中文摘要

密文检索技术是一种允许服务提供商对用户事先加密的文件进行检索的全文检索技术，可应用于云存储环境下，防止包括服务提供商在内的第三方获取用户文件的任何信息，以解决用户检索加密文件的需求。该技术是云存储技术中的核心安全构件。针对于此，本文设计一种密文检索方案用以满足新的功能需求，并据此实现相关密文检索系统以验证其正确性和可用性。具体来说，本文的研究内容包括以下三个方面：

1．分析传统信息检索系统的基本功能需求，包括布尔检索与排名检索。以此为依据，着重分析目前已知的密文检索方案为满足功能需求所做的设计。进而，分析已有系统中存在的不足与缺陷，提出新的设计目标。

2．设计一种密文检索方案。首先，在构造安全索引时使用优秀的编码方法来压缩索引记录，使得安全索引的空间开销更小，并优化了检索的效率。其次，使用同态加密算法的加法同态性质来支持多关键字的相关度计算，检索结果的排序工作由客户端完成。从而使得该方案可以同时支持布尔检索与多关键字排名检索。

3．实现一种由密文检索辅助工具和安全检索网站构成的密文检索系统。对于实现过程中出现的一些编码问题，提出了解决的方法。最后通过分析系统存在的不足，进而指出后续的研究方向。

总之，本研究设计了一种新的密文检索方案，实现相关密文检索系统。在方法上，体现了以下创新：

1．密文检索方案的索引空间开销小，检索效率高；

2．密文检索系统同时支持布尔检索与多关键字排名检索。

关键字：密文检索，多关键字排名检索，同态加密，云存储

ABSTRACT

Searchable encryption technology is a kind of full text retrieval technology, which allows service provider search the files user encrypted in advance. It can be applied in cloud storage environment to meet the need of searching encrypted files for users. Moreover, it can prevent the third party, including the service provider, to obtain any information of users' files. This technology is the core security components in cloud storage technology. Thus, we design a new searchable encryption scheme to meet the need of new features, and implement a searchable encryption system to verify its correctness and usability. Concretely, our contributions are summarized as follow.

1．We analyze the basic functional requirements of traditional information retrieval system, including boolean retrieval and ranked retrieval. We focus on the analysis of the design of existing searchable encryption schemes which meet the functional requirements.

2．We design a searchable encryption scheme. First of all, we use an efficient encoding method to construct secure index, which makes the space of secure index smaller, and optimize the efficiency of retrieval. Moreover, we use a homomorphic encryption algorithm, which has an additive homomorphic property, to support multiple keywords relevancy computation, and the sort of retrieval results performed by client. Therefore the scheme can support both boolean retrieval and multi-keyword ranked retrieval.

3．We implement a searchable encryption system, which is composed of searchable encryption tools and secure search website. In addition, we put forward the solution for some encoding problems appeared in the process of implementation. Finally, we analyze the disadvantage of the system, and point out the future research directions.

In conclusion, we design a new searchable encryption scheme, and implement a related searchable encryption system. Our work has the following advantages:

1．The index space of this searchable encryption scheme is small, and retrieval efficiency is high.

2．The searchable encryption system support both boolean retrieval and multi-keyword ranked retrieval.

Keywords: searchable encryption, multi-keyword ranked retrieval, homomorphic encryption, cloud storage

目录

1. 绪论

1.1 背景

密文检索技术是一种允许服务提供商对用户事先加密的文件进行检索的全文检索技术，可防止包括服务提供商在内的第三方获取用户文件的任何信息，以解决用户检索加密文件的需求。

目前，云存储作为一种新兴的网络存储服务被越来越多的人使用。但鉴于云存储服务的安全性，用户一般不会将私密文件存放在由服务提供商托管的数据中心。若想将私密文件存放到服务器上，用户不得不对私密文件使用传统的对称加密算法如 AES 进行加密。然而，服务器无法对使用 AES 加密的文件进行任何操作，也就是说服务器无法提供除上传和下载以外的任何服务，这在一定程度上弱化了云存储服务的能力。对于大规模数据集合的管理与维护，高效的检索能力显得尤为重要。密文检索技术的出现，对解决这一问题提供一种安全的手段。

在传统的非交互式信息检索中，一般分为布尔检索与排名检索两种方式[1]。从直观上看，密文检索技术也应该同时提供这两种检索方式，使得用户可以像操作明文那样，根据具体的应用场景来选择适当的检索方式。

为了满足这一需求，大量的密文检索方案已经被提出，其中包括只支持布尔检索的[2, 3, 4, 5]，支持单关键字排名检索的[6, 7]，以及支持多关键字排名检索的[8, 9, 10]。

1.2 应用场景

密文检索技术可以满足对加密文档集合进行安全检索的需求。特别地，对于目前越来越常见的将数据外包存储在第三方服务器的应用场景，密文检索技术提供一种安全高效的方法使得用户可以检索加密的文档集合。

场景一：用户 Olivia 由于不放心将明文文件存放在数据中心，因此将文件加密后上传到数据中心。但 Olivia 很多时候只有有限的带宽可以连接到数据中心，她希望可以在不下载全部文件的前提下，找到所需的文件[8]。

场景二：Waters 等人[11]使用 Goh[3]提出的密文检索方案，构建了一个加密的日志系统，使得只有具备生成关键字陷门权限的管理员才可以检索日志系统的内容。

场景三：用户可以在具有较高运算能力的计算机上生成安全索引和密文文件集合，并上传到服务器。在外出的时候，只需事先在手机上配备密钥，即可检索并下载到所需的文件[4]。

1.3 本文的主要工作与结构安排

1.3.1 主要工作与贡献

分析目前已知的密文检索方案。依据传统信息检索系统的基本功能需求，分析目前已知的密文检索方案的设计思路与不足，进而提出新的设计目标。

设计密文检索方案。结合信息检索领域的相关知识，提出一种密文检索方案，在功能上可同时支持布尔检索与多关键字排名检索。新方案在构造安全索引时使用优秀的编码方法来压缩索引记录，而且使用同态加密算法的加法同态性质来实现多关键字的相关度计算，检索结果的排序工作由客户端完成。

实现密文检索系统。实现一种由密文检索辅助工具与安全检索网站构成的密文检索系统，在实践中验证了新方案的正确性与可用性。对于实现过程中出现的编码问题，提出了解决的方法。通过分析，指出后续的研究方向。

1.3.2 本文结构安排

第一章分析密文检索技术的相关应用场景与功能需求，着重表明密文检索技术可应用在以云存储为代表的应用场景中以满足用户检索加密文件的功能需求。

第二章对密文检索技术进行功能需求分析。首先，对传统信息检索系统的功能需求进行分析。以此为依据，分析目前已知的密文检索方案的设计思路和存在的不足。

第三章设计一种密文检索方案。在 SSE-1[5]的基础上，使用优秀的编码方法对索引结构进行优化，提出一个可支持布尔检索的安全索引。进而引入同态加密算法以支持多关键字相关度的计算，提出一个可同时支持布尔检索和多关键字排名检索的密文检索方案。此外，分析修改文档集合对安全索引的影响。最后通过与目前已知的密文检索方案进行对比，分析新方案的利弊。

第四章实现一个由密文检索辅助工具和安全检索网站构成的密文检索系统。最后分析系统在实现过程中出现的一些问题，进而指出后续的研究方向。

第五章总结全文。

2. 密文检索技术的分析

2.1 信息检索系统的基本功能需求

2.1.1 基本检索方式

在传统的信息检索系统中，一般提供两种基本的检索方式，即布尔检索（Boolean Retrieval）和排名检索（Ranked Retrieval）。与排名检索相比，布尔检索在表达上更加精确，但对用户的要求较高。在另一方面，排名检索返回的结果按照相关度进行排序。

布尔检索接受由布尔表达式构成的查询，即通过与（AND）、或（OR）、非（NOT）等逻辑操作符将词项连接起来的查询[13]。考虑到补运算是代价昂贵的，出于效率原因，系统可能选择使用减（相对补）操作代替非操作[12]。

排名检索是采用一个或多个词项构成自由文本查询（Free Text Query），返回的结果通过文档的相关度进行排序[13]。信息检索系统中最常见的项权重框架综合考虑项频（Term Frequency）和反比文档频率（Inverse Document Frequency），一般简称为 TF-IDF[12]。利用项权重，一般选用下面三个方法来计算文档与特定查询的相关度[1]。

坐标匹配（Coordinate Matching）。本质上是累加查询中各个关键字在特定文档中的项权重，来计算文档的相关度。若文档中没有出现该关键字，则记该关键字的对于文档的相关度为 0。

内积相似度。确定相关性的过程可以形式化为查询向量和文档向量求内积[1]。将文档和查询分别表示为一个 T 维向量，其中 T 是文档集合包含的关键字数量。对两个向量求内积得到该文档与查询的相似度，并以此为排序的依据。

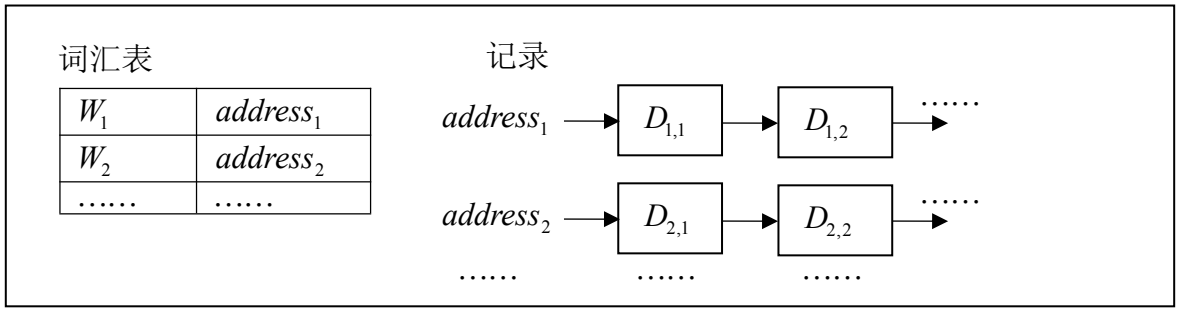
向量空间模型。与内积相似度相似，但通过求两个向量的夹角余弦来表示它们的相似度。由于这种方法一般能达到更好的检索效果，因此在信息检索领域，是应用最广泛的排名检索方法。

2.1.2 索引结构

在传统的信息检索系统中，为了实现以上两种检索方式，一般可选用三种索引结构：倒排索引（Inverted Index）、签名文件（Signature File）和位图。在大多数应用场景中，与倒排索引相比，签名文件和位图没有任何优势[1]。因此本节主要介绍倒排索引，但由于 Goh[3]提出的方案涉及到签名文件，所以也简略介绍签名文件。

倒排索引由两个元素组成，词汇表和记录[12]。词汇表保存从文档集合中提取的关键字及其记录地址，例如$<term_1,address_1>$。记录是一个列表，保存出现该关键字的文档标号集合，例如$(1,3,5,6)$，存放文档标号 1 的地址为 $address_1$。因此上述例子表示，$term_1$ 出现在文档标号为 1、3、5 和 6 的文档中。通常，词汇表使用类似哈希表这样支持快速检索的数据结构来实现，记录使用类似链表这样支持快速顺序访问的数据结构来实现。

显然，对于一个给定的关键字，使用倒排索引可以在 O(1)的时间取出对应的文档标号集合。基本倒排索引的结构如图 2.1 所示：



（图 2.1：基本倒排索引）

签名文件本质上使用一种称为布隆过滤器（Bloom Filter）的方法，用一个比特串来表示文档的关联签名[1]。为了生成文档描述符，对在文档中出现的每个关键字，分别使用多个不同的哈希函数生成多个哈希值，并在文档描述符（比特串）相应的位置置 1。在检索的时候，分别对查询包含的关键字重新进行哈希值计算并检查文档描述符的相应位置是否为 1。使用这种方法的好处在于节省索引的空间，但随着压缩倒排索引的出现，签名文件节省索引空间的好处在逐渐缩小[1]。在另一方面，对于一个给定的关键字，使用签名文件需要 O(N)的时间取出对应的文档标号集合，N 为文档数量，而且布隆过滤器具有误判性。

文献[12]还提到一些较新的索引结构，但由于在密文检索系统中难以应用，故不赘述。

2.2 密文检索技术的形式化定义

传统的密文检索方案主要是以 Curtmola 等人[5]提出的可检索对称加密方案（Searchable Symmetric Encryption）为代表。根据他们提出的形式化定义，密文检索技术一般可分为五个关键部分，即 $SSE=(Gen,Enc,Trpdr,Search,Dec)$。其中，

$K \leftarrow Gen(1^k)$：用户使用密钥生成算法，通过设置安全系数 k 来生成相应的私钥 K

$(I,C) \leftarrow Enc(K,D)$：用户利用私钥 K，对文件集合 $D=(D_1,D_2,\cdots,D_n)$ 使用加密算法，生成一个安全索引 I 和密文集合 $C=(C_1,C_2,\cdots,C_n)$

$t \leftarrow Trpdr(K,w)$：用户利用私钥 K，对一个给定的关键字 w 使用陷门算法，生成该关键字 w 的陷门

$X \leftarrow Search(I,t)$：服务器使用关键字 w 的陷门 t，对安全索引 I 进行检索，返回满足条件的结果集合 X

$D_i \leftarrow Dec(K,C_i)$：用户利用私钥 K，对密文文件 C_i 使用解密算法，得到原文件 D_i

2.3 密文检索技术的相关研究

2.3.1 支持布尔检索的密文检索方案

为了让服务器具备检索加密数据的能力，Song 等人[2]基于流密码提出一个密文检索方案。除了基于较弱的安全模型外，该方案的主要问题是算法复杂度为 O(S)，S 为文档集合的大小。这是由于该方案没有对文档集合构建索引，在每一次检索时都需要对全文进行匹配。

Goh[3]提出一种基于安全索引的密文检索方案，本质上是参考信息检索领域中签名文件的概念，利用多个私钥使用布隆过滤器作为陷门算法。检索时算法复杂度为 O(N)，N 为文档集合包含的文档数量。使用安全索引的好处在于，分离索引和加密文件。检索过程中只需要对索引文件进行操作，而原文件可独立应用高效安全的加密算法，甚至可以压缩存放以节省空间。Chang 等人[4]怀疑 Goh[3]提出的密文检索方案在使用布隆过滤器时可能存在静态分析的风险，提出另一个同样是基于索引的密文检索方案，尝试保证陷门不泄漏任何信息。

Curtmola 等人[5]提出可检索对称加密方案，安全性比上述方案都强。SSE-1 的索引结构类似于倒排索引，因此检索时的算法复杂度为 O(1)；SSE-2 在 SSE-1 的基础上进行扩展，本质上是将出现在不同文档中的同一关键字在安全索引中加密为“不同”的关键字，使得所有倒排记录表的长度为 1。因此进一步隐藏包含同一关键字文档的出现次数，增大服务器静态分析的难度，但索引所需的空间开销较大。

2.3.2 支持单关键字排名检索的密文检索方案

Swaminathan 等人[6]以 SSE-1 作为基本的密文检索方案，引入保序对称加密算法（Order-Preserving Symmetric Encryption）对事先计算的项权重进行加密。在检索时，服务器可以直接对加密后的项权重进行排序，从而支持排名检索。Wang 等人[7]使用与[6]相似的设计思路，但指出[6]基于 SSE-1 的检索效率并不高。他们重新基于倒排索引提出一个密文检索方案，与[6]相比算法效率更高。

使用保序对称加密算法可以使服务器拥有密文比较的能力，但服务器无法对密文进行其他任何操作。由此限制了项权重必须是事先计算的，无法做到动态计算项权重以支持多关键字排名检索。

2.3.3 支持多关键字排名检索的密文检索方案

Cao 等人[8]参照传统信息检索领域中实现排名检索的内积相似度方法，应用安全 K 近邻算法（Secure k-Nearest Neighbor）提出一个多关键字排名检索加密方案（Multi-keyword Ranked Search Encryption, MRSE）。在传统信息检索领域，K 近邻算法（k-Nearest Neighbor，kNN）[12]主要用于文本分类。在此处，每个文档和查询分别表示为一个二进制向量。在查询时，使用安全 K 近邻算法返回与查询向量最相关的 K 个文档。此方法的优点是使用较为成熟的排名检索模型保证检索结果的准确性，而且在服务器端

实现与密文排序相当的功能，仅通过一轮查询用户即可获得结果。但需要注意的是，传统的内积相似度方法是使用项权重而不是简单的二分量来计算相关度，因此该方案应用的是一个弱化的内积相似度方法。

Baldimtsi 等人[9]提出可以使用同态加密算法来允许服务器累加多个关键字的项权重以支持多关键字的排名检索。具体来说，他们使用一种具有双层加密性质和加法同态性质的广义 Paillier 加密体系（Generalized Paillier）。在使用 Paillier 算法的加法同态性质来允许服务器累加多个关键字项权重的同时，应用其双层加密性质设计一个秘密排序协议，引入一个不与服务器 S1 合谋的计算组件 S2 来帮助 S1 对文档的相关度进行排序。在此过程中，保证 S1 和 S2 都无法获得足够的信息推断出关于用户文档的任何信息，除非 S1 和 S2 合谋。虽然该方案可以很好地支持多关键字排名检索，但由于需要 S1 与 S2 合作完成密文排序，该方案的实现复杂度较高。

Jiss 等人[10]提出可以直接使用全同态加密算法[14]来加密项权重，由此可应用传统信息检索领域的向量空间模型，在得出相关度结果后，由客户端解密并进行排序。但使用全同态加密算法的问题在于计算的开销太大，特别是考虑到向量空间模型需要把文档表示为向量，其空间开销也很大。虽然他们在论文中对时间开销与传统的密文检索方案进行比较，得出不错的结果，但测试的数据规模实在太小，测试结果不足以说明问题。

2.4 本章小结

信息检索系统的基本功能需求包括布尔检索与排名检索。为了实现这两种基本的检索方式，一般采用倒排索引作为索引结构。检索时，只需检索索引即可得到结果。在设计密文检索方案时，同时实现以上两种检索方式显得尤为重要。

接着，本章描述密文检索技术的基本概念。一般地，密文检索技术可分为五个关键部分：密钥生成、加密文档集合与生成安全索引、生成关键字陷门、安全检索和解密密文文档。

最后，本章分析目前已知的密文检索方案。除了第一个密文检索方案[2]不对文档集合进行索引外，其他更新的方案[3, 4, 5, 6, 7, 8, 9, 10]都吸收信息检索领域的设计思想，对文档集合建立安全索引来提供更安全高效的检索手段。

因为布尔检索只需要关键字对应的文档标号集合，因此实现布尔检索是相对容易的。对于排名检索，索引还需要保存关键字对应的相关度集合。在传统信息检索系统中，依据相关度的结果排序是由服务器完成的。排序的核心操作是比较，那么，在密文检索系统中要支持单关键字的排名检索，需要服务器具备密文比较的能力。因此，文献[6, 7]引入保序加密算法使得服务器能对加密后的相关度密文进行比较。

对于多关键字的排名检索，在传统的信息检索系统中可应用不同的技术模型来实现。而文献[8]提出的密文检索方案借鉴文本分类的思想，通过找出与查询最相似的 K 个文档来实现多关键字排名检索。而延续之前单关键字的排名检索的设计思想，如果能将

多个关键字的相关度密文在服务器上累加，即服务器具有密文相加的能力，就可以实现多关键字排名检索。文献[9]提出的密文检索方案可以使服务器具有密文相加和密文排序的能力，从而支持多关键字排名检索。在另一方面，文献[10]提出的密文检索方案虽然在空间和效率上有待改进，但它并不像传统信息检索系统那样由服务器实现检索结果的排序，而是将其交由客户端实现。虽然与其他方案相比，其服务器能力相对较弱，但由客户端来实现相关度的排序，在一定程度上优化了检索的效率。

从上述分析可知，应用最新的密文检索方案，可支持布尔检索和多关键字的排名检索，这在一定程度上解决了信息检索系统的基本功能需求。

3.密文检索方案的设计

3.1 引言

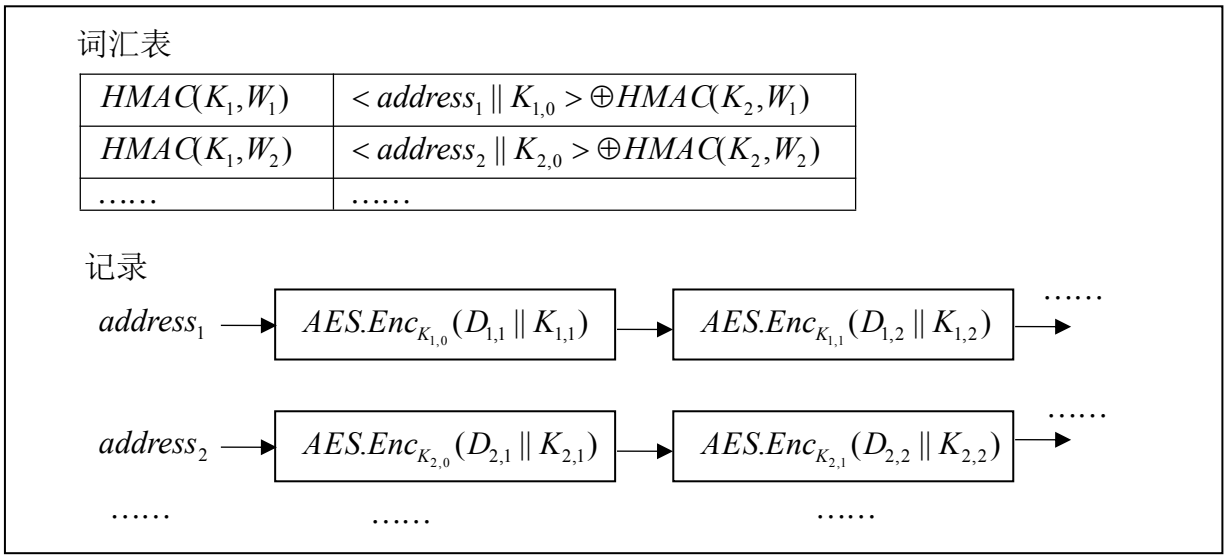
本章基于 SSE-1 的安全索引进行扩展，针对其存在的不足，引入压缩编码方法对文档标号集合进行压缩，使得索引空间开销更小，并优化了检索效率。

为了支持多关键字排名检索，使用同态加密算法加密在构造索引时事先计算的关键字项权重，然后利用同态加密算法的加法同态性质以支持多关键字的项权重累加，最后返回的检索结果在客户端解密后进行排序。另外，由于项权重密文各个单独存储时的空间开销较大，类比压缩文档标号集合的思想，将项权重集合编码为一个比特串，进一步降低索引的空间开销。

3.2 支持布尔检索

3.2.1 SSE-1 存在的不足

为了说明 SSE-1 存在的不足，本节首先描述一个简化的 SSE-1，SSE-1 的详细设计参见文献[5]。正如上一章提到，SSE-1 本质上是基于倒排索引而设计的一个安全索引结构（如图 3.1）。原文件使用对称加密算法进行加密，检索过程不涉及原文件。



（图 3.1：SSE-1 安全索引）

倒排索引由两个元素组成，词汇表和记录。记录保存关键字 W_i 的文档标号集合，记为 $D(W_i)$。对于每一个 $D(W_i)$，生成一个记录首密钥 $K_{i,0}$；对于文档标号集合中的各个元素 $D_{i,j}$ 做如下处理：生成一个对称加密密钥 $K_{i,j}$，并利用上一个密钥 $K_{i,j-1}$ 加密 $D_{i,j}$ 和 $K_{i,j}$，即 $AES.Enc_{K_{i,j-1}}(D_{i,j} \| K_{i,j})$，其中‖表示连接。词汇表保存关键字与及其对应的记录地址。在 SSE-1 中，关键字 W_i 存储为 $HMAC(K_1,W_i)$；记录地址 $address_i$ 与相应的记录首密钥 $K_{i,0}$ 共同存放，并使用 $HMAC(K_2,W_i)$ 异或其结果，即 $< address_i \| K_{i,0} > \oplus HMAC(K_2,W_i)$。

检索时，用户计算查询中的各个关键字的 $HMAC(K_1,W_i)$ 和 $HMAC(K_2,W_i)$ 并提交到服

务器；服务器使用 $HMAC(K_1,W_i)$ 找到相关的关键字索引项，并利用 $HMAC(K_2,W_i)$ 恢复出记录地址 $address_i$ 和记录首密钥 $K_{i,0}$，然后依次解密得到对应的文档标号集合。

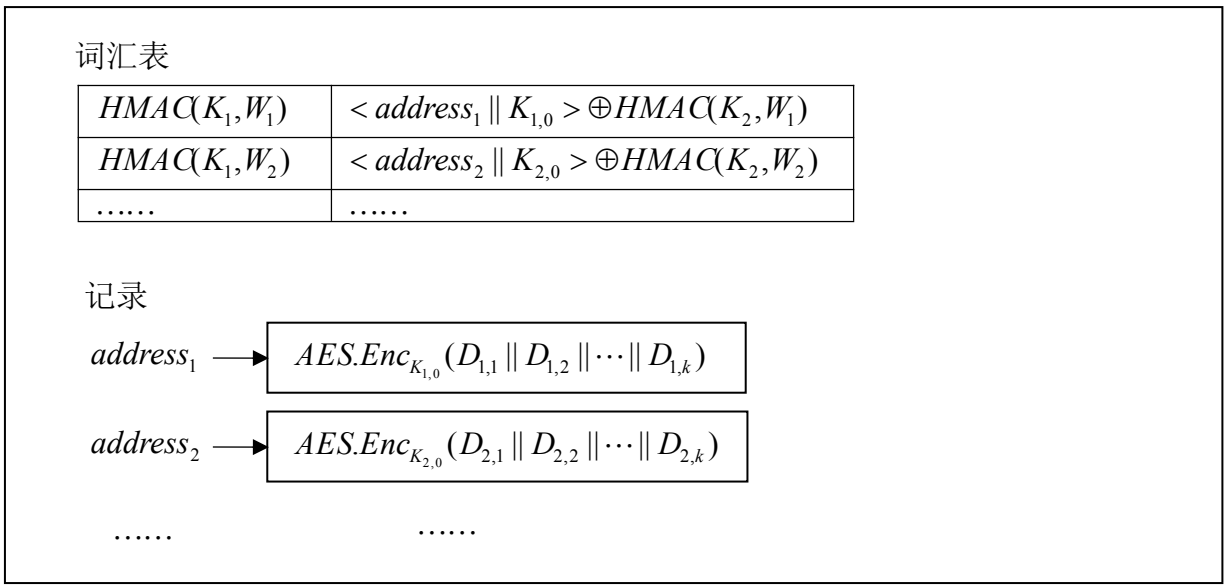
无论使用何种检索方式，对于特定关键字 W_i，检索时必须取出其文档标号集合。而 SSE-1 为了保证服务器无法获取出现关键字的文档数量，对词汇表中的每个关键字都分配不同的对称加密密钥，而且对文档标号集合的各部分均使用不同的密钥独立加密，在检索过程中必须多次解密才能获得整个文档标号集合，时间开销较大。正如在 2.2.2 中提到，Wang 等人[7]指出文献[6]使用的 SSE-1 算法效率不高，并基于倒排索引提出另一个类似的安全索引结构，获得更好的算法效率。

另外，由于像 AES 这样的对称加密算法是基于块加密的算法，一般采用填充模式。而文档标号一般很小，加密后的密文很可能需要进行填充。因此整个记录累积起来的“额外”存储空间就不容忽视。

因此，虽然 SSE-1 检索时的算法复杂度为 O(1)，但加密解密文档标号集合的效率并不能令人满意。在另一方面，加密后的文档标号集合与原来的文档标号集合相比，需要更多的存储空间。

3.2.2 压缩文档标号集合

鉴于检索时相应关键字的整个文档标号集合都要被完全解析出来，因此可以先将整个记录的文档标号连接在一起，然后使用一个记录密钥对整个记录做对称加密，来获得更好的效率，即 $AES.Enc_{K_{i,0}}(D_{i,1}\|D_{i,2}\|\cdots\|D_{i,k})$。索引结构如图 3.2 所示。



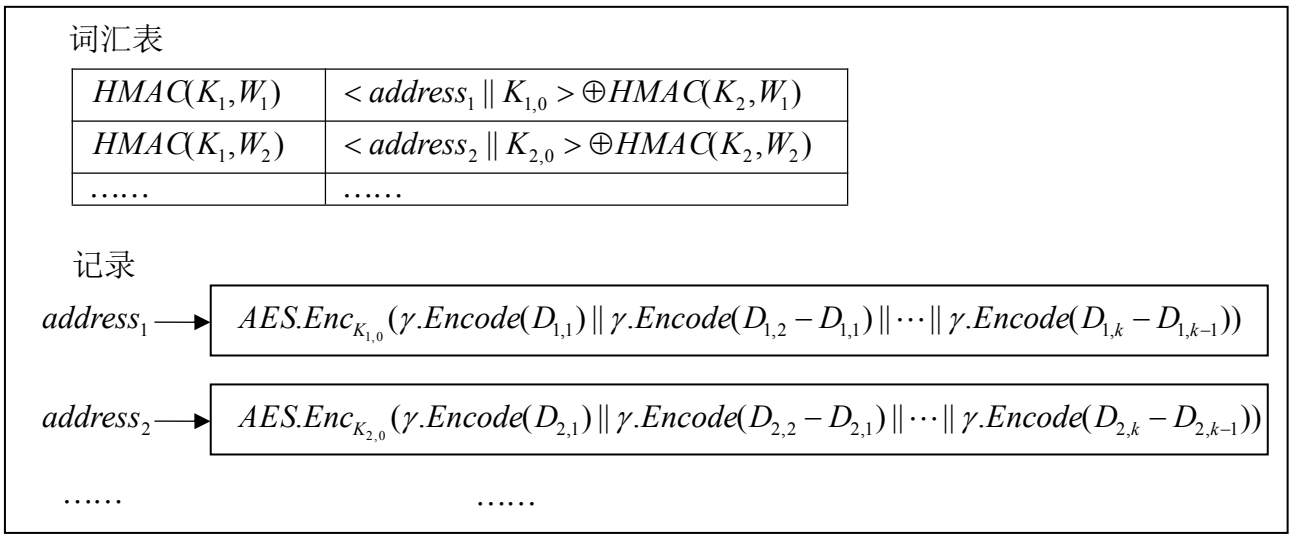
（图 3.2：文档标号集合集中存储的 SSE-1）

另外，在传统信息检索系统中，一般会对倒排记录表进行压缩存储，以获得更好的空间利用率。受此启发，引入一种与最优编码长度差距在常数倍以内的无参编码方法，γ 编码[13]，对文档标号集合进行压缩存储，然后再利用记录密钥进行加密。

具体来说，一个整数 G 的 γ 编码可分为长度和偏移(offset)两部分。G 的偏移实际上是 G 的二进制编码，但其最前端的 1 被省去。例如，9 的二进制编码为 1001，那么其偏

移是 001。G 的长度是指使用一元编码[13]编码后的偏移长度。整数 N 的一元编码为 N 个 1 后面接 1 个 0 的二进制串。例如，3 的一元编码为 1110。因此，9 的 γ 编码为 1110001。解码时，首先读入一元编码直到遇到 0，然后根据一元编码表示的整数 N 继续读入 N 位二进制码，即可恢复出 γ 编码所表示的整数。

由于 γ 编码对小数编码所需的位数比大数更少，使用 γ 编码对倒排记录表进行压缩的最佳实践是，除了首文档标号外，其他文档标号保存为与上一个文档标号的差值。文献[1]中还提到一些与 γ 编码相比具有更高压缩比的编码方法，这里选用 γ 编码的原因是 γ 编码在具有足够好的压缩比的同时，编码和解码的效率较高。此外，由于 γ 编码是前缀编码，因此可将整个文档标号集合中的元素不加边界地编码为一个比特串，而不影响解码结果。至此，优化后的 SSE-1 安全索引如图 3.3 所示：



（图 3.3：优化后的 SSE-1 安全索引）

在另一方面，SSE-1 生成索引时需要向倒排记录表随机填充无用块以达到随机化的效果，这时会产生额外的空间开销。但由于使用 γ 编码编码后的倒排记录表的长度并不仅与倒排记录表中含有的文档标号数量有关，更与文档标号之间的差值有关，因此不需要随机填充无用块即可达到随机化的效果。例如，关键字 W_1 对应的文档标号集合为 (1,2,3,4,5,6,7,8)，则使用 γ 编码编码后得到的比特串为 00000000；关键字 W_2 对应的文档标号集合为 (1,21)，则使用 γ 编码编码后得到的比特串为 0111100100。显然单从编码后的比特串长度无法推断文档标号集合的基数。

3.2.3 优化后的 SSE-1

优化后的 SSE-1 在支持布尔检索的同时，索引的空间开销更小，检索效率更高。在另一方面，由于检索过程与原文件无关，因此对原文件做压缩处理后再做加密运算，在解密的时候步骤与之对应，这样服务器的空间利用率更高。也正是由于检索过程不需要原文件的参与，因此可以使用安全性和效率都有保证的 AES 加密算法加密原文件，这是使用安全索引的优势。Li 等人[18]实现了一个基于整数全同态加密密文检索系统，该系统使用基于整数的全同态加密算法加密原文件，并在检索过程中检索全文。但由于全同

态加密算法加密的基本单位是比特，单就时间与空间上的开销来说就非常大。

具体算法构造如图 3.4 所示：

$K \leftarrow Gen(1^k)$：

根据安全参数 k 选取 K_1, K_2, K_3，其中 K_1 为 AES 密钥；K_2, K_3 为 HMAC 密钥，输出 $K = (K_1, K_2, K_3)$

$(I, C) \leftarrow Enc(K, D)$：

1. 扫描文档集合 $D = (D_1, D_2, \cdots, D_n)$，并生成明文倒排索引 I

2. 将文档集合中的各个文档 D_j 进行压缩，并使用 AES 算法加密，即 $AES.Enc_{K_1}(Compress(D_j))$，得到密文集合 $C = (C_1, C_2, \cdots, C_n)$

3. 将词汇表中存储的关键字 W_i 替换为 $HMAC(K_2, W_i)$

4. 对每个关键字 W_i 对应的记录生成一个记录密钥 $K_{i,0}$，并将词汇表中存储的记录地址 $address_i$ 替换为 $< address_i \| K_{i,0} > \oplus HMAC(K_3, W_i)$，其中 $\|$ 表示连接操作，$\oplus$ 表示异或操作

5. 对于每个记录，对首个文档标号 $D_{i,1}$ 进行 γ 编码，后续对每个文档标号 $D_{i,j}$ 与前一个文档标号 $D_{i,j-1}$ 的差值进行 γ 编码，最后表示为一个比特串，再使用记录密钥 $K_{i,0}$ 加密，即 $AES.Enc_{K_{i,0}}(\gamma.Encode(D_{i,1}) \| \gamma.Encode(D_{i,2} - D_{i,1}) \| \cdots \| \gamma.Encode(D_{i,k} - D_{i,k-1}))$

6. 参考 SSE-1 随机填充无用索引块。假设 V 是 $|D|$ 个文档中可能包含的最大关键字数目，向安全索引填充无用索引块直至总索引块的数目为 V。

7. 输出(I, C)

$t \leftarrow Trpdr(K, W)$：

计算 $HMAC(K_2, W_i)$ 和 $HMAC(K_3, W_i)$，分别记为 τ 和 η，输出 $t = (\tau, \eta)$

$X \leftarrow Search(I, t)$：

1. 将 t 解析为 τ 和 η；在安全索引 I 中查找 τ，若没有匹配项，则不存在包含关键字 W_i 的文档

2. 若存在匹配项，在安全索引 I 中取出 $\varphi = < address_i \| K_{i,0} > \oplus HMAC(K_3, W_i)$，计算 $< address_i \| K_{i,0} > = \varphi \oplus \eta$，使用 $address_i$ 找到关键字 W_i 相应的记录，记为 $x = AES.Enc_{K_{i,0}}(\gamma.Encode(D_{i,1}) \| \gamma.Encode(D_{i,2} - D_{i,1}) \| \cdots \| \gamma.Encode(D_{i,k} - D_{i,k-1}))$，然后依次使用 $K_{i,0}$ 解密和 γ 编码解码，即 $\gamma.Decode(AES.Dec_{K_i}(x))$，最后利用得到的文档标号差值恢复出文档标号集合

3. 返回满足条件的文档标号集合 $X = (X_1, X_2, \cdots, X_k)$

$D_j \leftarrow Dec(K, C_j)$：

计算 $Decompress(AES.Dec_{K_1}(C_j))$，得到 D_j

（图 3.4：优化后的 SSE-1）

这里对算法的细节作一些补充说明。K_2是用于构造检索时的索引入口，K_3是用于构造保存记录地址与记录密钥的陷门。对于每个不同的记录使用独立的 AES 记录密钥，主要是为了最大程度地隐藏每一个关键字具体出现的文档数量。在上述的方案中，服务器当此仅当用户检索特定关键字 W_i 时，才可知道包含 $HMAC(K_2,W_i)$ 的文档数量，这在一定程度上增加了服务器静态分析索引内容的难度。若将文档标号集合明文存放，这时服务器可以对安全索引进行扫描分析。根据大量关键字生成的 $HMAC(K_2,W_i)$ 对应的文档集合基数，有可能推测出个别 $HMAC(K_2,W_i)$ 对应的明文关键字。

3.3 支持多关键字排名检索

3.3.1 引入同态加密算法

为了支持多关键字排名检索，需要一个具有加法同态性质的同态加密算法。这里选用原始的 Paillier[17]加密算法，加密事先计算的 $TF\text{-}IDF_{i,j}$，然后利用 Paillier 的加法同态性质使服务器具有将多个关键字的 $TF\text{-}IDF_{i,j}$ 密文累加的能力，最后由客户端解密 $TF\text{-}IDF_{i,j}$ 并进行排序。

特别地，本文使用文献[12]推荐的一个 TF-IDF 权重框架，其中 $f_{i,j}$ 表示关键字 W_i 在文档 D_j 中的出现次数，N 表示文档集中的文档数量，n_i 表示出现关键字 W_i 的文档数量。计算公式如下所示：

$$TF-IDF_{i,j}=(1+\log(f_{i,j}))\times\log(\frac{N}{n_i})$$

不使用具有更强大功能的全同态加密算法[14]的主要原因是，使用全同态加密算法的时间和空间开销太大，特别是使用全同态加密算法实现像文献[9]那样的密文排序协议所需开销相当大。文献[15]使用文献[16]中的开源同态加密库 HElib，实现了对两密文的比较运算，测试结果显示其时间和空间开销很大。

具体来说，本文采用的 Paillier 加密算法如图 3.5 所示：

$(PU,PR)\leftarrow Gen(1^k)$：

1. 根据安全系数 k，随机选取两个不相等的比特长度为 k 的素数 p 和 q，计算 $N=p\times q$；
2. 计算 $\lambda=lcm(p-1,q-1)$，其中 $lcm(x,y)$ 表示计算 x 和 y 的最小公倍数。随机选取 $g\in Z^*_{N^2}$，使其满足 $\gcd(L(g^\lambda \bmod N^2),N)=1$，其中 $L(x)=\frac{x-1}{N}$；
3. 计算 $\mu=L(g^\lambda \bmod N^2)^{-1} \bmod N$；
4. 输出公钥 $PU(N,g)$，私钥 $PR(N,\lambda,\mu)$。

$c\leftarrow Enc(PU,m)$：

1. m 为待加密信息，满足 $m\in Z_N$；

2. 随机选取 $r \in Z_N^*$；

3. 计算密文 $c = g^m \times r^N \bmod N^2$。

$m \leftarrow Dec(PR, c)$：

1. c 为密文信息，满足 $c \in Z_{N^2}^*$；

2. 计算明文 $m = L(c^\lambda \bmod N^2) \times \mu \bmod N$。

$m_1 + m_2 \leftarrow Dec(PR, (Enc(PU, m_1) \times Enc(PU, m_2) \bmod N^2))$：

$$c_x = Enc(PU, m_1) \times Enc(PU, m_2) \bmod N^2$$
$$= (g^{m_1} \times r_1^N \bmod N^2) \times (g^{m_2} \times r_2^N \bmod N^2) \bmod N^2$$
$$= g^{m_1+m_2} \times (r_1 r_2)^N \bmod N^2$$
$$m_1 + m_2 = Dec(PR, c_x)$$

（图 3.5：Paillier 加密算法）

由此可知，服务器只需要掌握 N^2 即可具备将多个关键字的 $TF\text{-}IDF_{i,j}$ 密文相加的能力。

3.3.2 编码相关度集合

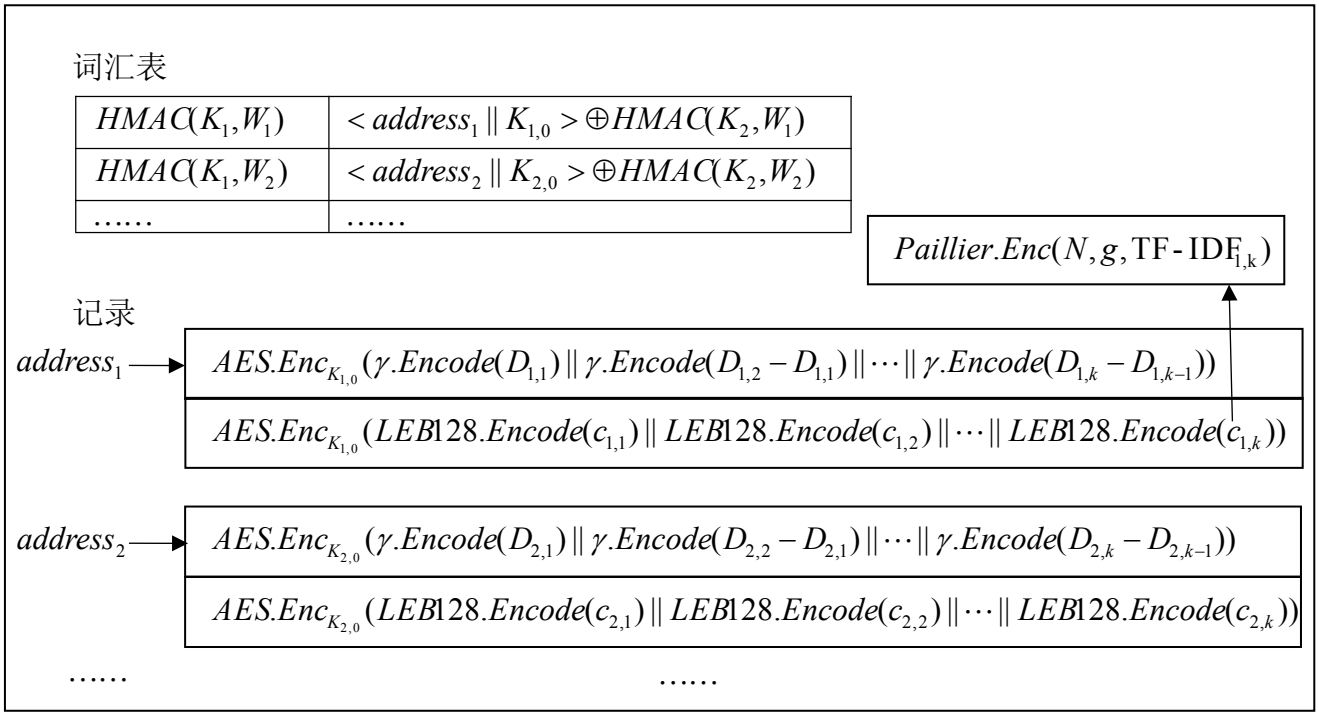
由于使用 Paillier 加密的相关度密文是一个大整数，为了方便存储和快速检索，需要使用某种编码方法将相关度集合编码存储。一种比较容易想到的方法是使用 BASE64[1]编码每个大整数所表示的字节并使用空格隔开连接成一个字符串。假设每个大整数密文具有相同的比特长度 x，总共有 N 个大整数，理论上的空间开销为 $X = x \times N$。采用此方法的空间开销 $X_1 = \frac{4}{3} \times x \times N + (N-1)$，与原来相比，需要的额外空间为 $\frac{1}{3} \times x \times N + (N-1)$，并不能令人满意。

类比 3.2.3 使用 γ 编码编码文档标号集合为一个字符串的思想，引入 LEB128[2]编码相关度集合。LEB128 是一种编码大整数的变长编码方式，基本思想是将大整数的二进制表示从低位开始编码，以 7 比特为一个基本单位，用一个字节进行编码，其中字节的首比特作为标识位。若标识位为 1，则表示此整数还包含后续字节；若标识位为 0，则表示此整数读取完毕。例如，整数 21332 的二进制表示为 1 0100110 1010100，则其 LEB128 编码为 11010100 10100110 00000001；在解码的时候，每次读取 1 个字节，直到读取到字节的首比特为 0 为止。丢弃所有字节的首比特，从低位开始解码，得到的结果为 101001101010100，也就是整数 21332。使用 LEB128 编码的主要原因是使用它能够使一组大整数不加边界地编码，而且编码后的空间开销较小。采用此方法的空间开销 $X_2 = \frac{8}{7} \times x \times N$，与原来相比，需要的额外空间为 $\frac{1}{7} \times x \times N$，显然比第一种方法要好。

为了增加服务器静态分析的难度，需要在相关度集合密文后随机填充比特串，然后

1 http://en.wikipedia.org/wiki/Base64
2 http://en.wikipedia.org/wiki/LEB128

再使用 AES 算法加密。这是因为与之前使用 γ 编码压缩文档标号集合不同，使用 LEB128 编码后的比特串长度与原来成正比。而使用 Paillier 加密后的 $TF\text{-}IDF_{i,j}$ 密文具有近似的长度，因此服务器可以根据密文长度推测出密文数量，进而推测出出现关键字的文档数量。尽管如此，实际上添加的随机比特串并不会影响 LEB128 解码后的结果。这是因为在检索时首先会通过解密前面的文档标号集合获取到正确的文档数量，然后再据此解码相关度集合。新的安全索引如图 3.6 所示：



（图 3.6：新安全索引）

3.3.3 新密文检索方案

在新安全索引中，文档标号集合和相关度集合是分开编码存放的。这样的存储方式对布尔检索还有额外的好处。因为在布尔检索中不涉及到相关度的计算，将两者分开编码存放后，节省了使用记录密钥对相关度集合解密的开销。

具体算法构造如图 3.7 所示：

$(PR,PU) \leftarrow Gen(1^k)$：

根据安全参数 k 选取 K_1, K_2, K_3，(N,g,λ,μ,N^2)，其中 K_1 为 AES 密钥；K_2, K_3 为 HMAC 密钥，(N,g,λ,μ,N^2) 为 Paillier 的参数，输出 $PR=(K_1,K_2,K_3,N,g,\lambda,\mu)$，$PU=(N^2)$。

$(I,C) \leftarrow Enc(PR,D)$：

1. 扫描文档集合 $D=(D_1,D_2,\cdots,D_n)$，并生成明文倒排索引 I

2. 将文档集合中的各个文档 D_j 进行压缩，并使用 AES 算法加密，即 $AES.Enc_{K_1}(Compress(D_j))$，得到密文集合 $C=(C_1,C_2,\cdots,C_n)$

3.将词汇表中存储的关键字 W_i 替换为 $HMAC(K_2,W_i)$

4.对每个关键字 W_i 对应的记录生成一个记录密钥 K_i，并将存储的记录地址 $address_i$ 替换为 $<address_i \| K_{i,0}> \oplus HMAC(K_3,W_i)$，其中||表示连接操作，⊕表示异或操作

5.记录分为两部分。第一部分保存文档标号集合。对首个文档标号 $D_{i,1}$ 进行 γ 编码，后续对每个文档标号 $D_{i,j}$ 与前一个文档标号 $D_{i,j-1}$ 的差值进行 γ 编码，最后表示为一个比特串，再使用记录密钥 $K_{i,0}$ 加密，即

$$AES.Enc_{K_i}(\gamma.Encode(D_{i,1}) \| \gamma.Encode(D_{i,2}-D_{i,1}) \| \cdots \| \gamma.Encode(D_{i,k}-D_{i,k-1}))$$

6.第二部分保存事先计算好的 $TF\text{-}IDF_{i,j}$ 集合，并使用 Paillier 加密算法加密，即 $c_{i,j}=Paillier.Enc(N,g,\mathrm{TF\text{-}IDF_{i,j}})$，然后再使用 LEB128 对密文进行编码，即

$$B=LEB128.Encode(c_{i,1}) \| LEB128.Encode(c_{i,2}) \| \cdots \| LEB128.Encode(c_{i,k})$$

7.设置一个阀门值 SS，若 LEB128 编码后的比特串长度小于 SS，则在后面随机填充比特串至 SS。最后使用 AES 加密比特串 B，即 $AES.Enc_{K_{i,0}}(B)$。

8.参照 SSE-1 随机填充无用索引块。假设 V 是 $|D|$ 个文档中可能包含的最大关键字数目，向安全索引填充无用索引块直至总索引块的数目为 V。

9.输出(I,C)。

$t \leftarrow Trpdr(PR,W)$：

计算 $HMAC(K_2,W_i)$ 和 $HMAC(K_3,W_i)$，分别记为 τ 和 η，输出 $t=(\tau,\eta)$

$X \leftarrow BooleanSearch(I,t)$：

1.将 t 解析为 τ 和 η；在安全索引 I 中查找 τ，若没有匹配项，则不存在包含关键字 W_i 的文档

2.若存在匹配项，在安全索引 I 中取出 $\varphi=<address_i \| K_{i,0}> \oplus HMAC(K_3,W_i)$，计算 $<address_i \| K_{i,0}>=\varphi \oplus \eta$，使用 $address_i$ 找到关键字 W_i 相应的记录，记为 $x=AES.Enc_{K_{i,0}}(\gamma.Encode(D_{i,1}) \| \gamma.Encode(D_{i,2}-D_{i,1}) \| \cdots \| \gamma.Encode(D_{i,k}-D_{i,k-1}))$，然后依次使用 $K_{i,0}$ 解密和 γ 编码解码，即 $\gamma.Decode(AES.Dec_{K_i}(x))$，最后利用文档标号差值恢复出文档标号集合

3.返回满足条件的文档标号集合 $X=(X_1,X_2,\cdots,X_k)$

$(X,S) \leftarrow RankedSearch(PR,PU,I,t)$：

1、2 步与 BooleanSearch 一致；

3.对 $TF\text{-}IDF_{i,j}$ 集合 B 先使用 AES 解密然后使用 LEB128 解码，即 $(c_{i,1} \| c_{i,2} \| \cdots \| c_{i,k})=LEB128.Decode(AES.Dec_{K_{i,0}}(B))$，然后再根据具体情况，利用 Paillier 加密的加法同态性累加同一文档 D_j 的相关度，即 $c_x \times c_y \bmod N^2$

4.返回满足条件的文档标号集合 $X=(X_1,X_2,\cdots,X_k)$，以及 $S=(S_1,S_2,\cdots,S_k)$

5.使用 Paillier 算法解密 S，即 $Paillier.Dec(N,\lambda,\mu,S)$，并按照文档相关度大小对文档标号集合进行排序

$D_j \leftarrow Dec(K,C_j)$：

计算 $Decompress(AES.Dec_{K_1}(C_j))$，得到 D_j

（图 3.7：新密文检索方案）

3.4 修改安全索引

3.4.1 引言

本节进一步讨论在已有文档集合的前提下，用户新增、更新或删除文件对安全索引的影响。与传统信息检索系统不同，服务器在缺乏客户端的帮助下无法识别安全索引中的任何内容，因此，提供一种高效便捷的修改安全索引方法是密文检索系统的一个重要组成部分。

首先，分析在传统信息检索系统中新增索引的技术手段，提出一种对排名检索的准确性略有损失和泄漏轻微信息的高效索引新增方法。然后，进一步讨论删除文件对安全索引的影响，并由此提出一种修复上述缺陷的改进手段。最后，提出更新文件的高效方法。

3.4.2 新增文件

在传统信息检索系统中，服务器一般对新增的文档集合生成一个新索引，在检索时，同时检索两个索引。根据具体情况，服务器也有可能把新索引与旧索引合并。在密文检索系统中，由于服务器在缺乏用户的帮助下无法识别安全索引中的任何内容，修改索引的工作只能由用户完成。因此，在新增文件时，采用与原来初始化索引时相同的方法，但提供一个不与旧文档标号冲突的开始文档标号，然后将新索引上传到服务器。服务器同时保存新索引与旧索引，在检索时，同时检索两个索引。

对于排名检索来说，由于计算项权重时所选取的参数是针对局部文档集合而不是全局文档集合，因此准确性会有所偏差。在另一方面，服务器可以静态分析两个索引中是否存在相同关键字，因此可推测出两个文档集合共同包含的密文关键字集合。

3.4.3 删除文件

为了支持删除操作，服务器要为每个用户维护一个文档标号列表。当用户需要删除文件时，服务器删除该加密文件，和用户文档标号列表中相应的文档标号。由于删除操作并不涉及到索引的修改，在检索结果返回前，需要先检查用户文档标号列表，确保检索返回的文档标号集合真实存在于服务器。虽然废弃的文档标号索引会占用服务器的空间，但不会影响到检索结果的正确性。此外，若用户每隔一段时间将全部文件重新生成一次索引，则不仅可以清理无效索引，还可修复此前新增文件时所造成的缺陷。

3.4.4 更新文件

更新文件，可分解为删除文件和新增文件两个操作。理论上，服务器无法判断两个加密的文件是否相同，因此服务器无法自动识别上传的文件是新文件还是旧文件的新版本。因此，采用先删除文件，再新增文件的方法，是一种高效的更新文件方法。

3.5 本章小结

本章首先分析 SSE-1 的安全索引记录在时间和空间上还存在可改进的地方，然后引入 γ 编码对文档标号集合进行压缩，由此提出一种优化后的密文检索方案。进而引入 Paillier 算法，利用其加法同态性质使服务器具有累加项权重密文的能力，检索结果的排序工作由客户端完成。更进一步，使用 LEB128 编码对项权重密文进行不分边界地编码，再将其使用 AES 加密，获得更好的空间利用率。至此，提出一个同时支持布尔检索和多关键字排名检索的密文检索方案。

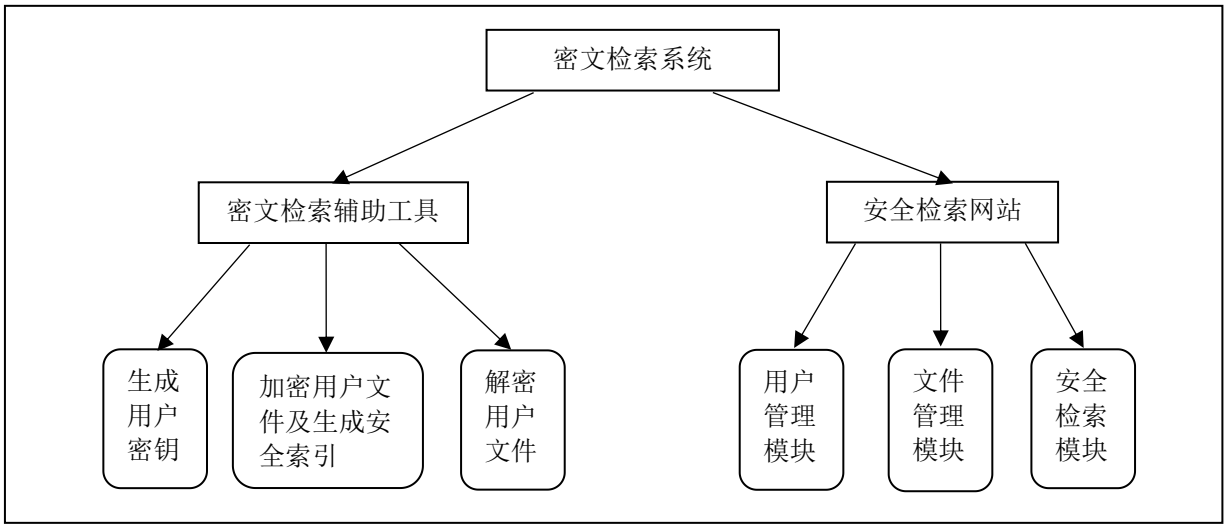
与提供相同检索能力的密文检索方案[8, 9, 10]相比，本章提出的密文检索方案索引的空间开销更小，检索效率更高。文献[8, 9]中的密文检索方案，由于排名检索结果的排序工作由服务器完成，检索过程需要对密文进行更多的操作，而本章提出的密文检索方案把检索结果的排序交由客户端完成，获得了更高的检索效率。诚然，文献[8, 9]中的密文检索方案由于服务器具有更强的能力，更容易进行扩展，但在目前只支持布尔检索和多关键字排名检索的前提上，文献[8, 9]中的密文检索方案并不具备明显优势。文献[10]中的密文检索方案虽然将检索结果的排序交由客户端完成，但由于需要对全同态加密后的密文进行向量运算，因而检索效率并不高。

本章提出的密文检索方案主要是从实践的角度出发，在放弃使服务器具有密文排序的能力后，获得更高的检索效率，而且更易于实现。此外，本章更进一步讨论新增、删除以及更新文件对安全索引的影响，由此提出一种以牺牲轻微排名检索准确性以及安全性为代价的高效索引修改方法。

4. 密文检索系统的实现

4.1 系统架构

本系统由密文检索辅助工具和安全检索网站构成。密文检索辅助工具有三个主要功能，分别是生成用户密钥、加密用户文件及生成安全索引、和解密用户文件。安全检索网站分为三个主要模块，分别是用户管理模块、文件管理模块和安全检索模块。密文检索系统架构如图 4.1 所示：



（图 4.1：密文检索系统的系统架构）

为了使用密文检索功能，需要将密文检索辅助工具和安全检索网站搭配使用。考虑到这两个主要构件在功能上独立性较强，这里有必要说明一下用户基本的检索流程。具体来说，一个新用户的初次检索流程如图 4.2 所示：

1.注册和下载密文检索辅助工具。用户首先在安全检索网站上注册，然后下载密文检索辅助工具。

2.生成用户密钥和上传用户公钥。用户使用密文检索辅助工具生成用户密钥。密钥分为两个部分，用户私钥，需要用户自己保存；需要上传到服务器的用户公钥，用于服务器提供多关键字排名检索功能。用户登录安全检索网站，将用户公钥更新到用户信息中。

3.加密文件及生成安全索引，并将其上传到安全检索网站。用户通过指明用户密钥，使用密文检索辅助工具将文档集合压缩加密并生成安全索引。用户登录安全检索网站上传密文文档集合和安全索引。

4.安全检索。在检索前，用户必须保证已上传公钥。当用户在检索页面指定用户密钥后，浏览器读取其中内容并保存为当前检索页面的临时变量。提交查询后，检索结果以列表的形式返回，用户可下载需要的文件。

5.解密密文文件。用户使用密文检索辅助工具解密并解压下载的文件，得到原文件。

（图 4.2：用户基本的检索流程）

4.2 密文检索辅助工具

4.2.1 生成用户密钥

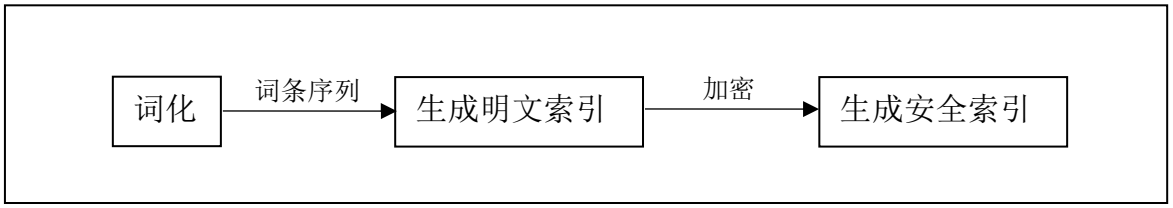
为了使用密文检索功能，用户需要生成一个 AES 密钥、两个 HMAC 密钥以及一对 Paillier 公私钥。特别地，本系统选用 Java 的 crypto 库提供的 AES/ECB/PKCS5Padding 加密算法和 HmacSHA256 加密算法。此外，基于 Java 的 BigInteger 结构构造了一个 Paillier 算法。

在使用生成用户密钥功能后，用户得到 keyFile 和 serverKey。具体来说，keyFile 保存用户私钥，其中包含 7 个参数，分别是 $K_1, K_2, K_3, N, g, \lambda, \mu$。serverKey 保存用户公钥，对应 Paillier 加密算法的 N^2，需要上传到服务器。

由于生成的密钥是二进制数据的，为了显示和读取的方便，使用 BASE64 进行编码存储。BASE64 是一种广泛应用的存储二进制数据的编码方法，用来保存密钥非常合适。

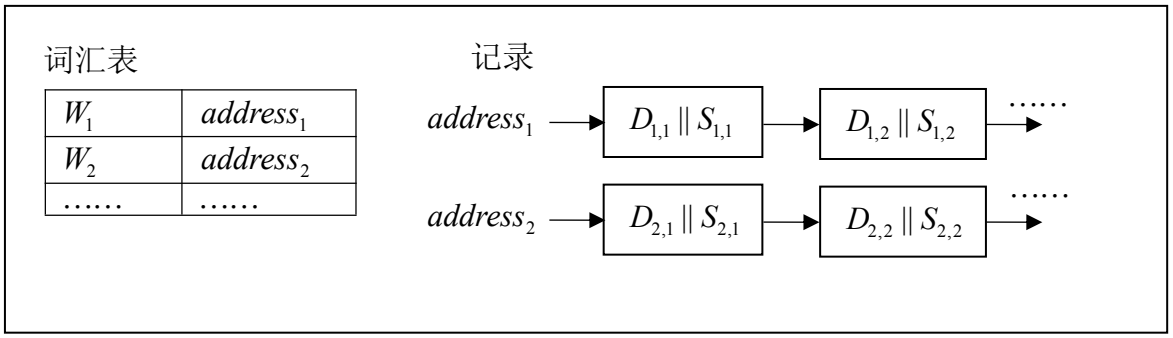
4.2.2 加密用户文件及生成安全索引

为了实现的简便，目前只支持对 TXT 类型的文件进行索引，对其他文件进行索引超出本文的研究范围。由于检索过程不涉及原文件，为了节省存储空间，采用先压缩后加密的方式处理原文件。这里使用 apache 软件基金会的 commons-compress 库[3]的 GZIP 压缩方法，对原文件进行压缩，即 $C_i = AES.Enc_{K_1}(GZIP.Compress(D_i))$。



（图 4.3：安全索引的生成流程）

生成安全索引的流程如图 4.3 所示，主要涉及几个关键步骤：词化（Stemming）[13]、生成明文索引以及生成安全索引。词化，是一个将文档中的词条（Token）分离出来的过程。本系统只是简单地将每个英文单词或中文单字抽取出来，并忽略大小写，不支持对短语进行索引。得到词条序列后，使用一般信息检索领域生成倒排索引的方法，生成明文索引。具体来说，生成的明文索引如图 4.4 所示：

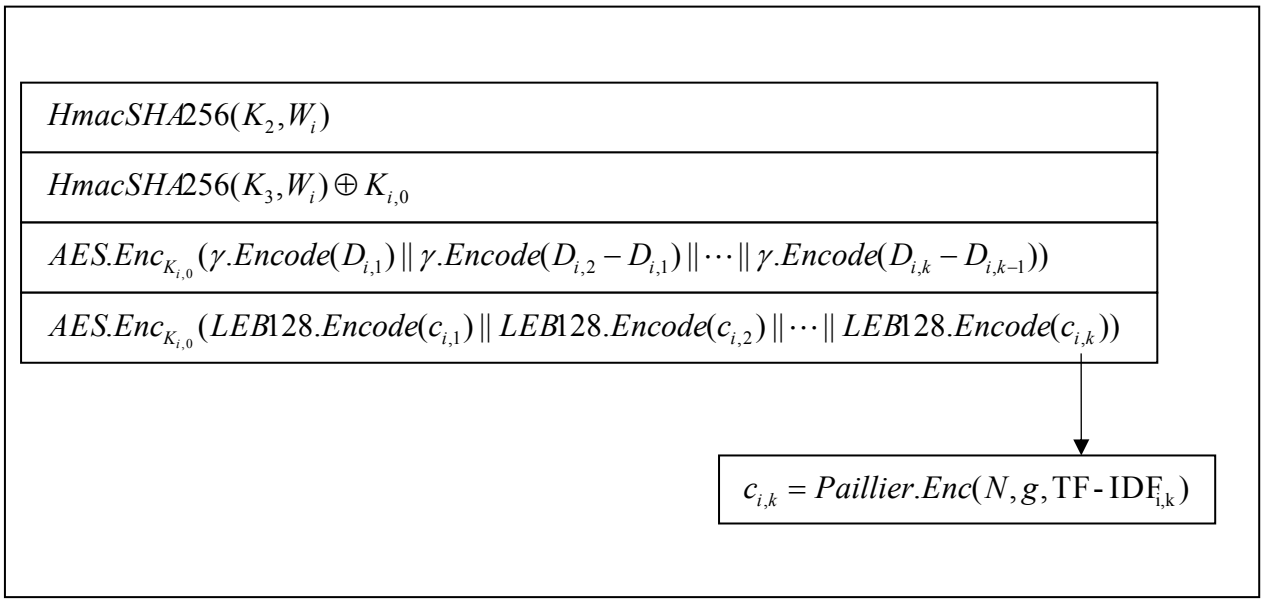


（图 4.4：明文索引）

3 http://commons.apache.org/proper/commons-compress/

文献[13]介绍了一些索引构建的技术，由于索引构建不是本文的核心，因此本系统只使用最简单的索引构建技术，假设用户的内存能够满足在生成索引过程中的内存需要。因此，本系统使用一个 HashMap 来存储倒排索引的词汇表，使用 ArrayList 存储倒排索引的记录。为了支持排名检索，记录需要同时保存文档标号 $D_{i,j}$ 及其对应的项权重 $S_{i,j}$。

将明文索引加密后，可得到安全索引。另外，为了方便多个用户的索引管理，安全索引存储在服务器的数据库里，也就是说安全索引的一个索引项对应数据库中的一个表记录，因此索引结构不再区分词汇表和记录。具体来说，安全索引由四个部分组成：关键字入口、陷门、加密的文档标号集合和加密的相关度集合，如图 4.5 所示：



（图 4.5：安全索引）

这里对安全索引的结构做一些补充说明。生成陷门时，由于 $K_{i,0}$ 长度较小，为了方便在检索时恢复 $K_{i,0}$，计算异或运算时只截取了 $HmacSHA256(K_3, W_i)$ 的低 $|K_{i,0}|$ 位。此外，由于明文索引保存的相关度是实数，Paillier 加密只能对整数进行加密，因此在加密前，截取相关度的两位小数，即 $floor(X \times 100)$。

在另一方面，由于 γ 编码是基于比特操作的编码方法，本系统使用 ArrayList<Byte> 构造了一个基于比特操作的数据结构 BitArray，基本思想是利用 1 个 Byte 来存储 8 比特，利用 ArrayList 来实现变长数组。然而，由于 AES 加密算法是基于 Byte 加密的，若使用原始的 γ 编码方法，解码后可能出现错误的结果。例如：文档标号集合为 1、3、7，用 γ 编码编码后的比特串应为 010011000，总长度为 9 个比特，加密的时候占用两个字节，因此加密前的比特串为 0100110000000000。但对该比特串解码的结果为 1、3、7、8、9、10、11、12、13、14，这是因为填充的 0 解码后为 1。因此需要对 γ 编码的编码方法作适当修改，当最后一个字节没有被完全使用，则填充 1，在上面的例子中就是 0100110001111111。由于 γ 编码解码时读取一元编码读到 0 才表示读完，因此填充的 1 不会对解码造成影响。

为了便于上传索引文件到服务器，需要把安全索引存储在一个文档中。因此，引入BASE64 分别编码安全索引的四个部分，并以空格隔开。虽然使用 BASE64 编码后的数据是原来的 4/3，但由于服务器在读取索引文件后会将其使用 BASE64 解码再保存到数据库，因此这部分的额外空间开销是暂时的。此外，为了检索结果的可读性，需要在索引文件中额外需要保存各个文档标号对应的文档名称。在服务器读取索引文件时，一并将它们读取到数据库。

4.2.3 解密用户文件

当用户下载到所需的密文文件后，通过指定用户密钥，即可解密密文文件。与加密相对应，此处先用 AES 解密，再用 commons-compress 库的 GZIP 方法解压缩得到原文件，即 $D_i = GZIP.Decompress(AES.Dec_{K_1}(C_i))$。

4.3 安全检索网站

4.3.1 用户管理模块

安全检索网站与传统信息检索系统不同，隔离了多个用户之间的索引和文档集合，单独给每个用户提供检索功能。具体来说，一个用户在数据库中表示为一个数据库表记录，每个用户的 ID 是唯一的，因此可通过用户 ID 来区分不同用户的索引和文档。

具体来说，用户的账户数据存储在表 account(id, username, password, addKey, capacity)，其中 id 表示用户 ID，用户注册时由系统自动生成；username 表示用户名；password 表示盐和加盐后的用户密码；addKey 表示用户检索时所需的用户公钥；capacity 表示用户可用上传空间。

这里需要特别说明的是 password 域。出于账户安全的考虑，用户密码并不明文保存在数据库中，而是采用加盐操作。本系统使用 PBKDF2（Password-Based Key Derivation Function 2）[4]算法来实现密码加盐操作。PBKDF2 是一种专门用于密码派生的算法，具有良好的安全性。具体来说，本系统选用 Java 的 crypto 库的 PBKDF2WithHmacSHA1 算法，在用户注册时随机生成一个“盐”，然后使用此算法计算加盐后的密码，将盐和密码以空格隔开同时保存在 password 域。

在验证密码时，将提交的密码和数据库中的盐重新计算加盐后的密码，然后与旧密码进行比较，若一致则登录成功。在修改密码时，验证旧密码步骤和上面类似，比对成功后重新生成盐与新密码进行计算，最后将结果存储到数据库的 password 域。

用户的文档数据存储在表 doc(id, accountId, docId, name)，其中 id 表示文档的数据库 ID，由系统自动生成；accountId 表示拥有文档的用户 ID；docId 表示文档在用户文档集合中的 ID，依赖于用户生成索引时的配置；name 表示文档名称。

用户的索引数据存储在表 term(id, accountId, name, trapdoor, docIds,

4 http://en.wikipedia.org/wiki/PBKDF2

scores)，其中 id 表示索引项的 ID，由系统自动生成；accountId 表示拥有此索引项的用户 ID；name 表示安全索引中的关键字入口，由于长度固定且较短，为了便于数据库检索，以字符串的形式保存；trapdoor 表示安全索引中的陷门，以二进制的形式保存；docIds 表示安全索引中的加密文档标号集合，以二进制的形式保存；scores 表示安全索引中的加密相关度集合，以二进制的形式保存。特别地，为了提高检索效率，在数据库中对(accountId, name)进行索引。

4.3.2 文件管理模块

文件管理模块提供用户管理上传文件的功能。文件管理模块包括三个功能：上传文件、下载文件以及删除文件。

上传文件时须包含加密文档集合及其对应的安全索引，否则检索功能不能正常使用。对于加密文档集合，服务器单独保存在各个用户独立的目录里。对于安全索引，服务器首先读取索引文件中的文档标号及其对应的文档名称，然后使用 BASE64 解码安全索引，最后保存到数据库。此外，服务器根据用户上传的文件大小，更新用户剩余的可用容量。在另一方面，为了防止用户提交具有相同文件名的文件而造成的新文件覆盖旧文件情况，加密文件会在服务器上重新命名为 docId+name，确保不会出现旧文件被错误替换的情况。对于非首次上传文件的情况，除了在使用密文检索辅助工具生成索引时给出开始文档标号外，其他方面并无不同。

对于下载文件，服务器只需在用户目录中将特定文件发送给用户即可。删除文件包括删除用户目录的文件以及对应的数据库表记录，然后更新用户剩余的可用空间。

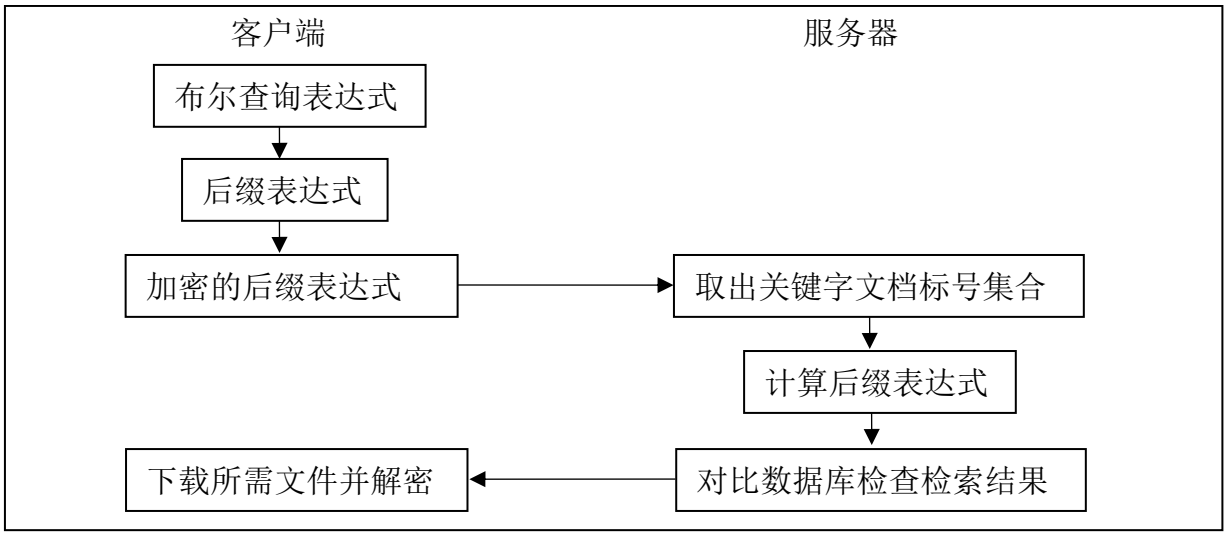
4.3.3 安全检索模块

安全检索模块是安全检索网站的核心。用户在登录并上传文件后，可进行安全检索。在检索前，用户需要确保已上传用户公钥到服务器。为了在浏览器上加密用户提交的查询，用户在检索前需要指定密钥文件。出于密钥安全的考虑，浏览器会将读取到的密钥保存为当前页面的临时变量。为了提供最大的便利性，只要用户不主动离开检索页面，就无需再次指定密钥文件。

在另一方面，用户提交查询时无需手动选择检索类型，因为浏览器会通过判断查询是否包含布尔符号来自动确定检索类型。特别地，本系统使用开源的 jsSHA 库[5]来提供 HmacSHA256 功能，以及基于开源的 BigInteger 库[6]实现 JS 端的 Paillier 算法。

布尔检索的流程（图 4.6）和算法构造（图 4.7）如下所示：

5 https://github.com/Caligatio/jsSHA
6 https://github.com/peterolson/BigInteger.js



（图 4.6：布尔检索流程）

1.用户在搜索栏输入布尔表达式：$W_1(\&|-)W_2(\&|-)\cdots(\&|-)W_k$，其中$(\&|-)$指布尔符号&、|或-，分别表示交集、并集和相对补集，可使用括号改变运算顺序。

2.浏览器将中缀的布尔表达式转换为后缀表达式$W_1W_2(\&|-)W_3,\cdots,(\&|-)$。在此过程中，检查关键字 W_i 是否为中文。若 W_i 是中文，将 W_i 分拆为中文单字，并从中插入布尔符号&；若 W_i 是英文单词，则做忽略大小写处理。

3.浏览器利用 HmacSHA256 密钥 K_2 和 K_3，将后缀表达式除布尔符号外的关键字 W_i 按顺序计算 $HmacSHA256(K_2,W_i)$ 和 $HmacSHA256(K_3,W_i)$，得到关键字序列和陷门序列：$HmacSHA256(K_2,W_1),HmacSHA256(K_2,W_2),(\&|-),HmacSHA256(K_2,W_3),\cdots,(\&|-)$、$HmacSHA256(K_3,W_1),HmacSHA256(K_3,W_2),(\&|-),HmacSHA256(K_3,W_3),\cdots,(\&|-)$，然后使用 BASE64 编码并提交到服务器。

4.服务器对它们使用 BASE64 解码后，对关键字序列进行预处理。对关键字序列中的每一个关键字，服务器利用 $HmacSHA256(K_2,W_i)$ 和用户 ID 在数据库表 term 中查找对应记录。若表记录存在，将其 trapdoor 字段与解码后的 $HmacSHA256(K_3,W_i)$ 进行异或操作恢复出 AES 记录密钥，即 $K_{i,0}=HmacSHA256(K_3,W_i)\oplus trapdoor$，然后利用记录密钥对 docIds 字段进行解密并使用 γ 编码解码，通过差值计算可得到该关键字的文档标号集合，即 $\gamma.Decode(AES.Dec_{K_{i,0}}(docIds))$。因此，得到各个关键字对应的文档标号集合。

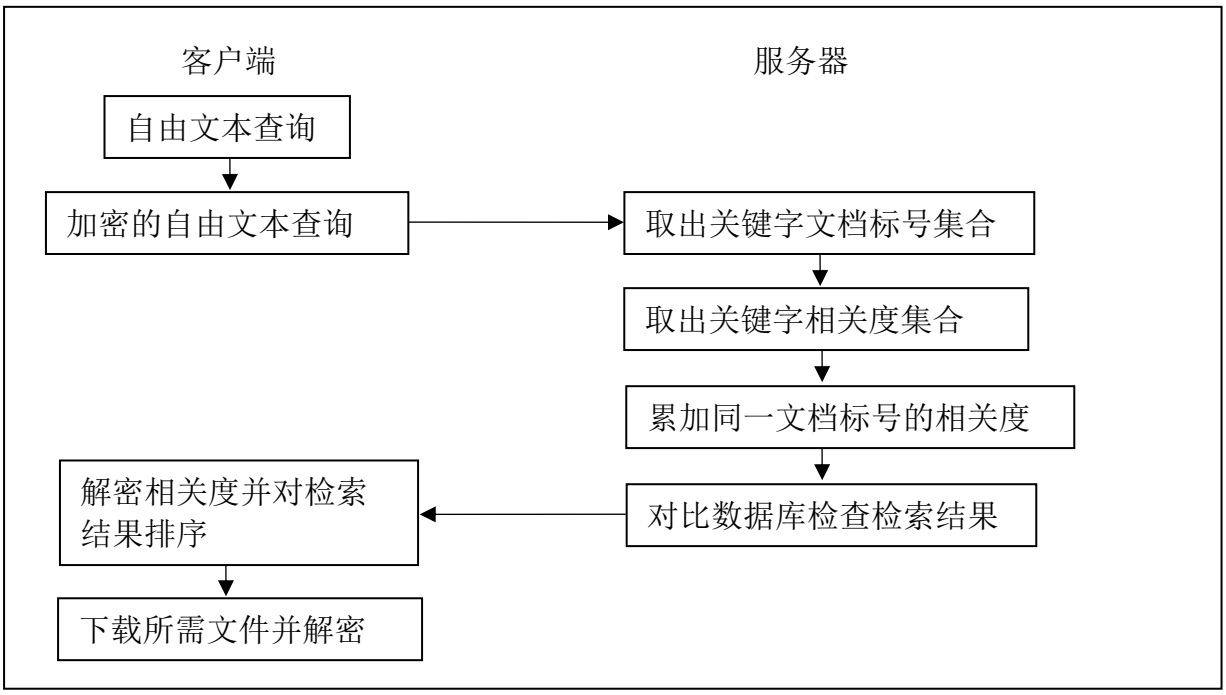
5.服务器计算后缀表达式并得出索引检索结果。计算后缀表达式基于三个集合操作，交集，并集和相对补集。进而，依据运算优先级计算后缀表达式，可得到检索结果。

6.服务器将检索结果与数据库表 doc 进行比对，确保返回结果的正确性，然后将结果发送给用户。

7.用户从获取到的检索结果中，下载所需的文件。

（图 4.7：布尔检索算法构造）

排名检索的流程（图 4.8）和算法构造（图 4.9）如下所示：



（图 4.8：排名检索流程）

1.用户在搜索栏输入查询：$W_1W_2\cdots W_k$

2.浏览器检查关键字 W_i 是否为中文。若 W_i 是中文，则将 W_i 分拆为中文单字；若 W_i 是英文单词，则做忽略大小写处理。

3.浏览器利用 HmacSHA256 密钥 K_2 和 K_3，将查询中的关键字 W_i 按顺序计算 $HmacSHA256(K_2,W_i)$ 和 $HmacSHA256(K_3,W_i)$，得到关键字序列和陷门序列：$HmacSHA256(K_2,W_1),\cdots,HmacSHA256(K_2,W_k)$、$HmacSHA256(K_3,W_1),\cdots,HmacSHA256(K_3,W_k)$，然后使用 BASE64 编码并提交到服务器。

4.服务器对它们使用 BASE64 解码后，如布尔检索那样恢复出文档标号集合，并取出表记录中的 scores 字段。进而依次使用 AES 解密和 LEB128 解码得到使用 Paillier 加密后的相关度集合，即 $(c_{i,1}\|c_{i,2}\|\cdots\|c_{i,k})=LEB128.Decode(AES.Dec_{K_{i,0}}(scores))$。服务器使用一个 HashMap 缓存结果，然后利用 Paillier 的同态加密性质累加各个文档的相关度。最后得到包含文档标号及其对应相关度的检索结果。

5.服务器将检索结果中的文档标号与数据库表 doc 进行比对，确保返回的结果正确性，然后将结果发送给用户。

6.浏览器利用 Paillier 密钥对检索结果中的文档相关度进行解密，即 $Paillier.Dec(N,\lambda,\mu,S)$，然后根据文档相关度对检索结果进行排序。

7.用户从获取到的检索结果中，下载所需的文件。

（图 4.9：排名检索算法构造）

4.4 系统测试与分析

测试平台的关键配置：(1)CPU：i5-2410M（双核 2.3GHz）；(2)内存 4G；(3)Ubuntu 14.04 系统。对于检索操作，多次使用不同的查询来测试。具体来说，通过多次对 rfc1-500 和 rfc501-1000 两个数据集合分别进行测试，得出如下测试数据：

rfc1-500 文档集合：a)原文档集合达大小为 5.0M，生成索引文件大小为 15.5M，密文文档集合大小为 1.6M；生成明文索引所需时间为 3540MS，生成安全索引所需时间为 19571MS；安全索引包含 42451 个索引项。b)服务器将索引文件读取到数据库所需时间为 7638MS；排名检索所需时间为 1625MS；布尔检索所需时间为 1491MS。

rfc501-1000 文档集合：a)原文档集合达大小为 14.8M，生成索引文件大小为 31.2M，密文文档集合大小为 4M；生成明文索引所需时间为 5280MS，生成安全索引所需时间为 38565MS；安全索引包含 82837 个索引项。b)服务器将索引文件读取到数据库所需时间为 18269MS；排名检索所需时间为 2221MS；布尔检索所需时间为 1784MS。

在提取出明文索引后，生成安全索引和加密密文文档这两个过程互不影响，因此分别使用两个不同的线程来完成。由于生成安全索引的时间开销远比加密密文文档大，将文档压缩后再加密的优势就非常明显，即没有明显增加时间开销的同时减小空间开销。

生成的索引文件一般约为原文档集合总大小的 2.5 倍，这主要有两个原因。首先，本系统使用的词化算法相对简单，造成生成的索引项较多；其次，使用 Paillier 加密的相关度是一个大整数，与明文相比，所需的存储空间非常大。

从上述数据可以看出，即使索引文件很大，检索的效率还是相对高效的。排名检索与布尔检索所需时间相差不大的原因是，排名检索需要做相关度集合的计算，而布尔检索需要对文档集合做集合运算。

4.5 本章小结

本章实现一种由密文检索辅助工具和安全检索网站构成的密文检索系统，在功能上同时支持布尔检索与多关键字排名检索。针对实现过程中出现的一些编码问题，给出了解决的方法。在另一方面，本系统还对不同用户的索引进行隔离，并向用户提供文件管理功能。

与此同时，本系统还存在着一些不足。因为词化和索引构建等部分并不是本文的研究重点，在实现时进行适当的简化。具体来说，在词化时只是简单地将每个中文单字和英文单词抽离出来，使得安全索引占用的空间较大。在索引构建时也只是简单地将索引保存在内存中，没有考虑内存不足以至于需要在磁盘上保存临时索引再合并的过程。

在另一方面，限于我们的认识，暂时还没有找到令人满意的构造安全短语索引结构的方法，这在一定程度上弱化本系统的可用性。由于在索引构建时没有对短语进行索引，因此本系统无法对短语进行检索，在中文语境下问题尤为突出。例如，使用排名检索的方式来检索“上海”，检索结果包含的文档主要有 3 种情况：只包含“上”的，只

包含“海”的，同时包含“上”和“海”的。使用布尔检索时，检索结果同时包含“上”和“海”。但总的来说，检索结果并不能保证返回的文档包含“上海”这个短语。

在传统信息检索系统中，实现短语查询的一种常用方式是采用位置信息索引[13]。在位置信息索引中，倒排记录除了存储文档标号外，还需要存储文档中出现的关键字位置信息，即：$docId:<position_1,position_2,\cdots>$。在检索的时候，通过比较不同关键字的位置信息，确定它们是否相邻来达到检索短语的功能。例如，关键字“Searchable”的倒排记录表为$1:<1,78>$，关键字“encryption”的倒排记录表为$1:<2>$。在检索短语“Searchable encryption”的时候，通过逐个比较它们的倒排记录表，可以发现“Searchable”和“encryption”的位置相邻。要在密文检索系统中提供对短语的检索，服务器需要具备密文相加和密文比较的能力。本文提出的密文检索方案无法做到这一点。

5. 总结

密文检索技术是一种允许服务提供商对用户事先加密的文件进行检索的全文检索技术，防止包括服务商在内的第三方无法获取文件的任何信息。应用密文检索技术，用户可以将私密文件加密后存放到由服务提供商托管的数据中心上，而且可以对加密后的文件进行全文检索。

本文首先通过分析传统信息检索系统的基本功能需求，对目前已知的密文检索方案进行深入的剖析，并进一步分析它们的优势与不足。特别地，目前最新的密文检索方案可支持布尔检索和多关键字排名检索。据此，本文的设计目标是从实践的角度出发，设计一种支持布尔检索与多关键字排名检索的密文检索方案，进一步使得索引空间开销更小，检索效率更高。

为了达到这一目标，本文使用信息检索系统中常用来压缩索引的γ编码来优化 SSE-1 的安全索引，然后引入 Paillier 加密算法并利用其加法同态性质以支持多关键字的相关度密文相加，然后由客户端来解密相关度密文并对检索结果进行排序。为了更进一步缩小安全索引的空间开销，本文引入 LEB128 编码来编码 Paillier 加密后的相关度密文，再由 AES 加密编码后的比特串。至此，一种新的密文检索方案被设计出来。

与最新的密文检索方案[8, 9]相比，虽然服务器的能力被弱化，但检索的效率得到提高。具体来说，文献[8, 9]提出的密文检索方案使服务器具有密文排序的能力，即对于多关键字的排名检索，检索结果的排序由服务器完成；而本文提出的密文检索方案，检索结果的排序由客户端完成。诚然，出于未来可能出现的扩展需求，服务器具备更强的能力会有更大的优势。然而，在支持布尔检索和多关键字排名检索这一功能需求上，检索结果的排序由客户端来完成，在一定程度上简化实际应用的难度，而且检索效率会有所提高。

随后，我们利用这一密文检索方案，实现一种由密文检索辅助工具和安全检索网站构成的密文检索系统，在实践中验证了新方案的可用性和正确性。此外，详细分析密文检索系统中的各个功能模块及其相关的实现技术。

最后，通过测试与分析发现系统并不能支持短语的检索。限于我们的认识，暂时无法解决这一问题。事实上，应用文献[9]提出的密文检索方案有望能解决这一难题。因为文献[9]提出的密文检索方案使服务器具有密文相加和密文排序的能力，而密文排序的关键就是两密文的比较。正如 4.5 的分析，密文短语查询要求服务器具有密文比较和密文相加的能力。

在另一方面，除了布尔检索和排名检索，信息检索领域还存在一些更加先进的检索模型，例如概率模型、扩展布尔检索模型、广义向量模型、神经网络模型等等。希望通过对信息检索领域和密码学领域的更深入研究，后续可以使密文检索系统进一步满足更强的检索功能需求。

参考文献

[1] Witten I H, Moffat A, Bell T C. Managing gigabytes: compressing and indexing documents and images[M]. Morgan Kaufmann, 1999.

[2] Song D X, Wagner D, Perrig A. Practical techniques for searches on encrypted data[C]. Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. IEEE, 2000: 44-55.

[3] Goh E J. Secure Indexes[J]. IACR Cryptology ePrint Archive, 2003, 2003: 216.

[4] Chang Y C, Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data[C]. Applied Cryptography and Network Security. Springer Berlin Heidelberg, 2005: 442-455.

[5] Curtmola R, Garay J, Kamara S, et al. Searchable symmetric encryption: Improved definitions and efficient constructions[J]. Journal of Computer Security, 2011, 19(5): 895-934.

[6] Swaminathan A, Mao Y, Su G M, et al. Confidentiality-preserving rank-ordered search[C]. Proceedings of the 2007 ACM workshop on Storage security and survivability. ACM, 2007: 7-12.

[7] Wang C, Cao N, Li J, et al. Secure ranked keyword search over encrypted cloud data[C]. Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on. IEEE, 2010: 253-262.

[8] Cao N, Wang C, Li M, et al. Privacy-Preserving Multi-keyword Ranked Search over Encrypted Cloud Data[J]. Proc. IEEE INFOCOM, 2011: 829-837

[9] Baldimtsi F, Ohrimenko O. Sorting and searching behind the curtain[J]. IACR Cryptology ePrint Archive, 2014, 2014: 1017

[10] Varghese J, Varghese L. Homomorphic Encryption for Multi-keyword based Search and Retrieval over Encrypted Data[J]. International Journal of Application or Innovation in Engineering & Management(IJAIEM), 2014, 3(8): 138-146.

[11] Waters B R, Balfanz D, Durfee G, et al. Building an Encrypted and Searchable Audit Log[C]. NDSS. 2004, 4: 5-6.

[12] Baeza-Yates R, Ribeiro-Neto B. Modern information retrieval[M]. Second Edition. Addison-Wesley Professional. 2011.

[13] Manning C D, Raghavan P, Schütze H. Introduction to information retrieval[M]. Cambridge: Cambridge university press, 2008.

[14] Gentry C. A fully homomorphic encryption scheme[D]. Stanford University,

2009.

[15] Togan M, Plesca C. Comparison-based computations over fully homomorphic encrypted data[C]. Communications (COMM), 2014 10th International Conference on. IEEE, 2014: 1-6.

[16] Halevi S, Shoup V. Design and Implementation of a Homomorphic-Encryption Library. 2013. available at http://researcher.ibm.com/researcher/files/us-shaih/he-library.pdf. Accessed April 2015

[17] Paillier P. Public-key cryptosystems based on composite degree residuosity classes[C]. Advances in cryptology—EUROCRYPT'99. Springer Berlin Heidelberg, 1999: 223-238.

[18] Li J, Chen S, Song D. Security structure of cloud storage based on homomorphic encryption scheme[C]. Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on. IEEE, 2012, 1: 224-227.

致谢

衷心感谢我的指导老师王立斌副教授。自大一以来，王老师一直在学习和生活上对我给予帮助。与王老师的交流和讨论，加深了我对计算机科学的认识，使我对计算机科学产生了极大的兴趣。不仅如此，与王老师一起愉快地交流文学作品、讨论跑步心得，这些都是不可多得的时光。在学习上，王老师总是不厌其烦地解答我的疑问，对我提出的问题指明方向。在生活上，王老师独特的见解使我受益匪浅。每当我陷入低谷的时候，王老师适时的指导总能使我不再局限于自身的视野去看问题，让我可以更加理性地思考问题，对我的成长起到重大作用。在四年大学生涯里，我很幸运能遇到王老师，更加感谢他对我的无私的帮助。

感谢大学四年来教导我的老师，特别感谢黄定老师、黄煜廉老师、张奇支老师等等。与他们的交流和讨论，对我的成长不可或缺。

最后感谢我的家人，感谢他们一直以来对我的支持和关心。