doi:10.11959/j.issn.1000-436x.2017166

高效可扩展的对称密文检索架构

吴志强[1]，李肯立[1]，郑蕙[2]
（1. 湖南大学信息科学与工程学院，湖南 长沙 410082；2. 湖南商学院旅游管理学院，湖南 长沙 410205）

摘 要：现有可搜索加密方案通常索引的构建、检索效率不高，数据节点的维护不便，难以适应分布式密文检索要求。针对现有密文检索方案出现的一些问题，提出了可拆分密文倒排索引架构，改善了密文索引的并行构建性能，简化了数据增删维护过程，增强了与传动 NoSQL 系统的兼容性。采用检索结果集中化倒排索引方法，提高分布式密文检索系统检索效率。所提方案满足被广泛采用的选择关键词攻击下的不可区分性(IND-CKA)安全标准。结合 Cassandra 对多种性能进行了验证，实验数据表明，本架构对分布式、海量密文数据环境具有很好的适用性。

关键词：可搜索加密；倒排索引；Cassandra；隐私保护
中图分类号：TP3-0 **文献标识码**：A

Efficient and scalable architecture for searchable symmetric encryption

WU Zhi-qiang[1], LI Ken-li[1], ZHENG Hui[2]
(1. College of Information Science and Engineering, Hunan University, Changsha 410082, China;
2. School of Tourism Management, Hunan University of Commerce, Changsha 410205, China)

Abstract: The existing encryption schemes were usually poor of index construction and maintenance, which was difficult to meet the retrieval requirements for distributed ciphertext. Aiming at the above shortage of existing schemes, a structure of separable ciphertext inverted index was proposed for improving the performance of parallel constructing cryptograph index, simplifying the data maintenance process, and increasing the compatibility with the traditional NoSQL system. The method of centralized inverted index was adopted to improve the retrieval efficiency of distributed retrieval system. Then, the proposed scheme was proved to meet the widely adopted IND-CKA security standard. Finally, Cassandra was combined to evaluate the several performance parameters, and the experimental results show proposed architecture has good applicability to the distributed and massive encrypted data environment.

Key words: searchable encryption, inverted index, Cassandra, privacy preserving

1 引言

1.1 背景与动机

为了更低的成本、更强的可扩展性和更高的可靠性，企业将存储与计算外包到诸如 Amazon EC2、Microsoft Azure、Google App Engine 等云计算环境中。存储于第三方公有云服务提供商的数据，存在各种泄露的风险，如黑客的攻击、内部云数据管理员的泄密等。

2013 年，黑客利用 Struts2 安全漏洞攻击京东购物商城，导致数千万用户名、密码、身份证等用户资料外泄。为了保护数据隐私，数据拥有者不得不将数据加密。对数据查询与计算的时候，需将全部数据从云服务端下载到本地，然后再进行解密、查询与计算操作。虽然此方案最为可靠与安全，但随着数据资源的日益激增，此种方案已经不再可

收稿日期：2016-12-19；修回日期：2017-05-13
通信作者：郑蕙，zhdilly@163.com
基金项目：国家自然科学基金资助项目（No.61672221）
Foundation Item: The National Natural Science Foundation of China (No.61672221)

取。为了充分利用云计算的优势，可搜索加密技术(SE, searchable encryption)等被提出。云端不需对数据解密，就可以提供对密文数据直接检索与计算的服务。具体地，数据拥有者为全部文档建立密文索引，将密文数据和密文索引都存储于第三方不可信云端，并将密钥共享给数据使用者。当数据使用者需要对数据检索的时候由使用者动态生成一个令牌（trapdoor）提交至云端，云端直接检索后将密文结果返回给数据使用者。在此过程中，必须要保证云端得不到任何关于文档和关键词的有效内容信息。

许多工作都对可搜索加密技术进行了研究，并在安全、效率、功能与数据的动态更新能力之间进行不同程度取舍，以适应用户需求。在数据量较大、数据集在线激增的背景下，由于受到硬件条件的制约，集中式环境无法再存储和处理过多的数据，海量密文被分散存储在分布式系统的多个节点上，因此产生了一系列新的问题：如何快速地增量预处理数据；云端如何管理与维护多个节点之间的密文数据，使密文一直保持较好的检索效率；如何仍然保持传统的分布式数据库架构具备的业务在线拓展、节点动态增删、负载均衡、故障恢复等功能。本文着重研究了增量索引的构建效率问题、数据局部化访问问题、云端索引数据维护问题，提出并验证了可拆分密文倒排索引方案。

1.2 安全风险

假设云端是不可信的或诚实但好奇的（honest-but-curious），可以正常执行用户事先规定的流程，但可能会动态跟踪并记录用户的全部查询请求、查询返回的结果，记录进程运行中产生的全部临时数据，并试图结合实际经验综合分析推理，以获取更多的用户数据信息，比如查询的关键词、文档的内容、文档的关键词个数等。

1.3 相关研究及其缺陷

文献[1]最早提出了可搜索加密方案，为无索引方式密文检索，只适合少量文档环境。文献[2]采用每文档一个 Bloom Filter 索引方式，一次检索需要遍历和文档数相同的密文索引，也只适合少量文档环境。后来，众多方案都对索引进行了改进，优化了检索的效率与数据动态增删能力，但仍然存在较多局限性。文献[3~9]没有充分利用云端的资源并行构建索引，索引并行构建效率较低，而且索引之间存在较多相互依赖关系，进一步降低了并行索引的效率，索引的初始化过程会消耗用户端较多的时间。当索引被拆分到多个数据节点时，文献[5~9]没有实现数据的局部化访问，这样会使一次检索很可能会遍历全部的数据节点，实际运行效率较低。文献[3,4]中 SSE-1、DSSE 采用了密文倒排索引方案，仍然需要拓展才能实现数据的局部化访问，以减少查询过程中随机访问存储器的次数。数据的局部化访问特性将在第 3 节进行分析。而且多数方案在索引被加密后，云端失去了独立优化索引数据分布的能力。如果需要优化索引数据的分布，云端所有的数据节点都依赖用户端的令牌才能操作索引，从而优化密文索引的分布较为困难。

可搜索加密技术按照密钥的生成可分为对称可搜索加密和非对称可搜索加密。对称可搜索加密采用 AES、3DES 等算法生成对称密钥，进行加密与计算。对称可搜索加密方案一般只是提供单用户环境下的加密数据检索的功能，都具有较高的检索速度。非对称可搜索加密体制目前一般采用 RSA、Elgamal[10]、Diffie-Hellman[11,12]等算法生成公钥与私钥，参与云端与本地的各种计算。非对称可搜索加密技术容易提供对多用户权限的灵活控制，适合数据共享环境，然而，非对称可搜索加密技术在大数据场景下计算成本较高[13]。

多属性的密文数据库（如 CryptDB[14]）能够实现密文数据的 SQL 查询等较多功能，不过在文献[15]中，多属性的密文数据库被证明为不安全的。同态加密技术[16]、ORAM 技术[17]也能够实现云端安全计算功能，但这些技术在大数据场景下计算或通信开销太大。

1.4 本文的方法

本文采用了与文献[3,4,7]中类似的密文倒排索引技术，关键词可以通过倒排索引快速找到对应的文档集合，具有亚线性时间检索效率。不同的是，本文采用了分布式可拆分与可合并的密文倒排索引技术，将密文关键词对应的文档集合加密后分组，并通过噪音分组对齐，消除最主要的密文统计特征：词频较高的词产生的统计特征。本文并不像文献[3,4]中 SSE-1 一样采用密文链表加密数据节点指针，而使用的是（*key*, *value*）集合类型倒排索引形式，从而实现了对可扩展 NoSQL 系统的兼容[18]，且密文索引数据节点可以被动态拆分与增量合并。此可拆分密文倒排索引容易兼容各种成熟的可扩展分布式系统架构，云端对索引的维护也相对容

易。因索引之间相互并无依赖关联，可利用云端资源并行计算提高数据预处理效率，在普通的 Intel Core I7 920 CPU 上实现了 16.1 min 索引、加密并上传 140 MB 数据，共计约 24 万个(*key*, *value*)元组，对应约 6 300 万个(*w*, *id*)元组，其中，*w* 指单词，*id* 指文档编号。将检索结果以集合的方式集中存储，一次检索只需要访问一个或少数几个节点，大大提高了系统吞吐率，在较低的硬件配置下仍然拥有平均每关键词 100 ms 和每文档 4 μs 的检索效率。本文暂且不考虑前向隐私与后向隐私[19]，以换取较高效的检索性能和优越的可扩展性。

假设分布式系统共有 q 个节点，每个节点拥有 p 个处理器，表 1 中详述了本文方案和其他 SSE 方案的多个性能指标对比。参数 n 指全部文档个数；参数 m 指全部文档中唯一关键词个数；参数 $M=\max_w|R(w)|$；M_0 为将文档集分组后每组的 M 值；参数 $N=\sum_w|R(w)|$；参数 r 指一次查询中平均返回的文档记录集条数；参数 d_w 指对某一个关键词全部的历史删除与插入次数和；参数 t_q 为节点之间的并发通信延迟时间。动态增删指的是可以在运行过程中动态增加与删除数据。检索通信指的是一次查询需要发送的报文大小。检索时间估算暂不考虑分布式系统数据节点复制策略带来的整体性能提升。索引局部化指的是一次检索访问的索引和结果集尽可能集中在少数节点中。云端索引维护指的是云端能够不依赖客户端独立优化索引数据分布的能力。

1.5　本文的贡献

本文的主要贡献如下：1) 提出了一种满足 IND-CKA 安全的可拆分密文倒排索引架构，云端能够对索引快速并行合并、拆分与优化，适合较大规模数据场景；2) 提出了检索结果集中存储型可搜索加密方案；3) 提出了可以兼容目前主流的 NoSQL 分布式数据库的密文索引方法；4) 提出了与 Cassandra 分布式系统相结合的方法，并对各种性能进行了验证。

2　标记与符号

$a||b$ 表示字符串 a 和字符串 b 的连接；$|A|$表示集合内元素的个数；null 或{}表示空集合；H 是一个多项式时间安全伪随机函数，简称伪随机函数，

表 1　　**一些 SSE 方案性能指标对比**

方案名称	动态增删	索引大小	并行索引时间	检索时间，检索通信	索引局部化	云端索引维护
CGKO06-1(SSE-1)[3]	否	$O(N+m)$	—	$O(r)$, $O(1)$	—	无
CGKO06-2(SSE-2)[3]	否	$O(Mn)$	$O\left(\frac{Mn}{p}\right)$	$O(r)$,$O(M)$	否	无
CK10[5]	否	$O(Mn)$	$O\left(\frac{Mn}{p}\right)$	$O(r)$,$O(r)$	否	无
LSDHJ10[6]	是	$O(mn)$	$O\left(\frac{mn}{p}\right)$	$O(m)$,$O(1)$	否	无
KO12[9]	否	$O(Mn)$	$O\left(\frac{Mn}{p}\right)$	$O(m)$,$O(1)$	否	无
KPR12(DSSE)[4]	是	$O(N+m)$	—	$O(r)$,$O(1)$	—	无
KP13(PDSSE)[8]	是	$O(mn)$	$O\left(\frac{mn}{p}\right)$	$O\left(\frac{r\lg n}{pq}+t_q\right)O()$,$O(1)$	否	无
CJJJKRS-1[7]	否	$O(N)$	$O\left(\frac{N}{p}\right)$	$O\left(\frac{r}{pq}+t_q\right)$, $O(1)$	否	无
CJJJKRS-2[7]	否	$O(N)$	$O\left(\frac{N}{p}\right)$	$O\left(\frac{r}{pq}+t_q\right)$,$O(r)$	否	无
CJJJKRS-3[7]	是	$O(N)$	$O\left(\frac{N}{p}\right)$	$O\left(\frac{r+d_w}{pq}+t_q\right)$,$O(1)$	否	无
本文(OSSE)	是	$O(N+mM_0)$	$O\left(\frac{N+mM_0}{pq}\right)$	$O\left(\frac{r+M_0}{p}\right)$,$O(1)$	是	有

指的是敌手对由不同的 2 个字符串 w_1、w_2 计算得到的 $H(w_1)$、$H(w_2)$在多项式时间内无法区分。采用带有密钥 K 的伪随机函数 H，对字符串 w 加密，记 $H_K(w)= H(K||w)$，或直接记为 $H(w)$。本文使用了 2 个带有密钥的伪随机函数 H、G 和一个没有带密钥的伪随机函数 V。*Enc*、*Dec* 是多项式时间安全对称加密、解密算法，指的是敌手对由不同的 2 个字符串 w_1、w_2 被密钥 K 加密后得到的结果 $Enc_K(w_1)$、$Enc_K(w_2)$在多项式时间内无法区分，且对任意的 K 和 w 都有 $Dec_K(Enc_K(w))=w$。记 $E(w, K)=Enc_K(w)$。$Negli(k)$是可忽略的，指的是对任意一个指定的 ε，都存在一个 k_0 当 $k\geqslant k_0$ 时 $Negli(k)\leqslant\varepsilon$。将一个文档 F 看作是不同唯一关键词 w_i 的集合 $F=\{w_1, w_2, \cdots, w_m\}$。倒排索引指的是对文档 $F_1, F_2, \cdots, F_n$ 建立的关键词—文档编号索引。一个关键词 w 对应的拥有此关键词的文档编号集合记为 $R(w)$，相应的密文文档编号集合记为 $R'(w)$。一个文档 id 对应的此文档的不同关键词集合记作 $D(id)$。在不考虑索引中噪音的情况下，如果对任意一个关键词 w 能够从索引 I_1 中找到索引项，那么也能够从 I_2 中找到相同结果集的索引项；同样，能从 I_2 找到索引项，那么也能从 I_1 找到相同结果集的索引项，并记为 $I_1=I_2$。

3 定义

3.1 可拆分对称可搜索加密

定义 1 (可拆分对称可搜索加密)一个可拆分对称可搜索加密算法是由 5 个多项式时间算法构成的元组 $SSE = (KeyGen, BuildIndex, Search, CombineIndex, SplitIndex)$。

SSE.KeyGen 是根据安全参数 k 产生密钥的随机概率算法。输入为 k，输出为随机生成的一组各不相同的密钥 K，每一个密钥的位长度都为 k。此算法分别为对称加密算法 *Enc*、对称解密算法 *Dec* 和带有密钥的伪随机算法 H、G 等提供不同密钥。

SSE.BuildIndex 是根据一组明文文本文件 F 和密钥集 K，得到密文索引 I 和加密的密文文档集合 C 的算法。(I, C) 将被存储于云端，此算法运行在用户端。

SSE.Search 是一个根据检索关键词 w 和用户的密钥集 K，检索得到拥有此关键词的全部文档集合 F_w 的算法。输入为(w，K)，输出为 F_w。其中，用户根据 w 和 K 由 *Trapdoor* 子算法计算令牌 $Trapdoor_K(w)$，提交云端对索引(I, C)检索并得到密文结果集 C_w，最终用户对密文文档集合解密得到 F_w。此算法运行在用户端和云端。

SSE.CombineIndex 是云端独立索引合并的算法，用于快速地将 2 个索引合并成一个索引。输入为索引 I_1 和索引 I_2，输出为合并后的索引 I_3。其中，I_1、I_2、I_3 都是能被 *Search* 算法有效检索的索引，I_3 和 I_1 可以指向同一个索引。*CombineIndex* 过程并不需要用户的密钥 K，也不再需要用户参与。

SSE.SplitIndex 是云端独立索引拆分的算法，用于快速地从一个索引中拆分出一个或多个索引。输入为原始索引 I_1 和待拆分的索引 I_2，输出为拆分出的索引 I_3。其中，(I_1, I_2, I_3) 都是能被 *Search* 算法有效检索的索引，I_3 和 I_1 可以指向同一个索引。*SplitIndex* 过程并不需要用户的密钥 K，也不再需要用户参与。

设 Δ 为字典，如果可拆分对称可搜索加密方案 *SSE* 是正确的，那么对 $\forall k\in N, n\in N, w\in\Delta, F=\{F_1, F_2, \cdots, F_n\}$，$F\subset 2^{\Delta}$，以及 *SSE.KeyGen*、*SSE.BuildIndex*、Enc 输出的 K 和(I, C)，都有以下结论。

1) $Dec_K(Search(I, C, Trapdoor_K(w)))=F_w$ 成立，其中，F_w 为搜索的结果文档集合。

2) 如果不考虑索引中添加的噪音，那么对由 *SSE.BuildIndex* 分别输出的任意 2 个没有共同关键词的索引 I 和 I_1 都有 $SplitIndex(CombineIndex(I, I_1), I_1)=I$。

3) 如果不考虑索引中添加的噪音，那么对于任意 2 个文档 F_1 和 F_2，都有 $BuildIndex_K(F_1\cup F_2)= CombineIndex(BuildIndex_K(F_1), BuildIndex_K(F_2))$。

3.2 IND-CKA 安全

采用文献[2]提出的安全标准来定义可搜索加密安全。令 $SSE=(KeyGen, BuildIndex, Search, CombineIndex, SplitIndex)$是一个动态的基于索引的可搜索加密方案。考虑如下概率的实验 $CKA_{SSE,A}(k)$，其中，A 是一个有状态的敌手。挑战者运行 $KeyGen(k)$ 产生密钥 K。敌手产生 2 个文档 $D=\{d_0, d_1\}$，文档的唯一关键词个数相同，文档长度相同，且有共同关键词，并将文档提交挑战者，挑战者随机选择一个文件 D_b，然后加密得到$(I, C)=BuildIndex_K(D_b)$，交给敌手。敌手可以一次性提交多个查询，对每一个查询 q_i 输入关键词 w，交由挑战者计算 $Trapdoor_K(w)$。敌手只能查询文档 d_0 和 d_1 中共同

拥有的关键词。敌手可以使用 *CombineIndex* 算法与 *SplitIndex* 算法对索引分析。最终敌手 A 输出结果 b'。如果 $b=b'$，实验结果输出 1，否则输出 0，其中，b 由均匀概率密度的算法生成。

SSE 是 IND-CKA 安全的指的是对所有的多项式敌手 A，都有

$$\Pr[CKA_{SSE,A}(k)=1]\leqslant\frac{1}{2}+Negli(k)$$

3.3　数据访问局部化

数据访问局部化特性指的是在分布式密文检索系统中，一次检索需要访问的密文索引或数据集中于局部的少数节点中。系统最大并发度指的是分布式系统整体对外提供检索并发连接的最大能力。没有数据访问局部化的 SSE 方案，数据被随机加密分散存储在各个节点，查询的时候需要遍历几乎所有的数据节点，容易产生通信性能瓶颈问题，系统最大并发度受限。如果一个数据节点最大支持 Y 个并发连接，拥有 q 个数据节点，且不考虑数据复制策略，不考虑连接池等缓冲技术，系统的最大检索并发度并没有随着节点数的增加而提高，仍然为 $O(Y)$。另外，针对相同大小的数据，假设数据都存储于磁盘，从磁盘顺序集中读写的速率远远高于随机访问磁盘的效率，因此，拥有数据访问局部化特性的 SSE 方案往往查询效率较高。云端索引维护能力在本文中定义为云端对不同时期用户提交的索引不再依赖客户端就能优化数据分布的能力，以实现更好的数据局部化与维持较高的检索效率。

4　构建可拆分倒排索引

本文的核心设计是为文档建立可拆分的密文倒排索引，目标是构建密文（*key*，*value*）索引集合以适应现有的各种 NoSQL 架构，从而保持原有 NoSQL 系统的高可扩展性以及高吞吐量等特性。对文档编号对称加密，避免泄露文档中包含的单词分布信息，并使用对齐与拆分技术消除主要的密文统计信息。倒排索引中采用了类似文献[3,4]中提及的技术，但并不支持对密文倒排链表指针加密，否则系统将与之存在相同的缺陷：云端丧失了独立快速索引合并与拆分的能力，以及丧失了对传统 NoSQL 系统的兼容性。本文方案并不加密数据指针，只采用伪随机函数、对称加密函数、文档对齐技术和噪音添加技术仍然可以满足 IND-CKA 安全。本文采用了与 Cassandra 分布式系统相结合的方法，在 Cassandra 中构建非关系型表，用于存储密文索引 I 和密文文档 C。为了方便起见，本文将此方案暂称为 OSSE 方案。

4.1　密文倒排索引

为实现对海量数据快速检索，通常需建立高效的索引，目前常用的方案有 3 种：倒排索引（inverted index）、类 B+树索引和 Bloom Filter 索引[2,7]。传统的明文倒排索引如图 1 所示，在图 1 中，通过 w_3 可以快速地查找出它在文档 1 中出现了 1 次，出现位置为 2，在文档 2 中出现了 2 次。

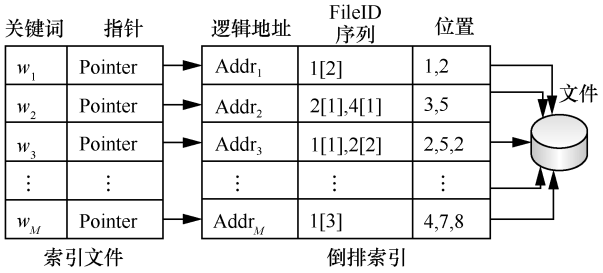


图 1　传统的倒排索引示例

传统的明文倒排索引关键词使用顺序表存储，以快速二分查找，在 $O(\text{lb}m)$时间内快速找到需要的关键词，其中，m 是唯一关键词个数。顺序表结构容易横向拆分。索引文件和反向索引文件之间使用逻辑地址和指针关联，便于大型索引的纵向拆分与合并，适合分布式环境。

本文仍然采用倒排索引架构，如图 2 所示。在图 2 中，密文索引关键词集合记为：$W=\{H_{k_1}(w_i)|i=1\cdots m\}$，其中，$w_i$ 为一个关键词，H 为一个带有密钥 k_1 的伪随机函数，假设$|W|=m$。E 是一个对称加密算法，G 也是一个带有密钥 k_2 的伪随机函数。k_1、k_2 都是用户的不同密钥。若用户需要搜索 w_2 关键词，那么他首先计算 $H_{k_1}(w_2)$值和 $G_{k_2}(w_2)$值提交云端，方便起见，记为$(H(w_2)$、$G(w_2))$。如果密文元素为顺序表存储，那么可以采用折半查找方法，就可以在 $O(\text{lb}m)$时间内通过 $H(w_2)$值找到逻辑指针所在位置；如果密文元素是以队列的方式存储，那么就可以在 $O(1)$的时间内找到逻辑指针和结果。然后在倒排索引中找到对应的逻辑地址，得到被加密的文档列表。最后采用 $G(w_2)$作为对称解密的密钥，对被加密的文档列表逐个解密，得到文档 ID 集合。这里文档 ID 集合为{2, 4}，记 $R(w_2)$为关键词对应的文档集合，也就是 $R(w_2)=\{2，4\}$；记 $R'(w_2)$为密文关键词编号集合，也就是 $R'(w_2)=\{E(2,G(w_2)), E(4, G(w_2))\}$。此

检索算法的输入参数为单词 w 的检索令牌($H(w)$, $G(w)$)，以及倒排索引 I，输出为检索到的密文文件集 C_w。为了安全起见，密文索引中已经去除了词频信息，并插入了空文档编号 0 对齐密文。

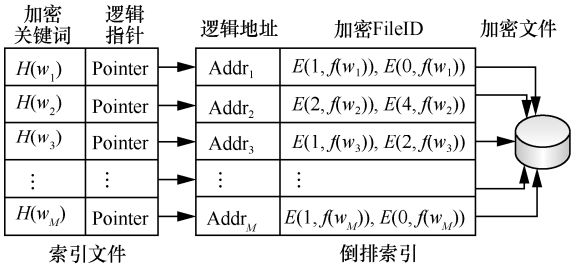


图 2 混淆的可拆分密文倒排索引示例

针对此密文倒排索引方案示例，需要进一步消除此索引中的密文统计特征，才能满足安全要求。

4.2 构建 IND-CKA 密文索引

本文采用如算法 1 所示的伪代码构建密文索引，输入文档集合 F 和密钥 K，输出密文倒排索引 I 和密文文件集 C。算法 2 是关键词检索伪代码，它可以检索由算法 1 生成的（I，C）得到结果集。算法 1 主要由 3 个步骤构成：1) 构建可拆分密文倒排索引；2) 拆分后对齐密文索引，消除高词频词语对应的文档编号统计特征；3) 添加伪关键词噪音混淆密文索引。

本文算法中用到了数个子算法，其中，*Analysis* 算法用于分析统计一个文档集 F，将其中的关键词按照词频的低、中、高取出唯一值，平均分为 3 类 $\{W_1, W_2, W_3\}$，如 *Analysis*({1, 1, 2, 2, 2, 3, 4})={{3, 4}, {1}, {2}}; $R(w)$表示取出关键词 w 对应的全部文档编号集合；$R'(w)$表示关键词 w 对应的全部密文文档编号集合，因包含噪音，体积可能与 $R(w)$不同；$D(id)$表示获得文档编号为 id 的不同关键词集合；H、G 是由密钥 $K=\{K_1,K_2\}$分别加密的伪随机函数，V 为伪随机函数；*SubGroup*(A, i, j)算法表示将集合 A 平均分为 j 块，然后取出其中第 i 块，如 *SubGroup*({1, 2, 3, 4}, 2, 2)={3, 4}；*rnd* 算法为随机数生成算法，*rnd*()表示一个随机整数，*rnd*([1, 100])表示生成一个 1 到 100 之间的随机整数，*rnd*([1, 100]，10)表示生成 10 个 1 到 100 之间的不同随机整数；W 表示全部的不同关键词；*IDs* 表示全部的文档编号。

首先要构建密文倒排索引，使可以根据关键词 w 生成 $H(w)$，然后通过倒排索引立刻检索得到一个文档集合 $R(w)$，按照图 2 的类似方法，构造并实现密文倒排索引。云端使用了一个伪随机算法 V，用于根据 $H(w)$值对文档编号集合分组索引计算，将较大的文档集合拆分，分别存储在多个集合中。将文档集 F 的词语按词频分类，分为低、中、高 3 种类型，表示为 *Analysis*(F)=$\{W_1, W_2, W_3\}$，对每一类词语分别处理。假设词频高的词语在所有文档中的出现率也高，也就是$|R(w)|$较大，将它拆分，分别存储于 3 个集合中；将词频中的词语的 $R(w)$拆分存储于 2 个集合中；词频低的词语的 $R(w)$存储于一个集合中。于是，一个关键词 w 的索引 *key* 分别可能为 $V(H(w)||1)$、$V(H(w)||2)$、$V(H(w)||3)$，一个关键词 w 的密文结果集 *value* 为 $\{E(id_1, G(w)), E(id_2, G(w)),\cdots,E(id_r, G(w))\}$，此 *value* 被拆分存储于多个集合中。具体拆分 *value* 的方法为：加入一个集合编号 j, $j\in[1, 3]$，使结果集的元素成为唯一的不同值 $E(j||id, G(w))$，索引 *key* 则相应的是 $V(H(w)||j)$。

先考虑文档数较少的情况，采用文档编号对齐方法，对齐每一个 $R'(w)$集合使大小都为 $\frac{M}{3}$，以消除文档编号统计特征。其中，$M = Max_{w\in W} |R(w)|$，具体的对齐方法为插入“空文档编号”。空文档编号为 0 开头的字符串，且假设原始文档中不存在 0 开头的文档编号。空文档编号分为 2 类，一类为混淆空文档编号，另外一类为有穷状态空文档编号。对于第一类混淆的空文档编号，表示为 $0||x||id$，加密后变为 $E(0||x||id, G(w))$。对每一个密文关键词 w，首先对齐 $R'(w)$分组集合最多 α 次，且称 α 为 OSSE 方案的混淆系数。第一次对齐后，$R'(w)=R'(w) \cup E(0||1||id, G(w)) \cup E(0||2||id, G(w)) \cup \cdots \cup E(0||max||id, G(w))$，其中，$max\leqslant\alpha$。对于第二类有穷状态空文档编号，表示为 $0||j||i$，j 属于分组编号 [1,3]，i 为[1,M]中的某一个值，加密后变为 $E(0||j||i, G(w))$，如果 $r=\frac{M}{3}-|R'(w)|$，那么第二次对齐：$R'(w)=R'(w) \cup E(0||t_1,G(w)) \cup E(0||t_2,G(w)) \cdots \cup E(0||t_r, G(w))$，其中，$t_i$ 为[1, M]之间的唯一随机数。如果 M 比较小，那么强制设置 M 大于一个常数 M_0 即可。经过对齐后，每一个 $R'(w)$分组的长度都为 $\frac{M}{3}$。

在用户没有发送令牌查询时，云端无法区分是空文档编号还是正常的文档编号，也不能统计关键词对应的不同文档个数来区分不同关键词。只有当

用户对此关键词 w 查询的时候，此关键词对应的空文档编号被对称解密，并忽略所有 0 开头的文档编号，才能准确获得 $R(w)$集合。

当文档数较大时，需要将文档按每组最多 B 个分组索引，最后云端将使用 *OSSE.CombineIndex* 方法分组合并索引。分组合并后，文档编号集合的长度将会处于一个不一致的状态，这会产生部分统计特征。如果为了绝对的安全，需要对齐全部的密文 $R'(w)$索引，但这种极端情况会导致长度等于全部文档个数 n，在较大数据集情况下，会造成检索效率的降低。本文方法为按词频拆分，消除词频高的词的统计特征，并使用纯随机空文档编号和有穷状态空文档编号对齐 $R'(w)$分组，使密文文档编号统计信息处于一个可以被接受的范围。对于第一类空文档编号，合并后会导致索引增大；对于第二类空文档编号，索引合并后每一个分组最多只会增加 M 个元素。

除了需要文档编号对齐之外，还需要关键词对齐。关键词对齐是指在密文索引中插入伪关键词和对应的随机集合值，使密文文档编号和密文词语的分布出现一致化。算法 1 中采用了植入随机数 *rnd*() 的随机密文(*key*, *value*)方法，来实现关键词个数的隐藏。*LW* 指的是每个文档中唯一关键词个数的最大值加上一个随机值。其中，*allign*(*LW*)表示采用伪关键词对齐关键词个数为 *LW* 的整数倍。伪关键词是无法被检索的，主要用于消除密文关键词数的统计特征。

算法 1　*OSSE.BuildIndex*

输入　文档集 F，密钥 K

输出　(I, C)// I 为密文索引，C 为密文文档集

1) $I \leftarrow \{\}$;
2) $k \leftarrow 0$;
3) for each $W_0 \in \{W_1, W_2, W_3\}$ do
4) $k++$;
5) for each $w \in W_0$ do
6) for j=1 to k do
7) $value \leftarrow \{\}$;
8) $key \leftarrow V(H(w)||j)$;
9) $X \leftarrow SubGroup(R(w),j,k)$;
10) for each $id \in X$ do
11) $item \leftarrow E(j||id,\ G(w))$;
12) $value \leftarrow value \cup \{item\}$;
13) $max \leftarrow GetAllign(M, \alpha)$;
14) for l = 1 to max do
15) $item \leftarrow E(0||l||id, G(w))$;
16) $value \leftarrow value \cup \{item\}$;
17) for each $i \in partsOf[1, M]$ do
18) $item \leftarrow E(0||j||i, G(w))$;
19) $value \leftarrow value \cup \{item\}$;
20) $I \leftarrow I \cup \{(key, value)\}$;
21) for i = 1 to $allign(LW)$ do
22) $key \leftarrow H(rnd())$;
23) $value \leftarrow \{E(rnd()_i,\ K)|i = 1 \text{ to } \frac{M}{3}\}$;
24) $I \leftarrow I \cup \{(key, value)\}$;
25) $C \leftarrow Enc_K(F)$;
26) return (I, C);

4.3　实现快速检索

算法 2(*OSSE.Search*)是密文关键词检索伪码。其中，*SubString*(*v*, 1)指的是取出字符串 v 内部第一个字符之后的子字符串。用户输入一个关键词 w，生成令牌$((H(w), G(w))$提交云端，云端最多 3 次访问索引便可以检索到全部的文档编号集合，且只需要通过判断解密后的文档编号集合是否以 0 开头即可判断是否是噪音信息。因 OSSE 的索引 I 是(*key*, *value*)的集合，可以与各种 NoSQL 系统兼容，如果不考虑噪音和解密过程时间消耗，则具有与各种明文 NoSQL 系统同样的检索效率。因需要对结果集顺序解密，将会额外消耗和结果集大小呈线性关系的时间，但此时间消耗在其他 SSE 方案[3,4,7]中一样存在。

算法 2　*OSSE.Search*

输入　索引 I，关键词 w，密钥 K

输出　文档编号集合 *result*

1) Client: $(k_1, k_2) \leftarrow (H(w), G(w))$;
2) 发送(k_1, k_2)到云端;
3) Server:
4) $result \leftarrow \{\}$;
5) for j = 1 to 3 do
6) $key \leftarrow V(k_1||j)$;
7) $v' \leftarrow I.getValue(k_1)$;
8) if $v' \neq$ null then
9) $v \leftarrow Dec(v', k_2)$;
10) if not $v.startWith(0)$ then
11) $id \leftarrow SubString(v, 1)$;
12) if not $result.Contains(v)$ then

13) $result \leftarrow result \cup \{id\}$;
14) else
15) break;
16) return *result*;

4.4 索引合并与索引拆分

OSSE.CombineIndex（索引合并）是指将 2 个不同时期创建的索引快速独立合并，*OSSE.SplitIndex*（索引拆分）指的是从一个索引中分割出一部分索引，这里假设只考虑同一组用户同一个密钥不同时期创建的索引的情况。索引合并和索引拆分的伪码分别由算法 3 和算法 4 所示。

索引合并与索引拆分算法用数行伪代码可以实现，主要得益于 OSSE 索引的（*key*, *value*）集合特性，且 *value* 集合不需要对称解密就可以直接进行集合的合并操作。索引的合并和索引的拆分效率与选用的 NoSQL 存储架构有关。

当一个巨型索引包含 n 个文档，以每 B 个文档为一组分组索引，最后分组合并索引，且 $M \leqslant B$，设置一个常数 α，如果一个关键词 w 对应的 $R(w)$有 r 个结果，可以推断，如果 w 是常用词，那么 OSSE 索引 w 的密文文档编号集合大小最多为 $3(r(1+\alpha)+M)$；如果 w 是不常用词，那么检索结果会落入一个集合中，集合大小最多为 $r(1+\alpha)+M$。于是单线程 *OSSE.Search* 检索时间估算为 $O\left(\frac{M}{3}\right) \sim 3O(r(1+\alpha)+M)$。

4.5 索引的动态维护

基于索引合并和索引拆分算法，可以得出 *OSSE.Remove* 和 *OSSE.Update* 算法，这些算法可用于对索引的动态维护。*OSSE.Remove* 算法是云端对索引数据部分删除的算法，与 *OSSE.Split* 不同的是，*Remove* 算法只会将当前文档编号拥有的关键词数据删除，为集合的差运算。*OSSE.Update* 算法等同于先 *Remove* 再 *CombineIndex* 的算法。

算法 3 *OSSE.CombineIndex*
输入 索引 I_1, I_2
输出 索引 I_1
1) for each $(k, v) \in I_2$ do
2) if I_1.*ContainsKey*(k) then
3) $value \leftarrow I_1$.*GetValue*(k) $\cup \{v\}$;
4) I_1.*SetValue*(k, *value*);
5) else
6) $I_1 \leftarrow I_1 \cup \{(k, v)\}$;

算法 4 *OSSE.SplitIndex*
输入 索引 I_1, I_2
输出 索引 I_1
1) for each $(k, v) \in I_2$ do
2) if I_1.*ContainsKey*(k) then
3) I_1.*Remove*(k);

为了在索引增加或更新的时候不泄露大致的 $|R(w)|$信息，不支持单独更新一个文档或少量文档，需要每组文档数至少达到 B 个才生成新的索引并增量提交云端，且可以提交部分历史文档的增量索引，使密文空文档编号难以被统计攻击识别。

4.6 多关键词检索与检索结果排序

对于多个关键词的检索，云端可以分别采用单关键词的方案检索，然后对检索的结果计算交集。用户将查询请求 $Q=(q_1, q_2, \cdots, q_n)$提交给云端查询，云端将用户的查询请求分解为 q（∧，∨）操作，先实现对每一个子数据的查询，然后对全部的结果进行（∧，∨）操作得到最终结果。

根据 TF-IDF 算法，每一个文档检索到的结果相关性分值与关键词在当前文档中出现的次数成正比，与关键词在其他文档中出现的总次数成反比。由于检索结果中没有任何词频信息，于是无法采用常用的 TF-IDF 算法对检索结果排序。本文采用近似算法对多个关键词检索的结果排序。将关键词在一个文档中的出现词频 *tf* 都设置为 1，反向词频 *idf* 设置为结果集的长度$|R(w)|$，其中，$R(w)$表示对一个关键词 w 检索得到的全部文档编号集合，且 $R(w)$中不包含空文档编号。如果一个连接查询拆分后需要查询的关键词集合为 U，可以计算多个关键词连接查询的结果集的相关性，对某个文档 d 的结果近似分值 $score(d)$为

$$score(d) = \sum_{w \in U, w \in d} \mathrm{lb}\left(\frac{n}{|R(w)|}\right)$$

其中，n 为全部的文档个数。分数越高，检索结果越相关。

5 OSSE 综合分析

5.1 可拆分密文索引分析

定理 1 OSSE 方案是可拆分的对称可搜索加密。

证明 为方便起见，记 *BuildIndex* 为 *BI*，*CombineIndex* 为 *CI*，*SplitIndex* 为 *SI*。

1) 在 OSSE 索引中，因一个关键词对应的 $R(w)$集合可能分别存储于 3 个集合中，云端可以通过伪

随机算法 V 遍历查找这 3 个集合，那么将枚举全部可能的文档 ID。又因为伪文档编号是以 0 开头的编号，云端采用 $G(w)$密钥解密后可以识别是否为伪文档编号，于是可以得到包含此文档编号的准确集合 F_w，$Search(I, C, Trapdoor_K(w))=Enc_K(F_w)$，最终有 $Dec_K(Search(I, C, Trapdoor_K(w)))=F_w$ 成立。

2) 如果不考虑索引中添加的噪音，分析 *OSSE.BuildIndex* 分别输出的任意 2 个没有共同关键词的索引 I_1 和 I_2，I_1 索引由 F_1 文档集合的关键词集合 W_1 索引而成，I_2 索引由 F_2 文档集合的关键词集合 W_2 索引而成。因 OSSE 索引 I_1、I_2 都为(*key*，*value*)的集合，相同 *key* 对应的 *value* 可以合并，于是 $CI(I_1, I_2)=I(W_1) \cup I(W_2)$，$SI(I_1, I_2) = I(W_1)-I(W_2)$，又因 $W_1 \cap W_2=\varnothing$，于是 $SI(CI(I_1, I_2), I_2)=I_1$。

3) 如果不考虑索引中添加的噪音，对于任意 2 个文档 F_1 和 F_2，都有 $BI_K(F_1 \cup F_2)=I(W_1 \cup W_2)$，$CI(BI_K(F_1),BI_K(F_2))=CI(I(W_1),I(W_2))=I(W_1 \cup W_2)$，于是 $BI_K(F_1 \cup F_2)= CI(BI_K(F_1),BI_K(F_2))$。定理 1 得证。

5.2　可扩展性分析

可拆分密文倒排索引的可扩展性体现在如下几个方面：1) 增量构建大型索引；2) 并行构建大型索引；3) 对索引的动态维护；4) 与传统分布式 NoSQL 较好兼容。

为了安全起见，目前，大多数的 SSE 系统首次构建索引 *BuildIndex* 算法都是在客户端完成。受限于客户端的处理能力与存储能力，一般普通客户端无法快速完成巨型索引的构建。于是，对于海量数据，都需要增量构建索引、增量提交云端。目前的一些动态可搜索加密方案，虽然可以增量构建索引，但是都是有状态的增量构建索引，云端需要依赖客户端的增量状态，才能增加索引，这会使索引合并的效率降低。如文献[4, 7]，云端对增量索引的合并操作依赖客户端的 Trapdoor 令牌。可拆分密文倒排索引、增量索引可以任意两两独立合并，是无状态的增量构建索引，多个增量索引之间可以不分先后高粒度并行合并，可以异步、并行增量构建。但如果是不可拆分的有状态的增量索引，虽然也可以部分并行构建，但索引之间会相互存在一些依赖关系，使并行构建与并行合并粒度降低。

可拆分密文倒排索引，相对不可拆分密文倒排索引，在索引快速增加、删除与更新之后，仍然保持较高的检索效率。然而，如果索引之间存在依赖关系，在多次动态增删索引之后可能面临相关索引也需要重构的尴尬，否则将无法继续高效检索。如在文献[7]中，多次增删之后顺序标签之间会产生不连续的碎片，需要继续重构不连续的标签为连续的顺序标签以提高检索效率，导致可扩展性降低。否则只能将全部的删除和修改记录当作新的记录添加到索引中，但这同时带来了索引无限增大和需要检索全部历史修改词的问题。

5.3　与传统数据库兼容性分析

将 OSSE 索引看成(*key*, *value*)的集合，于是可以与各种数据库系统尤其是分布式 NoSQL 系统较好兼容。对于海量数据，密文索引 I 和密文文档 C 可以存储于分布式 NoSQL 数据表中。本文在最后介绍了与 NoSQL 系统 Cassandra 相结合的方法。

OSSE 方案兼容了成熟的分布式 NoSQL 系统以后，将拥有如下优势：1) 可以复用数据的分布式存储；2) 可以复用高效的并发访问；3) 可以复用数据的负载均衡；4) 可以复用数据的错误监测、故障恢复、故障转移、灾难备份等功能。

5.4　检索结果局部化

OSSE 密文倒排索引可以看作是(*key*, *value*)的集合，*key* 为关键词，*value* 为密文文档编号集合。文献[7]也采用了类似的(*w*, *id*)集合方式建立倒排索引，*w* 指的是关键词与结果集位置的联合索引，*id* 则为一个文档编号结果。(*key*, *value*)与(*w*, *id*)之间的关系是，一个 *w* 最多被拆分为 3 个 *key*，一个 *value* 则包含多个 *id*。文献[7]方案用于分布式环境下，如果采用数据表横向拆分方法，当系统节点数较多时，由于采用了伪随机函数，同一个关键词在不同文档中的索引会被随机均匀分散存储在众多不同节点上，导致一次查询会随机访问几乎和结果数相同的节点数，使通信过于复杂。这不仅容易造成网络拥塞，而且节点之间通信的延迟也会导致检索时间成倍增长。如果需要检索结果局部化，需要使用其他方法进行拓展。

与文献[7]不同的是，本文方案为结果集中存储型分布式方案。本文将检索结果 *value* 以集合的形式表示，将集合数据集中存储于一个节点，从而检索的时候不需要遍历全部的分布式节点，这将大大地提高系统吞吐率。对于 512 KB 大小的结果集，本文方案只需访问 1~3 个节点就可以读取到全部的数据。事实上，目前常用的分布式数据库系统也是采用数据本地化的策略来优化数据访问的。一种提

高访问效率的常见方案就是将3个节点的数据合并存储于一个节点上，其余2个节点分别作为复制备份节点，这种方案对结果集中型存储较为适合。

5.5 混淆系数 α

为了使索引多次合并与拆分后仍然具有较少的统计信息泄露，本文引入了混淆系数 α。在第4节中提及的第二类有穷状态空文档编号，在极端情况下，由于取值范围受限，索引多次合并与拆分后，云端通过对比统计能够大概率识别第二类空文档编号。对于第一类混淆空文档编号，尽管进行了较多的索引合并与拆分，任意一个密文文档编号结果都至少存在 α 个密文结果与之不能区分，于是以存储空间的消耗为代价提高了安全性能。

5.6 空间消耗分析

如果关键词对齐后最多有 m 个唯一关键词，文档编号都采用 $\frac{M}{3}$ 个元素对齐，那么索引占用的空间是 $O(\frac{mM}{3})$。如果将 n 个文档分组，每组 B 个文档索引，采用 $M\leqslant B$ 长度空文档编号对齐后，$N=\sum_w|R(w)|$，其中，N 为全部可能的检索结果集合元素数量，那么索引的存储空间消耗最多为 $O(N(1+\alpha)+mM)$。

5.7 检索效率分析

由上文已经得出，OSSE 在单节点情况下检索效率估算为 $O(\frac{M}{3})$~$3O(r(1+\alpha)+M)$。考虑到 *value* 集合内部元素相互并不关联，可以完全并行工作。假设使用 p 核处理器并行处理，检索效率将变为 $O\left(\frac{M}{3p}\right)$~$3O\left(\frac{r(1+\alpha)+M}{p}\right)$。

6 安全性分析

OSSE 方案采用了可拆分密文倒排索引方式加密和混淆。本节将结合文献[2,3]中的敌手攻击模型与定义对本文方案的安全级别定量分析。

6.1 泄露函数

为了严格地分析 OSSE 方案可能泄露给云端的信息，定义如下的泄露函数。

L_1：1) 历史匹配记录(search pattern)；2) 历史检索得到的文档编号集合(access pattern)。

L_2：1) 文档个数及其编号，密文数据大小；2）关键词对应的文档集个数，发生此泄露的概率约为 $\frac{1}{M}$~$\frac{1}{1+\alpha}$，其中，α 为混淆系数。

6.2 IND-CKA 安全

定理 2 假设 H、G、V 都是多项式时间安全伪随机函数，*Enc*、*Dec* 都是多项式时间安全对称加解密算法，那么 *OSSE*={*KeyGen*, *BuildIndex*, *Search*, *CombineIndex*, *SplitIndex*}方案5元组具备(L_1, L_2)上的 IND-CKA 安全。

证明 本文将做一个实验 $Sim_{\{OSSE,A,S\}}(k)$，论证存在一个多项式时间的仿真者 S，可用于模拟敌手的实验 $Real_{\{OSSE,A\}}(k)$的输出，使2个实验得到的输出结果不存在多项式时间区分者可以区分。对仿真者 S，根据敌手 A 的历史检索记录，将产生输出 $v^*=(I^*,c^*,t^*)=((B^*,T^*),c_1^*,\cdots,c_n^*,t_1^*,\cdots,t_q^*)$，$I^*$为模拟的密文索引，$c^*$为模拟的密文文档集合，$t^*$为模拟的历史查询令牌集合，共计 q 个。其中，I^*索引分为2个部分：关键词索引表 B^*和倒排索引表 T^*。B 与 B^*用于存储索引的 *key*，T 与 T^*用于存储索引的 *value*。设经过伪随机函数计算后，数值的长度都为 Len_1，且采用对称加密算法加密长度为 Len_1 的文档编号后的长度统一变为 Len_2，索引中共有 m 个关键词。

1) (仿真 B^*) 首先生成$\{rd_1, rd_2, \cdots, rd_q\}$个随机数。如果 q=0，那么仿真者在 B^*中插入 Len_1 长度的随机数，共计$|W|$个，$|W|$为 B 中元素个数。如果 q>0，由于仿真者 S 根据敌手的输出视图知道每一个查询关键词 w 对应的明文文档编号集合 $R(w)$，并生成 $(rd_1, R(w_1))$, $(rd_2, R(w_2))$, $\cdots$, $(rd_q, R(w_q))$，准备加密后插入到索引 I^*中，其中，rd_i 是长度都为 Len_1 的随机数。首先，插入$\{V(rd_1\|1), V(rd_2\|1), \cdots, V(rd_q\|1)\}$到 B^*中，如果敌手输出视图中还有中频词或高频词的查询，则分别对每个词对应插入 $V(rd_i\|2)$、$V(rd_i\|3)$到 B^*中，如果共计生成了 x 条记录，那么再插入其他$|W|-x$条长度为 Len_1 随机数值到 B^*中。

2) (仿真 T^*) 如果 q=0，那么对每一个关键词随机产生 M 个随机文档编号，插入 T^*中，注意到 $\frac{M}{3}$ 为敌手视图中关键词对应的密文文档集个数，因被空文档编号对齐了，所以都是 $\frac{M}{3}$ 个。如果 q>0，那么产生$\{rd'_1, rd'_2, \cdots, rd'_q\}$个 Len_1 长度的随机数作为密钥，仿真者 S 使用 *Enc* 算法分别加密 $R(w_1)$, $R(w_2)$, $\cdots$, $R(w_q)$集合，仿真者从敌手输出视图知道敌手查询的

w_i属于低频词还是高频词，用 $F(w_i)$表示，{1}为低频词，{1, 2}为中频词，{1, 2, 3}为高频词。其中，$R^*(W_i)=\bigcup_{\{j\in F(w_i),id\in R(w_i)\}} Enc(j\|id,rdi')$。将 $R^*(w_1)$, $R^*(w_2)$, …, $R^*(w_q)$分别插入到 T^*中。对于每一个关键词，如果对应插入了 y 条记录，则将剩余的 M–y 个位置用长度为 Len_2 的空文档编号对齐。

3) (仿真 t_i*)这里将(rd_i, rd_i')设置为 t_i*。

4) (仿真 c_i*) 因为原始密文的长度在敌手的视图中是已知的，这里生成和 c_i 长度一样的随机字符串 c_i*。

于是，本文构造了一个仿真的索引 I*，且可以用仿真的令牌 t_i*查询得到仿真的结果。现在证明这个仿真的输出和敌手的原始视图输出是无法多项式时间区分的。

1)（B 和 B*）无论 q=0，还是 q>0，B*中都是长度一致为 Len_1 的随机字符串，而 B 中都是由伪随机算法 H 得到的长度一致为 Len_1 的字符串。根据伪随机算法 H 的特性，无法在多项式时间区分 B 和 B*中任意一条记录。

2)（T 和 T*）如果 q=0，T*中都是长度一致为 Len_2 的随机字符串，而 T 中都是采用对称加密算法 Enc 加密的长度为 Len_2 的字符串，根据对称加密算法 Enc 的特性，无法多项式时间区分 T 与 T*。如果 q>0，T 中可以查询得到 $R(w_1)$, $R(w_2)$, …, $R(w_q)$集合，而 T*中同样可以使用 t_i*包含的对称密钥 rd_i'对应的密文集合 $R(rd_i)$解密得到完全一样的集合。

3)（t_i 和 t_i*）由于 t_i 是由伪随机算法 H 和 G 组合计算得到的结果，而 t_i*是随机数，两者长度都一致，于是无法在多项式时间区分 t_i 与 t_i*。

4)（c_i 与 c_i*）由于 c_i 是采用多项式时间安全对称加密算法 Enc 加密的文档，而 c_i*是随机的长度一致的字符串，于是多项式时间不可以区分 c_i 与 c_i*。

由以上的分析得出，$Real_{\{OSSE,A\}}(k)$实验的输出和 $Sim_{\{OSSE,A,S\}}(k)$仿真实验的输出在多项式时间内无法区分，于是定理 2 得证。

7　实验与评测

为了评估 OSSE 方案的综合性能，使用约 5 000 行 C#语言代码实现 OSSE 方案的客户端，云端采用去中心化架构的 NoSQL 系统和 Cassandra 非关系型分布式数据库系统[20]。因 Cassandra 已经经过上千个节点长期的测试与改进，具有线性可扩展性、容错性、故障检测、故障恢复、负载均衡、容灾备份等众多特点，相比 Hbase、Redis、Lucene 等，本文方案与 Cassandra 去中心化分布式系统架构结合更为方便，可部分复用 Cassandra 系统的架构，快速构造大型分布式密文检索系统。本文分析多个因素，用于评估参数对性能的综合影响。实验的环境为 5~8 台台式电脑，CPU 为 Intel i7 2.67 GHz，24 GB 内存，部分节点采用 512 GB SSD 固态硬盘，操作系统为 Windows 7 64 位版本和 Ubuntu14.04 64 位版本，采用 Mono 提供对 Microsoft.NET 的跨平台支持。使用 Microsoft .NET Framework 的 System.Security.Cryptography 空间提供的 AES 算法与 SHA256 算法，前者用于实现对称加密数据与对称加密节点，后者用于实现伪随机算法。文档编号（包括伪文档编号）在加密之前统一对齐成 15 B，加密后采用 Base64 编码存储。5 台电脑中，1 台为 Seed 节点，3 台为工作节点，1 台为仿真客户端，并使用 100 Mbit/s 网络互联。Cassandra 为 3.9 版本，支持类似关系型数据库查询的 CQL 查询语言，使用 DataStax 提供的访问 API 与之交互。本文将做多个实验，分别用于验证索引的并行构建效率、索引的增量构建效率和数据的检索效率。

7.1　初始化与配置

为 Cassandra 系统建立一个 KeySpace，在此空间内建立 2 个表：InvertedIndex 表和 Files 表。InvertedIndex 表用于存储 OSSE 密文倒排索引，Files 表用于存储密文原始文件。为了能够更加快速地批量插入与更新数据，将 Cassandra 的批量处理报警阈值和批量处理限制阈值调整为最大值 5 000 KB，并设置复制因子为 1，设置一致性级别为 Any。使用类似 update InvertedIndex set v=v+{'2', '8'} where k='abc'的 CQL 指令实现密文文档编号集合的插入、合并与更新。使用 BatchStatement 类批量处理命令，以每 50 个命令为一组分组提交 Cassandra，实现较高的处理效率。设置 B=100，也就是以 100 个文件为一组分组分别索引，最后合并全部索引。

7.2　数据集

使用文献[4]中的 Enron Email 测试数据集 20150507 版本，此数据集也被许多可搜索加密方案使用。此数据集解压后包含 517 401 个文本文档邮件，共 1.32 GB，每个文档大小从数 KB 到数百 KB 不等。将此数据集分为 2 组，第一组为字母 a~c 开头目录内包含的全部文档，共计 55 366 个文件，140 MB 大小，记为 D_1；第二组为全部的 51 万个

文档，记为 D_2。

7.3 并行索引性能评估

可拆分密文倒排索引可以任意独立两两快速合并，具有高粒度并行索引特性。常用的可搜索加密方案，如果全部利用客户端的资源进行索引，在客户端内存资源不充足条件下（比如 8 GB 内存），索引 140 MB 文本数据通常需要数小时到 1 天时间不等[3,4,7,8,19,21]。OSSE 方案支持云端高粒度并行索引，可以充分利用云端的存储与计算资源，索引 140 MB 数据仅需要 16.1 min。

本文使用如下策略，将文档集按每组 100 个文件拆分为 N 个任务包，这 N 个任务包分别由 20~50 个进程分别处理，每一个任务包采用 *OSSE.BuildIndex* 算法预处理数据，然后采用一个集中化的进程管理器按照 CPU 的使用率、网络带宽的使用率和内存的使用率对全部的进程统一管理和动态调度，使全部的系统处理资源任何时候都不处于闲置状态，也不处于过度饱和状态。同时，将数据增量提交到仿真的云端，利用 Cassandra 提供的分布式计算缓解本地索引的负荷。得益于可拆分对称可搜索加密的高粒度并行特性，多个任务之间可以不分先后并行工作，索引 140 MB 文本文件仅需要 16.1 min，索引 Enron 数据集全部文件也仅消耗约 3 h，相比文献[4, 8]加密方案快数倍。

在索引预处理实验过程中，网络一直为 100%满负荷状态，但 CPU、内存都存在闲置，实验的整体性能仍然具备提升空间。

影响索引效率的一个重要参数为混淆系数 α，图 3 为不同的混淆系数 α 分别在数据集 D_1、D_2 下的索引速度。

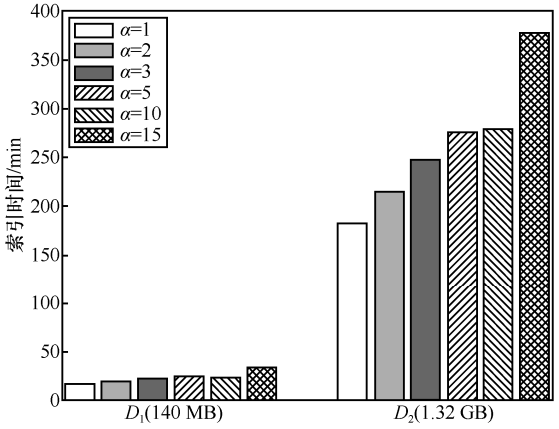


图 3 不同混淆系数 α 下的 OSSE 并行构建索引效率

当混淆系数 α 由 1 变为 15，索引 140 MB 数据所消耗的时间也由 16.1 min 增长至 34 min。当 α=1，处理 D_1 数据集全部独立任务会产生 391 万条临时(*key*, *value*)记录，合并后最终产生 24 万条(*key*, *value*)记录与 6 319 万条(*w*, *id*)记录，如图 4 所示。实验数据表明，OSSE 方案具有较高的数据预处理效率。

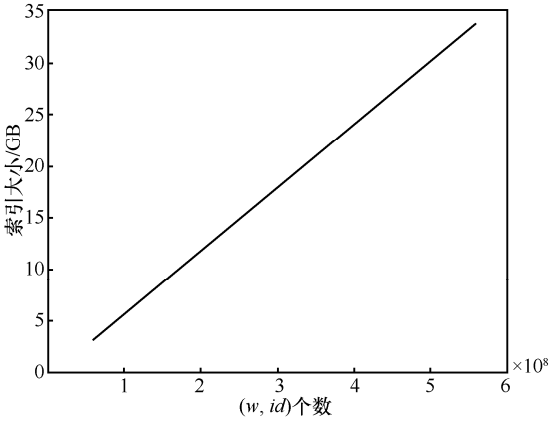


图 4 索引占用空间与(*w*, *id*)个数之间的关系

7.4 索引维护性能评估

可拆分密文倒排索引具备线性可扩展性，数据节点的线性增加也会带来索引维护性能的近似线性提升。本文采用 Cassandra 提供的 CQL 指令仿真实现 *OSSE.CombineIndex*、*OSSE.SplitIndex* 与 *OSSE.Remove* 算法，索引合并的效率、索引更新与删除的效率和 CQL 的执行效率总体一致。CQL 的执行效率一般在每秒 1 万~10 万条之间[20]，且每条 CQL 指令可以增删多个 *id*，4 个节点每秒可以更新约 6 万个(*w*, *id*)，8 个数据节点能提供每秒更新约 12 万个(*w*, *id*)的能力。图 5 描述了在此实验环境下，分布式系统节点个数与更新(*w*, *id*)速度之间的关系。

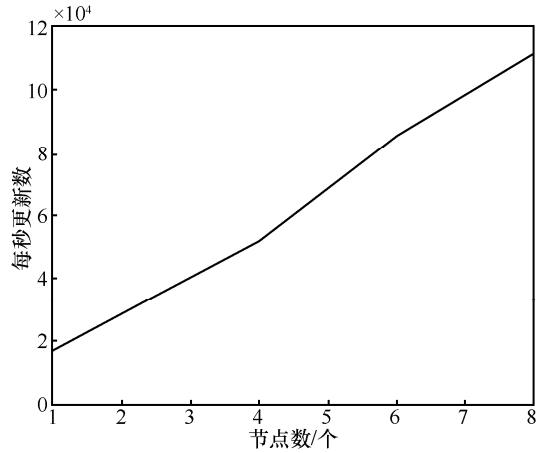


图 5 节点个数与更新(*w*, *id*)速度之间关系

由实验数据可知，可拆分密文倒排索引数据结构具有较好的索引维护效率。其他 SSE 方案目前暂未明确提及线性可扩展的能力，目前暂无法比较。文献[8]提供了基于节点加密红黑树的快速并行密文检索方案（PDSSE），红黑树索引具有线性扩展的潜力，但仍然遗留 2 个问题。一个问题是需要减少检索过程中大量访问网络与磁盘的次数；另外一个问题是红黑树索引消耗过多的存储空间，只适合小关键词空间场景。

7.5　检索时间评估

分别采用不同关键词组合，来检索 D_2 密文数据集。为了使时间有可比性，暂不分析原始数据解密时间和数据传输时间，只分析数据获取时间与数据计算时间。数据获取时间指的是从分布式计算节点的磁盘读取到全部相关密文索引的时间，而数据计算时间指的是对文档编号解密并过滤空文档编号的时间。图 6 中，采用 3~6 线程，并行搜索“the”关键词，得到 410 189 个结果，消耗时间 1 636 ms，平均每 4 μs 得到一个 *id*，获取与计算约各占一半的时间。获取时间由分布式系统的磁盘与网络性能决定，对于本地局部化存储的数据，数据访问效率更高。计算时间是相互无关元素之间的集合运算，可以高度并行处理，相比文献[3,4]倒排索引中的密文链表结构，并行性能更优秀。由于对(*key*, *value*)值采用伪随机函数加密后，*key* 产生了随机性，会消耗较多的分布式系统网络资源，导致索引合并效率适当降低。但一个 *key* 得到的 *value* 集合值是单节点连续存储，所以并没有带来检索效率的降低，在大规模节点集情况下，仍然能够实现较高效率的检索。

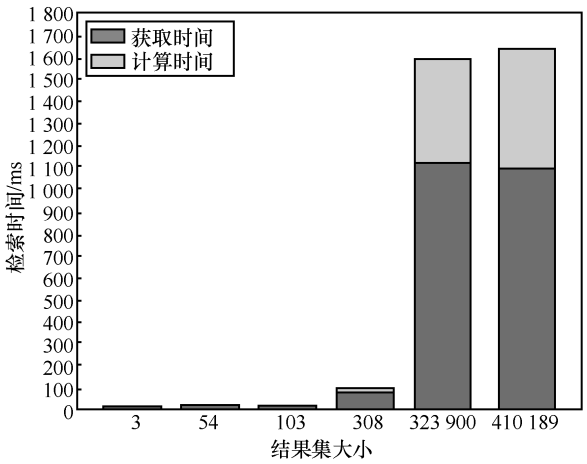


图 6　OSSE 算法结果集大小与检索时间关系

文献[7]中的密文检索算法(CJJJKRS)是一种标签顺序递增密文并行检索算法，密文检索效率相对其他方案[3~6,19,21]都高。采用同样的开发环境将文献[7]中的算法 CJJJKRS-1 实现，并索引全部 D_2 数据集。结果集大小与检索时间的关系如图 7 所示。当数据结果集较小时，如果数据都已经在内存中，算法的访问效率较高；当数据结果集较大，需要反复访问磁盘时，算法效率急剧下降，即使拥有部分缓存和并发加速访问，检索消耗的时间仍然是 OSSE 算法时间的数倍。

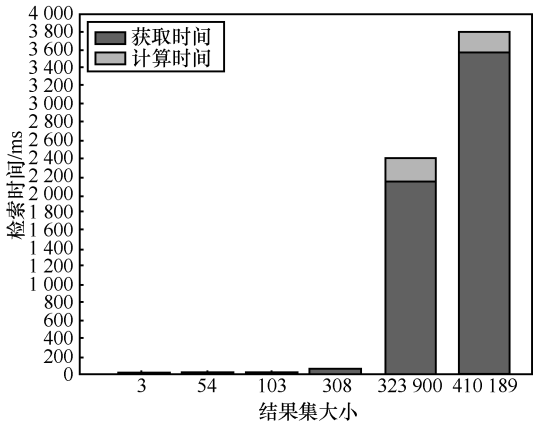


图 7　CJJJKRS-1 算法结果集大小与检索时间关系

据文献[20]中提供的数据，如果节点数充足并采用数据复制策略，拥有大的缓存，采用光纤网络，即使在大数据集场景下，基于关键词—文档的密文倒排索引检索延迟仍然可以保持在毫秒级。

7.6　统计特征

OSSE 索引在索引的多次合并之后，存在一定的密文文档编号统计特征，本文通过实验观察统计特征。图 8 描述了密文词语编号与密文文档 ID 集

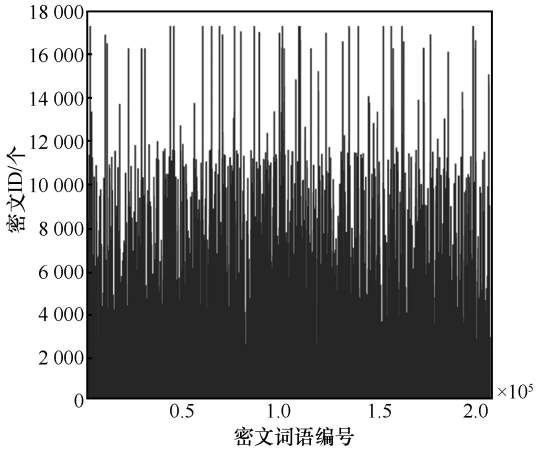


图 8　密文统计特征

合长度$|R'(w)|$之间的统计特征，它由 D_1 数据集经混淆因子 α=10 混淆索引而成，共产生了 24 万个(*key*, *value*)与 6 321 万个(*w*, *id*)。由于词频高的词被拆分后，统计特征和词频低的词一样，而且存在噪音信息，泄露的文档个数并非是直接的结果集大小，无法通过分析文档编号集合简单区分，因此，泄露的密文统计特征仍然处于可接受范围。

7.7 实验总结与展望

实验表明，OSSE 架构具有高度并行的数据预处理效率、优越的可扩展性、较好的兼容性与较高的检索效率，可以适应分布式系统环境。本文架构可为其他可搜索加密方案提供思路，实现更多的密文云计算功能，如密文模糊匹配[22,23]、密文子字符串搜索[24]、密文范围查询、密文 Boolean 查询[13]和多关键词查询等功能[25,26]；可与文献[27]结合，隐藏文档大小及个数以提高安全性；可与文献[28]结合降低 search pattern 泄露风险；也可以和中心化架构的 HBASE 结合或与传统关系型数据库比（如 MySQL）等结合，以适用不同客户环境。由于 Cassandra 牺牲了部分一致性，换来了更高的写入与读取效率，相比采用关系型数据库方案，可拆分倒排索引与 Cassandra 结合更适用于一致性要求并不十分严格的大规模密文数据场景。受限于实验条件，暂没有实验多用户、大规模数据节点的部署，未来将进行更多实验和改进。

8 结束语

本文提出并实现了一个线性可扩展、混淆对称可搜索加密架构 OSSE，使用了可拆分密文倒排索引技术以及噪音混淆技术，能够满足 IND-CKA[2]安全要求，具有索引的快速并行构建、快速合并与动态维护能力，且与一些 NoSQL 非关系型数据库系统结合比较方便。实验验证了本文架构适合分布式环境，具有亚线性时间的检索效率与优越的可扩展性，适合充分利用云计算资源处理海量密文数据。

参考文献：

[1] SONG D X, WAGNER D, PERRIG A. Practical techniques for searches on encrypted data[C]//IEEE Symposium on Security and Privacy IEEE Computer Society. 2000:44-55.

[2] GOH E J. Secure Indexes[J]. IACR Cryptology ePrint Archive, 2003, 2003: 216.

[3] CURTMOLA R, GARAY J, KAMARA S, et al. Searchable symmetric encryption: improved definitions and efficient constructions[C]//ACM Conference on Computer and Communications Security. ACM, 2006: 895-934.

[4] KAMARA S, PAPAMANTHOU C, ROEDER T. Dynamic searchable symmetric encryption[C]//ACM Conference on Computer & Communications Security. 2015: 965-976.

[5] CHASE M, KAMARA S. Structured encryption and controlled disclosure[M]. Advances in Cryptology-ASIACRYPT 2010. Springer Berlin Heidelberg, 2010:577-594.

[6] LIESDONK P V, SEDGHI S, DOUMEN J, et al. Computationally efficient searchable symmetric encryption[C]//Secure Data Management, VLDB Workshop, SDM 2010. Singapore, 2010: 87-100.

[7] CASH D, JAEGER J, JARECKI S, et al. Dynamic searchable encryption in very-large databases: data structures and implementation[C]// Network and Distributed System Security Symposium. 2014.

[8] KAMARA S, PAPAMANTHOU C. Parallel and dynamic searchable symmetric encryption[M]. Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2013:258-274.

[9] KUROSAWA K, OHTAKI Y. UC-secure searchable symmetric encryption[J]. Lecture Notes in Computer Science, 2012, 7397:258-274.

[10] ELGAMAL T. A public key cryptosystem and a signature scheme based on discrete logarithms[J]. IEEE Transactions on Information Theory, 1985, 31(4):469-472.

[11] DIFFIE W, HELLMAN M E. New directions in cryptography[J]. IEEE Transactions on Information Theory, 1976, 22(6): 644-654.

[12] CASH D, JARECKI S, JUTLA C, et al. Highly-scalable searchable symmetric encryption with support for boolean queries[M]. Advances in Cryptolog-CRYPTO 2013. Springer Berlin Heidelberg, 2013: 353-373.

[13] FANG L, SUSILO W, GE C, et al. Public key encryption with keyword search secure against keyword guessing attacks without random oracle[J]. Information Sciences An International Journal, 2013, 238(7): 221-241.

[14] POPA R A, REDFIELD C M S, ZELDOVICH N, et al. CryptDB: protecting confidentiality with encrypted query processing[C]//SOSP. 2011:85-100.

[15] NAVEED M, KAMARA S, WRIGHT C V. Inference attacks on property-preserving encrypted databases[C]//ACM Sigsac Conference on Computer and Communications Security. 2015: 644-655.

[16] DIJK M V, GENTRY C, HALEVI S, et al. Fully homomorphic encryption over the integers[C]//International Conference on Theory and Applications of Cryptographic Techniques. Springer-Verlag, 2010: 24-43.

[17] STEFANOV E, VAN DIJK M, SHI E, et al. Path ORAM: an extremely simple oblivious RAM protocol[C]//ACM Sigsac Conference on Computer & Communications Security. 2013:299-310.

[18] CATTELL. Scalable SQL and NoSQL data stores[J]. ACM Sigmod Record, 2010, 39(4):12-27.

[19] STEFANOV E, PAPAMANTHOU C, SHI E. Practical dynamic searchable encryption with small leakage[C]//Network and Distributed System Security Symposium. 2014.

[20] LAKSHMAN A, MALIK P. Cassandra: a decentralized structured storage system[J]. ACM Sigops Operating Systems Review, 2010, 44(2): 35-40.

[21] BELLARE M, BOLDYREVA A, O'NEILL A. Deterministic and efficiently searchable encryption[C]//International Cryptology Conference on Advances in Cryptology. Springer-Verlag, 2006:535-552.

[22] LI R, LIU A X, WANG A L, et al. Fast range query processing with strong privacy protection for cloud computing[J]. IEEE/ACM Transactions on Networking, 2016,24(4):2305-2318.

[23] LI J, WANG Q, WANG C, et al. Fuzzy keyword search over encrypted data in cloud computing[J]. In INFOCOM, 2012, 3(9):1 - 5.

[24] CHASE M, SHEN E. Substring-searchable symmetric encryption[J]. Proceedings on Privacy Enhancing Technologies, 2015(2): 263-281.

[25] CAO N, WANG C, LI M, et al. Privacy-preserving multi-keyword ranked search over encrypted cloud data[J]. IEEE Transactions on Parallel & Distributed Systems, 2011, 25(1):829-837.

[26] XIA Z, WANG X, SUN X, et al. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data[J]. IEEE Transactions on Parallel & Distributed Systems, 2015, 27(2): 1.

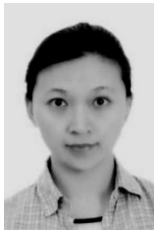
[27] NAVEED M, PRABHAKARAN M, GUNTER C A. Dynamic searchable encryption via blind storage[C]//S&P. 2014:639-654.

[28] LIU C, ZHU L, WANG M, et al. Search pattern leakage in searchable encryption: attacks and new construction[J]. Information Sciences, 2014, 265(5):176-188.

作者简介：

吴志强（1977-），男，湖南涟源人，湖南大学博士生，主要研究方向为网络安全、并行计算。

李肯立（1971-），男，湖南娄底人，湖南大学教授、博士生导师，主要研究方向为并行计算、网格计算和 DNA 计算等。



郑蕙（1978-），女，湖南长沙人，湖南商学院讲师，主要研究方向为大数据安全、旅游电子商务。