计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220267
60(12): 2725−2736, 2023

# 一种基于 SRE 的对称可搜索加密方案

黄一才　郁　滨　李森森
（信息工程大学　郑州　450001）
（huangyicai3698@163.com）

# A Searchable Symmetric Encryption Scheme Based on SRE

Huang Yicai, Yu Bin, and Li Sensen
（*Information Engineering University*, *Zhengzhou* 450001）

**Abstract**　DSSE (dynamic searchable symmetric encryption), which has forward/backward privacy and high search efficiency, supports addition and deletion of encrypted index. Relevant research has been a hot spot and many new schemes have been constructed in recent years, such as Aura. Aiming at the problems of high ciphertext storage overhead and mistaken deletion in Aura scheme, a more strict correctness definition of SRE (symmetric revokable encryption) primitive is given, and the condition of mistaken deletion is analyzed theoretically. In addition, an insertion position selection algorithm is designed to avoid node reuse due to Hash collision. On this basis, a searchable symmetric encryption scheme based on SRE is constructed by adding a deletion list and using a t-puncturable pseudorandom function. Puncturing all unused nodes at one time, the scheme not only effectively reduces the computing overhead on the cloud server during search phase, but also avoids revealing unused keys and gains better security. Finally, the scheme is analyzed in terms of search efficiency, storage overhead, communication overhead and security. Theoretical analysis and experimental results show that the scheme can effectively reduce the space overhead of ciphertext storage on the server, avoid the mistaken deletion, and improve the search efficiency on large-scale nodes.

**Key words**　cloud security; searchable symmetric encryption (SSE); symmetric revokable encryption (SRE); symmetric puncturable encryption; backward privacy

**摘　要**　具有前向隐私和后向隐私的动态对称可搜索加密(dynamic searchable symmetric encryption, DSSE)方案能够支持动态添加和删除密文索引且具有较高的搜索效率，一直是近年来研究的热点. 针对 Aura 方案中存在密文存储开销大和误删除的问题，给出了对称可撤消加密（symmetric revokable encryption, SRE）原语更严格的正确性定义，从理论上分析了误删除发生的条件，通过设计穿刺密钥位置选择算法，避免了哈希碰撞导致的节点位置重用. 在此基础上，构造了基于 SRE 对称可搜索加密方案. 方案利用多点可穿刺伪随机函数实现一次穿刺所有未使用节点，既有效降低了搜索时服务器的计算开销，又可避免提前暴露未使用密钥，提高方案的安全性. 最后，从搜索效率、存储开销、通信开销和安全性等方面对方案进行了分析. 理论分析和实验结果表明，所提方案不仅能够减小服务器密文存储时的空间开销，避免误删除索引，而且在大规模节点下具有更高的搜索效率.

**关键词**　云安全；对称可搜索加密；对称可撤消加密；对称可穿刺加密；后向隐私

**收稿日期**：2022-03-31；**修回日期**：2023-02-24
**基金项目**：国家自然科学基金项目(61772547)
This work was supported by the National Natural Science Foundation of China (61772547).
**通信作者**：李森森(lss589@163.com)

中图法分类号 TP391

可搜索加密无需解密就能实现数据搜索. 依据搜索时所采用的加密算法加密方案可分为公钥可搜索加密(public key encryption with keyword search, PEKS)和对称可搜索加密(searchable symmetric encryption, SSE)这 2 类. 其中 SSE 方案因采用对称算法，在数据量较大的云存储环境下往往具有更高的搜索效率.

## 1 相关工作

2000 年，Song 等人[1]提出了可搜索加密的基本概念，并基于对称密码算法，构造对称可搜索加密方案. 文献 [2−6] 分别构造了正排索引、倒排索引、双向索引和树形索引结构的对称可搜索加密方案，提高了方案的搜索效率.

相比静态方案，支持密文数据更新，如添加、删除等操作的动态方案具有更好的实用性. 文献 [6] 利用 OMAP 结构隐藏服务器密文字典的访问地址，实现了一种安全且支持动态更新的对称可搜索加密方案，但 OMAP 结构较高的通信开销降低了方案的实用性[7]. 文献 [8−10] 分别采用并行计算和优化 I/O 效率的方式进一步提高了方案的搜索效率. 文献 [11] 设计 2 级索引链结构，实现了固定大小的客户端存储空间开销.

安全性一直是动态方案构造中研究的热点问题. 为方便对动态方案的安全性进行分析，Curtmola 等人[4]通过泄露函数给出了自适应安全对称可搜索加密方案的定义. 文献 [12] 给出了方案动态更新时前向及后向隐私的正式定义. 文献 [13] 构造了一种前向隐私的可搜索加密方案，但方案需要进行大量的指数运算，搜索效率不高. 文献 [14−16] 提出了执行效率更高的前向隐私方案. 文献 [17−18] 提出了同时满足前向和后向隐私的对称可搜索加密方案，文献 [19] 提出了利用 SGX 技术实现的后向隐私，但方案的安全性需要基于特殊软件平台，一定程度上降低了方案的通用性.

方案的构造过程通常需要结合应用场景在搜索效率、存储和通信开销、安全性、功能性等方面进行多种妥协. 其中利用可穿刺加密技术构造的前向/后向隐私方案，具有交互少、安全性高的特点，近年来越来越受到人们的重视. 文献 [20] 基于可穿刺加密技术，在 Janus 方案[12]基础上，采用可穿刺伪随机函数[21]代替公钥算法，设计了对称可穿刺可搜索加密方案 Janus++，提高了方案在增加和删除密文索引时的效率，且方案在实现后向隐私的同时，具有更小的通信开销.

Sun 等人[22]指出 Janus++方案中使用的对称可穿刺加密(symmetric puncturable encryption, SPE)无法抵抗共谋攻击，为此定义了基于可穿刺伪随机函数的对称可撤消加密(symmetric revokable encryption, SRE)密码学原语，并利用对称可撤消加密构造一种单轮交互的后向安全对称可搜索加密方案 Aura. Aura 方案使用 GGM 树实现可穿刺伪随机函数，并在此基础上引入 Bloom Filter 记录删除索引，实现了 TypeII 级后向隐私. 实验证明，该方案能够进一步提高搜索响应速度. 然而该方案存在 2 个方面的不足：1)由于 Bloom Filter 存在假阳性[23]，会导致索引误删除. 2)服务器在每次搜索前需要生成所有密钥序列，增加了服务器计算开销，降低了搜索效率，且使非可信服务器提前掌握了所有密钥信息，存在一定的安全隐患.

本文基于 Aura 方案[22]提出了一种改进的可穿刺对称可搜索加密方案，具有 3 个方面的优势：

1) 搜索效率高. 通过在客户端增加存储节点使用标志，解密时仅传输已使用节点密钥，减少服务器端计算开销，提高搜索效率，且节点平均搜索时间不受节点规模影响.

2) 避免误删除. 结合 Bloom Filter 设计思想，计算可穿刺密钥节点位置选择算法，在提高节点利用率的同时，防止因哈希碰撞而误删索引.

3) 服务器密文存储规模小.Aura 方案中密文规模与 Bloom Filter 中哈希函数个数相关，通过设计插入位置选择算法为每个索引选择唯一的节点位置，避免密文规模扩大，减少服务器密文存储开销.

## 2 预备知识

首先简要介绍方案设计中用到的相关密码学基础知识，包括多点可穿刺伪随机函数、对称加密算法及前向与后向隐私等.

### 2.1 多点可穿刺伪随机函数

**定义 1.** 多点可穿刺伪随机函数[21-22]设$t(\cdot)$表示一个确定的多项式，密钥空间为$\mathcal{K}$的伪随机函数 $MF$: $\mathcal{K}\times\mathcal{X}\to\mathcal{Y}$，若集合$S$满足$|S|\leqslant t$，存在另外一个密钥空间$\mathcal{K}_{\mathrm{p}}$以及 3 个多项式时间算法($Setup$, $Punc$, $Eval$)：

1) $k\leftarrow Setup\left(1^{\lambda}\right)$. 输入安全参数$\lambda$，输出伪随机

函数密钥$k \in \mathcal{K}$.

2）$k_S \leftarrow Punc(k, S)$. 输入密钥$k \in \mathcal{K}$，集合$S \subset \mathcal{X}$ s.t.$|S| \leqslant t(\lambda)$，输出可穿刺密钥$k_S \in \mathcal{K}_p$.

3）$y \leftarrow Eval(k_S, x)$. 输入可穿刺密钥$k_S \in \mathcal{K}_p$，$x \in \mathcal{X}$，输出$y \in \mathcal{Y}$或$\perp$.

则称$MF$为多点可穿刺伪随机函数 $t$-Punc-PRF（$t$-puncturable pseudorandom function）.

**定义 2.** $MF$正确性[22]. 对所有$S \subset \mathcal{X}$ s.t.$|S| \leqslant t(\lambda)$，$x \in \mathcal{X} \backslash S$，$k \leftarrow Setup\left(1^\lambda\right)$，$k_S \leftarrow Punc(k, S)$，对所有$k \in \mathcal{K}$使

$$Pr[Eval(k_S, x) \neq MF(k, x)] \leqslant \nu(\lambda)$$

成立，其中$\nu(\lambda)$为可忽略函数，则称函数$MF$满足正确性条件.

### 2.2 对称加密算法

**定义 3.** 对称加密算法[22]. 设明文空间$\mathcal{M}$、密钥空间$\mathcal{K}$和密文空间$C$，则对称加密（symmetric encryption, SE）算法可用 3 个多项式时间算法的三元组表示，记为$SE = (Gen, Enc, Dec)$，其中：

1）$k \leftarrow Gen\left(1^\lambda\right)$. 输入秘密参数$\lambda$，输出密钥$k \in \mathcal{K}$.

2）$ct \leftarrow Enc(k, m)$. 输入密钥$k$、明文消息$m$，输出密文$ct \in C$.

3）$m \leftarrow Dec(k, ct)$. 输入密钥$k$、密文$ct$，成功解密输出$ct$对应明文$m$，否则输出$\perp$.

**定义 4.** $SE$正确性[22]. 若算法对任意明文$m \in \mathcal{M}$，$k \leftarrow Gen\left(1^\lambda\right)$且$ct \leftarrow Enc(k, m)$，满足

$$Pr[Dec(k, ct) = m] = 1,$$

则称算法$SE$满足正确性条件.

### 2.3 前向隐私与后向隐私

可搜索加密方案前向隐私是指更新查询不会泄露正在更新的关键字或文档对中所涉及的关键字信息，主要针对文档添加操作，确保新添加文档不会包含在以前查询令牌的检索结果中. 文献 [24] 指出针对不具备前向隐私的方案可有效实施文件注入攻击，快速恢复用户检索信息.

后向隐私要求搜索和更新操作仅泄露数据库中的当前文档（不包括已删除的文档），主要针对文档删除操作，确保当前查询不会包含过去已删除的文档索引. 文献 [4] 根据泄露信息不同，将安全强度由高到低分为 Type Ⅰ，Type Ⅱ，Type Ⅲ. Type Ⅰ后向隐私允许泄露当前关键词所匹配的文档索引、插入时间及更新关键词的更新总数；Type Ⅱ在 Type Ⅰ基础上，增加了泄露关键词的操作类型和时间；Type Ⅲ相对于 Type Ⅱ，增加了泄露删除操作对应的那个添加操作.

## 3 搜索模型

本文方案中各主要符号定义如表 1 所示. 为简化方案构造，假定：

**Table 1 Symbol Definition**

**表 1 符号定义**

| 符号 | 含义 | 符号 | 含义 |
|---|---|---|---|
| $EDB$ | 密文数据库 | $d_{\max}$ | 最大搜索深度 |
| $w_i$ | 文档包括关键词 | $MSK[w_i]$ | 文档$w_i$当前主密钥 |
| $\lambda$ | 安全参数 | $C[w_i]$ | 存储$w_i$查询次数 |
| $b$ | 节点规模，即关键词允许插入文档数 | $\boldsymbol{A}[w_i]$ | 已使用节点列表 |
| $N$ | 插入文档索引数 | $\boldsymbol{D}[w_i]$ | 删除节点列表 |
| $n$ | 信息指纹宽度 | $K_s$ | 索引加密安全参数 |
| $nw$ | 插入关键词个数 | $K_t$ | 标签生成安全参数 |
| $h$ | 哈希函数 | $K_{add}$ | 数据加密密钥 |

1）服务器为“诚实且好奇（honest but curious）”的. 即服务器能忠实运行用户发出的请求，但也会尽可能地窥探用户个人隐私.

2）只有数据拥有者（data owner, DO）是完全可信的，且能够访问系统中所有的明文信息；各实体之间以及云存储服务器内部各通信信道均不可信，且存在各种可能的网络攻击行为.

3）数据拥有者能够对文档进行解密，但在访问服务器前并不确定云服务器中存有哪些文件，以及是否包含其想要的文件.

4）不考虑多客户端应用场景. 仅对方案搜索效率优化与避免误删除进行设计，对多客户端场景下的特殊性不做专门研究.

### 3.1 场景描述

对称可搜索加密应用场景由安全存储服务器和数据拥有者 2 部分组成，其中安全存储服务器包括 1 个存储用户密文文件的存储服务器和 1 个实现数据密文搜索的密文索引搜索服务器. DO 既能够向安全存储服务器中添加新的密文文件，又可以实现对数据密文查询. 应用场景如图 1 所示.

方案的工作过程包括数据动态更新和数据查询 2 个阶段，其中数据动态更新通常包括数据添加、删除和更新 3 种操作，为简化描述过程，这里以数据添加为例，具体描述方案的工作过程.

1）数据添加. 用于向密文索引数据库中加入新的数据文件索引.DO 先将数据文件加密后存储在文

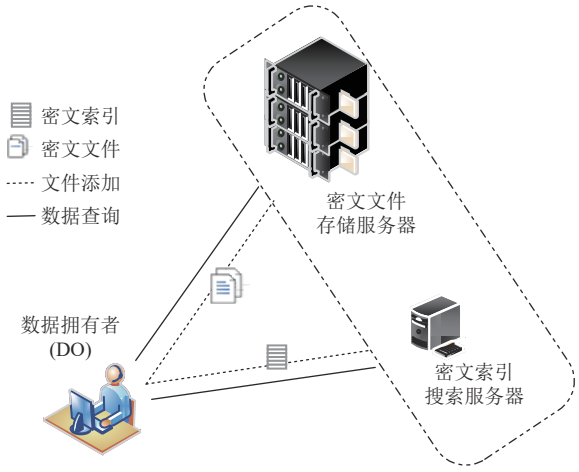


Fig. 1 Application scenario of dynamic searchable symmetric encryption

图 1 动态对称可搜索加密应用场景

件存储服务器,并返回访问入口信息;然后 DO 通过对明文数据进行处理,生成文件访问入口信息与关键词对应关系,并按照一定索引结构生成密文索引,将密文索引存储于密文索引服务器.

2)数据查询. 要求密文服务器不用解密具体密文文件,便能从所有密文文件中返回符合用户查询需求的文件信息,这也是可搜索加密方案的核心. 此过程一般是 DO 通过关键词生成查询令牌,然后通过密文索引服务器获取密文文件的入口地址信息,最终从密文文件服务器获取密文文件.

### 3.2 方案形式化描述

动态对称可搜索加密(dynamic searchable symmetric encryption, DSSE)的形式化定义通常由一组多项式时间算法组成.

**定义 5.** DSSE 方案. 设数据库由$DB=(id_i, W_i)_{i=1}^{d}$文档地址或关键词集构成,其中$id_i$为第$i$文档,$W_i$为文档$i$包含的关键词集合,$d$为数据库中文档的数量,则动态对称可搜索加密方案可用一个三元组多项式时间算法$(Setup, Search, Update)$表示,其中:

1)初始化算法:$(K, \sigma, EDB)\leftarrow Setup\left(1^{\lambda}, DB\right)$. 算法输入安全参数$\lambda$和明文索引数据库$DB$,输出密钥$K$、关键词状态$\sigma$和密文索引数据库 $EDB$. 在 DSSE 方案中,通常设$EDB=\varnothing$.

2)查询算法:$(DB(w), \sigma', EDB')\leftarrow Search(K, \sigma, w; EDB)$. 输入密钥$K$、关键词状态$\sigma$和检索关键词$w$,输出包含检索关键词$w$的所有文档标识$DB(w)$,以及搜索操作后关键词状态$\sigma'$和密文索引数据库$EDB'$. 通常算法包含客户端生成查询令牌和服务器根据令牌搜索查询结果 2 部分.

3)更新算法:$(\sigma', EDB')\leftarrow Update(K, \sigma, w, ind, op; EDB)$. 输入密钥$K$、关键词状态$\sigma$、更新关键词$w$及对应索引$ind$,更新操作类型$op$,其中$op\in\{\text{add}, \text{del}\}$,输出更新后的密文索引数据库$EDB'$、关键词状态$\sigma'$.

**定义 6.** DSSE 正确性[25]. 对于所有的 $DB$ 和 $W$,若任意的$w\in W$,满足

$$Search(EDB, w)=DB(w),$$

则称 DSSE 方案满足正确性条件.

### 3.3 SRE 语法定义

**定义 7.** SRE 算法定义[21]. 设$\mathcal{MSK}$表示密钥空间,$\mathcal{T}$表示标签空间,则一个 SRE 算法可用 4 个多项式时间算法表示,记为$SRE=(KGen, Enc, KRev, Dec)$,其中:

1)$msk\leftarrow KGen\left(1^{\lambda}\right)$. 输入安全参数$\lambda$,输出系统密钥$msk\in\mathcal{MSK}$.

2)$ct\leftarrow Enc(msk, m, T)$. 输入系统密钥$msk$,消息$m\in\mathcal{M}$及标签列表$T\subseteq\mathcal{T}$,输出消息$m$标签$T$下的密文$ct$.

3)$sk_{\mathrm{R}}\leftarrow KRev(msk, R)$. 输入系统密钥$msk$及撤消列表$R\subseteq\mathcal{T}$,输出撤消后的密钥$sk_{\mathrm{R}}$,仅能解密不属于标签$R$对应的密文.

4)$m\leftarrow Dec(sk_{\mathrm{R}}, ct, T)$. 输入撤消后的密钥$sk_{\mathrm{R}}$、密文$ct$及对应标签$T$,输出消息$m$(解密失败输出$\perp$).

**定义 8.** $SRE$ 正确性. 对所有安全参数$\lambda\in\mathbb{N}_{+}$,消息$m\in\mathcal{M}$及对应标签$T\subseteq\mathcal{T}$、撤消列表$R\subseteq T$ s.t.$R\cap T=\varnothing$,算法$SRE$正确解密的概率$Pr[SRE.Dec(sk_{\mathrm{R}}, ct_m, t)=m]=\begin{cases}1, t\in\mathcal{T}\text{且}t\notin R,\\ 0, \text{其他},\end{cases}$即对于未撤消加密节点均可正确解密,则称算法$SRE$满足正确性条件.

相比文献 [21] 给出的正确性定义中,解密发生错误的概率以一个不可忽略函数$\nu(\lambda)$为上界,定义 8 要求能够对$t\in\mathcal{T}$且$t\notin R$的所有密文$ct_m$及对应标签$T$,均可正确解密.

**定义 9.** 撤消操作. 对所有安全参数$\lambda\in\mathbb{N}_{+}$,消息$m$对应标签$t\in T$ s.t.$T\subseteq\mathcal{T}$和撤销列表$R\subseteq\mathcal{T}$,消息$m\in\mathcal{M}$,撤消加密操作定义为$R=R\cup\{t\}$.

**性质 1.** 撤消操作的有效性. 对所有安全参数$\lambda\in\mathbb{N}_{+}$,消息$m$对应标签$t\in T$ s.t.$T\subseteq\mathcal{T}$和撤销列表$R\subseteq\mathcal{T}$,消息$m\in\mathcal{M}$撤消加密,则$Pr[SRE.Dec(sk_{\mathrm{R}}, ct_m, t)=m]=0$,即撤消加密后无法通过撤消后的密钥$sk_{\mathrm{R}}$、密文$ct_m$和对应标签$t$完成解密.

证明. 令消息$m$撤消前的撤消列表为$R'$,根据定义 9,撤消后撤消列表$R=R'\cup\{t\}$,则有$t\in\mathcal{T}$且$t\in R$.

根据正确性定义,$Pr[SRE.Dec(sk_{\mathrm{R}}, ct_m, t)=m]=0$.

证毕.

**性质 2.** 撤消操作的正确性. 对所有安全参数

$\lambda \in \mathbb{N}_+$, 已插入消息 $\mathcal{M}=\{m_1, m_2, \cdots, m_n\}$, 对应标签 $T=\{t_1, t_2, \cdots, t_n\}$和撤销列表 $R \subseteq \mathcal{T}$, 其中$n$为已插入消息数.$\forall m_i, m_j \in \mathcal{M}$, 对应标签分别为 $t_i, t_j \in T$ s.t.$i \neq j, t_i \neq t_j$. 当$t_i \in R, t_j \notin R$, 则

$$Pr\left[SRE.Dec\left(sk_{\mathrm{R}}, ct_{m_i}, t_i\right)=m_i\right]=0$$且 $Pr[SRE.Dec(sk_{\mathrm{R}}, ct_{m_j}, t_j)=m_j]=1$, 即撤消解密后不会破坏其他已加密密文解密的正确性.

证明. 因为$t_i \in R$, 由性质 1 可知, $Pr[SRE.Dec(sk_{\mathrm{R}}, ct_{m_i}, t_i)=m_i]=0$成立.

因为$t_j \in T$且$t_j \notin R$, 则由定义 8 可知, $Pr[SRE.Dec(sk_{\mathrm{R}}, ct_{m_j}, t_j)=m_j]=1$成立. 证毕.

撤消消息$m \in \mathcal{M}$对应密文$ct_m$的解密操作, 将$ct_m$对应标签$t_m$加入撤消列表, 即进行消息删除操作.

**定理 1.** 误删除条件. 若方案满足正确性条件, 发生误删除的充要条件是$\exists j \neq i$,且$t_i = t_j$.

证明. 设$\forall m_i, m_j \in \mathcal{M}$, 对应标签分别为$t_i, t_j \in T$ s.t.$i \neq j$, $m_i$撤消前撤消列表为$R^*$.

1) 充分性

设消息$m_i$撤消后, $m_j$也被撤消, 其中$j \neq i, t_i \neq t_j$. 撤消前, $t_i, t_j \in \mathcal{T}$且$t_i, t_j \notin R^*$. 当$m_i$撤消后, 依据性质 2, 撤消列表$R = R^* \cup \{t_i\}$.

因为$j \neq i$且$t_i \neq t_j$, $t_j \notin R^*$, 所以$t_j \notin R^* \cup \{t_i\} = R$, 则 $Pr\left[SRE.Dec\left(sk_{\mathrm{R}}, ct_{m_j}, t_j\right)=m_j\right]=1$.

因为$m_i$撤消时, $m_j$也被撤消, 所以依据性质 1, 有 $Pr\left[SRE.Dec\left(sk_{\mathrm{R}}, ct_{m_j}, t_j\right)=m_j\right]=0$.

矛盾, 即假设不成立.

2) 必要性

由定义 9 得到, 删除$m_i$后, 撤消列表$R = R^* \cup \{t_i\}$, 即$t_i \in R$. 因为$t_i = t_j$, 所以$t_j \in R$.

根据正确性定义, 可得

$$Pr\left[SRE.Dec\left(sk_{\mathrm{R}}, ct_{m_j}, t_j\right)=m_j\right]=0,$$

则由定义 9 可知, 消息$m_j$也已删除. 证毕.

可以看出, 防止选择相同标签加密是避免误删除的有效手段.

## 4 方案构造

插入位置选择算法为每个插入文件索引寻找唯一的节点位置. 在此基础上, 构造基于 SRE 的对称可搜索加密方案.

### 4.1 插入位置选择算法

插入位置选择算法用于在给定范围内, 为消息$m$随机选择一个未使用的节点位置$p_m$.

设$h_p(x)$, $h_{fp}(x)$分别表示 2 个哈希函数, $\boldsymbol{Lb}$表示长度为$b$的状态列表, $\boldsymbol{Lb}[i]=(f_i, fp_i)$. 其中, $f_i$为占用标志, 宽度为 1 b, 用于记录列表$\boldsymbol{Lb}$中位置$i$的占用情况, 占用时标志置为true, 否则置为 false; $fp_i$为位置$i$上存储的消息指纹, 一般为消息$m$的哈希值低$n$ 比特($n$为预设指纹宽度).图 2 所示为消息$m$选择插入位置过程示意图.

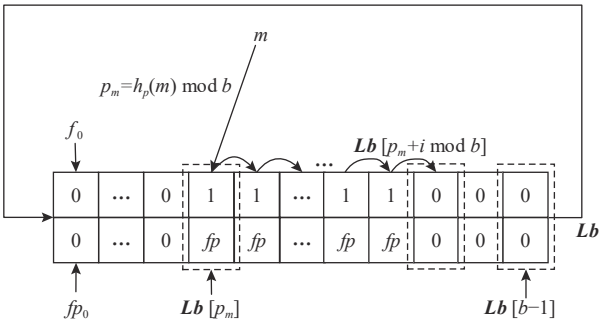


Fig. 2 Design idea of insertion position selection algorithm

图 2 插入位置选择算法设计思想

算法记为$InsPos(b, d_{\max}, n)$, 其中$b$表示列表规模, $d_{\max}$表示最大搜索深度, $n$表示消息指纹宽度, 且$b, d_{\max}, n \in \mathbb{N}_+$, $d_{\max} < b$, 则算法可以用一个三元组$InsPos=(Setup, Insert, Test)$表示.

1)初始化: $\boldsymbol{Lb} \leftarrow Setup(b, n, \boldsymbol{Lb}_{\mathrm{init}})$. 输入列表规模$b$、初始状态$\boldsymbol{Lb}_{\mathrm{init}}$, 输出状态列表$\boldsymbol{Lb}$. 工作过程为:

① 分配位长为$b+nb$的比特状态列表$\boldsymbol{Lb}$.

② 若$\boldsymbol{Lb}_{\mathrm{init}}=\perp$, 令 $\boldsymbol{Lb}[i]=(\mathrm{false}, \perp)$, 否则$\boldsymbol{Lb}[i]=\boldsymbol{Lb}_{\mathrm{init}}[i]$, 其中$0 \leqslant i \leqslant b-1$.

③ 返回比特列表$\boldsymbol{Lb}$.

2)插入点位置选择: $(Rst, p_m, \boldsymbol{Lb}') \leftarrow Insert(m, d_{\max}, \boldsymbol{Lb})$. 输入待插入消息$m$、最大搜索深度$d_{\max}$、状态列表$\boldsymbol{Lb}$; 输出为插入结果$Rst$、插入位置$p_m$、插入后状态列表$\boldsymbol{Lb}'$. 工作过程为:

① 计算消息$m$的插入位置$p_m = h_p(m) \bmod b$, $fp_m = h_{fp}(m)$.

② 令$Rst=\mathrm{false}$, 若$f_{p_m}=\mathrm{false}$, 则$Rst=\mathrm{true}$, 运行③, 否则$\exists i \leqslant d_{\max}$, 使$f_{p_m}=\mathrm{false}$成立(其中$p_m=(p_m+i) \bmod b$), 则$p_m=(p_m+i) \bmod b$, $Rst=\mathrm{true}$.

③ 若$Rst=\mathrm{true}$, 则修改$\boldsymbol{Lb}$列表$f_{p_m}=\mathrm{true}$, $fp_{p_m}=fp_m$.

④ 返回插入结果$Rst$、插入位置$p_m$、最新状态列表$\boldsymbol{Lb}'=\boldsymbol{Lb}$, 即$(Rst, p_m, \boldsymbol{Lb}')$.

当$Rst=\mathrm{true}$时表示在列表中找到唯一的节点位置$p_m$, 否则表示在搜索深度$d=d_{\max}$下未能找到空闲节点, 返回$p_m$为第 1 哈希的节点位置. 此时若选择此

节点位置伪随机数作为加密密钥,将导致密钥重用,并在删除该节点时因同时删除多个文件加密密钥而造成误删除.

3)插入位置测试: $(rstPos, fp_m) \leftarrow Test(m, d_{\max}, \boldsymbol{Lb})$. 通过对消息指纹对比获取消息$m$在列表$\boldsymbol{Lb}$中的位置. 输入为消息$m$、最大搜索深度$d_{\max}$、状态列表$\boldsymbol{Lb}$;输出为消息$m$插入位置$rstPos$,消息指纹$fp_m$. 工作过程为:

① 计算消息$m$的插入位置$p_m = h_p(m) \bmod b$和消息指纹$fp_m = h_{fp}(m)$.

② 令$rstPos=-1$,若$\exists 0 \leqslant i \leqslant d_{\max}$, $\boldsymbol{Lb}\left[(p_m+i) \bmod b\right] = (rstPos, fp_m)$,则$rstPos = (p_m+i) \bmod b$.

③ 返回$(rstPos, fp_m)$. 当$rstPos \geqslant 0$时,表示找到消息$m$的插入位置.

在可搜索加密方案构造中,$Test$算法主要用于从列表删除索引时查找索引准确位置. 搜索时服务器通过存储密文对应标签,直接获取节点在列表中的位置,因此不需要重新通过$Test$算法计算. $Test$算法的性能仅影响节点删除时的执行速度,不影响添加和搜索效率.

插入指纹生成时使用的哈希函数、位置选择以及位置深度搜索时使用的哈希函数可以设为相同也可以设为不同,列表占用标志$f_i$可以通过判断指纹位置是否为0判断该位置是否已被占用. 然而指纹生成时采用哈希函数的质量将直接影响方案指纹大小设定,选择优秀的哈希函数作为消息指纹生成算法能有效降低指纹占用空间的大小. 同时方案中,由于$f_i$仅需1 b,占用空间较小,且能够方便对指定位置占用情况进行判断,提高算法执行效率.

### 4.2 SRE的一般构造

设$SE = (Gen, Enc, Dec)$为标准对称加密算法,其密钥空间为$\mathcal{Y}$, $MF: \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$为多点穿刺伪随机函数,记为$MF = (Setup, Punc, Eval)$[20-21], $InsPos(b, d_{\max}, n) = (Setup, Insert, Test)$表示插入位置选择算法,则对称可撤消加密算法$SRE = (KGen, Enc, KRev, Dec)$的具体描述为4个步骤.

1) $msk \leftarrow KGen(1^\lambda, b, n, \boldsymbol{Lb}_{\text{init}})$:输入安全参数$\lambda$和整数$b, n \in \mathbb{N}_+$, $\boldsymbol{Lb}_{\text{init}}[i] = (\text{false}, \perp)$,其中$0 \leqslant i \leqslant b-1$,系统主密钥$msk$的生成过程有2步:

① 生成$\boldsymbol{Lb} \leftarrow InsPos.Setup(b, n, \boldsymbol{Lb}_{\text{init}})$.

② 生成$sk \leftarrow MF.Setup\left(1^\lambda\right)$,输出$msk = (sk, \boldsymbol{Lb})$.

2) $(ct_m, t) \leftarrow Enc(d_{\max}, msk, m, t)$. 输入$msk = (sk, \boldsymbol{Lb})$和消息$m \in \mathcal{M}$对应标签$t \in \mathcal{T}$,通过2个步骤生成密文$ct_m$:

① $(Rst, t) \leftarrow InsPos.Insert(m, d_{\max}, \boldsymbol{Lb})$和$sk_t = MF(sk, t)$.

② 生成$ct_m = SE.Enc(sk_t, m)$,返回密文$ct_m$和对应标签$t$.

3) $sk_R \leftarrow KRev(msk, R)$. 输入$msk = (sk, \boldsymbol{Lb})$和撤消标签列表$R = \{t_1, t_2, \cdots, t_\tau\}$,满足$\tau \leqslant NL$, $NL$为$\boldsymbol{Lb}$中已使用节点数. 通过2个步骤生成$R$下撤消的密钥$sk_R$:

① 计算$\boldsymbol{Lb}$所有置1的索引节点位置集合$I_R = \{t : t \in [b]$且$t \notin R\}$.

② 计算集合$sk_{I_R} \leftarrow MF.Punc(sk, I_R)$,返回$sk_R = (sk_{I_R}, I_R)$.

4) $m \leftarrow Dec(sk_R, ct_m, t)$. 输入密文$ct_m$、对应标签$t$,以及$R$下撤消后的密钥$sk_R$,按2个步骤恢复消息:

① 若$t \notin I_R$,解密失败,运行②.

② 若$t \in I_R$,计算$sk_t = MF.Eval(sk_I, t)$,计算$m = SE.Dec(sk_t, ct_m)$.

### 4.3 方案描述

设$\Sigma_{\text{add}} = (Setup, Search, Update)$为一个前向安全的对称可搜索加密方案,用于动态增加密文索引,并保证前向数据隐私,$SRE = (KGen, Enc, KRev, Dec)$为对称可撤消加密算法(构造方法详见4.2节). 算法1详细描述基于$SRE$和$\Sigma_{\text{add}}$的可搜索加密方案$\Sigma_{\text{add}} = (Setup, Search, Update)$的工作过程.

**算法1.** 初始化$\Sigma.Setup\left(1^\lambda\right)$.

输入:安全参数$\lambda$;

输出:密钥集$K$,关键词状态$\sigma$,密文索引数据库$EDB$.

客户端:

① $(EDB_{\text{add}}, K_{\text{add}}, \sigma_{\text{add}}) \leftarrow \Sigma_{\text{add}}.Setup\left(1^\lambda\right)$;

② $K_{\text{s}}, K_{\text{t}} \xleftarrow{\$} \{0,1\}^\lambda$, $EDB_{\text{cache}} \leftarrow \varnothing$, $MSK, C \leftarrow \perp$, $\boldsymbol{A}, \boldsymbol{D} \leftarrow InsPos.Setup(b, n, \perp)$;

③ return$((K_{\text{add}}, K_{\text{s}}, K_{\text{t}}), (\sigma_{\text{add}}, MSK, C, \boldsymbol{A}, \boldsymbol{D}), (EDB_{\text{add}}, EDB_{\text{cache}}))$.

客户端选择2个随机数$K_{\text{s}}, K_{\text{t}} \xleftarrow{\$} \{0,1\}^\lambda$,利用方案$\Sigma_{\text{add}}$的$Setup$算法生成$(EDB_{\text{add}}, K_{\text{add}}, \sigma_{\text{add}})$,并将集合$MSK, C, EDB_{\text{cache}}$和列表$\boldsymbol{A}, \boldsymbol{D}$初始化.

初始化完成后,客户端存储密钥$K = (K_{\text{add}}, K_{\text{s}}, K_{\text{t}})$、关键词状态$\sigma = (\sigma_{\text{add}}, MSK, C, \boldsymbol{A}, \boldsymbol{D})$,服务器存储密文索引数据库$EDB = (EDB_{\text{add}}, EDB_{\text{cache}})$.

**算法2.** 查询$\Sigma.Search(K, w, \sigma; EDB)$.

输入:客户端输入密钥集$K$,关键词状态$\sigma$,查询关键词$w$,服务器输入密文索引数据库$EDB$;

输出:匹配结果集$DB(w)$及更新后关键词状态$\sigma'$.

Step 1. 生成查询令牌$\Sigma.Search(K, w, \sigma; EDB)$.

客户端：

① $c \leftarrow C[w]$, $msk_w \leftarrow MSK[w]$, $\boldsymbol{D} \leftarrow \boldsymbol{D}[w]$, $\boldsymbol{A} \leftarrow \boldsymbol{A}[w]$;

② if $c = \perp$ then

③ return $\varnothing$;

④ end if

⑤ 计算 $sk_R = (sk_A, \boldsymbol{A}) \leftarrow SRE.KRev((msk_w, \boldsymbol{A}), \boldsymbol{D})$;

/* $\boldsymbol{A}$ 来自 $\boldsymbol{A}[w]$, $\boldsymbol{D}$ 来自 $\boldsymbol{D}[w]$*/

⑥ 发送 $sk_R$ 和 $tkn = h(K_s, w)$ 至服务器;

⑦ $msk_w' \leftarrow SRE.KGen\left(1^\lambda, \boldsymbol{A}\right)$;

/*搜索后更新 $w$ 对应的 $msk$*/

⑧ $MSK[w] \leftarrow msk_w'$, $C[w] \leftarrow c+1$.

Step 2. $\Sigma.Search(K, w, \sigma; EDB)$结果搜索.

客户端 & 服务器：

① 运行 $\Sigma_{\text{add}}.Search(K_{\text{add}}, w \| i, \sigma_{\text{add}}; EDB_{\text{add}})$, 服务器得到关于密文和标签对的列表 $((ct_1, t_1), (ct_2, t_2), \cdots, (ct_\ell, t_\ell))$.

服务器：

② 服务器使用 $sk_R$ 按如下步骤解密所有密文 $\{(ct_i, t_i)\}$;

③ for $i \in [1, \ell]$ do

④ $ind_i = SRE.Dec(sk_R, ct_i, t_i)$;

⑤ if $ind_i \neq \perp$ then

⑥ $NewInd \leftarrow NewInd \cup \{ind_i, t_i\}$;

⑦ else

⑧ $DelInd \leftarrow DelInd \cup \{t_i\}$;

⑨ end if

⑩ end for

⑪ $OldInd \leftarrow EDB_{\text{cache}}[tkn]$;

⑫ $OldInd \leftarrow OldInd \setminus \{(ind, t) : \exists t_i \in DelInd \text{ s.t.} t = t_i\}$;

⑬ $Res \leftarrow NewInd \cup OldInd$, $EDB_{\text{cache}}[tkn] \leftarrow Res$;

⑭ return $Res$.

查询算法用于实现对关键词 $w$ 的搜索操作, 主要包括客户端生成查询令牌和结果搜索 2 个步骤, 即生成查询令牌和结果搜索.

1) 查询令牌生成. 查询令牌生成算法在客户端运行, 输入 $(K, w, \sigma)$, 通过查询本地 $MSK, C, \boldsymbol{A}, \boldsymbol{D}$ 的值, 生成查询令牌 $tkn$ 和穿刺密钥 $sk_R$, 并发送到服务器.

Step1 中, 客户端执行行①~⑥, 根据待查询的关键词 $w$, 从 $MSK$ 获取密钥 $msk_w$, 使用 $SRE$ 算法生成撤消加密密钥 $sk_R$, 使用伪随机函数 $F$ 生成查询令牌 $tkn$, 将 $(tkn, sk_R)$ 发送至密文索引服务器. 执行行⑦⑧, 更新本地存储状态 $\sigma$, 隐藏查询特征. 同时由于行⑦⑧与 Step2 在时间上可并行执行, 并不会对查询效率造成明显影响.

2) 结果搜索. 服务器接收到查询请求, 解析查询令牌 $tkn$ 和穿刺密钥 $sk_R$, 运行 $\Sigma_{\text{add}}$ 搜索算法查询密文索引数据库 $EDB$, 获取关键词 $w$ 未穿刺节点所有密文索引, 将搜索结果返回客户端.

Step2 中, 通过调用 $\Sigma_{\text{add}}$ 方案的 $\Sigma_{\text{add}}.Search$ 和 $SRE$ 算法, 计算查询关键词 $w$ 相关的索引, 去除已删除索引, 更新服务器端结果缓存 $EDB_{\text{cache}}$, 并向客户端返回查询结果.

**算法 3.** 更新 $\Sigma.Update(K, \sigma, op, w, ind; EDB)$.

输入：客户端输入密钥集 $K$、关键词状态 $\sigma$、操作类型 $op$、更新关键词 $w$ 及对应文档索引 $ind$, 服务器输入为密文索引数据库 $EDB$;

输出：更新后的关键词状态 $\sigma'$, 密文索引数据 $EDB'$.

客户端：

① $msk_w \leftarrow MSK[w]$, $i \leftarrow C[w]$, $\boldsymbol{D} \leftarrow \boldsymbol{D}[w]$, $\boldsymbol{A} \leftarrow \boldsymbol{A}[w]$;

② if $msk_w \leftarrow \perp$ then

③ $(msk_w, \boldsymbol{A}) \leftarrow SRE.KGen\left(1^\lambda\right)$;

④ $MSK[w] \leftarrow msk_w$, $\boldsymbol{D}[w] \leftarrow \boldsymbol{D}$;

⑤ $i \leftarrow 0$, $C[w] \leftarrow i$;

⑥ end if

⑦ if $op$=add then

⑧ 计算 $(rst, t, \boldsymbol{A}') \leftarrow F_{K_i}(w, ind)$, $InsPos.Insert(w, d_{\max}, \boldsymbol{A})$;

⑨ $ct \leftarrow SRE.Enc(d_{\max}, (msk_w, \boldsymbol{A}), ind, t)$;

⑩ $\boldsymbol{A} \leftarrow \boldsymbol{A}'$; /*更新添加列表*/

⑪ 运行 $\Sigma_{\text{add}}.Update(K_{\text{add}}, \text{add}, w \| i, (ct, t), \sigma_{\text{add}}; EDB_{\text{add}})$; /*客户端&服务器运行*/

⑫ else (i.e., $op$ = del)

⑬ $\boldsymbol{D} \leftarrow \boldsymbol{D}[w]$;

⑭ if $pos_w \geqslant 0$ then

⑮ $\boldsymbol{D}' = InsPos.Insert(w, d_{\max}, \boldsymbol{D})$;

⑯ $\boldsymbol{D}[w] \leftarrow \boldsymbol{D}'$; /*更新删除列表*/

⑰ end if

⑱ end if

数据更新包括插入和删除 2 类, 即 $op$ = add 和 $op$ = del.

插入文档索引时, 客户端查找主密钥列表, 找出 $w$ 对应密钥信息, 利用节点位置选择算法查找插入点位置, 并利用 $SRE.Enc$ 算法对索引进行加密, 修改本地密钥位置使用标志. 将密文和标签对 $(ct, \boldsymbol{A})$ 发送至

服务器，服务器运行$\Sigma_{\text{add}}.Update$算法更新密文索引数据库$EDB$.

删除文档索引时，通过$InsPos.Test$查找需要删除的节点位置，并在删除列表记录删除标记，在下次生成搜索密钥时对应位置密钥进行穿刺，服务器因无法恢复对应节点密钥信息而无法解密，从而实现后向隐私.

## 5 方案分析

下面从插入位置选择算法和搜索方案分析2个方面对方案进行理论分析.

### 5.1 插入位置选择算法分析

需要记录每个位置使用情况，至少需要1 b，对于一个节点规模为$b$列表，算法工作时需要分配$b$比特存储空间. 插入时的时间复杂度为多项式时间$\boldsymbol{O}(n)$. 为了进一步对算法的性能进行评价，现给出节点利用率定义.

**定义 10.** 节点利用率. 当$InsPos.Insert(x, d_{\max}, \boldsymbol{Lb})$首次输出$Rst=\text{false}$时，节点利用率$\rho=N/b$，其中$N$为$\boldsymbol{Lb}$中元素为1的个数.

由定义10可以看出，因为$0<N\leqslant b$，所以$0<\rho\leqslant 1$. 随着搜索深度$d_{\max}$的增大，插入点生成时间增大，空间利用率逐步升高. 当$d_{\max}=b$时，搜索插入点位置时间最长，节点利用率$\rho=1$. 在实际方案构造过程中，提高节点利用率可有效减小节点规模、减少搜索时的通信开销.

由定理1可知，插入消息$m_i$后导致发生误删除的概率与消息$m_i$在插入位置选择算法中$Rst=\text{false}$的概率相等. 当最大搜索深度为$d_{\max}$时，第$i(i\in[b])$次选择插入位置时$InsPos.Insert$算法输出发生$Rst=\text{false}$的概率等于$\boldsymbol{Lb}$在$p_m$后连续$d_{\max}+1$个节点为1的概率.

设位置选择时每次均是从空余节点中随机选择一个位置，则设指纹长度足够的情况下，节点规模为$b$、搜索深度为$d_{\max}$的列表在第$N+1(N\geqslant d_{\max})$次插入失败的概率为

$$Pr[Rst=\text{false}]=C_N^{d_{\max}}\times\prod_{i=0}^{d_{\max}-1}\frac{d_{\max}-i}{b-i}=\prod_{i=0}^{d_{\max}-1}\frac{N-i}{b-i}\leqslant\left(\frac{N-d_{\max}+1}{b-d_{\max}+1}\right)^{d_{\max}}.$$

### 5.2 搜索方案分析

结合本文方案的实现过程，从搜索效率、存储开销、通信开销和安全性等方面进行分析.

1)搜索效率

搜索时服务器仅需要计算使用的密钥节点，相比每次均需要计算GGM树所有节点密钥的Aura方案，平均搜索时间更低. 实际上GGM树节点计算随着节点规模的扩大而迅速增加，在节点规模较大、但使用节点较少的场景中，每个索引平均搜索时间将迅速增加. 本文方案平均搜索时间受节点规模影响较小，在节点规模较大但使用节点较少时具有明显的优势.

2)存储开销

客户端需要为每个节点存储指纹、使用标志、删除标志，通常使用标志和删除标志仅需1 b存储空间，则客户端需要预先为每个节点分配$n+2$比特存储空间，其中$n$为每个节点指纹宽度.

服务器中需要存储索引密文和节点位置信息，相比Aura中密文大小为索引密文大小的整数倍，不会导致密文规模扩张，存储空间占用更低，如表2所示.

3)通信开销

本文方案采用文献[22]的节点压缩算法对传输节点进行压缩，但与Aura删除前仅需传输单个节点信息不同，由于每次选择节点位置是随机的，在插入文档索引较少时，节点压缩率也较低，随着使用节点数量增多，传输节点数量将逐步降低.

在删除索引时，Aura由于中间使用的索引位置被删除，通常需要发送更多的节点.

**Table 2 Scheme Comparison**

**表2 方案对比**

| 方案 | 特点 | 搜索效率 | 存储扩展 | 通信开销 | 误删除 |
|---|---|---|---|---|---|
| Janus[12] | 非对称 | 低 | 不扩展 | 较高 | 不会 |
| Janus++[20] | SPE | 较高 | 不扩展 | 较高 | 可忽略 |
| Aura[22] | SRE | 高 | 扩展 | 相对较低 | 不可忽略 |
| 本文方案 | SRE | 高 | 不扩展 | 相对较低 | 不会 |

4)安全性

本文方案通过在服务器端增加$EDB_{\text{cache}}$保存上次查询结果，并在每次查询后更新查询密钥，以保护数据访问特征. 下面通过真实游戏$Real$和理想游戏$Idea$对方案安全性进行证明.

$Real_{\mathcal{A}}^{\Sigma}(k)$. 攻击者$\mathcal{A}$选择数据库$DB$，DO运行$Setup(DB)$算法生成检索索引$EDB$并发送给攻击者. $\mathcal{A}$按一定规则选择一系列查询$q$，DO运行$Trapdoor(q)$算法生成的$\mathcal{T}_q$并发送给$\mathcal{A}$. 服务器运行$Search(EDB, \mathcal{T}_q)$，将运行的所有结果发送给$\mathcal{A}$. 最后，$\mathcal{A}$输出一个

比特$b$.

$Idea^{\Sigma}_{\mathcal{A},\mathrm{S}}$. 模拟器初始化查询数组$q$. $\mathcal{A}$选择数据库$DB$, DO 运行$\mathrm{S}(\mathcal{L}(DB))$算法生成检索索引并发送给$\mathcal{A}$. 接着, $\mathcal{A}$按一定规则的选择查询$q$. 模拟器记录下查询$q[i]$, 运行$S(\mathcal{L}(q, DB))$. 最后, $\mathcal{A}$输出一个比特$b$.

理想游戏$Idea^{\Sigma}_{\mathcal{A},\mathrm{S}}$中, $\mathcal{A}$得到的结果均是模拟器根据泄露信息所模拟出的结果, 若$\mathcal{A}$输出的比特概率分布之差是可忽略的, 则认为理想游戏与真实游戏是计算上不可区分的.

**定理 2.** 方案$\Sigma$后向隐私自适应安全. 设$sp(w)$为关键词$w$的搜索特征, $TimeDB(w)$表示关键词$w$有关索引(去除删除索引), $DelTime(w)$表示关键词$w$上的删除操作的时间戳, 方案$\Sigma$的泄露函数$\mathcal{L}_{\mathrm{BS}}$定义为

$$\mathcal{L}_{\mathrm{BS}} = (\mathcal{L}^{Search}_{\mathrm{BS}}, \mathcal{L}^{Update}_{\mathrm{BS}}),$$

其中:

$$\mathcal{L}^{Search}_{\mathrm{BS}} = (sp(w), TimeDB(w), DelTime(w)),$$

$$\mathcal{L}^{Update}_{\mathrm{BS}}(op, w, ind) = op.$$

若$\Sigma_{\mathrm{add}}$为前向自适应隐私的对称可搜索加密方案, SRE 满足 IND-sREV-CPA 安全, 则方案$\Sigma$为随机预言机模型下对所有多项式时间攻击者$\mathcal{A}$是 Type Ⅱ 后向隐私自适应安全的.

证明. 设多项式时间攻击者$\mathcal{A}$在模拟器$S$中所取得的最大优势记为$Adv_{\mathcal{A},S}(\lambda)$, $G_i$表示 Game $i$, $Pr[E]$表示事件$E$发生的概率. 证明定理 2 的关键是针对自适应$\mathcal{A}$构造一个高效的模拟器$S$, 使得$Pr\left[Real^{\Sigma}_{\mathcal{A}}(\lambda)=1\right] - Pr[Idea^{\Sigma}_{\mathcal{A},S}=1] \leqslant Adv^{\Sigma}_{\mathcal{A},S}(\lambda)$成立, 其中$Adv^{\Sigma}_{\mathcal{A},S}(\lambda)$是可忽略的.

利用构造法, 从真实游戏构造一个满足要求的高效模拟器$S$, 进而证明定理 2.

Game 0. $G_0$为完全按照方案$\Sigma$运行的真实游戏, 则有

$$Pr\left[Real^{\Sigma}_{\mathcal{A}}(\lambda)=1\right] = Pr[G_0=1].$$

Game 1. 在$G_0$基础上, 将伪随机函数$F$使用随机数代替, 并将生成的随机数存入数组中, 在下次调用时重新找出生成的随机数. 在攻击者视野下, 相同的输入均会获得相同的随机数.

由于伪随机函数计算上具有不可区分性, 不防设使用真随机数代替函数$F$, 对所有多项式时间攻击者$B_1$获得的最大优势为$Adv^{\mathrm{prf}}_{F,B_1}(\lambda)$, 其中$Adv^{\mathrm{prf}}_{F,B_1}(\lambda)$是可忽略函数.

方案$\Sigma$中每次新加入的关键词和生成密文标签均采用随机数代替, 则攻击者$B_1$能够获得 2 倍的优势, 即

$$Pr[G_0=1] - Pr[G_1=1] \leqslant 2\times Adv^{\mathrm{prf}}_{F,B_1}(\lambda).$$

Game 2. 在$G_1$基础上, 将前向安全 SSE 方案$\Sigma_{\mathrm{add}}$使用模拟器$S^{\Sigma}_{\mathrm{add}}$代替, 同时在服务器端维护一个更新操作列表$\boldsymbol{L}_{\mathrm{add}}$, 在攻击者发出查询操作时再执行更新操作. 由于$\Sigma_{\mathrm{add}}$满足前向隐私, 更新操作不会泄露关键词和索引的有关信息.

$\boldsymbol{L}_{\mathrm{add}}$包含了方案$\Sigma$搜索时泄露的信息$\mathcal{L}^{Search}_{\mathrm{BS}}$, 即搜索特征$sp(w)$、索引密文及对应标签、更新时间$u.\boldsymbol{L}_{\mathrm{add}}$反映了关键词$w$通过方案$\Sigma_{\mathrm{add}}$执行添加操作更新的历史记录, 并作为方案$\Sigma_{\mathrm{add}}$的模拟器$S^{\Sigma}_{\mathrm{add}}$的输入. 针对方案$\Sigma_{\mathrm{add}}$的多项式时间的攻击者$B_{\mathrm{add}}$, 存在一个模拟器$S^{\Sigma}_{\mathrm{add}}$, 满足

$$Pr[G_1=1] - Pr[G_2=1] \leqslant Adv^{\mathrm{SSE},\mathcal{L}_{\mathrm{FS}}}_{\Sigma_{\mathrm{add}},B_{\mathrm{add}},S^{\Sigma}_{\mathrm{add}}}(\lambda),$$

其中$Adv^{\mathrm{SSE},\mathcal{L}_{\mathrm{FS}}}_{\Sigma_{\mathrm{add}},B_{\mathrm{add}},S^{\Sigma}_{\mathrm{add}}}(\lambda)$是可忽略的.

Game 3. 在$G_2$基础上, 在执行可穿刺加密操作前将已删除文档索引置为 0.由于 SRE 是 IND-sREV-CPA 安全的[22], 因此对攻击者$B_3$存在一个高效的模拟器$S^{\mathrm{SRE}}$, 满足

$$Pr[G_2=1] - Pr[G_3=1] \leqslant Adv^{\mathrm{IND-sREV-CPA}}_{\mathrm{SRE},B_3,S^{\mathrm{SRE}}}(\lambda),$$

其中$Adv^{\mathrm{IND-sREV-CPA}}_{\mathrm{SRE},B_3,S^{\mathrm{SRE}}}(\lambda)$是可忽略的.

Game 4. 使用更新列表获得泄露信息$TimeDB$和$DelHist$, 重建索引添加列表$\boldsymbol{L}_{\mathrm{add}}$、使用标志$\boldsymbol{A}$和删除标志$\boldsymbol{D}$, 并作为模拟器的输入. 若假设每个文档索引只会被添 1 次和删除 1 次, 则标签不会被重复使用, 也就不必为了保证一致性而为每个密文索引存储对应的密文标签.

在$G_4$中, 仅对模拟器的工作方式进行了修改, 即

$$Pr[G_4=1] = Pr[G_3=1].$$

为构造理想游戏的模拟器, 不能直接调用关键词$w$, 而改用查询特征$sp[w]$来生成令牌信息. 同时, 不再保持维护更新记录的列表, 而是利用搜索查询中的泄露信息$TimeDB(w)$, $DelTime(w)$重建添加列表$\boldsymbol{L}_{\mathrm{add}}$和节点使用标志$\boldsymbol{A}$、删除标志$\boldsymbol{D}$. 这样利用$\mathcal{L}_{\mathrm{BS}}$有关信息, 构造了一个高效的模拟器$S$, 使得

$$Pr[G_4=1] = Pr\left[Idea^{\Sigma}_{\mathcal{A},S,\mathcal{L}_{\mathrm{BS}}}(\lambda)=1\right].$$

将以上多项式时间攻击者$B_1$, $B_{\mathrm{add}}$, $B_3$对方案$\Sigma$攻击所获得的优势相加, 可得

$$\left|Pr\left[Real^{\Sigma}_{\mathcal{A}}(\lambda)=1\right] - Pr\left[Idea^{\Sigma}_{\mathcal{A},S,\mathcal{L}_{\mathrm{BS}}}(\lambda)=1\right]\right| \leqslant 2\times Adv^{\mathrm{prf}}_{F,B_1}(\lambda) + Adv^{\mathrm{SSE},\mathcal{L}_{\mathrm{FS}}}_{\Sigma_{\mathrm{add}},B_{\mathrm{add}},S^{\Sigma}_{\mathrm{add}}}(\lambda) + Adv^{\mathrm{IND-sREV-CPA}}_{\mathrm{SRE},B_3,S^{\mathrm{SRE}}}(\lambda) = Adv^{\Sigma}_{\mathcal{A},S}(\lambda),$$

其中$Adv^{\Sigma}_{\mathcal{A},S}(\lambda)$是可忽略的, 即定理 2 成立. 证毕.

## 6 方案测试

方案性能测试所采用的软/硬件环境配置如表 3 所示.

**Table 3 Environment Configuration of Software/ Hardware for Testing**

**表 3 测试软/硬环境配置**

| 环境 | 配置项 | 配置参数 |
|---|---|---|
| 硬件 | 处理器 | Intel® Core™ i7-8550U CPU @ 1.80 GHz 1.99 GHz |
| | 内存 | 8 GB |
| 软件 | 系统 | Windows 11 家庭中文版 wsl（Ubuntu 20.04） |
| | 编程语言 | C++ |
| | 开发环境 | VSCode 1.64.0 |

### 6.1 插入点位置选择算法测试

插入点位置选择算法中，消息位置、消息指纹所使用的哈希函数均采用 SpookyHash V2 算法，随机生成消息 $m$，发生第 1 次插入失败(即未找到可用节点或发生指纹冲突)时，每个搜索深度下随机生成 500 组索引，节点利用率$\rho$的平均值随指纹宽度$n$和搜索深度$d_{\max}$的变化关系如图 3 所示.

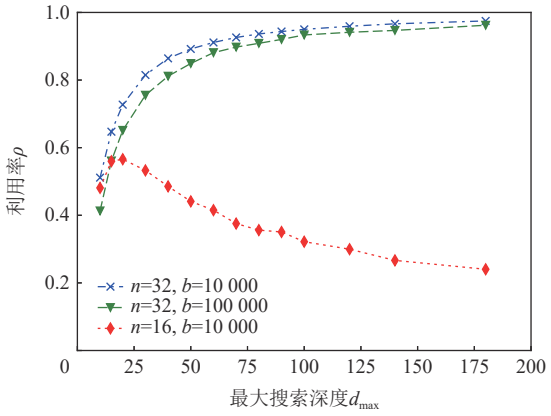


Fig. 3 The curve of node utilization rate

图 3 节点利用率曲线

可以看出，在$n=32$, $d_{\max}=100$时能够保证较高的节点利用率.

### 6.2 搜索方案性能测试

对称加密算法采用 AES，哈希算法选用计算速度较快的 SpookyHash V2，伪随机数发生器采用 GGM 算法实现，采用支持分布式部署的非关系数据库 rocksDB 作为密文索引数据库. 测试中，随机生成文档索引，利用$\Sigma.Update$算法添加至密文索引数据库.

如表 4 所示，从存储空间开销可以看出，Aura 在

**Table 4 Actual Storage Overhead of Schemes**

**表 4 方案实际存储开销**

| 方案 | 节点规模 | 参数设置 | 存储开销 | |
|---|---|---|---|---|
| | | | 客户端开销/（B·节点$^{-1}$） | 服务器开销/MB |
| Aura[22] | 200 000 | 3 个哈希函数 | | 19.6 |
| | 500 000 | | | 19.6 |
| | 800 000 | | | 19.6 |
| 本文方案 | 800 000 | $n=32$ $d_{\max}=100$ | 10 | 11.3 |

注：$n$为指纹宽度，$d_{\max}$为搜索深度，关键词个数$nw=20$，索引数$N=100000$.

服务器中因需要同时存储多个密文备份，服务器占用空间开销更大. 相比 Aura 方案，本文方案每个密文索引不存在密文扩展，实际平均每条索引占用存储空间开销更小. 在客户端，密文删除操作需要为每个节点存储一个指纹，存储客户端存储空间开销为$O(b)$. 节点实际占用空间包含了 rocksDB 内部存储结构数据，方案实际占用的空间相比测试值要更小. 另外，采用质量更好的指纹哈希算法，降低每个节点的指纹长度，也可进一步快速降低客户端中存储空间的大小.

由于 GGM 树随着节点规模的扩大，其计算时间迅速增加，如图 4 所示，本文方案通过增加节点使用列表，避免每次搜索时服务器需要生成全部穿刺密钥，大大减少在匹配节点较少时搜索的响应时间，随着匹配节点数的增多，方案平均搜索时间趋于一致.

如图 5 所示，在节点规模$b=60\,000$时，客户端运行查询算法时向服务器发送的节点个数随节点利用率$\rho$的变化情况. 在插入文档索引较少时，发送数据随文档索引增加而增加，当$\rho>0.3$时，发送的节点数逐渐提高.

图 6 为删除索引后查询到的匹配节点个数变化情况，可以看出本文方案不会发生误删除. 图 7 为删

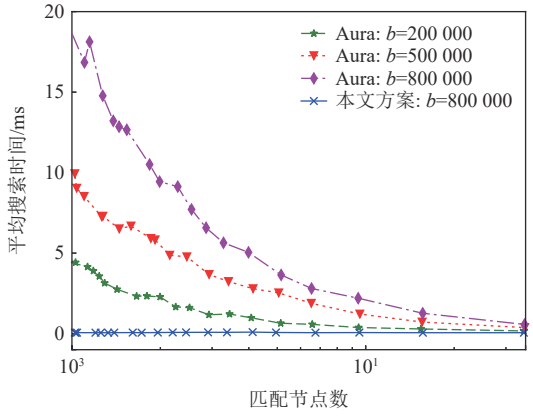


Fig. 4 The comparison of actual search time ($N=100\,000$)

图 4 实际搜索时间对比（$N=100\,000$）

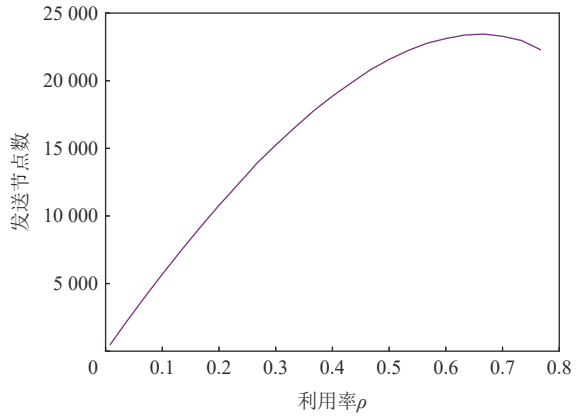


Fig. 5 Communication overhead of our scheme ($n = 32$, $b = 60\,000$)

图 5 本文方案通信开销（$n = 32$, $b = 60\,000$）

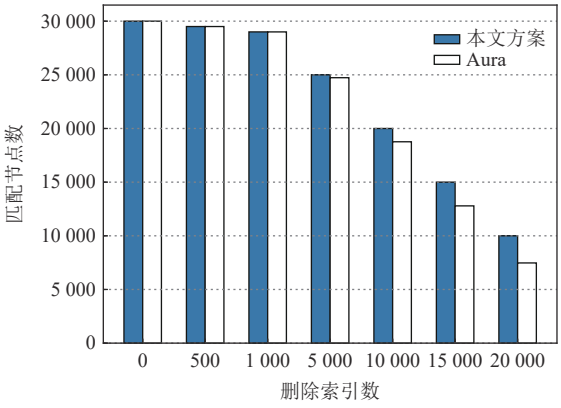


Fig. 6 Search results after index deletion ($b = 60\,000$, $N = 30\,000$)

图 6 删除索引后搜索结果（$b = 60\,000$, $N = 30\,000$）

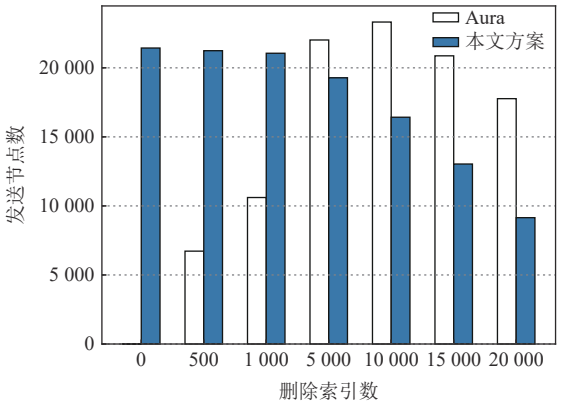


Fig. 7 Scheme communication overhead comparison after index deletion ($b = 60\,000$, $N = 30\,000$)

图 7 删除索引后方案通信开销对比（$b = 60\,000$, $N = 30\,000$）

除索引时方案通信开销变化情况. 从图 7 可以看出，在频繁发生索引删除的 DSSE 应用场景下，本文方案通信开销也随着删除节点增多而逐渐降低.

## 7 总 结

本文结合可搜索加密方案的应用场景，给出了动态对称可搜索方案的形式化描述. 同时给出了 SRE 更严格的正确性定义，并通过理论分析证明了避免误删除发生的条件. 在设计插入点位置选择算法的基础上，构造了基于 SRE 的对称可搜索加密方案.

大多数情况下，关键词的使用密钥节点数远小于未使用节点数，通过为每个关键词添加节点插入标志，记录每个关键词实际使用的节点位置信息，查询时只发送已使用节点，可以大大降低服务器恢复密钥时的时间开销. 理论分析和实现结果表明，本文方案进一步优化了 Aura 方案搜索效率，有效避免了节点误删除问题. 然而方案需要在客户端存储更多节点信息，增加了客户端数据存储开销，同时在删除索引较少的场景下，本文方案具有更大的通信开销. 下一步将在减少客户端存储开销、方案通信开销以及多客户端场景下的扩展等问题上展开进一步研究.

**作者贡献声明**：黄一才提出了算法思路和实验方案，并撰写了论文；郁滨提出指导意见并修改论文；李森森参与了实验方案制定及部分数据分析与整理.

## 参 考 文 献

[1] Song Xiaodong, Wagner D, Perrig A. Practical techniques for searches on encrypted data[C] // Proc of the 21st IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2000: 44−55

[2] Goh E J. Secure indexes [EB/OL]. Cryptography Archive. 2003[2021-03-01]. http://eprint.iacr.org/2003/216

[3] Naveed M, Prabhakaran M, Gunter C A. Dynamic searchable encryption via blind storage[C] // Proc of the 35th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2014: 639−654

[4] 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

[5] Hahn F, Kerschbaum F. Searchable encryption with secure and efficient updates[C] // Proc of the ACM Conf on Computer and Communications Security. New York: ACM, 2014: 310−320

[6] Ghareh C J, Papadopoulos D, Papamanthou C, et al. New constructions for forward and backward private symmetric searchable encryption[C] // Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security(CCS'18). New York: ACM, 2018: 1038−1055

[7] Naveed M. The fallacy of composition of oblivious RAM and searchable encryption[EB/OL]. 2015[2021-06-20]. https://eprint.iacr.org/2015/668

[8] Kamara S, Papamanthou C. Parallel and dynamic searchable symmetric encryption[G] // LNCS 7859: Proc of the 17th Financial Cryptography and Data Security. Berlin: Springer, 2013: 258–274

[9] Song Xiangfu, Dong Changyu, Yuan Dandan, et al. Forward private searchable symmetric encryption with optimized I/O efficiency[J]. IEEE Transactions on Dependable and Secure Computing, 2020, 17(5): 912−927

[10] Bossuat A, Bost R, Fouque P A, et al. SSE and SSD: Page-efficient searchable symmetric encryption[EB/OL]. 2021[2021-10-21].https://eprint.iacr.org/2021/716

[11] He Kun, Chen Jing, Zhou Qinxi, et al. Secure dynamic searchable symmetric encryption with constant client storage cost[J]. IEEE Transactions Information Forensics and Security, 2021, 16: 1538−1549

[12] Bost R, Minaud B, Ohrimenko O. Forward and backward private searchable encryption from constrained cryptographic primitives[C] // Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2017: 1465−1482

[13] Bost R. ∑οφος: Forward secure searchable encryption[C] // Proc of ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 1143−1154

[14] Etemad M, Küp ü A, Papamanthou C, et al. Efficient dynamic searchable encryption with forward privacy[J]. Proceedings on Privacy Enhancing Technologies, 2018(1): 5−20

[15] Wang Qiao, Guo Yu, Huang Hejiao, et al. Multi-user forward secure dynamic searchable symmetric encryption[G] // LNCS 11058: Proc of Symp on NSS 2018. Berlin: Springer, 2018: 125−140

[16] Li Jin, Huang Yanyu, Wei Yu, et al. Searchable symmetric encryption with forward search privacy[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 18(1): 460−474

[17] Huang Ke, Dong Xiaolei, Cao Zhenfu, et al. Dynamic searchable symmetric encryption schemes with forward and backward security[C/OL] // Proc of IOP Conf Series: Materials Science and Engineering. 2020[2023-01-16].https://iopscience.iop.org/article/10.1088/1757−899X/715/1/012062

[18] Zuo Cong, Sun Shifeng, Liu J K, et al. Forward and backward private DSSE for range queries[J]. IEEE Transactions on Dependable and Secure Computing, 2022, 19(1): 328−338

[19] Amjad G, Kamara S, Moataz T. Forward and backward private searchable encryption with SGX[C] // Proc of ACM Conf on Euro-Sec'19. New York: ACM, 2019: 1143−1154

[20] Sun Shifeng, Yuan Xingliang, Joseph K L, et al. Practical backward-secure searchable encryption from symmetric puncturable encryption[C] //Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security(CCS'18). New York: ACM, 2018: 763−780

[21] Hohenberger S, Koppula V, Waters B. Adaptively secure puncturable pseudorandom functions in the standard model[G] // LNCS 9452: Proc of Advances in Cryptology (ASIACRYPT 2015). Berlin: Springer, 2015: 79–102

[22] Sun Shifeng, Steinfeld R, Lai Shangqi, et al. Practical non-interactive searchable encryption with forward and backward privacy[C/OL] // Proc of Symp on Network and Distributed Systems Security (NDSS). 2021[2022-01-23].https://dx. doi.org/10.14722/ndss.2021.24162

[23] Ray I G, Rahulamathavan Y, Rajarajan M. A new lightweight symmetric searchable encryption scheme for string identification[J]. IEEE Transactions on Cloud Computing, 2020, 8(3): 672−684

[24] Zhang Yupeng, Jonathan K, Charalampos P. All your queries are belong to us: The power of file-injection attacks on searchable encryption[C] // Proc of Symp on USENIX Security. Berkeley, CA: USENIX Association. 2016: 707–720

[25] Wang Yunling, Chen Xiaofeng. Research on searchable symmetric encryption[J]. Journal of Electronics & Information Technology, 2020, 42(10): 2374−2385 (in Chinese)
(王赟玲, 陈晓峰. 对称可搜索加密技术研究进展[J]. 电子与信息学报, 2020, 42(10): 2374−2385)

**Huang Yicai**, born in 1985. PhD candidate. His main research interests include IoT, secure cloud storage system, and searchable encryption.

**黄一才**, 1985 年生. 博士研究生. 主要研究方向为物联网、安全云存储系统、可搜索加密.

**Yu Bin**, born in 1964. PhD, professor, PhD supervisor. His main research interests include the design and analysis of algorithms, visual cryptography, and network security.

**郁　滨**, 1964 年生. 博士, 教授, 博士生导师. 主要研究方向为算法设计与分析、视觉密码、网络安全.

**Li Sensen**, born in 1993. PhD candidate. His main research interests include network security and wireless communication technology.

**李森森**, 1993 年生. 博士研究生. 主要研究方向为网络安全、无线通信技术.