196.2382251]

[9] Wang X, Chan H, Shi E. Circuit ORAM: On tightness of the goldreich-ostrovsky lower bound. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. ACM Press, 2015. 850−861. [doi: 10.1145/2810103.2813634]

[10] Oblivious RAM. https://en.wikipedia.org/wiki/Oblivious_ram

[11] Ohrimenko O, Costa M, Fournet C, *et al*. Observing and preventing leakage in MapReduce. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. ACM Press, 2015. 1570−1581. [doi: 10.1145/2810103.2813695]

[12] Dinh TTA, Saxena P, Chang EC, *et al*. M^2R: Enabling stronger privacy in MapReduce computation. In: Proc. of the 24th Symp. on USENIX Security. USENIX Association, 2015. 447−462.

[13] Goldreich O, Ostrovsky R. Software protection and simulation on oblivious RAMs. Journal of the ACM (JACM), 1996,43(3): 431−473. [doi: 10.1145/233551.233553]

[14] Kushilevitz E, Lu S, Ostrovsky R. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In: Proc. of the 23rd Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2012. 143−156. [doi: 10.1137/1.9781611973099.13]

[15] Gentry C, Halevi S, Jutla C, *et al*. Private database access with he-over-oram architecture. In: Proc. of the 13th Int'l Conf. on Applied Cryptography and Network Security. Springer-Verlag, 2015. 172−191. [doi: /10.1007/978-3-319-28166-7_9]

[16] Stefanov E, Shi E, Song D. Towards practical oblivious RAM. In: Proc. of the 17th Network and Distributed System Security Symp. 2011.

[17] Stefanov E, Dijk MV, Shi E, *et al*. Path ORAM: An extremely simple oblivious RAM protocol. In: Proc. of the 20th ACM Conf. on Computer and Communications Security. ACM Press, 2013. 299−310. [doi: 10.1145/2508859.2516660]

[18] Garg S, Mohassel P, Papamanthou C. TWORAM: Round-optimal oblivious RAM with applications to searchable encryption. IACR Cryptology ePrint Archive, 2015. 1010.

[19] Moataz T, Mayberry T, Blass EO. Constant communication ORAM with small blocksize. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. ACM Press, 2015. 862−873. [doi: 10.1145/2810103.2813701]

[20] Stefanov E, Shi E. Oblivistore: High performance oblivious cloud storage. In: Proc. of the 34th IEEE Symp. on Security and Privacy. IEEE, 2013. 253−267. [doi: 10.1109/SP.2013.25]

[21] Zahur S, Wang X, Raykova M, *et al*. Revisiting square-root ORAM: Efficient random access in multi-party computation. In: Proc. of the 37th IEEE Symp. on Security and Privacy. IEEE, 2016. 218−234. [doi: 10.1109/SP.2016.21]

[22] Batcher KE. Sorting networks and their applications. In: Proc. of the Spring Joint Computer Conf. ACM Press, 1968. 307−314. [doi: 10.1145/1468075.1468121]

[23] Ajtai M, Komlós J, Szemerédi E. An $O(n\log n)$ sorting network. In: Proc. of the 15th Annual ACM Symp. on Theory of Computing. ACM Press, 1983. 1−9. [doi: 10.1145/800061.808726]

[24] Goodrich MT. Randomized shellsort: A simple oblivious sorting algorithm. In: Proc. of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010. 1262−1277. [doi: 10.1137/1.978161197 3075.101]

[25] Ren L, Fletcher CW, Kwon A, *et al*. Constants count: Practical improvements to oblivious RAM. In: Proc. of the 24th USENIX Conf. on Security Symp. USENIX Association, 2015. 415−430.

[26] Devadas S, Dijk MV, Fletcher CW, *et al*. Onion ORAM: A constant bandwidth blowup oblivious RAM. In: Proc. of the 13th Theory of Cryptography Conf. Springer-Verlag, 2016. 145−174. [doi: 10.1007/978-3-662-49099-0_6]

[27] Shi E, Chan THH, Stefanov E, *et al*. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In: Proc. of the 17th Int'l Conf. on Theory and Application of Cryptology and Information Security, Vol.7073. 2011. 197−214. [doi: 10.1007/978-3-642- 25385-0_11]

[28] Hoang T, Ozkaptan CD, Yavuz AA, *et al*. S3ORAM: A computation-efficient and constant client bandwidth blowup ORAM with shamir secret sharing. In: Proc. of the 24th ACM Conf. on Computer and Communications Security. ACM Press, 2017. 491−505. [doi: 10.1145/3133956.3134090]

[29] Maffei M, Malavolta G, Reinert M, *et al*. Privacy and access control for outsourced personal records. In: Proc. of the 36th IEEE Symp. on Security and Privacy. IEEE, 2015. 341−358. [doi: 10.1109/sp.2015.28]

[30] Maffei M, Malavolta G, Reinert M, *et al*. Maliciously secure multi-client ORAM. IACR Cryptology ePrint Archive, 2017. 329. [doi: 10.1007/978-3-319-61204-1_32]

[31] Williams P, Sion R, Carbunar B. Building castles out of mud: Practical access pattern privacy and correctness on untrusted storage. In: Proc. of the 15th ACM Conf. on Computer and Communications Security. ACM Press, 2008. 139−148. [doi: 10.1145/1455770. 1455790]

[32] Williams P, Sion R. Access privacy and correctness on untrusted storage. ACM Trans. on Information and System Security, 2013, 16(3):12. [doi: 10.1145/2535524]

[33] Williams P, Sion R, Tomescu A. Privatefs: A parallel oblivious file system. In: Proc. of the 19th ACM Conf. on Computer and Communications Security. ACM Press, 2012. 977−988. [doi: 10.1145/2382196.2382299]

[34] Bindschaedler V, Naveed M, Pan X, *et al*. Practicing oblivious access on cloud storage: The gap, the fallacy, and the new way forward. In: Proc. of the 22nd ACM Conf. on Computer and Communications Security. ACM Press, 2015. 837−849. [doi: 10.1145/ 2810103.2813649]

[35] Sahin C, Zakhary V, Abbadi ElA, *et al*. Taostore: Overcoming asynchronicity in oblivious data storage. In: Proc. of the 37th IEEE Symp. on Security and Privacy. IEEE, 2016. 198−217. [doi: 10.1109/SP.2016.20]

[36] Ostrovsky R, Shoup V. Private information storage. In: Proc. of the 29th Annual ACM Symp. on Theory of Computing. ACM Press, 1997. 294−303.

[37] Lu S, Ostrovsky R. Distributed oblivious RAM for secure two-party computation. In: Proc. of the 10th Conf. on Theory of Cryptography. Springer-Verlag, 2013. 377−396. [doi: 10.1007/978-3-64 2-36594-2_22]

[38] Keller M, Scholl P. Efficient, oblivious data structures for MPC. In: Proc. of the 21st Int'l Conf. on Theory and Application of Cryptology and Information Security. Springer-Verlag, 2014. 506−525. [doi: 10.1007/978-3-662-45608-8_27]

[39] Wang XS, Nayak K, Liu C, *et al*. Oblivious data structures. In: Proc. of the 21st ACM Conf. on Computer and Communications Security. ACM Press, 2014. 215−226. [doi: 10.1145/2660267.2660314]

[40] Boyle E, Gilboa N, Ishai Y. Function secret sharing. LNCS, 2015,9057:337−367. [doi: 10.1007/978-3-662-46803-6_12]

[41] Maas M, Love E, Stefanov E, *et al*. Phantom: Practical oblivious computation in a secure processor. In: Proc. of the 20th ACM Conf. on Computer and Communications Security. ACM Press, 2013. 311−324. [doi: 10.1145/2508859.2516692]

[42] Goodrich MT, Mitzenmacher M. Privacy-Preserving access of outsourced data via oblivious RAM simulation. In: Proc. of the 38th Int'l Colloquium on Automata, Languages, and Programming. Springer-Verlag, 2011. 576−587. [doi: 10.1007/978-3-642-22012-8_46]

[43] Pagh R, Rodler FF. Cuckoo hashing. In: Proc. of the 9th Annual European Symp. on Algorithms, Vol.2161. Springer-Verlag, 2001. 121−133. [doi: 10.1007/3-540-44676-1_10]

[44] Moataz T, Blass EO, Mayberry T. CHf-ORAM: A constant communication ORAM without homomorphic encryption. Technical Report, 2015/1116, Cryptology ePrint Archive, 2015.

[45] Dautrich J, Stefanov E, Shi E. Burst ORAM: Minimizing ORAM response times for bursty access patterns. In: Proc. of the 23rd Symp. on USENIX Security. USENIX Association, 2014. 749−764. [doi: 10.1007/BF02483924]

[46] Gentry C, Goldman KA, Halevi S, *et al*. Optimizing ORAM and using it efficiently for secure computation. In: Proc. of the 13th Int'l Symp. on Privacy Enhancing Technologies, Vol.7981. Springer-Verlag, 2013. 1−18. [doi: 10.1007/ 978-3-642-39077-7_1]

[47] Mayberry T, Blass EO, Chan AH. Efficient private file retrieval by combining ORAM and PIR. In: Proc. of the 20th Network and Distributed System Security Symp. 2014. [doi: 10.14722/ndss.2014.23033]

[48] Dautrich J, Ravishankar C. Combining ORAM with PIR to minimize bandwidth costs. In: Proc. of the 5th ACM Conf. on Data and Application Security and Privacy. ACM Press, 2015. 289−296. [doi: 10.1145/2699026.2699117]

[49] Apon D, Katz J, Shi E, *et al*. Verifiable oblivious storage. In: Proc. of the 17th Int'l Conf. on Practice and Theory in Public-Key Cryptography. Springer-Verlag, 2014. 131−148. [doi: 10.1007/978-3-642-54631-0_8]

[50] Damgard I, Jurik M. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Proc. of the 4th Int'l Workshop on Practice and Theory in Public Key Cryptography, Vol.1992. Springer-Verlag, 2001. 119−136. [doi: 10.1007/ 3-540-44586-2_9]

[51] Goldberg I. Improving the robustness of private information retrieval. In: Proc. of the 28th IEEE Symp. on Security and Privacy. IEEE, 2007. 131−148. [doi: 10.1109/SP.2007.23]

[52] Abraham I, Fletcher CW, Nayak K, *et al*. Asymptotically tight bounds for composing ORAM with PIR. In: Proc. of the 20th Int'l Conf. on Practice and Theory in Public-Key Cryptography. Springer-Verlag, 2017. 91−120. [doi: 10.1007/978-3-662-54365-8_5]

[53] Boneh D, Mazieres D, Popa RA. Remote oblivious storage: Making oblivious RAM practical. Manuscript, 2011. http://dspace.mit.edu/bitstream/handle/1721.1/62006/MIT-CSAIL-TR-2011-018.pdf

[54] Fletcher CW, Naveed M, Ren L, *et al*. Bucket ORAM: Single online roundtrip, constant bandwidth oblivious RAM. IACR Cryptology ePrint Archive, 2015. 1065.

[55] Chung KM, Pass R. A simple ORAM. IACR Cryptology ePrint Archive, 2013. 243.

[56] Chung KM, Liu Z, Pass R. Statistically-Secure ORAM with $O(\log^2 n)$ overhead. In: Proc. of the 20th Int'l Conf. on Theory and Application of Cryptology and Information Security. Springer-Verlag, 2014. 62−81. [doi: 10.1007/978-3- 662-45608-8_4]

[57] Sanchez AM. Toward efficient data access privacy in the cloud. IEEE Communications Magazine, 2013,51(11):39−45. [doi: 10.1109/MCOM.2013.6658650]

[58] Moataz T, Mayberry T, Blass EO, *et al*. Resizable tree-based oblivious RAM. In: Proc. of the 19th Int'l Conf. on Financial Cryptography and Data Security. Springer-Verlag, 2015. 147−167. [doi: 10.1007/978-3-662-47854-7_9]

[59] Goodrich MT, Mitzenmacher M, Ohrimenko O, *et al*. Privacy-Preserving group data access via stateless oblivious RAM simulation. In: Proc. of the 23rd Annual ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2012, 13(Suppl 1):157−167. [doi: 10.1137/1.9781611973099.14]

[60] Williams P, Sion R, Sotakova M. Practical oblivious outsourced storage. ACM Trans. on Information and System Security, 2011, 14(2):20. [doi: 10.1145/2019599.2019605]

[61] Williams P, Sion R. Single round access privacy on outsourced storage. In: Proc. of the 19th ACM Conf. on Computer and Communications Security. ACM Press, 2012. 293−304. [doi: 10.1145/2382196.2382229]

[62] Stefanov E, Shi E. Multi-Cloud oblivious storage. In: Proc. of the 33rd ACM Conf. on Computer and Communications Security. ACM Press, 2013. 247−258. [doi: 10.1145/2508859.2516673]

[63] Sun XN, Jiang H, Xu QL. Multi-User binary tree based ORAM scheme. Ruan Jian Xue Bao/Journal of Software, 2016,27(6): 1475−1486 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5002.htm [doi: 10.13328/j.cnki.jos.005002]

[64] Yao AC. Protocols for secure computations. In: Proc. of the 23rd Annual Symp. on Foundations of Computer Science. IEEE, 1982. 160−164. [doi: 10.1109/SFCS.1982.88]

[65] Goldwasser S, Micali S, Wigderson A. How to play any mental game, or a completeness theorem for protocols with an honest majority. In: Proc. of the 19th Annual ACM Symp. on Theory of Computing, Vol.87. ACM Press, 1987. 218−229. [doi: 10.1145/28395.28420]

[66] Faber S, Jarecki S, Kentros S, *et al*. Three-Party ORAM for secure computation. In: Proc. of the 21st Int'l Conf. on Theory and Application of Cryptology and Information Security. Springer-Verlag, 2015. 360−385. [doi: 10.1007/978-3- 662-48797-6_16]

[67] Doerner J. Scaling ORAM for secure computation. In: Proc. of the 24th ACM Conf. on Computer and Communications Security. ACM Press, 2017. 523−535. [doi: 10.1145/3133956.3133967]

[68] Wang XS, Huang Y, Chan THH, *et al*. SCORAM: Oblivious RAM for secure computation. In: Proc. of the 21st ACM Conf. on Computer and Communications Security. ACM Press, 2014. 191−202. [doi: 10.1145/2660267.2660365]

[69] Xiao W, Dov G, Jonathan K. Simple and efficient two-server ORAM. IACR Cryptology ePrint Archive, 2018. 005.

[70] Thang H, Ceyhun DO, Gabriel H, Attila AY. Efficient oblivious data structures for database services on the cloud. IACR Cryptology ePrint Archive, 2017. 1238.

[71] Zhao C, Dong X, Li F. Oblivious ram: A dissection and experimental evaluation. In: Proc. of the 42nd Int'l Conf. on Very Large Databases. ACM Press, 2016. 1113−1124. [doi: 10.14778/2994509.2994528]

[72] Boyle E, Chung KM, Pass R. Oblivious parallel RAM and applications. In: Proc. of the 13th Theory of Cryptography Conf. Springer-Verlag, 2016. 175−204. [doi: 10.1007/978-3-662-49099-0_7]

[73] Malkhi D, Nisan N, Pinkas B, *et al*. Fairplay—A secure two-party computation system. In: Proc. of the 13th Symp. on USENIX Security. USENIX Association, 2004. 20−20.

[74] Ben-David A, Nisan N, Pinkas B. FairplayMP: A system for secure multi-party computation. In: Proc. of the 15th ACM Conf. on Computer and Communications Security. ACM Press, 2008. 257−266. [doi: 10.1145/1455770.1455804]

[75] Bogdanov D, Laur S, Willemson J. Sharemind: A framework for fast privacy-preserving computations. In: Proc. of the 13th European Symp. on Research in Computer Security, Vol.5283. Springer-Verlag, 2008. 192−206. [doi: 10.1007/978-3-540-88313-5_13]

附中文参考文献:

[63] 孙晓妮,蒋瀚,徐秋亮.基于二叉树存储的多用户 ORAM 方案.软件学报,2016,27(6):1475−1486. http://www.jos.org.cn/1000-9825/5002.htm [doi: 10.13328/j.cnki.jos.005002]

吴鹏飞(1994−),男,江苏淮安人,博士生,主要研究领域为分布式系统安全,隐私保护,大数据安全.

沈晴霓(1970−),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为操作系统与虚拟化安全,云计算,大数据安全与隐私,可信计算.

秦嘉(1993−),男,硕士生,主要研究领域为加密共享,隐私保护.

钱文君(1994−),女,博士生,主要研究领域为可信计算,系统安全,大数据计算安全,隐私保护.

李聪(1990−),男,博士生,CCF 学生会员,主要研究领域为公钥密码,区块链,云计算安全.

吴中海(1968−),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为大数据系统与分析,大数据与云安全,嵌入式系统.