中国科学 : 信息科学 2025 年 第 55 卷 第 9 期: 2319–2338
SCIENTIA SINICA Informationis
论文

后泄露安全下的密钥泄露可验证可搜索加密技术研究

王蔚[1], 雷淑芳[1], 刘东立[1], 徐鹏[2*], 杨天若[3]

1. 华中科技大学计算机科学与技术学院, 武汉 430074
2. 华中科技大学网络空间安全学院, 武汉 430074
3. 郑州大学计算机与人工智能学院, 郑州 450001
* 通信作者. E-mail: xupeng@hust.edu.cn

收稿日期: 2025–02–10; 修回日期: 2025–05–12; 接受日期: 2025–06–16; 网络出版日期: 2025–09–08

国家自然科学面上基金 (批准号: 62372201, 62272186) 资助项目

摘要 可搜索加密技术可实现加密数据的安全搜索功能保障用户在云服务器上的安全搜索, 然而实际应用中, 密钥的泄露风险严重威胁数据的安全与隐私. Bamboo 机制在密钥泄露后实现密钥更新, 并通过两层加密和隐藏链式结构保障密钥泄露后至更新前的密文安全, 但未考虑如何主动检测到密钥泄露, 从而触发密钥更新. 因此, 本文提出了一种基于密钥可更新技术的密钥泄露可验证可搜索加密方案 KLVSE, 通过引入双向验证机制, 验证密钥泄露事件, 检测敌手潜在的非法搜索行为, 确保在密钥泄露后及时触发密钥更新, 保障了密钥泄露后安全. 实验证明, 相比传统的定期密钥更新策略, 本方案在保证安全的同时以足够小的运行时延实现了密钥泄露的验证, 有效保障了用户数据的安全和隐私.

关键词 可搜索加密, 密钥泄露可验证, 密钥可更新, 密钥泄露后安全

1 引言

可搜索加密 [1] 是一种通过密文索引保障检索关键字隐私的加密技术, 确保加密数据能够安全地托管于云服务器上, 并实现由云服务器对加密的关键字陷门检索相关数据, 承载网络信息空间数据安全隐私保护作用. 它通过融合可搜索加密与云服务, 在保障客户隐私的同时显著减轻了客户端在数据存储与计算方面的负担.

在采用可搜索对称加密技术的方案中, 客户利用密钥将明文数据加密成密文并将其外包存储到云服务器上. 执行搜索时, 客户端根据特定的搜索关键字生成对应的搜索陷门发送给云服务器, 云服务器根据该搜索陷门检索相关的密文, 并将搜索的结果以密文的形式返回给客户端. 最后客户端使用相同的密钥解密接收到的密文得到原始的明文数据. 在上述过程中, 客户检索隐私的保障依赖于客户密钥的安全性, 因此密钥的安全成为焦点. 然而, 现实生活中存在大量密钥安全受损的实例, 比如在资源

引用格式: 王蔚, 雷淑芳, 刘东立, 等. 后泄露安全下的密钥泄露可验证可搜索加密技术研究. 中国科学: 信息科学, 2025, 55: 2319–2338, doi: 10.1360/SSI-2025-0053
Wang W, Lei S F, Liu D L, et al. Research on key leakage-verifiable searchable encryption with post-compromise security. Sci Sin Inform, 2025, 55: 2319–2338, doi: 10.1360/SSI-2025-0053

受限的物联网应用中客户端容易遭受攻击导致密钥泄露 [2], 2018 年 HTTPS 证书经销商 Trustico 曝出 23000 个 HTTPS 证书的密钥在网络攻击下泄露 [3] 的安全事件, Imperva 公司由于云 API 密钥被盗继而泄露了客户的数据 [4]. 可搜索加密中同样存在敌手恶意窃取密钥的情况, 给客户的数据安全和检索隐私造成侵害.

为了降低密钥泄露的影响, 系统需在密钥泄露后重建数据库并更新密钥. 可更新加密 [5] 是一种能在密钥泄露场景下保障密钥泄露后数据安全的技术. 该技术通过密钥更新, 将旧密文更新为新密钥下的密文, 并在确保更新密钥后加密的数据无法被旧密钥破解的同时, 保证现存密文的数据安全, 从而达到密钥泄露后安全.

然而, 在密钥更新操作中, 系统并不具备及时发现密钥泄露的能力. 例如, 文献 [6] 面向需要更新密钥的情形 (如员工离职后撤销其访问数据权限), 通过使用基于密钥同态伪随机函数 (key homomorphic PRF) 的对称代理重加密技术 (proxy re-encryption, PRE) 对存储在云服务器的加密数据进行密钥更新, 从而保证数据的隐私和安全. 但该方案仅支持定期更新密钥, 本质上未解决密钥泄露检测的问题, 在实际应用中依然有安全隐患. 文献 [7] 提出的 ROSE 方案虽然引入了密钥可更新伪随机函数, 但仅将该原语用于构造具有健壮性和前后向安全性的实例化方案, 并未涉及密钥泄露检测问题. 文献 [8] 则仅提及在发现密钥泄露之后方可使用密钥可更新伪随机函数实现密钥更新, 但同样未明确如何发现泄露. 上述情况表明在敌手窃取密钥后, 客户端无法及时获知密钥失窃.

针对这一现象, 本文将研究如何在动态可搜索对称加密运行中检测到密钥泄露的行为. 首先本文将敌手发起攻击的方式分为在线搜索和离线搜索两种:

– 在线搜索指敌手向云服务器发送在线搜索查询, 由云服务器完成检索并返回搜索结果;

– 离线搜索指敌手下载云服务器端的密文数据库副本, 并利用获取的密钥在本地自行搜索.

针对敌手上述两种攻击方式, 采取的抵御策略不同. 在线搜索情况下适宜执行密钥泄露检测, 如文献 [9] 中采用认证和参数独立/定期同步方式检测密钥泄露, 文献 [10] 中应用区块链技术提供了具有不可篡改性和透明度的密钥使用记录, 监控受保护应用程序的密钥使用情况以检测密钥是否泄露. 离线搜索由敌手在本地自行完成, 云服务器无法感知攻击行为, 这将导致密钥泄露状态难以被检测, 从而无法及时更新密钥. 为了确保密钥泄露能被检测, 则需满足云服务器具备唯一密文检索能力, 使得敌手无法通过下载密文数据库副本完成搜索, 仅能通过云服务器发起在线搜索攻击. 依据以上分析, 由于泄漏检测技术一般在云服务器上进行, 本文提出构想: 当敌手获取密钥后, 如果不对密文数据库进行检索, 则对数据安全不产生威胁; 只有敌手利用泄露的密钥对密文数据库进行关键字检索时, 才会造成相应的数据隐私泄密.

确保在线搜索为敌手唯一攻击方式之后, 本文重点考虑构建高效及时的密钥泄露发现机制, 通过特定验证算法验证密钥泄露的发生, 进而触发密钥更新, 以保障密钥泄露后安全.

为实现上述构想, 本文提出基于密钥可更新技术的密钥泄露可验证可搜索加密方案 (key leakage-verifiable searchable encryption, KLVSE). 针对现有方案中存在的敌手攻击方式不明确、缺乏有效的密钥泄露检测方法的问题, 本文首先在威胁模型中分析了敌手在线搜索和离线搜索两种攻击方式, 通过设置密文索引与文件的同步映射机制, 保证云服务器唯一搜索的能力, 有效阻断敌手的离线攻击, 将敌手的攻击方式限定为在线搜索这一单一途径. 在此基础上, 通过引入双向验证机制, 检测敌手的非法搜索行为, 从而验证密钥泄露事件的发生, 及时触发密钥更新, 保障密钥泄露后安全. 实验显示, 与最新的密钥可更新但缺乏验证反馈机制的 Bamboo 方案 [8] 相比, 我们的方案在搜索时间方面额外增加的时延几乎可以忽略不计, 总体上增加损耗 1% 左右, 最小 0.07%, 最大不足 1.7%; 同时验证时间开销控制在 10 μs 以内且保持恒定, 显示了方案的高效性.

2　相关工作

2000 年, Song 等 [1] 首次提出了利用关键字检索加密数据的可搜索加密方案, 并首次定义了可搜索对称加密 (searchable symmetric encryption, SSE), 它使用相同的密钥加密和搜索数据, 可以实现快速高效的密文数据搜索, 但当前密钥可更新的可搜索加密方案均未实现及时的密钥更新策略.

2.1　动态可搜索对称加密

2012 年, Kamara 等 [11] 在 SSE 技术的基础上进一步引入了动态可搜索对称加密 (dynamic searchable symmetric encryption, DSSE), 允许密文索引的动态更新, 比如添加/删除文件索引, 但这同时也会引发额外的信息泄露以及前向/后向安全性的问题, 前向安全性确保新更新的条目无法与以前的更新和搜索查询关联 [12], 后向安全性确保无法通过后续的搜索查询找到已删除的条目 [13]. 之后的大量工作在原始 DSSE 模型的基础上进行改进与优化 [7,8,13~18], 在安全性和效率上进一步提高. 例如, Chen 等 [19] 提出 Bestie 方案以简洁的结构实现了索引结构动态更新的高效率. 此外, 还有关于 SSE 场景下健壮性问题的研究工作 [20].

2.2　可更新加密与密钥可更新

基于可更新加密与密钥可更新技术, 云服务器能够借助客户端提供的密钥更新令牌将原先使用旧密钥加密的密文更新为新密钥下的密文, 但不向敌手泄露任何有关明文的信息. 密钥可更新可以使用独立于密文的可更新加密 [5,21,22], 比如 2023 年, Yang 等 [23] 在公钥体制下依赖于区块链共识机制更新密钥. 对称体制下可以使用代理重加密技术, 如 2020 年 Boyd 等 [21] 提出的代理重加密方案允许密钥持有者生成新的加密密钥更新密文, 而无需访问原始密钥或解密数据, 但这种方法无法保证自密钥泄露后至密钥更新完成过程中生成的密文安全性. 另一种方法是改造伪随机函数 (pseudorandom function, PRF), 将基础的 PRF 改造成具备密钥更新功能的密码原语. 比如 2013 年, Boneh 等 [6] 利用密钥同态伪随机函数构建对称密钥代理重加密, 定期进行密钥轮换以实现可更新加密. 2022 年, Xu 等 [7] 引入密钥可更新伪随机函数, 并利用该原语设计了一种具有前向安全和后向安全的密钥可更新 DSSE 方案. 2023 年, Chen 等 [8] 提出 Bamboo 方案, 利用密钥可更新 PRF 实现密钥更新, 保障密钥泄露后安全. 但是上述方案的共同缺陷是密钥只能以定期频率更新, 无法实现密钥泄露后的及时反馈和密钥更新.

2.3　可验证的可搜索加密

2012 年, Chai 等 [24] 首次提出基于加密数据的可验证关键字搜索方案, 对搜索结果进行验证. 由此, 可验证的 SSE (verifiable SSE, VSSE) 被广泛提出与应用.

在 VSSE 中, 验证的内容主要包括正确性、完整性、完备性、时新性等. 正确性是指查询返回的每一个文件 id 是正确的搜索结果, 比如方案 [25,26] 支持静态数据库中的正确性验证, 方案 [27] 利用默克尔树 (Merkle tree) 验证数据所有者、用户和云服务器三方操作的正确性; 文献 [16,28~30] 利用增量哈希、累加器、同态 MAC 等技术实现动态数据库上的正确性验证; 此外还有基于区块链技术 [31~34] 和可信硬件的正确性验证 [35]. 完整性指查询返回结果包含了符合查询条件的所有文件标识 id, 比如方案 [16,24] 使用 proof 在保证正确性的同时也实现了完整性验证. 完备性则是保证查询返回结果同时满足正确性和完整性的定义, 防止恶意云服务器返回部分结果或被篡改的结果, 一些工作直接将正确性、完整性与完备性的定义杂糅共同使用 [16,24], 也有工作单独提出囊括了正确性与完整性定义的完备性验证 [17,31,35]. 时新性验证 [18,27,32,36] 指云服务器返回的是最新的数据结果, 而非历史数据库版本中的查询结果. 然而, 目前验证技术主要用于验证云服务器的搜索结果, 并未考虑验证密钥是否泄露.

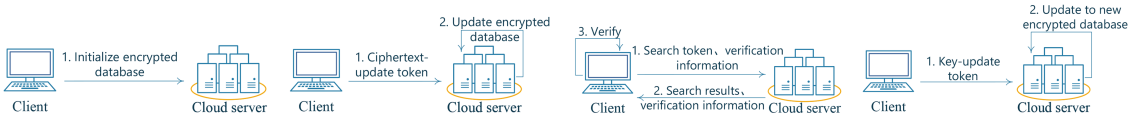

图 1　(网络版彩图) 系统模型主体模块运行流程图. (a) 初始化; (b) 更新密文; (c) 搜索与验证; (d) 更新密钥.

Figure 1　(Color online) Operation flowchart of the main modules of the system model. (a) Initialize; (b) ciphertext update; (c) search & verify; (d) key update.

表 1　主要符号.

Table 1　Main symbols.

Symbol	Description
λ	System security parameter
$a_{\max}$	The padding constant used for hiding the real search result size
K	The key for searchable symmetric encryption
State	Private state of the client
C	Searchable ciphertext
EDB	Encrypted database
w	Keyword
id	File identifier
R_u	Random number for generating verification information
ID	User/client identity
$\mathbf{H}_{\mathrm{L}}(m,k)/\mathbf{H}_{\mathrm{R}}(m,k')$	Keyed hash function
δ	Verification information sent by the client to the cloud server
β	Verification information sent by the cloud server to the client
$e \xleftarrow{\$} \mathcal{X}$	Uniformly sampling an element e from a distribution or set $\mathcal{X}$
tk	Search token
Δ	Key-update token

3　系统模型

本节将介绍密钥泄露可验证可搜索加密 KLVSE 的系统模型, 包含角色介绍、威胁模型、密钥泄露可验证可搜索加密系统及其安全模型, 其中威胁模型分析了两种不同能力的敌手产生的威胁情况, 并设置了相应的机制阻断敌手的离线攻击, 以便安全模型的定义.

图 1 展示了 KLVSE 系统各模块运行流程. (a) 初始化: 客户端和云服务器初始化参数和密文数据库. (b) 更新密文: 客户端向云服务器发送密文更新令牌, 云服务器完成密文更新. (c) 搜索与验证: 客户端向云服务器发送搜索令牌和验证信息, 云服务器完成客户端身份验证和密文搜索并返回结果, 客户端验证云服务器的搜索状态与本地是否一致. (d) 更新密钥: 客户端向云服务器发送密钥更新令牌, 云服务器完成密钥更新.

为方便阅读和理解, 本文主要符号的说明如表 1 所示.

3.1　角色

在 KLVSE 中, 存在客户端和云服务器两种角色.

• **客户端.** 客户端将数据加密存储于外包云服务器上, 通过向云服务器发起密文更新查询和搜索查询请求来存取相应的密文数据, 并能实现验证以判断是否触发密钥更新.

• **云服务器.** 云服务器负责存储和处理密文数据. 它接收密文更新查询后会根据密文更新令牌添

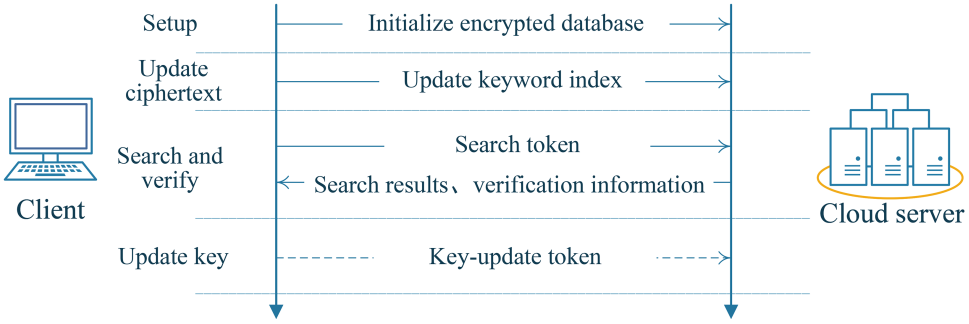

图 2　(网络版彩图) 支持密钥可更新的密钥泄露可验证可搜索加密系统.

Figure 2　(Color online) Key leakage-verifiable searchable encryption system.

加或删除外包密文数据; 接收搜索查询后会根据搜索令牌找到匹配的结果, 并将搜索结果和生成的验证信息返回给客户端; 接收密钥更新查询后会根据密钥更新令牌将原有的密文转换为采用新密钥加密的新密文.

3.2　威胁模型

针对 KLVSE 上两种攻击方式, 本文将敌手分为两类.

– **敌手 S (Spy).** 能窃听客户端与云服务器之间的通信, 通过短暂控制客户端等手段窃取客户端的密钥, 并向云服务器发起搜索查询.

– **敌手 T (Thief).** 具有 **敌手 S** 的能力, 同时还能下载密文数据库副本到本地, 并利用获取的密钥自行检索.

为了抵御敌手造成的数据安全威胁, 关键在于检测敌手冒充客户端发起了非法搜索行为并及时更新密钥. 具体而言, 如果敌手发起了非法搜索, 则合法客户端能通过验证机制检测到云服务器的状态与本地不同步, 表明密钥发生泄露, 验证不通过, 由此触发密钥更新. 分析两类敌手如下.

(1) **敌手 S** 能窃取密钥并利用该密钥向云服务器发起搜索查询请求. 在验证机制下, **敌手 S** 的搜索行为会引起云服务器端的状态发生变化, 使得合法客户端随后发起搜索查询时无法验证通过云服务器返回的验证信息, 从而及时更新密钥, 防止 **敌手 S** 利用获取的密钥持续进行非法操作.

(2) **敌手 T** 能窃取密钥并下载云端密文数据库的副本以便本地离线搜索. 与 **敌手 S** 相比, **敌手 T** 能力更强, 且其离线搜索的攻击方式不易被察觉. 因此为了抵御 **敌手 T**, 需要有必要的机制阻断 **敌手 T** 的离线搜索方式: 设置索引与文件的映射机制, 保证云服务器唯一搜索的能力. 具体来说, 云服务器端存储文件索引, 而实际文件存储在数据中心, 通过设计索引与文件的同步映射机制, 使得 **敌手 T** 无法通过下载数据库的副本进行离线搜索, 只能在线向云服务器发送搜索请求. 每次搜索时, 需要云服务器从数据中心取回与索引对应的文件; 搜索完成后, 文件索引与文件的映射关系会被打乱重置. 显然 **敌手 T** 无法代替云服务器完成上述操作, 从而确保即使 **敌手 T** 复制了云服务器端的密文数据库, 也无法同步更新密文索引与文件之间的映射关系, 即无法获取正确的文件. 在此机制下, **敌手 T** 的攻击被阻断, 此时可以通过检测和抵御 **敌手 S** 的方式来应对 **敌手 T** 的安全威胁.

综上, 通过对 **敌手 T** 能力的阻断, 使得 **敌手 T** 和 **敌手 S** 均可通过对搜索行为的验证机制来检测, 以判断密钥是否泄露, 一旦发现密钥泄露, 立即触发密钥更新, 从而抵御敌手利用泄露密钥产生的非法搜索威胁.

3.3　密钥泄露可验证可搜索加密系统

如图 2 所示, KLVSE 模型分为 4 个阶段.

- **建立阶段.** 客户端和云服务器运行 Setup 协议, 根据系统安全参数进行初始化.

• **密文更新阶段.** 客户端和云服务器运行 DataUpdate 协议, 客户端使用密钥生成可搜索密文, 并传输至云服务器存储在密文数据库中.

• **搜索与验证阶段.** 客户端和云服务器运行 Search 协议, 客户端根据搜索关键字, 使用检索密钥生成搜索令牌; 云服务器根据搜索令牌进行关键字搜索, 并生成相应的验证信息, 将搜索结果及验证信息返回给客户端. 之后, 客户端运行 Verify 协议, 对验证信息处理分析, 判断是否产生密钥泄露.

• **密钥更新阶段.** 根据验证结果判断是否更新密钥, 若验证显示密钥泄露, 则运行 KeyUpdate 协议, 触发密钥更新.

具体而言, KLVSE 细分为 5 个协议.

(1) Setup 协议. 客户端输入安全参数 λ、最大填充值 $a_{\max}$, 输出密文数据库并发送给云服务器, 主要功能是根据系统安全参数完成系统函数、用户内部状态和密文数据库的初始化.

(2) DataUpdate 协议. 客户端输入密钥 K、私有状态 **State**, 生成密文 **C** 并发送给云服务器, 云服务器输出更新后的密文数据库 **EDB**, 主要功能是生成可搜索密文并存储于密文数据库.

(3) Search 协议. 客户端输入密钥 K、私有状态 **State**、关键字 w、随机数 R_u、用户 ID, 输出搜索结果、验证信息, 主要功能是云服务器根据搜索令牌完成关键字搜索, 生成搜索结果和验证信息发送给客户端, 客户端完成解密.

(4) Verify 协议. 客户端输入随机数 R_u、用户 ID、验证信息、搜索计数器, 输出验证结果 1/0 代表验证通过与否, 主要功能是验证是否产生密钥泄露, 以判断是否需要触发密钥更新.

(5) KeyUpdate 协议. 客户端输入密钥 K、私有状态 **State**, 输出更新后的新密文数据库, 主要功能是完成密钥更新, 并更新旧密文为新密文.

正确性. 如果对于任何 $\lambda \in \mathbb{N}$ 和 $(K_\Sigma, \mathbf{State}; \mathbf{EDB}) \leftarrow \text{Setup}(\lambda, a_{\max})$, 面向任意多项式时间内执行的 DataUpdate$(K_\Sigma, \mathbf{State}, \text{op}, (w, \text{id}); \mathbf{EDB})$, Search$(K_\Sigma, \mathbf{State}, w, R_u, \text{ID}; \mathbf{EDB})$, KeyUpdate$(K_\Sigma,$ **State**; **EDB**), 满足以下条件: 协议 Search$(K_\Sigma, \mathbf{State}, w, R_u, \text{ID}; \mathbf{EDB})$ 总是返回与特定关键字 w 配对的文件 id 集合, 其中这些 id 通过 DataUpdate$(K_\Sigma, \mathbf{State}, \text{op} = \text{add}, (w, \text{id}); \mathbf{EDB})$ 插入到 **EDB** 中, 且尚未被 DataUpdate$(K_\Sigma,$ **State**, op $=$ del$, (w, \text{id}); \mathbf{EDB})$ 删除, 而且若 Verify$(R_u, \text{ID}, \beta, \text{srchcnt}_u)$ 验证通过则不执行密钥更新, 若验证不通过则立即调用密钥更新, 则该 KLVSE 方案是正确的.

3.4　安全模型

此节定义了 KLVSE 的安全模型, 首先给出适应性安全性的定义, 之后根据适应性安全性和相关泄露函数给出密钥泄露后安全 (post-compromise security) 定义.

3.4.1　适应性安全性

本文定义的适应性安全性使得敌手无法区分真实和理想的 KLVSE 游戏. 其中, 真实 KLVSE 游戏是真实运行的 KLVSE 方案, 而理想 KLVSE 游戏是理想状态下运行的 KLVSE 方案的模拟器, 其输入为对应的泄露函数, 即泄露的信息. 由于诚实且好奇的云服务器和敌手都试图获取客户的隐私信息, 下面分别定义 KLVSE 对云服务器 $\mathcal{A}_{\text{Srv}}$ 和敌手 $\mathcal{A}_{\text{Adv}}$ 的适应性安全性.

定义1 (针对 $\mathcal{A}_{\text{Srv}}$ 的适应性安全性)　给定泄露函数 $\mathcal{L}_{\text{Srv}} = (\mathcal{L}_{\text{Srv}}^{\text{Stp}}, \mathcal{L}_{\text{Srv}}^{\text{DaUpdt}}, \mathcal{L}_{\text{Srv}}^{\text{Srch}}, \mathcal{L}_{\text{Srv}}^{\text{KeyUpdt}})$, 如果对于任何足够大的安全参数 $\lambda \in \mathbb{N}$ 和概率多项式时间敌手 $\mathcal{A}_{\text{Srv}}$, 存在一个可有效运行的模拟器 $\mathcal{S} = (\mathcal{S}.\text{Setup}, \mathcal{S}.\text{DataUpdate}, \mathcal{S}.\text{Search}, \mathcal{S}.\text{KeyUpdate})$, 使得概率差 $|\Pr[\text{Real}_{\mathcal{A}_{\text{Srv}}}^{\Sigma}(\lambda)=1] - \Pr[\text{Ideal}_{\mathcal{A}_{\text{Srv}},\mathcal{S},\mathcal{L}_{\text{Srv}}}^{\Sigma}(\lambda) = 1]|$ 的值相对于 λ 是可忽略的, 则 KLVSE 方案 Σ 满足 $\mathcal{L}_{\text{Srv}}$-适应性安全性, 其中 $\text{Real}_{\mathcal{A}_{\text{Srv}}}^{\Sigma}(\lambda)$ 表示真实游戏, $\text{Ideal}_{\mathcal{A}_{\text{Srv}},\mathcal{S},\mathcal{L}_{\text{Srv}}}^{\Sigma}(\lambda)$ 表示理想游戏, 它们的形式化定义如下.

• $\text{Real}_{\mathcal{A}_{\text{Srv}}}^{\Sigma}(\lambda)$. 真实游戏实现了所有 KLVSE 协议. 运行协议 Setup 初始化 Σ 后, $\mathcal{A}_{\text{Srv}}$适应性地发起 DataUpdate, Search 和 KeyUpdate 查询, 并观察这些查询生成的真实记录. 最后, $\mathcal{A}_{\text{Srv}}$ 输出一个比特.

• $\mathrm{Ideal}^{\Sigma}_{\mathcal{A}_{\mathrm{Srv}},\mathcal{S},\mathcal{L}_{\mathrm{Srv}}}(\lambda)$. 理想游戏中, $\mathcal{A}_{\mathrm{Srv}}$ 与模拟器 $\mathcal{S}$ 交互并发布与真实游戏相同的查询. 模拟器 $\mathcal{S}$ 使用泄露函数 $\mathcal{L}_{\mathrm{Srv}} = (\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{Srch}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Srv}})$ 作为输入, 并分别通过运行 $\mathcal{S}$.Setup, $\mathcal{S}$.DataUpdate, $\mathcal{S}$.Search 和 $\mathcal{S}$.KeyUpdate 为 $\mathcal{A}_{\mathrm{Srv}}$ 模拟 KLVSE 的 Setup, DataUpdate, Search 和 KeyUpdate 协议的相应记录. 最后, $\mathcal{A}_{\mathrm{Srv}}$ 输出一个比特.

定义2 (针对 $\mathcal{A}_{\mathrm{Adv}}$ 的适应性安全性)　给定泄露函数 $\mathcal{L}_{\mathrm{Adv}} = (\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}})$, 如果对于任何足够大的安全参数 $\lambda \in \mathbb{N}$ 和概率多项式时间敌手 $\mathcal{A}_{\mathrm{Adv}}$, 存在一个可有效运行的模拟器 $\mathcal{S}' = (\mathcal{S}'.\mathrm{Setup}, \mathcal{S}'.\mathrm{DataUpdate}, \mathcal{S}'.\mathrm{Search}, \mathcal{S}'.\mathrm{KeyUpdate}, \mathcal{S}'.\mathrm{KeyLeak})$, 使得概率差 $|\Pr[\mathrm{Real}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}}}(\lambda) = 1] - \Pr[\mathrm{Ideal}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}},\mathcal{S}',\mathcal{L}_{\mathrm{Adv}}}(\lambda) = 1]|$ 的值相对于 λ 是可忽略的, 则 KLVSE 方案 Σ 满足 $\mathcal{L}_{\mathrm{Adv}}$-适应性安全性, 其中真实游戏 $\mathrm{Real}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}}}(\lambda)$ 和理想游戏 $\mathrm{Ideal}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}},\mathcal{S}',\mathcal{L}_{\mathrm{Adv}}}(\lambda)$ 形式化定义如下.

• $\mathrm{Real}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}}}(\lambda)$. 真实游戏完全实现了所有 KLVSE 协议. 敌手 $\mathcal{A}_{\mathrm{Adv}}$ 适应性地发起 DataUpdate, Search 和 KeyUpdate 查询, 并观察由这些查询生成的真实记录. 此外, $\mathcal{A}_{\mathrm{Adv}}$ 可以适应性地多次泄露密钥和私有状态. 最后, 它输出一个比特.

• $\mathrm{Ideal}^{\Sigma}_{\mathcal{A}_{\mathrm{Adv}},\mathcal{S}',\mathcal{L}_{\mathrm{Adv}}}(\lambda)$. 理想游戏中, 敌手 $\mathcal{A}_{\mathrm{Adv}}$ 发起与真实游戏相同的查询. $\mathcal{S}'$ 使用泄露函数 $\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Adv}}$, $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}$, $\mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}$ 和 $\mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}}$ 分别运行 $\mathcal{S}'$.Setup, $\mathcal{S}'$.DataUpdate, $\mathcal{S}'$.Search 和 $\mathcal{S}'$.KeyUpdate 为 $\mathcal{A}_{\mathrm{Adv}}$ 伪造相应的记录. 当发生真实的密钥泄露事件时, $\mathcal{S}'$ 使用泄露函数 $\mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}}$ 运行 $\mathcal{S}'$.KeyLeak 来模拟密钥泄露事件. 最后, $\mathcal{A}_{\mathrm{Adv}}$ 输出一个比特.

3.4.2　KLVSE 的密钥泄露后安全

本文沿用文献 [8] 中定义的信息泄露限制、时间戳定义, 以及允许泄露给云服务器 $\mathcal{A}_{\mathrm{Srv}}$ 和敌手 $\mathcal{A}_{\mathrm{Adv}}$ 的信息, 给出 KLVSE 的密钥泄露后安全定义.

令 $u = \mathrm{Time}(\mathbf{C})$ 表示检索密文 $\mathbf{C}$ 的相应时间戳 u, $\mathrm{CTRelation}(u^{\mathrm{KU}})$ 表示在时间戳 $u^{\mathrm{KU}} \in Q^{\mathrm{KU}}$ 时完成 KeyUpdate 查询后预更新密文和已更新密文之间的时间戳关系, $\mathrm{KUHist}(u)$ 记录每个不大于 u 的 KeyUpdate 时间戳 u^{KU} 及其相应的 $\mathrm{CTRelation}(u^{\mathrm{KU}})$, $\mathrm{CUHist}(u)$ 表示在最大时间戳 u^{KU} 时执行 KeyUpdate 查询生成的所有密文的原始数据, 若不存在这样的 u^{KU}, 则 $\mathrm{CUHist}(u) = \emptyset$. 令 $\mathrm{DUHist}(u)$ 表示自最大时间戳 u^{KU} 的 KeyUpdate 查询以来客户端发起的所有 DataUpdate 查询, 其中 u^{KU} 满足 $u^{\mathrm{KU}} \leqslant u$, 如果没有这样的 u^{KU}, 定义 $u^{\mathrm{KU}} = 0$. 令 $\mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}}$ 表示密钥泄露的泄露函数, $\mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}}$ 和 $\mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Srv}}$ 表示 KeyUpdate 的泄露函数, $\mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}$ 表示 Search 的泄露函数, $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}$ 和 $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}$ 表示 DataUpdate 的泄露函数.

根据对云服务器和敌手的适应性安全性以及上述泄露函数, 定义 KLVSE 的密钥泄露后安全如下.

定义3 (密钥泄露后安全)　如果一个 KLVSE 方案是 $\mathcal{L}_{\mathrm{Srv}}$-适应性安全和 $\mathcal{L}_{\mathrm{Adv}}$-适应性安全的, 同时满足以下在泄露函数 $\mathcal{L}_{\mathrm{Srv}}$ 和 $\mathcal{L}_{\mathrm{Adv}}$ 上的限制, 则该 KLVSE 方案是密钥泄露后安全的, 其中 NULL 表示空集, 即没有泄露.

(1) 对于 $\mathcal{L}_{\mathrm{Srv}} = (\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{Srch}}_{\mathrm{Srv}}, \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Srv}})$, 泄露函数 $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}$ 和 $\mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Srv}}$ 可以写成

$$\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}(\mathrm{op}, (w, \mathrm{id})) = \mathrm{NULL}, \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Srv}} = \mathcal{L}'_{\mathrm{Srv}}(\mathrm{KUHist}(U_{\mathrm{now}})), \text{其中 } \mathcal{L}'_{\mathrm{Srv}} \text{ 是无状态函数.} \tag{1}$$

(2) 对于 $\mathcal{L}_{\mathrm{Adv}} = (\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}}, \mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}})$, 泄露函数 $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}$, $\mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}$, $\mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}}$ 和 $\mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}}$ 可以写成

$$\begin{gathered}\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Adv}}(\mathrm{op}, (w, \mathrm{id})) = \mathrm{NULL},\ \mathcal{L}^{\mathrm{Srch}}_{\mathrm{Adv}}(w) = \mathrm{NULL},\ \mathcal{L}^{\mathrm{KeyUpdt}}_{\mathrm{Adv}} = \mathrm{NULL},\\ \mathcal{L}^{\mathrm{KeyLeak}}_{\mathrm{Adv}} = \mathcal{L}'_{\mathrm{Adv}}(\mathrm{CUHist}(U_{\mathrm{now}}), \mathrm{DUHist}(U_{\mathrm{now}})), \text{其中 } \mathcal{L}'_{\mathrm{Adv}} \text{ 是无状态函数.}\end{gathered} \tag{2}$$

值得注意的是, 密钥泄露后安全没有对 $\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Srv}}$, $\mathcal{L}^{\mathrm{Srch}}_{\mathrm{Srv}}$ 和 $\mathcal{L}^{\mathrm{Stp}}_{\mathrm{Adv}}$ 的泄露函数作出限制.

由密钥泄露后安全定义可知, 泄露函数 $\mathcal{L}^{\mathrm{DaUpdt}}_{\mathrm{Srv}}$ 同时满足 DSSE 上下文中的前向安全概念 [12], 即 DataUpdate 查询不泄露有关更新关键字的任何信息. 密钥泄露后安全定义了 DataUpdate 对于云服务

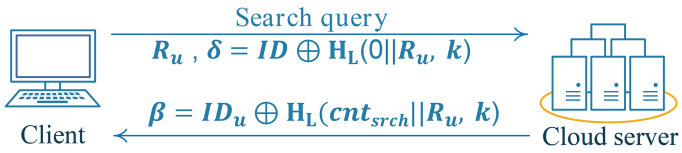

图 3　(网络版彩图) 双向验证机制.

Figure 3　(Color online) Bidirectional verification mechanism.

器 $\mathcal{A}_{\mathrm{Srv}}$ 和敌手 $\mathcal{A}_{\mathrm{Adv}}$ 的泄露函数都是 NULL, 显然满足前向安全性. 因此, 密钥泄露后安全定义包含了前向安全. 但密钥泄露后安全并没有明确限制 $\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Srch}}$ 的泄露函数. 如若需要可以设计具有不同 $\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Srch}}$ 的 KLVSE 实例, 以实现对云服务器的各种安全强度, 例如可搜索加密的后向安全. 因此, 密钥泄露后安全与后向安全是兼容的.

4　方案构造

本文以密钥泄露的验证机制和密钥可更新技术为支撑, 引入搜索计数器, 采用双向验证机制, 实现密钥泄露的验证, 从而构造 KLVSE.

4.1　双向验证机制

本文改进了 David 数字图书馆协议 [37], 将原协议中三方之间的验证简化为云服务器和客户端两方的验证, 并优化了参数设置以实现本方案密钥泄露验证的目标. 具体而言, 该协议中客户端和云服务器各自独立维护一个搜索计数器, 分别记录客户端发起的搜索查询次数和云服务器接收到的搜索查询次数, 每进行一次搜索, 就各自更新搜索计数器的值, 并双向验证客户端的合法性与云服务器端搜索计数器的异常动态, 以抵御和检测密钥泄露后敌手的非法搜索行为, 在云服务器和客户端间实现双重防护.

如图 3 所示, 该验证协议具体步骤如下.

(1) 客户端在初始化阶段预存一个随机数 R_u. 在每次发起搜索查询时, 使用自己的 ID、秘密值 k 和随机数 R_u, 计算一个待验证值 δ, 将其发送给云服务器, 然后更新 R_u.

(2) 云服务器查询自己的数据库, 若存在 ID 使得 δ 的计算公式成立, 则执行搜索查询, 并计算验证信息 β 发送给客户端; 否则判定对方非合法客户端 (一般认为是敌手), 不执行搜索查询.

(3) 若对方是非法客户端, 则无法获得搜索结果; 若是合法客户端, 则客户端根据本地的搜索计数器值计算 β', 若 $\beta' = \beta$, 则验证通过; 否则验证失败, 说明需要更新密钥.

在实际应用中, 客户端通常会频繁发起搜索查询, 因此可以认为客户端能够在敌手实施非法搜索后的极短时间内完成密钥泄露检测 —— 只要客户端发起一次新的搜索即可完成验证. 一旦发现搜索计数器同步异常, 系统将立即启动密钥更新, 从而将密钥泄露的影响窗口控制在最小范围.

4.2　密钥泄露可验证可搜索加密方案

结合改进后的双向验证协议, 本文提出 KLVSE 方案 (见算法 1). 该方案由 **Setup**, **DataUpdate**, **Search**, **Verify**, **KeyUpdate** 这 5 个部分组成.

• **Setup.** 该协议输入安全参数 λ、最大填充值 $a_{\max}$. 初始化相关的密码原语和私有状态 (第 1 和 2 行); 初始化 **EDB**, 将最大填充值 $a_{\max}$、随机数 R_u 和用户 ID 发送给云服务器 (第 3 行).

• **DataUpdate.** 该协议输入密钥 K、私有状态 **State** 加密 $(\mathrm{op}, (w, \mathrm{id}))$. 首先检索 **State**$[w]$ 状态的记录, 若记录为空, 则设置 tk_w 为 λ 长度的随机比特串, 并将 cnt_w 设置为 0 (第 1~4 行). 接着,

算法 1 KLVSE.

Setup $(\lambda, a_{\max})$

初始化可逆映射函数及其逆函数 $(\lambda, \mathbb{G}, q, \pi, \pi^{-1}) \leftarrow \mathbf{PGen}(\lambda, \lambda)$, 三个密码学哈希函数 $\mathbf{H_1} : \{0,1\}^* \to \mathbb{G}, \mathbf{H_2} : \{0,1\}^* \to \mathbb{G}$, $\mathbf{G} : \{0,1\}^* \to \mathbb{G}$ 与密钥哈希函数 $\mathbf{H}_{\mathrm{L}}(m,k) : \{0,1\}^* \to \{0,1\}^{\kappa}$, $\mathbf{H}_{\mathrm{R}}(m,k') : \{0,1\}^* \to \{0,1\}^{\kappa'}$, Diffie-Hellman 密钥交换协议参数 $\mathbb{G}'$, $g' \in \mathbb{G}'$;
初始化空映射 $\mathbf{State} \leftarrow \emptyset$, $\mathbf{EDB} \leftarrow \emptyset$, 密钥 $K_\Sigma = (K_1, K_2) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$, 搜索计数器 $\mathrm{srchcnt}_u \leftarrow 0$, 随机变量 $R_u \leftarrow \{0,1\}^\lambda$, $\mathrm{ID} \leftarrow \{0,1\}^\lambda$;
发送 $\mathbf{EDB}$, $a_{\max}, R_u, \mathrm{ID}$, 云服务器令 $\mathrm{srchcnt}_s \leftarrow 0$.

DataUpdate $(K_\Sigma, \mathbf{State}, \mathrm{op}, (w, \mathrm{id}); \mathbf{EDB})$

客户端:

从 $\mathbf{State}[w]$ 中检索记录 $(\mathrm{tk}_w, \mathrm{cnt}_w)$;
if $(\mathrm{tk}_w, \mathrm{cnt}_w) = (\mathrm{NULL}, \mathrm{NULL})$ **then**
  $\mathrm{tk}_w \leftarrow \{0,1\}^\lambda$, $\mathrm{cnt}_w \leftarrow 0$;
end if
$\mathrm{cnt}_w \leftarrow \mathrm{cnt}_w + 1$, $\mathrm{tk}'_w \leftarrow \{0,1\}^\lambda$;
计算密文 $L \leftarrow (\mathbf{H}_1(\mathrm{tk}'_w))^{K_1}$, $D \leftarrow (\pi(\mathrm{tk}_w) \cdot \mathbf{H}_2(\mathrm{tk}'_w))^{K_1}$, $C \leftarrow (\pi(\mathrm{op}||\mathrm{id}))^{K_2} \cdot (\mathbf{G}(\mathrm{tk}'_w))^{K_1}$;
更新 $\mathbf{State}[w] \leftarrow (\mathrm{tk}'_w, \mathrm{cnt}_w)$;
将密文 (L, D, C) 发送给云服务器;

云服务器:

$\mathbf{EDB}[L] \leftarrow (D, C)$.

Search $(K_\Sigma, \mathbf{State}, w, R_u, \mathrm{ID}; \mathbf{EDB})$

客户端:

从 $\mathbf{State}[w]$ 中检索记录 $(\mathrm{tk}_w, \mathrm{cnt}_w)$;
if $(\mathrm{tk}_w, \mathrm{cnt}_w) = (\mathrm{NULL}, \mathrm{NULL})$ **then** 中止;
用 Diffie-Hellman 密钥交换协议与服务器建立临时安全信道;
计算密文标签 $L \leftarrow (\mathbf{H}_1(\mathrm{tk}_w))^{K_1}$;
计算 $\mathrm{Msk}_D \leftarrow (\mathbf{H}_2(\mathrm{tk}_w))^{K_1}$ 和 $\mathrm{Msk}_C \leftarrow (\mathbf{G}(\mathrm{tk}_w))^{K_1}$;
计算验证信息 $\delta \leftarrow \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(0||R_u, k)$;
通过安全信道发送搜索令牌 $(K_1, L, \mathrm{Msk}_D, \mathrm{Msk}_C)$ 和验证信息 δ 至云服务器, 同时$\mathrm{srchcnt}_u \leftarrow \mathrm{srchcnt}_u + 1$;

云服务器:

$\delta' \leftarrow \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(0||R_u, k)$;
if $\delta' \neq \delta$ **then** 退出;
初始化空列表 $\mathcal{I} \leftarrow \emptyset$, 检索 $(D, C) \leftarrow \mathbf{EDB}[L]$;
while $(D, C) \neq$ (NULL, NULL) **do**
  解密搜索令牌 $\mathrm{tk} \leftarrow \pi^{-1}((\frac{D}{\mathrm{Msk}_D})^{K_1^{-1}})$;
  计算 $C' \leftarrow \frac{C}{\mathrm{Msk}_C}$ 并将 C' 插入 $\mathcal{I}$;
  计算 $L \leftarrow (\mathbf{H}_1(\mathrm{tk}))^{K_1}, \mathrm{Msk}_D \leftarrow (\mathbf{H}_2(\mathrm{tk}))^{K_1}$, $\mathrm{Msk}_C \leftarrow (\mathbf{G}(\mathrm{tk}))^{K_1}$;
  检索 $(D, C) \leftarrow \mathbf{EDB}[L]$;
end while
$\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{I}' = \{g_i \mid g_i \xleftarrow{\$} \mathbb{G}\}$;　　//其中 $\#\{\mathcal{I}'\} = a_{\max} - n$, n 表示找到的密文数量;
$\mathrm{srchcnt}_s \leftarrow \mathrm{srchcnt}_s + 1$, $\beta \leftarrow \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(\mathrm{srchcnt}_s||R_u, k)$, 更新 $R_u \leftarrow \mathbf{H}_{\mathrm{R}}(0||R_u, k')$;
通过安全信道将 $\mathcal{I}$ 和验证信息 β 返回给客户端;

客户端:

初始化空列表 $\mathcal{R}$;
for $i = 1$ to cnt_w **do**
  通过计算 $\mathrm{op}_i||\mathrm{id}_i \leftarrow \pi^{-1}(C'^{K_2^{-1}}_i)$ 解密 $C'_i \in \mathcal{I}$;
  if $\mathrm{op}_i = \mathrm{add}$ **then** 将 id_i 插入 $\mathcal{R}$ **else** 从 $\mathcal{R}$ 中删除 id_i;
end for
return $\mathcal{R}$.

Verify$(R_u, \mathrm{ID}, \beta, \mathrm{srchcnt}_u)$

客户端:

$\beta' \leftarrow \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(\mathrm{srchcnt}_u||R_u, k)$, $R_u \leftarrow \mathbf{H}_{\mathrm{R}}(0||R_u, k')$;
if $\beta' \neq \beta$ **then** 不通过 **else** 通过.

KeyUpdate $(K_\Sigma, \mathbf{State}; \mathbf{EDB})$

客户端:

用 Diffie-Hellman 密钥交换协议与服务器建立临时安全信道;
从 $\mathbb{Z}_q^*$ 中随机抽取密钥更新令牌 Δ, 更新 $K_1 \leftarrow K_1 \cdot \Delta$, $K_2 \leftarrow K_2 \cdot \Delta$, 通过安全信道发送 Δ 给云服务器;

云服务器:

for 所有 (L, D, C) **such that** $(D, C) \leftarrow \mathbf{EDB}[L]$ **do**
  更新密文 $L' \leftarrow L^\Delta$, $D' \leftarrow D^\Delta$, $C' \leftarrow C^\Delta$;
  $\mathbf{EDB}[L'] \leftarrow (D', C')$, 从 $\mathbf{EDB}$ 中移除旧密文 (L, D, C);
end for

关键字 w 对应的 cnt_w 加 1, 生成新的随机比特串 tk'_w, 计算密文并发送给云服务器 (第 5∼8 行). 最后云服务器将密文 $\mathbf{C}$ 存储在 $\mathbf{EDB}$ 中 (第 9 行).

• **Search.** 该协议输入密钥 K、私有状态 $\mathbf{State}$、关键字 w、随机数 R_u、用户 ID. 首先检索 $\mathbf{State}[w]$ 状态的记录, 若为空则中止 (第 1 和 2 行). 若不为空, 则建立安全信道, 计算搜索令牌和验证信息 $\delta = \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(0||R_u, k)$并发送给云服务器, 用于云服务器实现客户端身份验证和搜索, 此时客户端本地的搜索计数器 +1, 表明客户端发起了一次新的搜索查询 (第 3∼7 行).

云服务器接收搜索查询后, 计算 $\delta' = \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(0||R_u, k)$, 若 $\delta' \neq \delta$, 则表明对方非合法客户端, 不执行搜索查询 (第 8 和 9 行); 否则, 执行搜索查询, 根据客户端提供的搜索令牌倒序搜索所有关于 w 的密文 (第 10∼16 行). 之后为了隐藏真实密文数量, 填充密文数至最大填充值 (第 17 行). 云服务器完成搜索后, 搜索计数器 +1, 并计算验证信息 $\beta = \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(\mathrm{srchcnt}_s||R_u, k)$ 以便客户端后续的验证,

然后将搜索结果和 β 发送给客户端 (第 18 和 19 行). 最后客户端使用可逆映射函数的逆函数解密所有密文得到搜索结果 (第 20~25 行).

• **Verify.** 该算法由客户端执行, 首先计算 $\beta' = \mathrm{ID} \bigoplus \mathbf{H}_{\mathrm{L}}(\mathrm{srchcnt}_u || R_u, k)$, 更新 R_u 为新值 (第 1 行). 若 $\beta' = \beta$, 则验证通过, 说明云服务器端的搜索计数器与本地保持一致, 表明云服务器未被敌手搜索过, 即密钥未发生泄露; 否则验证失败, 说明需要更新密钥 (第 2 行).

• **KeyUpdate.** 该协议输入密钥 K、私有状态 **State**. 客户端建立安全信道, 生成密钥更新令牌 Δ, 更新本地密钥, 并提交密钥更新查询至云服务器 (第 1 和 2 行). 云服务器根据密钥更新令牌将所有旧密文更新为相应的新密文 (第 3~6 行).

上述方案利用双向验证机制嵌入并改造可更新的 DSSE 密码原语, 实现客户端和云服务器之间的相互验证, 确保客户端是合法的, 并且通过搜索计数器的独立更新可以验证密钥泄露事件是否发生. 针对验证不通过即密钥发生泄露的情况, 调用密钥更新协议, 将旧密文更新为新密钥设置下的新密文, 达到保障密钥泄露后安全的目标.

4.3　安全性分析

正确性. KLVSE 的正确性源于哈希函数 $\mathbf{H}_1$, $\mathbf{H}_2$, $\mathbf{G}$ 和密钥哈希函数 $\mathbf{H}_{\mathrm{L}}$, $\mathbf{H}_{\mathrm{R}}$的抗碰撞性、群 $\mathbb{G}$ 的代数特性, 以及可逆映射函数 π 的正确性. 具体来看, 当处理给定条目 $(\mathrm{op}, (w, \mathrm{id}))$ 的 DataUpdate 操作时, 密文 (L, D, C) 中加密操作类型 op、文件标识符 id 和上一条密文的搜索令牌 tk_w 的正确性由 $\mathbf{G}$ 和 $\mathbf{H}_2$ 的抗碰撞特性以及可逆映射函数 π 的正确性保证. 生成验证信息 δ 和随机数 R_u 时, 正确性由密钥哈希函数 $\mathbf{H}_{\mathrm{L}}$ 和 $\mathbf{H}_{\mathrm{R}}$ 的正确性保证.

当使用给定关键字 w 执行 Search 时, 可以通过搜索令牌中的标签 $L = (\mathbf{H}_1(\mathrm{tk}_w))^{K_1}$ 正确定位最新发布的密文 (L, D, C). 给定搜索令牌中包含的 K_1, Msk_D 和 Msk_C, 云服务器可以从 D 中解密先前密文中的搜索令牌 tk, 并部分解密 C 以获得 $(\pi(\mathrm{op}||\mathrm{id}))^{K_2}$. 然后, 云服务器利用相应的搜索令牌 tk, 借助哈希函数 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 定位并解密先前加密的密文. 通过这种方式, 云服务器可以精确地筛选出所有匹配的密文, 并将执行了部分解密的密文返回给客户端. 云服务器使用密钥哈希函数 $\mathbf{H}_{\mathrm{L}}$ 生成验证信息 β, 并利用 $\mathbf{H}_{\mathrm{R}}$ 更新随机数 R_u 以保证后续验证信息的正确性.

在接收到云服务器返回的搜索结果和验证信息之后, 客户端使用加密密钥 K_2 解密返回的结果, 同时使用密钥哈希函数 $\mathbf{H}_{\mathrm{L}}$ 生成本地的验证信息 β' 与云服务器端的验证信息 β 比较, 若验证通过则表明密钥未泄漏, 不执行密钥更新; 否则发出验证不通过的警告, 并立即调用密钥更新. 密钥哈希函数的抗碰撞性保障了验证结果的正确性.

在密钥更新过程中, 假设存在一个密文 $(L, D, C) = (L_0^{K_1}, D_0^{K_1}, C_0^{K_2} \cdot X_0^{K_1})$, 其中 K_1 是搜索密钥, K_2 是加密密钥. 执行带有密钥更新令牌 Δ 的协议 KeyUpdate 后, 密钥更新后的密文是 $(L_0^{K_1 \cdot \Delta}, D_0^{K_1 \cdot \Delta}, C_0^{K_2 \cdot \Delta} \cdot X_0^{K_1 \cdot \Delta})$, 两个新密钥分别是 $K_1 \cdot \Delta$ 和 $K_2 \cdot \Delta$. 令 $K_1' = K_1 \cdot \Delta$ 且 $K_2' = K_2 \cdot \Delta$, 显然新密钥 K_1' 和 K_2' 能够搜索和解密密钥更新后的密文.

泄露分析. 在云服务器和敌手不合谋的模型下, 根据云服务器 $\mathcal{A}_{\mathrm{Srv}}$ 和敌手 $\mathcal{A}_{\mathrm{Adv}}$ 的视角分析 KLVSE 的密钥泄露后安全, 分别定义 KLVSE 对于云服务器 $\mathcal{A}_{\mathrm{Srv}}$ 和敌手 $\mathcal{A}_{\mathrm{Adv}}$ 的泄露函数.

对于 $\mathcal{A}_{\mathrm{Srv}}$ 来说, Setup 协议只泄露了安全参数 λ 和最大填充值 $a_{\max}$; DataUpdate 协议没有泄露任何信息, 因此对于云服务器 $\mathcal{A}_{\mathrm{Srv}}$ 来说泄露函数 $\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Stp}}$ 和 $\mathcal{L}_{\mathrm{Srv}}^{\mathrm{DaUpdt}}$ 表示为:

$$\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Stp}}(\lambda, a_{\max}) = (\lambda, a_{\max}), \tag{3}$$

$$\mathcal{L}_{\mathrm{Srv}}^{\mathrm{DaUpdt}}(\mathrm{op}, (w, \mathrm{id})) = \mathrm{NULL}. \tag{4}$$

当发布关键字 w 的搜索查询时, Search 协议泄露了搜索模式 $\mathrm{sp}(w)$、与 w 匹配的文件 id 及其插入时间戳 (即 $\mathrm{TimeDB}(w)$)、w 的 DataUpdate 时间戳 (即 $\mathrm{DUTime}(w)$) 和 KeyUpdate 历史记录

KUHist(U_{now}); KeyUpdate 协议除了 KUHist(U_{now}) 外没有泄露任何信息. 泄漏信息 sp(w)、TimeDB(w) 和 DUTime(w) 定义为:

$$\text{TimeDB}(w) = \{(u, \text{id}) \mid (u, \text{add}, (w, \text{id})) \in Q^{\text{DU}} \text{ 且} \forall u', (u', \text{del}, (w, \text{id})) \notin Q^{\text{DU}}\}, \tag{5}$$

$$\text{sp}(w) = \{u \mid (u, w) \in Q^{\text{Srch}}\}, \tag{6}$$

$$\text{DUTime}(w) = \{u \mid (u, \text{op}, (w, \text{id})) \in Q^{\text{DU}}\}. \tag{7}$$

根据上述分析, 对于云服务器 $\mathcal{A}_{\text{Srv}}$ 在搜索和密钥更新时的泄露函数 $\mathcal{L}_{\text{Srv}}^{\text{Srch}}$ 和 $\mathcal{L}_{\text{Srv}}^{\text{KeyUpdt}}$ 表示为:

$$\mathcal{L}_{\text{Srv}}^{\text{Srch}}(w) = \mathcal{L}'_{\text{Srv}}(\text{sp}(w), \text{TimeDB}(w), \text{DUTime}(w), \text{KUHist}(U_{\text{now}})), \text{其中 } \mathcal{L}'_{\text{Srv}} \text{ 是无状态函数}, \tag{8}$$

$$\mathcal{L}_{\text{Srv}}^{\text{KeyUpdt}} = \mathcal{L}''_{\text{Srv}}(\text{KUHist}(U_{\text{now}})), \text{其中 } \mathcal{L}''_{\text{Srv}} \text{ 是无状态函数}. \tag{9}$$

对于 $\mathcal{A}_{\text{Adv}}$ 来说, 协议 Setup 泄漏安全参数 λ 和最大填充值 $a_{\max}$, 协议 DataUpdate, Search 和 KeyUpdate 没有泄漏, 因此对于敌手 $\mathcal{A}_{\text{Adv}}$ 的泄露函数 $\mathcal{L}_{\text{Adv}}^{\text{Stp}}$, $\mathcal{L}_{\text{Adv}}^{\text{DaUpdt}}$, $\mathcal{L}_{\text{Adv}}^{\text{KeyUpdt}}$ 和 $\mathcal{L}_{\text{Adv}}^{\text{Srch}}$ 表示为:

$$\mathcal{L}_{\text{Adv}}^{\text{Stp}}(\lambda, a_{\max}) = (\lambda, a_{\max}), \tag{10}$$

$$\mathcal{L}_{\text{Adv}}^{\text{DaUpdt}}(\text{op}, (w, \text{id})) = \text{NULL}, \tag{11}$$

$$\mathcal{L}_{\text{Adv}}^{\text{Srch}}(w) = \text{NULL}, \tag{12}$$

$$\mathcal{L}_{\text{Adv}}^{\text{KeyUpdt}} = \text{NULL}. \tag{13}$$

当密钥被泄露时, 若敌手 $\mathcal{A}_{\text{Adv}}$ 采用离线攻击, 则可以学习到自上次执行 DataUpdate 协议以来生成的密文的明文, 以及自上次执行 KeyUpdate 以来发布的 DataUpdate 查询. 使用 $\mathcal{L}_{\text{Adv}}^{\text{KeyLeak}}$ 来模拟发生密钥泄露事件时暴露给敌手的信息, 则 $\mathcal{L}_{\text{Adv}}^{\text{KeyLeak}}$ 表示为:

$$\mathcal{L}_{\text{Adv}}^{\text{KeyLeak}} = \mathcal{L}'_{\text{Adv}}(\text{CUHist}(U_{\text{now}}), \text{DUHist}(U_{\text{now}})), \text{其中 } \mathcal{L}'_{\text{Adv}} \text{ 是无状态函数}. \tag{14}$$

根据上述泄露函数的定义, 存在以下定理 1, 且 KLVSE 对云服务器的泄露函数满足前向安全和后向安全, 此时 Search 协议对于云服务器泄露了当前与查询关键字 w 匹配的文件 id、它们上传的时间以及所有关于 w 的 DataUpdate 发布的时间 [13].

定理1　假设哈希函数 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 是随机预言机, 并且 DDH 假设在 $\mathbb{G}$ 中成立, 则 KLVSE 是一个密钥泄露后安全的密钥可更新 DSSE 方案, 即

(1) KLVSE 是 $\mathcal{L}_{\text{Srv}}$-适应性安全的, 并且泄露函数 $\mathcal{L}_{\text{Srv}} = (\mathcal{L}_{\text{Srv}}^{\text{Stp}}, \mathcal{L}_{\text{Srv}}^{\text{DaUpdt}}, \mathcal{L}_{\text{Srv}}^{\text{Srch}}, \mathcal{L}_{\text{Srv}}^{\text{KeyUpdt}})$ 分别满足式 (3), (4) 和式 (8), (9);

(2) KLVSE 是 $\mathcal{L}_{\text{Adv}}$-适应性安全的, 且泄露函数 $\mathcal{L}_{\text{Adv}} = (\mathcal{L}_{\text{Adv}}^{\text{Stp}}, \mathcal{L}_{\text{Adv}}^{\text{DaUpdt}}, \mathcal{L}_{\text{Adv}}^{\text{Srch}}, \mathcal{L}_{\text{Adv}}^{\text{KeyUpdt}}, \mathcal{L}_{\text{Adv}}^{\text{KeyLeak}})$ 分别满足式 (10)∼(14).

证明中使用了判断性迪菲 – 赫尔曼 (Diffie-Hellman) 假设, 该假设定义如下.

定义4 (Decisional Diffie-Hellman (DDH) 假设)　设 $\mathbb{G}$ 为一个素数阶 q 的乘法循环群, g 是 $\mathbb{G}$ 的一个生成元, 其中 q 的长度为 λ 位. 如果对任意多项式时间敌手 $\mathcal{A}$, 其区分元组 (g, g^a, g^b, g^{ab}) 和 (g, g^a, g^b, g^c) 的概率相对于 λ 是可忽略的, 其中 $(a, b, c) \xleftarrow{\$} \mathbb{Z}_q^* \times \mathbb{Z}_q^* \times \mathbb{Z}_q^*$, 则说 DDH 假设在 $\mathbb{G}$ 中成立.

安全性证明. 为了证明敌手和云服务器不合谋模型下的密钥泄露后安全, 下面证明定理 1 成立, 即在相应的泄露函数下 KLVSE 满足 $\mathcal{L}_{\text{Srv}}$-适应性安全性和 $\mathcal{L}_{\text{Adv}}$-适应性安全性. 前文安全模型中使用真实游戏和理想游戏的不可区分性定义适应性安全性, 其中理想游戏使用泄露函数作为模拟器的输入, 因此为了证明方案的 $\mathcal{L}_{\text{Srv}}$-适应性安全性和 $\mathcal{L}_{\text{Adv}}$-适应性安全性, 仅需证明以泄露函数作为输入所构建的模拟器与真实方案下的对应协议不可区分.

算法 2 模拟器 $\mathcal{S}$ 的构建.

$\mathcal{S}.\mathrm{Setup}(\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Stp}}(\lambda, a_{\max}) = (\lambda, a_{\max}))$

1: 初始化可逆映射函数 $(\lambda, \mathbb{G}, q, \pi, \pi^{-1}) \leftarrow \mathbf{PGen}(\lambda,\ \lambda)$, Diffie-Hellman 密钥交换协议参数 $\mathbb{G}'$, $g' \in \mathbb{G}'$;
2: 初始化空映射 **SHist** 和 **EDB**, 空列表 **CList**, 时间戳 $u \leftarrow 0$, 搜索密钥 $K_1 \xleftarrow{\$} \mathbb{Z}_q^*$;
3: 将 **EDB** 发送给云服务器.

$\mathcal{S}.\mathrm{DataUpdate}(\mathcal{L}_{\mathrm{Srv}}^{\mathrm{DaUpdt}}(\mathrm{op}, (w, \mathrm{id})) = \mathrm{NULL})$

1: 累加时间戳 $u \leftarrow u + 1$;
2: 随机选择 3 个元素 $(L_0, D_0, C_0) \xleftarrow{\$} \mathbb{G} \times \mathbb{G} \times \mathbb{G}$;
3: 将记录 (u, L_0, D_0, C_0) 插入到 **CList**;
4: 将模拟的密文 $(L_0^{K_1}, D_0^{K_1}, C_0^{K_1})$ 发送给云服务器.

$\mathcal{S}.\mathrm{Search}(\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Srch}}(w) = (\mathrm{sp}(w), \mathrm{TimeDB}(w), \mathrm{DUTime}(w), \mathrm{KUHist}(U_{\mathrm{now}})))$

1: 用 Diffie-Hellman 密钥交换协议与服务器建立临时安全信道;
2: 累加时间戳 $u \leftarrow u + 1$, $\mathrm{DUTime}(w)$ 为空则中止;
3: 从 $\mathrm{sp}(w)$ 中检索最小时间戳 $u_{\min}^{\mathrm{srch}}$, 从 **SHist** 中检索记录 $(u_s, H_L, H_D, H_C, H_\delta, H_\beta) \leftarrow \mathbf{SHist}[u_{\min}^{\mathrm{srch}}]$;
4: **if** 记录 $(u_s, H_L, H_D, H_C, H_\delta, H_\beta)$ 不存在 **then** $(u_s, H_L, H_D, H_C, H_\delta, H_\beta) \xleftarrow{\$} \{0\} \times \mathbb{G} \times \mathbb{G} \times \mathbb{G} \times \{0,1\}^\kappa \times \{0,1\}^\kappa$;
5: **for** 所有 $u' \in \mathrm{DUTime}(w)$ 按升序排列, 其中 $u' > u_s$ **do**
6: 从 **CList** 中检索 (u', L_0, D_0, C_0);
7: $\mathrm{tk} \xleftarrow{\$} \{0,1\}^\lambda$, $m \xleftarrow{\$} \{0,1\}^\lambda$;
8: 编程随机预言机 $\mathbf{H}_1(\mathrm{tk}) \leftarrow H_L$, $\mathbf{H}_2(\mathrm{tk}) \leftarrow H_D$, $\mathbf{G}(\mathrm{tk}) \leftarrow H_C$, $\mathbf{H}_{\mathrm{L}}(m) \leftarrow H_\delta$;
9: 设置 $(H_L, H_D) \leftarrow (L_0, \frac{D_0}{\pi(\mathrm{tk})})$, $H_C \xleftarrow{\$} \mathbb{G}$;
10: **end for**
11: **for** 所有 $u' \in \mathrm{DUTime}(w)$ 按升序排列, 其中 $u' > u_s$ **do**
12: $m' \xleftarrow{\$} \{0,1\}^\lambda$, 编程随机预言机 $\mathbf{H}_{\mathrm{L}}(m') \leftarrow H_\beta$;
13: **end for**
14: 从 $\mathrm{DUTime}(w)$ 中检索最大时间戳 $u_{\max}^{\mathrm{DU}}$;
15: 更新 $\mathbf{SHist}[u_{\min}^{\mathrm{srch}}] \leftarrow (u_{\max}^{\mathrm{DU}}, H_L, H_D, H_C, H_\delta, H_\beta)$, 通过安全信道发送模拟的令牌 $(K_1, H_L^{K_1}, H_D^{K_1}, H_C^{K_1})$和 H_δ给云服务器;
16: 接收服务器结果和 H_β后, 输出从 $\mathrm{TimeDB}(w)$ 中提取的所有文件 id.

$\mathcal{S}.\mathrm{KeyUpdate}(\mathcal{L}_{\mathrm{Srv}}^{\mathrm{KeyUpdt}} = (\mathrm{KUHist}(U_{\mathrm{now}})))$

1: 用 Diffie-Hellman 密钥交换协议与服务器建立临时安全信道;
2: 累加时间戳 $u \leftarrow u + u$, 从 $\mathbb{Z}_q^*$ 中随机抽取密钥更新令牌 Δ;
3: 更新 $K_1 \leftarrow K_1 \cdot \Delta$, 通过安全信道发送 Δ 给云服务器.

(1) $\mathcal{L}_{\mathrm{Srv}}$-适应性安全性的证明.

证明　构建模拟器 $\mathcal{S}$ 证明 KLVSE 是 $\mathcal{L}_{\mathrm{Srv}}$-适应性安全的. 在 $\mathcal{S}$ 中, 哈希函数 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 建模为随机预言机. 泄露函数 $\mathcal{L}_{\mathrm{Srv}} = (\mathcal{L}_{\mathrm{Srv}}^{\mathrm{Stp}}, \mathcal{L}_{\mathrm{Srv}}^{\mathrm{DaUpdt}}, \mathcal{L}_{\mathrm{Srv}}^{\mathrm{Srch}}, \mathcal{L}_{\mathrm{Srv}}^{\mathrm{KeyUpdt}})$ 定义如定理 1 的 (1) 所示.

模拟器 $\mathcal{S}$ 的构建如算法 2 所示. 在模拟协议 Setup 中, $\mathcal{S}$ 初始化映射 **SHist**、列表 **CList** 和搜索密钥 K_1. 映射 **SHist** 维护搜索查询的状态以保证搜索过程的一致性. 列表 **CList** 记录了由 $\mathcal{S}$ 生成的密文. $\mathcal{S}$ 使用 K_1 在协议 $\mathcal{S}$.Search 中生成搜索令牌. 显然, 模拟协议 Setup 与 KLVSE. Setup 是一致的.

协议 $\mathcal{S}$.DataUpdate 通过从 $\mathbb{G}$ 随机采样 3 个元素并计算它们的 K_1 次幂来模拟密文. 由于 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 的单向性和抗碰撞性以及搜索令牌的随机性, 在客户端发出与此密文相关的 Search 查询之前, 由真实协议 DataUpdate 生成的密文 (L, D, C) 与随机数是不可区分的. 因此, 这个协议与真实的 KLVSE.DataUpdate 是不可区分的.

协议 $\mathcal{S}$.Search 模拟 KLVSE.Search. $\mathcal{S}$ 使用搜索模式 $\mathrm{sp}(w)$ 的最小时间戳 $u_{\min}^{\mathrm{srch}}$ 来指示映射 **SHist** 中最新模拟 w 的搜索令牌和验证信息 (第 3 和 4 行). 映射 **SHist** 还记录了时间戳 u_s 用于最新模拟的密文, 其对应的搜索令牌已被编程到随机预言机 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 中, 而验证信息编程到随机预言机 $\mathbf{H}_{\mathrm{L}}$ 中. 任何比 u_s 旧的时间戳上生成的 w 的模拟密文将不会在后续步骤中被重新编程到预言机中. 第 5∼13 行, $\mathcal{S}$ 为每个模拟密文随机生成一个搜索令牌, 并将搜索令牌编程到随机预言机 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 中, 将验证信息编程到随机预言机 $\mathbf{H}_{\mathrm{L}}$ 中. 编程过程遵循以下规则: 对于同一关键字的任何两个连续密文, 后一个密文加密前一个密文的搜索令牌. 根据这一规则, 云服务器可以使用最新模拟密文的搜索令牌正确地揭示同一关键字的整个隐藏链. 上述过程模拟了云服务器检索获得相关密文的过程, 保证了云服务器无法区分模拟的搜索过程与真实的搜索过程. 此外, 模拟搜索令牌中的 $H_L^{K_1}$, $H_D^{K_1}$和$H_C^{K_1}$也与真实搜索令牌中的 L, Msk_D 和 Msk_C 不可区分. 由于 L, Msk_D 和 Msk_C 的基数是通过随机选择

算法 3 模拟器 $\mathcal{S}'$ 的构建.

$\mathcal{S}'.\mathrm{Setup}(\mathcal{L}_{\mathrm{Adv}}^{\mathrm{Stp}}(\lambda, a_{\max}) = (\lambda, a_{\max}))$

初始化可逆映射函数 $(1 + l, \mathbb{G}, q, \pi, \pi^{-1}) \leftarrow \mathbf{PGen}(\lambda, l + 1)$, Diffie-Hellman 密钥交换协议参数 $\mathbb{G}', g' \in \mathbb{G}'$;
初始化空映射 **State**, **TdMap**, **OriCip** 和 **EDB**, 空列表 **CList**, 时间戳 $u \leftarrow 0$, 搜索密钥 $K_1 \xleftarrow{\$} \mathbb{Z}_q^*$ 和加密密钥 $K_2 \xleftarrow{\$} \mathbb{Z}_q^*$, 设置模拟密钥 $K_\Sigma \leftarrow (K_1, K_2)$;
将模拟数据库 **EDB** 发送给云服务器.

$\mathcal{S}'.\mathrm{DataUpdate}(\mathcal{L}_{\mathrm{Adv}}^{\mathrm{DaUpdt}}(\mathrm{op}, (w, \mathrm{id})) = \mathrm{NULL})$

模拟客户端:

累加时间戳 $u \leftarrow u + 1$ 并插入 **OriCip**$[u] \leftarrow u$;
随机选择 3 个元素 $(L_0, D_0, C_0) \xleftarrow{\$} \mathbb{G} \times \mathbb{G} \times \mathbb{G}$;
计算模拟密文$L \leftarrow L_0^{K_1}$, $D \leftarrow D_0^{K_1}$, $C \leftarrow C_0^{K_1}$;
将记录 $(0, u, L_0, D_0, C_0)$ 和 $(1, u, L, D, C)$ 插入到 **CList**;
将模拟密文 (L, D, C) 发送给模拟服务器;

模拟服务器:

将模拟密文存储到模拟数据库 **EDB**$[L] \leftarrow (D, C)$.

$\mathcal{S}'.\mathrm{Search}(\mathcal{L}_{\mathrm{Adv}}^{\mathrm{Srch}}(w) = \mathrm{NULL})$

模拟客户端:

用 Diffie-Hellman 密钥交换协议与模拟服务器建立安全信道;
累加时间戳 $u \leftarrow u + 1$;
$(e_1, e_2, e_3) \xleftarrow{\$} \mathbb{G} \times \mathbb{G} \times \mathbb{G}$, $r_\delta \xleftarrow{\$} \{0,1\}^\kappa$;
经安全信道发送 (K_1, e_1, e_2, e_3)和 r_δ给云服务器;

模拟服务器:

返回 $a_{\max}$ 个 $r_{\mathbb{G}} \xleftarrow{\$} \mathbb{G}$ 和随机数 $r_\beta \xleftarrow{\$} \{0,1\}^\kappa$ 给客户端.

$\mathcal{S}'.\mathrm{KeyUpdate}(\mathcal{L}_{\mathrm{Adv}}^{\mathrm{KeyUpdt}} = \mathrm{NULL})$

模拟客户端:

用 Diffie-Hellman 密钥交换协议与服务器建立临时安全通道;
累加时间戳 $u \leftarrow u + 1$, 从 $\mathbb{Z}_q^*$ 中随机抽取密钥更新令牌 Δ;
更新 $K_1 \leftarrow K_1 \cdot \Delta$, $K_2 \leftarrow K_2 \cdot \Delta$;
经安全信道发送 Δ 给云服务器;

模拟服务器:

for 所有密文 (L, D, C) 使得 $(D, C) \leftarrow$ **EDB**$[L]$ **do**
  $u \leftarrow u+1$, 更新密文 $L' \leftarrow L^\Delta$, $D' \leftarrow D^\Delta$, $C' \leftarrow C^\Delta$;
  插入密文 **EDB**$[L'] \leftarrow (D', C')$;
  将记录 $(1, u, L', D', C')$ 插入到 **CList**;
  从 **CList** 中检索 $(1, u', L, D, C)$;
  检索 $u_0 \leftarrow$ **OriCip**$[u']$ 并插入 **OriCip**$[u] \leftarrow u_0$;
  移除 **EDB**$[L]$;
end for

$\mathbf{S}'.\mathrm{KeyLeak}(\mathcal{L}_{\mathrm{Adv}}^{\mathrm{KeyLeak}} = (\mathrm{CUHist}(U_{\mathrm{now}}), \mathrm{DUHist}(U_{\mathrm{now}})))$

累加时间戳 $u \leftarrow u + 1$;
初始化 **LeakedData** $\leftarrow \mathrm{CUHist}(U_{\mathrm{now}}) \cup \mathrm{DUHist}(U_{\mathrm{now}})$;
for 每个在 **LeakedData** 中的关键字 w **do**
  初始化空列表 $\mathbf{DB}_w$;
  for 每个记录 $(u', \mathrm{op}, (w, \mathrm{id})) \in$ **LeakedData do**
    检索 $u_0' \leftarrow$ **OriCip**$[u']$ 并将 $(u_0', \mathrm{op}, \mathrm{id})$ 插入到 $\mathbf{DB}_w$;
  end for
  for 所有三元组 $(u_0', \mathrm{op}, \mathrm{id}) \in \mathbf{DB}_w$ 按 u_0' 升序排列 **do**
    检索搜索令牌 $\mathrm{tk}_w' \leftarrow$ **TdMap**$[u_0']$;
    if $\mathrm{tk}_w' \neq$ NULL **then continue else** $\mathrm{tk}_w' \xleftarrow{\$} \{0,1\}^\lambda$;
    插入 **TdMap**$[u_0'] \leftarrow \mathrm{tk}_w'$;
    从 **CList** 中检索记录 $(0, u_0', L_0, D_0, C_0)$;
    从 **State**$[w]$ 中检索记录 $(\mathrm{tk}_w, \mathrm{cnt}_w)$;
    if $(\mathrm{tk}_w, \mathrm{cnt}_w) = (\mathrm{NULL}, \mathrm{NULL})$ **then**
      $\mathrm{tk}_w \leftarrow \{0,1\}^\lambda$, $\mathrm{cnt}_w \leftarrow 0$;
      使用 $\mathbb{G}$ 中随机选择的元素编程随机预言机 $\mathbf{H}_1(\mathrm{tk}_w)$, $\mathbf{H}_2(\mathrm{tk}_w)$ 和 $\mathbf{G}(\mathrm{tk}_w)$;
    end if
    编程随机预言机 $\mathbf{H}_1(\mathrm{tk}_w') \leftarrow L_0$, $\mathbf{H}_2(\mathrm{tk}_w') \leftarrow \frac{D_0}{\pi(\mathrm{tk}_w)}$, $\mathbf{G}(\mathrm{tk}_w') \leftarrow \frac{C_0}{\pi(\mathrm{op}||\mathrm{id})\frac{K_2}{K_1}}$;
    更新 **State**$[w] \leftarrow (\mathrm{tk}_w', \mathrm{cnt}_w + 1)$;
  end for
end for
将 $(K_1, K_2,$ **State**$)$ 暴露给敌手 $\mathcal{A}_{\mathrm{Adv}}$.

的数字进行哈希处理的 (即与随机数不可区分), 因此模拟协议 Search 与真实的 Search 协议不可区分.

在模拟协议 KeyUpdate 中, 很明显云服务器 $\mathcal{A}_{\mathrm{Srv}}$ 在 $\mathcal{S}$.KeyUpdate 中观察到的内容与 $\mathcal{A}_{\mathrm{Srv}}$ 在 KLVSE.KeyUpdate 中可以观察到的内容是相同的. 因此, $\mathcal{S}$ 模拟了一个不可区分的协议 KeyUpdate.

综上, 构建的 $\mathcal{S}$ 模拟了一个与真实游戏在给定泄露函数 $\mathcal{L}_{\mathrm{Srv}}$ 视角下不可区分的理想游戏. 因此, KLVSE 是 $\mathcal{L}_{\mathrm{Srv}}$-适应性安全的.

(2) $\mathcal{L}_{\mathrm{Adv}}$-适应性安全性的证明.

证明　构建模拟器 $\mathcal{S}'$ 证明 KLVSE 是 $\mathcal{L}_{\mathrm{Adv}}$-适应性安全的, 其中泄露函数 $\mathcal{L}_{\mathrm{Adv}} = (\mathcal{L}_{\mathrm{Adv}}^{\mathrm{Stp}}, \mathcal{L}_{\mathrm{Adv}}^{\mathrm{DaUpdt}}, \mathcal{L}_{\mathrm{Adv}}^{\mathrm{Srch}}, \mathcal{L}_{\mathrm{Adv}}^{\mathrm{KeyUpdt}}, \mathcal{L}_{\mathrm{Adv}}^{\mathrm{KeyLeak}})$ 定义如定理 1 的 (2) 所示.

算法 3 描述了 $\mathcal{S}'$ 的构建过程. 在构建模拟器 $\mathcal{S}'$ 时使用了 DDH 假设, 同时哈希函数 $\mathbf{H}_1$, $\mathbf{H}_2$ 和 $\mathbf{G}$ 均建模为随机预言机.

模拟协议 Setup 初始化映射 **State** 和密钥 K_1, K_2 分别生成私有状态 **State** 和模拟密钥 K_Σ. $\mathcal{S}'$ 使用 K_Σ 和 **State** 来模拟密钥泄露事件. 列表 **CList** 记录密文及其对应的生成时间戳. 映射 **TdMap** 存储用于生成密文的搜索令牌. 映射 **OriCip** 将密文的时间戳映射到它所更新的原始 DataUpdate 查询的时间戳. 在敌手 $\mathcal{A}_{\text{Adv}}$ 的视角中, 模拟协议 Setup 与 KLVSE.Setup 是一致的.

在模拟的 DataUpdate 中, $\mathcal{S}'$ 通过从 $\mathbb{G}$ 中采样 3 个随机数并计算这些元素的 K_1 次幂来模拟密文 (第 2 和 3 行). 第 4 行, $\mathcal{S}'$ 将这 3 个元素和模拟的密文以及时间戳插入到 **CList** 中. 模拟的密文即使在 $\mathcal{A}_{\text{Adv}}$ 得到泄露的密钥和私有状态情况下, 也与通过 KLVSE.DataUpdate 生成的真实密文不可区分. 由于新生成的密文是用新的随机搜索令牌加密的, 除非 $\mathcal{A}_{\text{Adv}}$ 持有相应的搜索令牌, 否则密文与随机数不可区分. 而且, 除非 $\mathcal{A}_{\text{Adv}}$ 再次获得更新后的密钥和私有状态, 否则无法访问搜索令牌.

模拟的 Search 与 KLVSE.Search 不可区分. 在安全信道的保护下, $\mathcal{A}_{\text{Adv}}$ 无法了解搜索令牌和搜索结果. 模拟的客户端和云服务器只需要选择 $a_{\max}$ 个随机数据和一个随机数作为模拟验证信息并通过安全信道传输这些数据.

同理, 模拟协议 KeyUpdate 受到安全信道的保护, 即 $\mathcal{A}_{\text{Adv}}$ 无法获得 KeyUpdate 令牌. 没有密钥更新令牌, 即使 $\mathcal{A}_{\text{Adv}}$ 获得了密钥更新后密文或密钥更新前密文的某个密钥, 也无法判断密钥更新后的密文是否更新自密钥更新前的密文. 这种安全属性可以归结为群 $\mathbb{G}$ 中的 DDH 假设. 具体而言, 假设一个可搜索密文 (L, D, C) 可以写成 (g^x, g^y, g^z) 的形式, 其中 g 是 $\mathbb{G}$ 的生成元, $x, y, z \in \mathbb{Z}_q^*$. 那么, 即使 $\mathcal{A}_{\text{Adv}}$ 得知了底层的密钥 (K_1, K_2)、条目 $(\text{op}, (w, \text{id}))$、搜索令牌 tk'_w, 以及用于生成 (L, D, C) 的下一个搜索令牌 tk_w, $\mathcal{A}_{\text{Adv}}$ 也无法确切知道 x, y 和 z. 比如, 假设 $\mathbf{H}_1(\text{tk}'_w) = g^{h_1}$, $\mathbf{H}_2(\text{tk}'_w) = g^{h_2}$, $\mathbf{G}(\text{tk}'_w) = g^{h_3}$, $\pi(\text{tk}_w) = g^{n_1}$, 且 $\pi(\text{op}||\text{id}) = g^{n_2}$, 则有 $x = K_1 \cdot h_1$, $y = (n_1 + h_2) \cdot K_1$ 和 $z = K_2 \cdot n_2 + K_1 \cdot h_3$, 其中 h_1, h_2 和 h_3 对 $\mathcal{A}_{\text{Adv}}$ 来说是独立且随机的. 因此, 由于随机预言机的性质, x, y 和 z 对 $\mathcal{A}_{\text{Adv}}$ 来说是独立且随机的. 假设 $\mathcal{S}'$ 持有 KeyUpdate 令牌 $\Delta \in \mathbb{Z}_q^*$、一个随机比特 $b \in \{0, 1\}$、两个不同的密文 $(L_0, D_0, C_0) = (g^{x_0}, g^{y_0}, g^{z_0})$ 和 $(L_1, D_1, C_1) = (g^{x_1}, g^{y_1}, g^{z_1})$, 其中 $x_0, y_0, z_0, x_1, y_1, z_1 \in \mathbb{Z}_q^*$, 且 $\mathcal{A}_{\text{Adv}}$ 被给定 $(g^{x_0}, g^{y_0}, g^{z_0})$, $(g^{x_1}, g^{y_1}, g^{z_1})$ 和 $(g^{\Delta}, g^{x_b \cdot \Delta}, g^{y_b \cdot \Delta}, g^{z_b \cdot \Delta})$ (注意 $\mathcal{A}_{\text{Adv}}$ 实际上在真实和理想游戏中都无法访问 g^{Δ}), 则 $\mathcal{A}_{\text{Adv}}$ 成功猜测出正确的 $b^* \in \{0, 1\}$ 使得 $b^* = b$ 的优势可以直接归约为 $\mathcal{A}_{\text{Adv}}$ 攻破 3 个 DDH 假设实例之一的优势, 显然该优势是可忽略的. 因此在 $\mathcal{A}_{\text{Adv}}$ 看来, 模拟的 KeyUpdate 与真实的 KeyUpdate 协议是不可区分的.

当密钥泄露事件发生时, 模拟器 $\mathcal{S}'$ 将执行 KeyLeak 协议, 输入泄露的信息模拟该事件. 协议 KeyLeak 的目标是向 $\mathcal{A}_{\text{Adv}}$ 暴露密钥和私有状态, 使得 $\mathcal{A}_{\text{Adv}}$ 能够使用密钥和私有状态对密文数据库 **EDB** 进行检索. $\mathcal{S}'$ 首先从 $\text{CUHist}(U_{\text{now}})$ 和 $\text{DUHist}(U_{\text{now}})$ 中提取包含相同关键字 w 的条目, 并将这些条目存储到 DB_w 中 (第 4~7 行). 其次, 对于 DB_w 中的每个条目, $\mathcal{S}'$ 准备可以用来生成该条目的搜索令牌 tk'_w, 如果一个条目已经被分配了一个搜索令牌, $\mathcal{S}'$ 将跳过这个条目并处理下一个条目 (第 8~10 行). 然后, $\mathcal{S}'$ 使用 $\mathbf{State}[w]$ 中的信息来构建 w 的密文的隐藏链 (第 13~19 行), 它编程随机预言机来构建 w 的密文的隐藏链 (第 18 行), 该过程模拟了离线攻击的敌手在本地搜索相应密文索引的过程, 保证了搜索流程的不可区分. 最后, $\mathcal{S}'$ 向 $\mathcal{A}_{\text{Adv}}$ 暴露密钥 $K_\Sigma = (K_1, K_2)$ 和私有状态 **State**. 因此, $\mathcal{A}_{\text{Adv}}$ 无法区分虚拟世界中的密钥泄漏与真实世界的密钥泄漏事件.

在 KeyLeak 之后, 若 $\mathcal{A}_{\text{Adv}}$ 采用离线攻击的方式, 则能下载密文数据库的副本, 并使用暴露的搜索密钥 K_1 和加密密钥 K_2 以及私有状态 **State** 在密文数据库副本上执行搜索和解密. 但由于存在文件密文索引与文件的同步映射机制, $\mathcal{A}_{\text{Adv}}$ 下载的副本仅为文件索引而非真实文件, $\mathcal{A}_{\text{Adv}}$ 仅能解密得到文件索引. 若 $\mathcal{A}_{\text{Adv}}$ 采用在线攻击的方式, 则需使用泄露的密钥和私有状态向云服务器发起搜索查询.

综上, 构建的 $\mathcal{S}'$ 在 $\mathcal{A}_{\text{Adv}}$ 的视角中与真实游戏不可区分. 因此, KLVSE 是 $\mathcal{L}_{\text{Adv}}$-适应性安全的.

上述安全性分析证明了 KLVSE 分别使用定理 1 中定义的泄露函数 $\mathcal{L}_{\text{Srv}}$ 和 $\mathcal{L}_{\text{Adv}}$ 时是 $\mathcal{L}_{\text{Srv}}$-适应性安全和 $\mathcal{L}_{\text{Adv}}$-适应性安全的.

表 2　与其他方案的性能比较.

Table 2　Performance comparison with other schemes.

Scheme	DataUpdate	Search	Verify	KeyUpdate	Storage overhead
Fides^{KU}	$O(1)$	$O(a_w)$	–	$O(N)$	$O(W \log F)$
Mitra^{KU}	$O(1)$	$O(a_w)$	–	$O(1)$	$O(W \log F)$
Aura^{KU}	$O(1)$	$O(n_w)$	–	$O(N)$	$O(Wd_{\max})$
Bamboo	$O(1)$	$O(a_{\max})$	–	$O(1)$	$O(W \log F)$
KLVSE	$O(1)$	$O(a_{\max} + 2\delta)$	$O(1)$	$O(1)$	$O(W \log F + n_{\text{src}})$

5　性能评估

本节从理论和实验两方面分析了 KLVSE 方案的性能, 并与 Bamboo 方案 [8] 以及经过改造的 Fides 方案 [13] 可更新密钥版本、Mitra [38] 可更新密钥版本、Aura [39] 可更新密钥版本进行比较, 结合实现验证所增长的时间开销来看, KLVSE 方案达到了在高效搜索的同时使用少量开销完成密钥泄露验证的目标.

5.1　理论分析

表 2 展示了 KLVSE 与其他方案的理论开销比较. 其中, a_w, N, W, F, n_w, $d_{\max}$, $a_{\max}$, δ, n_{src} 分别表示密文更新查询总数、密文索引总数、关键字数量、文件数量、包含关键字 w 的文件数、最大删除查询数、密文最大填充值、验证签名大小、搜索计数器大小.

在搜索阶段, Fides^{KU} 与 Mitra^{KU} 方案的搜索效率与密文更新查询总数相关, Aura^{KU} 则取决于匹配关键字 w 的文件数, 而由于 Bamboo 与 KLVSE 均会采用密文最大填充值来隐藏真实密文数量, 因此搜索效率与 $a_{\max}$ 相关. 此外, KLVSE 需要生成验证签名 δ 和 β 用于验证云服务器和客户端搜索计数器的同步性, 以检测敌手的非法搜索行为, 这个过程需要比 Bamboo 方案多出计算验证签名的额外开销, 但基本不会影响线性搜索的整体开销.

在验证阶段, KLVSE 只需要常数级的计算开销完成一次比较操作. 在密文更新和密钥更新阶段, 由于不涉及验证密钥泄露的相关操作, KLVSE 与 Mitra^{KU}, Bamboo 方案具有一致的性能优势. 就客户端和云服务器的存储开销而言, KLVSE 需要维护搜索计数器, 因此比其他方案多占用少量额外的存储空间.

5.2　实验评测

本文在 Ubuntu 系统上对 KLVSE 与其他方案进行了 C++ 代码实现, 分别测试相关操作的运行时间. 实验核心配置为: 16.0 GB 内存, i7-10700 CPU @ 2.90 GHz, Ubuntu 20.04 x64. 客户端和云服务器间使用 TCP 套接字建立网络通信, 并用 SQLite 数据库存储客户端状态, PostgreSQL 数据库存储密文. KLVSE 方案使用 openssl 开发包 (提供了 SHA-256, SHA-384 和 SHA-512 密码散列函数) 和 Relic Toolkit (提供了 NIST-P256 椭圆曲线算法) 实现哈希函数、伪随机函数以及可逆映射函数. 实验选取搜索结果数量范围为1 万 $\sim$ 100 万, 评测了 KLVSE 与其他方案搜索查询的时间开销、带删除情况下的搜索时间开销 (未删除的数据集基数为 50000) 以及 KLVSE 的验证时间开销.

表 3 和 4 列出了8 组数据, 各方案的搜索性能表现如下: 除 Aura^{KU} 的客户端搜索时间保持基本恒定外, 其他方案的总搜索时间和客户端搜索时间均与搜索结果数量呈线性增长关系. 为了评估网络延迟对搜索性能的影响, 表 3 还列出了不同网络延迟 (delay 设置为 300 ms) 下各方案的总搜索时间, 结果显示网络延迟会小幅增加搜索时间, 但整体影响有限. 在搜索性能对比方面, Bamboo 和 KLVSE 的搜索时间开销显著优于 Fides^{KU}. 需要说明的是, 由于 Bamboo 和 KLVSE 在搜索时会填充搜索结果至最大值来隐藏搜索体量, 而 Mitra^{KU} 和 Aura^{KU} 并未隐藏搜索体量, 因此前两者的搜索效率略

表 3　总搜索时间 (s) 与搜索结果数量 n ($\times 10^4$) 的关系.

Table 3　Total search time cost (s) vs. result size n ($\times 10^4$).

Scheme	delay	$n=1$	$n=5$	$n=10$	$n=15$	$n=20$	$n=25$	$n=50$	$n=100$
Fides^{KU}	No delay	69.013	341.513	692.993	1021.672	1387.619	1742.912	3446.754	6869.744
	With delay	69.183	342.539	699.475	1029.212	1399.363	1752.110	3467.747	6877.078
Mitra^{KU}	No delay	11.072	55.235	110.335	164.353	220.792	274.632	553.195	1092.907
	With delay	11.278	56.514	111.858	166.666	224.099	278.992	559.868	1103.816
Aura^{KU}	No delay	33.087	43.086	54.974	66.911	78.922	90.764	155.581	309.208
	With delay	33.097	43.154	55.433	68.210	81.827	91.637	158.633	331.962
Bamboo	No delay	25.145	122.207	244.492	366.060	489.387	607.870	1229.463	2442.232
	With delay	25.199	123.535	249.728	369.750	499.905	625.029	1234.664	2466.137
KLVSE	No delay	25.203	122.589	244.974	366.523	489.793	608.302	1233.692	2446.657
	With delay	25.235	123.791	250.463	369.819	501.888	627.646	1238.249	2467.402

表 4　客户端搜索时间 (s) 与搜索结果数量 n ($\times 10^4$) 的关系.

Table 4　Client search time cost (s) vs. result size n ($\times 10^4$).

Scheme	$n=1$	$n=5$	$n=10$	$n=15$	$n=20$	$n=25$	$n=50$	$n=100$
Fides^{KU}	63.122	312.173	632.041	931.869	1265.703	1589.024	3122.327	6260.738
Mitra^{KU}	9.683	48.055	95.799	144.105	192.401	240.140	484.889	932.417
Aura^{KU}	0.221	0.224	0.221	0.225	0.226	0.227	0.226	0.229
Bamboo	3.578	17.916	35.905	53.743	72.202	90.273	189.198	360.740
KLVSE	3.625	18.121	36.437	54.521	72.956	91.093	189.903	363.364

低于后者. 值得注意的是, 与 Bamboo 方案相比, KLVSE 在总搜索时间和客户端搜索时间方面不仅没有显著的增加, 且能够达到几乎一致的性能表现. 例如在 no delay 情况下, 搜索结果数量为 10000 时, Bamboo 总搜索时间为 25.145 s, KLVSE 为 25.203 s, 增加损耗 0.23%; 搜索结果数量为 250000 时, Bamboo 总搜索时间为 607.870 s, KLVSE 为 608.302 s, 增加损耗 0.07%, 总搜索时间最大损耗不足 1%, 客户端搜索时间最大损耗不足 1.6%. 这表明在搜索阶段, 为了实现验证密钥泄露功能而嵌入维护搜索计数器、生成验证签名等操作不会明显加重客户端和云服务器的计算与通信时间开销, 表明本文工作牺牲极小的代价实现了验证密钥泄露的目标.

表 5 和 6 显示, 在不同删除比例下, Fides^{KU} 的总搜索时间和客户端搜索时间随删除比例增加呈线性下降趋势, Aura^{KU} 的搜索时间则随着删除比例提高逐渐趋于稳定, 而其他方案的总搜索时间和客户端搜索时间都随删除比例的提高而线性增加. 与上述单纯进行搜索查询的实验结论类似, 网络延迟对带删除的搜索效率影响微乎其微. 从性能表现来看, Bamboo 与 KLVSE 为隐藏搜索体量牺牲了一定的搜索效率, 因此总搜索时间均高于 Mitra^{KU} 和 Aura^{KU}, 但在客户端时间开销方面明显优于 Mitra^{KU}. 其次, 在低删除比例下, Bamboo 与 KLVSE 的性能显著优于 Fides^{KU}, 但随着删除比例持续升高, Fides^{KU} 逐渐呈现出一定优势. 值得注意的是, 与 Bamboo 相比, KLVSE 并无明显劣势, 总搜索时间最大损耗 0.3%, 客户端搜索时间最大损耗 1.7%. 这表明 KLVSE 为实现验证密钥泄露的维护和辅助操作牺牲的效率损失几乎可以忽略不计, 由极小代价实现了密钥泄露后安全.

如图 4 所示, 随着搜索结果数量从 1 万递增至 100 万, KLVSE 验证签名得到验证结果耗费的时间非常低, 平均验证时间约为 6 μs, 控制在 10 μs 以内, 且基本保持恒定. 这是因为验证操作仅涉及客户端重新生成本地最新的验证签名, 并与云服务器生成的验证签名相比较, 基本不受搜索结果数量的影响. 而从数据上看, 相较于其他核心操作的时间开销 (例如密文更新时间开销在毫秒级、搜索时间开销在秒至千秒级), 验证时间开销极低, 表明相比于缺乏 验证密钥泄露的 Bamboo 方案, 实现密钥泄露

表 5　不同删除比例下的总搜索时间 (s).

Table 5　Total search time cost (s) vs. percentage of deletion.

Scheme	delay	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Fides^{KU}	No delay	339.982	310.551	282.861	253.433	225.127	196.626	168.126	140.014	112.053	83.349
	With delay	340.364	311.986	283.756	254.544	225.671	197.524	169.166	140.272	111.865	83.358
Mitra^{KU}	No delay	54.744	60.485	65.691	71.339	76.720	82.184	87.709	92.895	98.676	104.039
	With delay	55.647	61.728	67.536	72.037	77.411	84.301	89.506	94.282	100.713	108.600
Aura^{KU}	No delay	42.734	23.639	25.081	25.611	25.922	26.600	25.651	26.051	24.805	24.234
	With delay	42.931	24.006	25.097	25.928	26.010	26.993	25.974	26.357	25.875	24.478
Bamboo	No delay	121.982	134.209	146.666	158.445	170.530	183.147	195.217	207.436	218.819	231.783
	With delay	123.950	134.960	147.229	160.394	170.862	183.814	198.718	210.275	223.806	236.019
KLVSE	No delay	122.344	134.585	146.934	158.659	170.929	183.456	195.579	207.779	219.143	232.098
	With delay	123.959	134.990	147.456	160.596	170.984	183.880	198.930	210.769	224.046	236.231

表 6　不同删除比例下的客户端搜索时间 (s).

Table 6　Client search time cost (s) vs. percentage of deletion.

Scheme	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
Fides^{KU}	310.580	278.670	248.573	216.692	185.950	154.949	123.809	92.787	62.100	30.942
Mitra^{KU}	47.963	52.842	57.449	62.391	67.098	71.906	76.723	81.386	86.254	91.076
Aura^{KU}	0.244	6.684	9.548	11.104	12.031	12.477	12.673	12.748	12.502	12.270
Bamboo	18.004	19.797	21.659	23.435	25.260	27.078	28.908	30.640	32.317	34.127
KLVSE	18.285	20.133	21.968	23.789	25.639	27.470	29.213	30.962	32.625	34.414

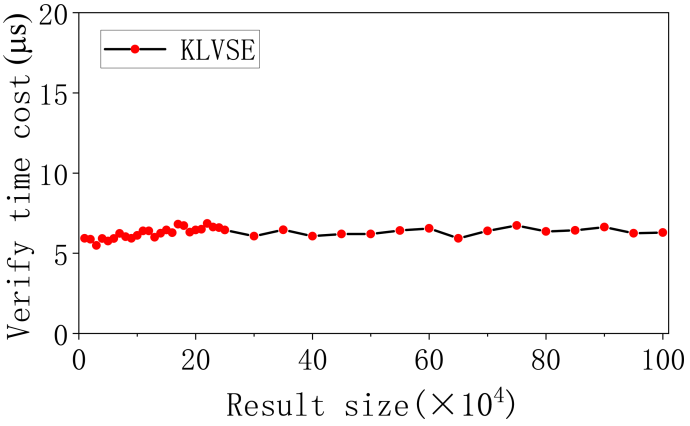

图 4　(网络版彩图) 总验证时间与搜索结果数量关系.

Figure 4　(Color online) Time cost of **Verify** vs. result size.

验证的相关操作不会显著增加客户端的负担, 在资源受限的场景下也能表现出较高的适用性.

6　总结

本文针对可搜索加密中的客户端密钥泄露问题, 引入验证机制, 构造了密钥泄露可验证可搜索加密模型, 实现了密钥泄露可验证的高效 DSSE 方案 KLVSE. 通过杜绝敌手下载数据库副本进行离线搜索的情况确保在线搜索为敌手唯一攻击方式, 进一步利用双向验证机制实现密钥泄露的验证, 从而达到检测敌手非法搜索行为的效果, 进而在密钥泄露后及时触发密钥更新, 保障密钥泄露后安全. KLVSE

在密钥可更新的可搜索加密方案基础上, 解决了密钥泄露检测的问题, 实现了在密钥泄露情况下的数据安全, 提高了系统的安全性. 实验显示, 与未实现密钥泄露验证的方案相比, KLVSE 以极低的代价即可实现验证功能, 对整体的计算和通信效率影响小. 验证机制的嵌入提高了系统对密钥泄露的及时发现和应对能力, 从而更有效地保障用户数据在存储和搜索方面的安全性和隐私性.

参考文献

1 Song D X, Wagner D, Perrig A. Practical techniques for searches on encrypted data. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2000. 44–55
2 Han J, Qi L, Zhuang J. Vector sum range decision for verifiable multiuser fuzzy keyword search in cloud-assisted IoT. IEEE Internet Things J, 2023, 11: 931–943
3 Leyden J. 23,000 HTTPS certs will be axed in next 24 hours after private keys leak. 2018. https://www.theregister.com/2018/03/01/trustico_digicert_symantec_spat
4 Muncaster P. Stolen cloud api key to blame for imperva breach. 2019. https://www.infosecurity-magazine.com/news/stolen-cloud-api-key-to-blame-for
5 Lehmann A, Tackmann B. Updatable encryption with post-compromise security. In: Proceedings of the 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2018. 685–716
6 Boneh D, Lewi K, Montgomery H, et al. Key homomorphic PRFs and their applications. In: Proceedings of the Annual Cryptology Conference (CRYPTO), 2013. 410–428
7 Xu P, Susilo W, Wang W, et al. ROSE: robust searchable encryption with forward and backward security. IEEE Trans Inform Forensic Secur, 2022, 17: 1115–1130
8 Chen T, Xu P, Picek S, et al. The power of bamboo: on the post-compromise security for searchable symmetric encryption. In: Proceedings of Network and Distributed System Security (NDSS) Symposium, 2023
9 Just M, van Oorschot P C. Addressing the problem of undetected signature key compromise. In: Proceedings of Network and Distributed System Security (NDSS) Symposium, 1999
10 Kuzmicheva S, Kiryakina M, Zapechnikov S. Methods and algorithms for detecting compromise of secret keys. In: Proceedings of Advanced Technologies in Robotics and Intelligent Systems, 2020. 275–283
11 Kamara S, Papamanthou C, Roeder T. Dynamic searchable symmetric encryption. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2012. 965–976
12 Bost R. $\sum o\varphi o\varsigma$: forward secure searchable encryption. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2016. 1143–1154
13 Bost R, Minaud B, Ohrimenko O. Forward and backward private searchable encryption from constrained cryptographic primitives. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017. 1465–1482
14 Zuo C, Sun S F, Liu J K, et al. Dynamic searchable symmetric encryption with forward and stronger backward privacy. In: Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS), 2019. 283–303
15 Song X, Dong C, Yuan D, et al. Forward private searchable symmetric encryption with optimized I/O efficiency. IEEE Trans Dependable Secure Comput, 2018, 17: 912–927
16 Zhang Z, Wang J, Wang Y, et al. Towards efficient verifiable forward secure searchable symmetric encryption. In: Proceedings of the 24th European Symposium on Research in Computer Security (ESORICS), 2019. 304–321
17 Shi Z, Fu X, Li X, et al. ESVSSE: enabling efficient, secure, verifiable searchable symmetric encryption. IEEE Trans Knowl Data Eng, 2022, 34: 3241–3254
18 Najafi A, Javadi H H S, Bayat M. Efficient and dynamic verifiable multi-keyword searchable symmetric encryption with full security. Multimed Tools Appl, 2021, 80: 26049–26068
19 Chen T, Xu P, Wang W, et al. Bestie: very practical searchable encryption with forward and backward security. In: Proceedings of the 26th European Symposium on Research in Computer Security (ESORICS), 2021. 3–23
20 Dou H, Dan Z, Xu P, et al. Dynamic searchable symmetric encryption with strong security and robustness. IEEE Trans Inform Forensic Secur, 2024, 19: 2370–2384
21 Boyd C, Davies G T, Gjøsteen K, et al. Fast and secure updatable encryption. In: Proceedings of the Annual International Cryptology Conference (CRYPTO), 2020. 464–493
22 Klooß M, Lehmann A, Rupp A. (R)CCA secure updatable encryption with integrity protection. In: Proceedings of the

38th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), 2019. 68–99

23 Yang N, Tang C, Zhou Q, et al. Dynamic consensus committee-based for secure data sharing with authorized multi-receiver searchable encryption. IEEE Trans Inform Forensic Secur, 2023, 18: 5186–5199

24 Chai Q, Gong G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In: Proceedings of the IEEE International Conference on Communications (ICC), 2012. 917–922

25 Kurosawa K, Ohtaki Y. UC-secure searchable symmetric encryption. In: Proceedings of the 16th International Conference (FC), 2012. 285–298

26 Ogata W, Kurosawa K. Efficient no-dictionary verifiable SSE. Cryptology ePrint Archive, 2016. https://eprint.iacr.org/2016/981.pdf

27 Zhu J, Li Q, Wang C, et al. Enabling generic, verifiable, and secure data search in cloud services. IEEE Trans Parallel Distrib Syst, 2018, 29: 1721–1735

28 Bost R, Fouque P A, Pointcheval D. Verifiable dynamic symmetric searchable encryption: optimality and forward security. Cryptology ePrint Archive, 2016. https://eprint.iacr.org/2016/062.pdf

29 Yuan D, Cui S, Russello G. We can make mistakes: fault-tolerant forward private verifiable dynamic searchable symmetric encryption. In: Proceedings of the 7th European Symposium on Security and Privacy (EuroS&P), 2022. 587–605

30 Lu H, Chen J, Ning J, et al. Verifiable conjunctive dynamic searchable symmetric encryption with forward and backward privacy. Comput J, 2023, 66: 2379–2392

31 Hu S, Cai C, Wang Q, et al. Searching an encrypted cloud meets blockchain: a decentralized, reliable and fair realization. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM), 2018. 792–800

32 Wu H, Song R, Lei K, et al. Slicer: verifiable, secure and fair search over encrypted numerical data using blockchain. In: Proceedings of the 42nd International Conference on Distributed Computing Systems (ICDCS), 2022. 1201–1211

33 Guo Y, Zhang C, Wang C, et al. Towards public verifiable and forward-privacy encrypted search by using blockchain. IEEE Trans Dependable Secure Comput, 2022, 20: 2111–2126

34 Xu C, Yu L, Zhu L, et al. Blockchain-based verifiable DSSE with forward security in multi-server environments. In: Proceedings of the 16th International Conference on Wireless Algorithms, Systems, and Applications (WASA), 2021. 163–171

35 Ferreira B, Portela B, Oliveira T, et al. Boolean searchable symmetric encryption with filters on trusted hardware. IEEE Trans Dependable Secure Comput, 2020, 19: 1307–1319

36 Najafi A, Bayat M, Haj Seyyed Javadi H. Fair multi-owner search over encrypted data with forward and backward privacy in cloud-assisted Internet of Things. Future Generation Comput Syst, 2021, 124: 285–294

37 Guo W, Wang L B. Lower cost David's digital library protocol. Modern Comput, 2009, 6: 29–31 [郭维, 王立斌. 低成本 David 的数字图书馆协议. 现代计算机, 2009, 6: 29–31]

38 Chamani J G, Papadopoulos D, Papamanthou C, et al. New constructions for forward and backward private symmetric searchable encryption. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2018. 1038–1055

39 Sun S, Steinfeld R, Lai S, et al. Practical non-interactive searchable encryption with forward and backward privacy. In: Proceedings of Network and Distributed System Security (NDSS) Symposium, 2021

Research on key leakage-verifiable searchable encryption with post-compromise security

Wei WANG[1], Shufang LEI[1], Dongli LIU[1], Peng XU[2*] & Laurence Tianruo YANG[3]

1. *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
2. *School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, China*
3. *School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China*

* Corresponding author. E-mail: xupeng@hust.edu.cn

Abstract Searchable encryption technology can enable secure search over encrypted data, ensuring the security of searching for users on a cloud server. However, the risk of key leakage in practical applications poses a significant threat to the security and privacy of data. The previous work Bamboo enables key update after key leakage and ensures the security of ciphertexts between key leakage and key update through two-layer encryption and hidden chain structure, but it does not consider how to detect key leakage to trigger key update actively. Therefore, this paper proposes a key leakage-verifiable dynamic searchable symmetric encryption (DSSE) scheme called KLVSE based on key-updatable technology. It introduces a bidirectional verification mechanism to verify key leakage events and detect potential illegal search behavior by the adversary. This ensures that key updates are triggered immediately after key leakage and guarantees post-compromise security. The experimental results demonstrate that compared to the traditional periodic key update strategy, this scheme achieves key leakage verification with a sufficiently small runtime delay while ensuring security, effectively safeguarding the security and privacy of user data.

Keywords searchable encryption, key leakage-verifiable, key-updatable, post-compromise security