一、SSE 删除学习笔记

目录

二、读书笔记

（一）《Dynamic Searchable Symmetric Encryption with Physical Deletion and Small Leakage》

时间：2017

会议：ACISP（C 会）

主要内容：

本文提出了一种同时支持逻辑删除与物理删除的方法，并且在搜索性能、存储开销、功能性、信息泄露达到了较的折中。

（1）在参考文献 1 的基础上，新增了 AddKeyword 及 DeleteKeyword 的功能，新定义了 IND-CKA2 安全（indistinguishability under adaptively chosen keyword attacks）

（2）本文的亮点及难点在于删除。本文支持 2 种逻辑及物理删除方式。逻辑删除时，仅将链表 K1 元素标志位置“1”；而物理删除在 K2、K3 之中删除元素重排链表。不同于以往方案，其在检索阶段进行。

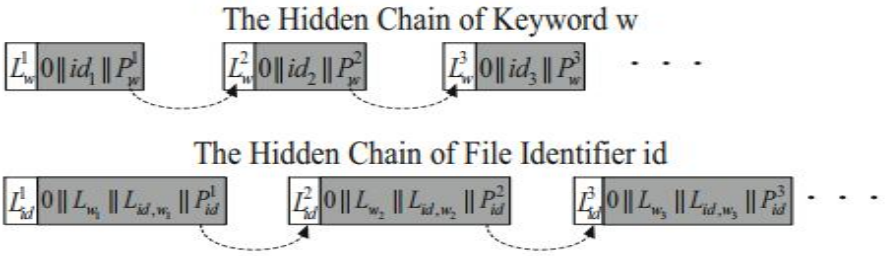
（3）本文仅仅支持添加关键词，而添加文件可以认为是多次添加关键词。

（4）本文给出了 3 个方案（包含 2 个过渡方案）。

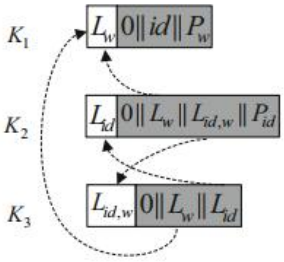
1）过渡方案 1 用于展示构造一个 hidden chain，其支持支持在链上插入密文及在脸上检索关键词；

2）过渡方案 2 用于展示如何同时支持逻辑删除及物理删除。

3）最终方案维护了 K1、K2、K3 个链表，K1 用于检索及逻辑删除、K2 用于删除文件及物理删除、K3 用于删除关键词及物理删除，每个链本身通过指针相连，如图：



每个 K1、K2、K3 节点的相互依赖关系如图。

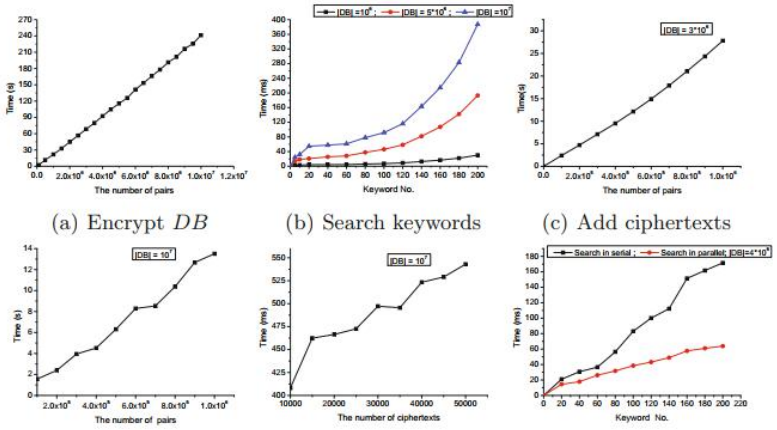


（5）作者对本文 DSSE 方案与之前方案进行了对比并进行了实验，如图

Table 2. Exact comparisons

Scheme	**Search** complexity	Storage complexity	Leakage functions				
			$\mathcal{L}_{Setup}$	$\mathcal{L}_{Search}$	$\mathcal{L}_{AddKeyword}$	$\mathcal{L}_{DeleteFile}$	$\mathcal{L}_{DeleteKeyword}$
KPR'12 [1]	$O(\lvert DB(w)\rvert)$	$O(\lvert DB\rvert + \lvert\mathcal{W}\rvert)$	$\lvert\mathcal{W}\rvert$, $\lvert DB\rvert$	①	×	③	×
KP'13 [2]	$O(\lvert DB(w)\rvert \cdot log\ \lvert\mathcal{ID}\rvert)$	$O(\lvert\mathcal{W}\rvert \cdot \lvert\mathcal{ID}\rvert)$	$\lvert\mathcal{W}\rvert \cdot \lvert\mathcal{ID}\rvert$	①	×	④	×
CJJ'14 [3]	$O(\lvert DB(w)\rvert)$	$O(\lvert DB\rvert)$	$\lvert DB\rvert$	①	②	×	⑤ at the worst case
Ours	$O(\lvert DB(w)\rvert)$	$O(\lvert DB\rvert)$	$\lvert DB\rvert$	①	②	④	⑤

① : $DB(w)$, $Old(w)$, and $New(w)$. ② : $New(id, w)$. ③ : $Old(id)$, $New(id)$ and a part of $DB(w)$ where $w \in DB(id)$.
④ : $Old(id)$ and $New(id)$. ⑤ : $Old(id, w)$ and $New(id, w)$. ×: means that the operation cannot be achieved. All symbols have been defined in Sects. 2 and 6.1.



(a) Encrypt DB (b) Search keywords (c) Add ciphertexts

(d) Delete ciphertexts (e) Physical deletion (f) Parallel search

（二）《A Multi-client Dynamic Searchable Symmetric Encryption System with Physical Deletion》

时间：2017

会议：ICICS（C 会）

主要内容：

本文工作基于文献【2】及【4】，结合文献【2】动态方案及文献【4】多用户方案（如图），通过 RSA 加密同时支持动态+多用户模式，并且证明了效率是较优的，且该方案为 IND-CPA2 安全.

Table 1. The communication and computation cost of some classical retrieval scheme

Scheme	Key size	Cipher. size	Search cost	Dynamic	Multi-client
Xu et al. [24]	$2\|k\|$	$O(\|DB\|)$	$O(\|DB(w)\|)$	√	
Sun et al. [25]	$3\|k\| + \|G\|$	$3O(\|DB(w)\|)$	$O(\|DB(w)\| \cdot exp)$		√
Ours	$2k + \|G\|$	$O(\|DB\|)$	$O(\|DB(w)\|)$	√	√

（三）《Updatable Searchable Symmetric Encryption with fine-grained delete functionality》

时间：2018

会议：CANDAR Workshops

主要内容：

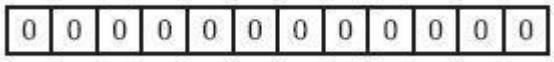
（1）背景知识（布隆过滤器）

来源：https://zhuanlan.zhihu.com/p/140545941

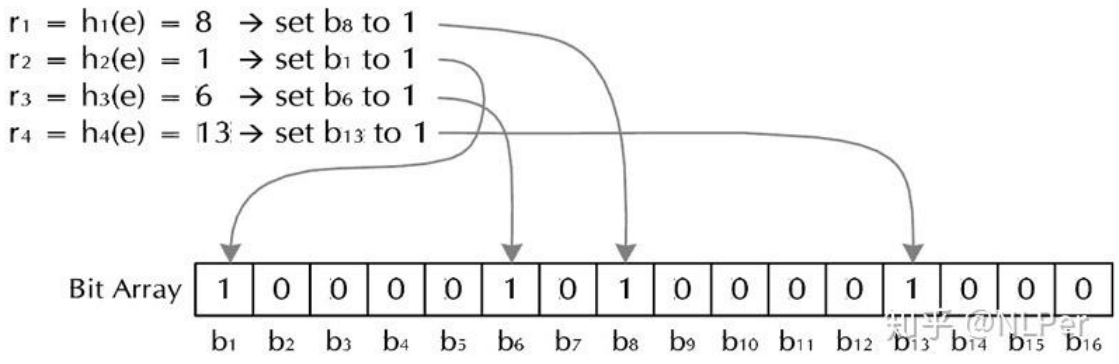
1）概述

Bloom Filter是由Bloom在1970年提出的一种多哈希函数映射的快速查找算法。通常应用在一些需要快速判断某个元素是否属于集合，但是并不严格要求100%正确的场合。基于一种概率数据结构来实现。

布隆过滤器维护一个m大的数组（数组元素只能为0或1），对于需要存储的关键词str，其通过k个不同的哈希函数$h_1,h_2,...,h_k$，将每个关键词都保存在k个数组的位置中。



2）Add操作



如图所示，对于关键词 str，分别计算 h（1，str），h（2，str）…… h（k，str）。然后将 BitSet 的第 h（1，str）、h（2，str）…… h（k，str）位设为 1。

3）check 操作

根据上图，我们对每个关键词的 check 操作采用同样的算法。

下面是检查关键词 str 是否被 BitSet 记录过的过程：

对于关键词 str，分别计算 h（1，str），h（2，str）…… h（k，str）。然后检查 BitSet 的第 h（1，str）、h（2，str）……h（k，str）位是否为 1，**若其中任何一位不为 1 则可以判定 str 一定没有被记录过。若全部位都是 1，则“认为”关键词“很可能”str 存在。**

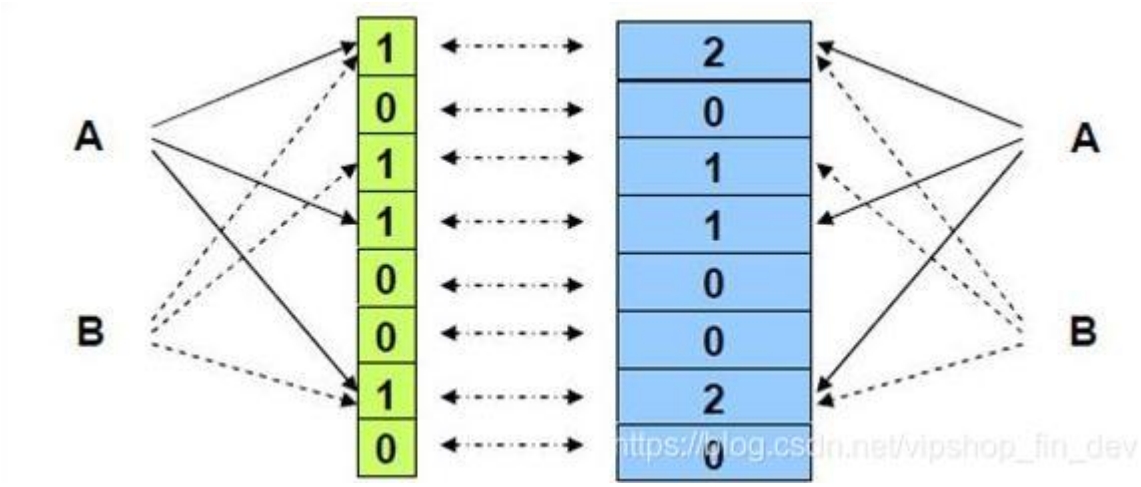
注意：若一个关键词对应的 Bit 不全为 1，则可以肯定该关键词一定没有被 Bloom Filter 记录过。（这是显然的，因为关键词被记录过，其对应的二进制位肯定全部被设为 1 了）；但是若一个关键词对应的 Bit 全为 1，实际上是不能 100%的肯定该关键词被 Bloom Filter 记录过的。（因为有可能该关键词的所有位都刚好是被其他关键词所对应）这种将该关键词划分错的情况，称为 wrong position。

4）check 操作

字符串加入了就被不能删除了，因为删除会影响到其他字符串。实在需要删除字符串的可以使用 Counting bloomfilter(CBF)，这是一种基本 Bloom Filter 的变体，CBF 将基本 Bloom Filter 每一个 Bit 改为一个计数器，这样就可以实现删除字符串的功能了。

（2）背景知识（计数式布隆过滤器及动态布隆计数器）

标准 Bloom filter 对于需要精确检测结果的场景将不再适用，而带计数器的 Bloom filter 的出现解决了这个问题。Counting Bloom filter 实际只是在标准 Bloom filter 的每一个位上都额外对应得增加了一个计数器，在插入元素时给对应的 k（k 为哈希函数个数）个 Counter 的值分别加 1，删除元素时给对应的 k 个 Counter 的值分别减 1。



Counting Bloom Filter 通过多占用几倍的存储空间的代价，给 Bloom Filter 增加了删除操作。这其中最关键的问题是 Counting Bloom filter 需要增加多少存储量？在论文中给出了相关计算，假设 counter 数组的长度为 m（对应 bloom filter 的位数组），Ci 表示 counter 数组中第 i 个 counter 的大小，即哈希函数映射到第 i 位的次数，则每个 counter 最少位数 N 为：

$$N = \sum_{i=1}^{m} [\log_2 C_i]$$

SBF（Spectral Bloom Filter）作为 Counting Bloom Filter 的一种实现，将所有 counter 排成一个位串，counter 之间完全不留空隙，然后通过建立索引结构来访问 counter，并达到了只使用 O(N) + O(m)位的存储目标，O(m)的构建时间。虽然 SBF 解决了动态 counter 的存储问题，但其引入了复杂的索引结构，这让每个 counter 的访问变得复杂而耗时。

为改进 SBF 的缺点，人们又发明了 DCF（Dynamic Count Filter），其使用两个数组来存储所有的 counter，它们的长度都为 m（即 bloom filter 的位数组长度）。第一个数组是一个基本的 CBF（即下图中的 CBFV，counting bloom filter vector），counter 的长度固定，为 x = log(M/n)，其中 M 是集合中所有元素的个数，n 为集合中不同元素的个数。第二个数组用来处理 counter 的溢出（即下图中的 OFV，overflow vector），数组每一项的长度并不固定，根据 counter 的溢出情况动态调整。

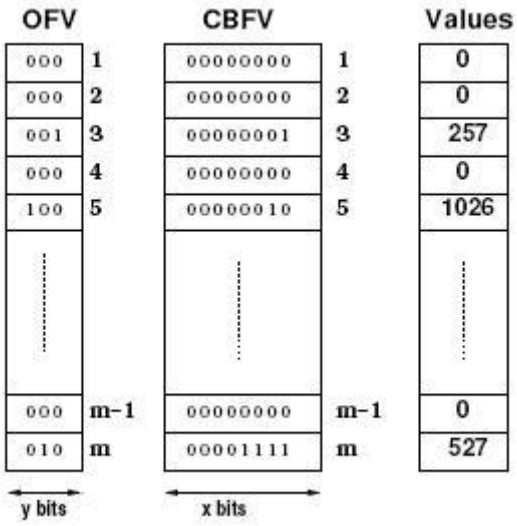
在查询一个 counter 时，DCF 要求两次内存访问。假设想查询位置为 j 的 counter 的值，我们先读出 CBFV 和 OFV 的值，分别为 Cj 和 OFj，那么 counter 的值就可以表示为 Vj = (2x×OFj ＋ Cj)。

在集合增加元素时，如果 OFV 的最大值从 2x － 1 增加到 2x，OFV 就需要给每一项增加 1 位，否则就会溢出。对应的，当 OFV 的最大值从 2x 减少到 2x － 1 时，OFV 就需要减少 1 位。每次 OFV 大小改变的时候都需要重新创建一个 OFV 数组，然后把旧 OFV 数组的值拷贝到新建的 OFV 数组中，最后把旧 OFV 数组的

空间释放掉。对于减少的情况，可以采用一些策略延迟 OFV 的重建，以避免一些临时性的减少导致 OFV 反复重建。

实际的使用场景中会有很多 CBF 的升级版。

比如 SBF（Spectral Bloom Filter）在 CBF 的基础上提出了元素出现频率查询的概念，将 CBF 的应用扩展到了 multi-set 的领域；dlCBF（d-Left Counting Bloom Filter）利用 d-left hashing 的方法存储 fingerprint，解决哈希表的负载平衡问题；ACBF（Accurate Counting Bloom Filter）通过 offset indexing 的方式将 Counter 数组划分成多个层级，来降低误判率。



原文链接：https://blog.csdn.net/vipshop_fin_dev/article/details/102647115

（3）主要内容

1）初始化：

1) $K \leftarrow KeyGen(1^s)$
Given a security parameter s, choose a pseudo-random function $f:\{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^s$, and output the secret key $K = (k_1, ..., k_r) \leftarrow \{0,1\}^{s \cdot r}$.

2) $T(w) \leftarrow Trapdoor(w, K)$
Given the secret key $K = (k_1, ..., k_r) \leftarrow \{0,1\}^{s \cdot r}$ and w, output the trapdoor $T(w) = (f(w, k_1), ..., f(w, k_r) \in \{0,1\}^{s \cdot r}$. 生成了r个密钥

3) $I_{D_{id}} \leftarrow BuiltIndex(D, K)$
Given the document D comprising of an unique identifier(document ID) $D_{id} \in \{0,1\}^n$, word list $(w_1, ..., w_t) \in \{0,1\}^{n \cdot t}$ and $K = (k_1, ..., k_r) \leftarrow \{0,1\}^{s \cdot r}$.

a) For each unique word $w_i (i = 0...t)$, implement the following steps

Keyword->X

(i) compute the trapdoor:
$(x_1 = f(w_i, k_1), ..., x_r = f(w_i, k_r)) \leftarrow \{0,1\}^{s \cdot r}$
using $Trapdoor(w, K)$ algorithm

(ID,X)->X'

(ii) compute the codeword:
$(y_1 = f(D_{id}, x_1), ..., y_r = f(D_{id}, x_r)) \leftarrow \{0,1\}^{s \cdot r}$

X'->Bloom filter

(iii) insert the codeword $y_1, ..., y_r$ into counting Bloom filter(CF) of D_{id}

b) Based on the encrypted document D, the sum of the lengths of the words is calculated and taken as u. Let v be the number of elements of the set of unique words. Randomly increment the elements of the array by the number of $(u - v) \times r$ in the index. This is equivalent to storing $u - v$ random words in the index.

u: 所有的关键词的数量
v: 不同关键词的数量

c) Output the index $I_{D_{id}} = (D_{id}, CF)$

2）检索：

4) $S(w) \leftarrow Search(T(w), I_{D_{id}})$
Given the trapdoor $T(w) = (f(w, k_1), ..., f(w, k_r)) \in \{0,1\}^{s \cdot r}$ and the index $I_{D_{id}} = (D_{id}, CF)$, for each document, implement the following steps.

a) Compute codeword:
$(y_1 = f(D_{id}, x_1), ..., y_r = f(D_{id}, x_r)) \leftarrow \{0,1\}^{s \cdot r}$.

b) Check if the bits at positions corresponding to $y_1, ..., y_r$ of CF is all 1.

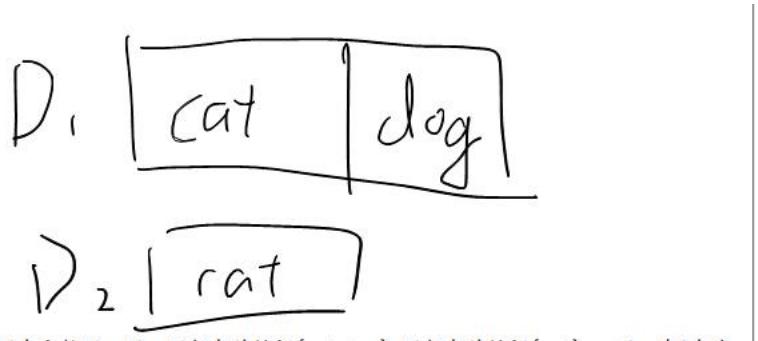
c) If so output 1, otherwise output 0 as $S(w)$.

3）更新：

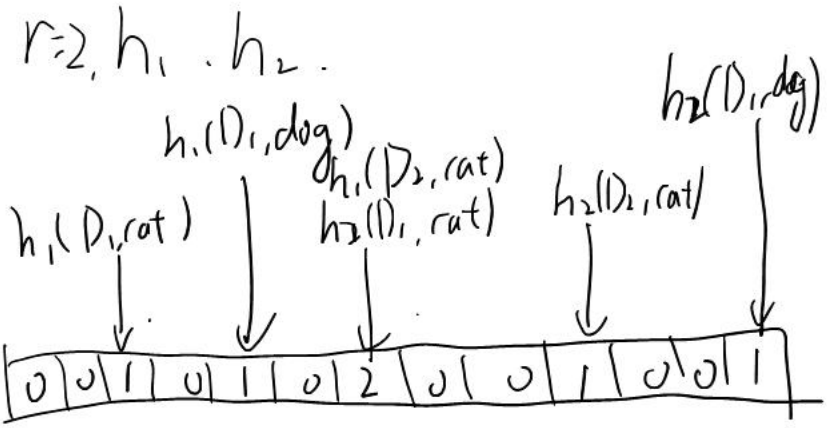
5) $I'_{D_{id}} \leftarrow Update(T(w), I_{D_{id}}, OP)$ 比第二章多了Update操作
Given the trapdoor $T(w) = (f(w,k_1),...,f(w,k_r)) \in \{0,1\}^{s \cdot r}$, the index $I_{D_{id}} = (D_{id}, CF)$ and operation $OP = \{add, del\}$, implement the following steps

a) Compute codeword: $(y_1 = f(D_{id}, x_1),...,y_r = f(D_{id}, x_r)) \leftarrow \{0,1\}^{s \cdot r}$.
b) If $OP = add$, check if the bits at positions corresponding to $y_1,...,y_r$ of CF is all 1.
c) If so skip to f), otherwise, insert the codeword $y_1,...,y_r$ into CF.
d) If $OP = del$, check if the bits at positions corresponding to $y_1,...,y_r$ of CF is all 1.
e) If so delete the codeword $y_1,...,y_r$ into CF, otherwise skip to f).
f) output $I'_{D_{id}} = (D_{id}, CF)$ as the updated index.

举例说明，在构造时

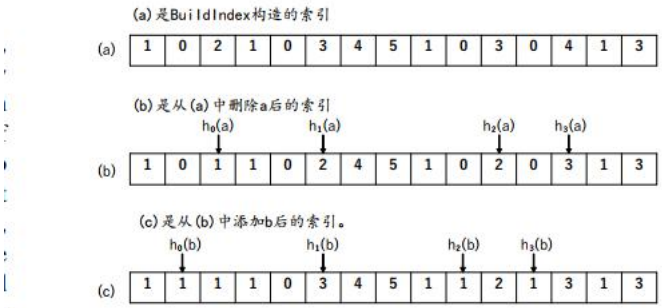


假设有2个文档D1，D2，D1包含关键词{cat,dog}，D2包含关键词{cat}，r=2，有2个哈希函数，则生成的布隆过滤器如图



检索时，trapdoor T(w) = (f(w, k1), ..., f(w, kr))；
对每个文档，计算(y1 =f(Did, x1), ..., yr =f(Did, xr))，若所有哈希值在数组相应位置的数值都不为0，则说明匹配。

在添加或删除时



4) 效率

初始化的复杂度与文档及关键词梳理呈线性关系，新增、删除、检索的计算开销是常量。

5) 安全性

更新算法会泄露更新后的索引信息，但是若不考虑更新函数，则可以证明是 IND-CKA 安全的。

个人观点：1、由于 IND-CKA 的安全性作者并未给出有效证明，其安全性不一定可信；

三、参考文献

1. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: ACM CCS 2012, pp. 965–976. ACM (2012)

2. Kamara, S., Papamanthou, C.: Parallel and dynamic searchable symmetric encryption. In: Sadeghi, A.-R. (ed.) FC 2013. LNCS, vol. 7859, pp. 258–274. Springer,Heidelberg (2013).doi:10.1007/978-3-642-39884-1 22

3. Cash, D., Jaeger, J., Jarecki, S., Jutla, C., Krawczyk, H., Ros, M.C., Steiner, M.: Dynamic searchable encryption in very-large databases: data structures and implementation. In: NDSS (2014)

4. Sun, S.-F., Liu, J.K., Sakzad, A., Steinfeld, R., Yuen, T.H.: An efficient noninteractive multi-client searchable encryption with support for Boolean queries. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9878, pp. 154–172. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45744-4 8