可验证对称加密搜索算法研究

（申请清华大学工程硕士专业学位论文）

培 养 单 位: 计 算 机 科 学 与 技 术 系
学　　科: 计 算 机 技 术
申　请　人: 朱　洁
指 导 教 师: 李　琦　副　研　究　员

二〇一八年五月

A Study of Verifiable Searchable Symmetric Encryption

Thesis Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the professional degree of

Master of Engineering

by

Zhu Jie

(Computer Technology)

Thesis Supervisor : Professor Li Qi

May, 2018

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘 要

云存储的发展使得用户可以方便地存储、获取与分享数据，例如提供了随时存取功能，减少了本地数据存储开销并且提供了灵活的服务。但与此同时，云存储也带来了很多安全问题，例如，数据隐私泄露等等。对称加密搜索的提出解决了云存储的数据隐私泄露问题，同时也保证了数据的可搜索性。通过使用对称加密搜索，用户可以将数据加密后再上传给云服务器，同时云服务器可以在加密数据上进行关键字搜索，从而确保数据隐私。

然而，现有对称加密搜索方案的假定都是云服务器半可信，即云服务器会遵守协议。不幸的是，这个假设在现实情况中往往是不成立的，因为云服务器有可能会遭受外部攻击，内部配置错误，软件错误，甚至内部攻击等等。所有这些因素都会导致云服务器背离原有协议。为了解决该问题，可验证对称加密搜索技术相应提出，它通过结果验证机制可以检测云服务器的恶意行为。但是，现有的可验证对称加密搜索方案都不完善，例如，不支持用户数据动态更新，依赖于特定的对称加密搜索方案，只支持单用户读写等等。

针对以上的问题，本文提出了一种通用的可验证对称加密搜索方案，该方案支持用户数据更新，可以为任何加密搜索方案提供结果验证功能，并且能够同时在单用户和多用户的场景下工作。本文的主要工作和创新点包括：

- 提出了一种单用户场景下的可验证对称加密搜索框架，并在此基础上设计了结果验证算法，该算法能同时保证数据新鲜性和数据完整性。该框架支持用户数据动态更新，并且将验证索引从对称加密搜索方案中解耦，使其可以为任何加密搜索方案提供结果验证功能。
- 提出了一种多用户场景下的可验证对称加密搜索框架。该框架支持单用户写，多用户读，确保了多用户场景下的数据新鲜性，并实现了数据共享场景下的结果验证。
- 通过安全性分析证明，本文提出的可验证对称加密搜索方案不泄露用户数据隐私信息。通过基于开源数据集的实验测试表明，方案引入的计算开销和通信开销很小，几乎可以忽略不计。

关键词：对称加密搜索，结果验证，云存储

Abstract

Cloud storage allows users to retrieve and share their data conveniently with well understood benefits, such as on-demand access, reduced data maintenance cost, and service elasticity. Meanwhile, cloud storage also brings serious data privacy issues, i.e., the disclosure of private information. In order to ensure data privacy without losing data usability, a cryptographic notion named Searchable Symmetric Encryption (SSE) has been proposed. By using SSE, users can encrypt their data before uploading to cloud services, and cloud services can directly operate and search over encrypted data, which ensures data privacy.

However, most existing SSE schemes are built based on the assumption that cloud services are semi-honest, which means cloud services will not deviate from the prescribed protocols. Unfortunately, this assumption does not always hold in practice, since cloud services may be subject to external attacks, internal misconfiguration errors, software bugs, and even insider threats. All these factors may cause the cloud services to deviate from the prescribed protocol and operate beyond the semi-honest model. To alleviate the issue, there have been studies on Verifiable Searchable Symmetric Encryption (VSSE), which functions against malicious cloud services by enabling results verification. But to our best knowledge, existing VSSE schemes exhibit very limited applicability, such as only supporting static database, demanding specific SSE constructions, or only working in the single-user model.

In this paper, we proposed a generic verifiable SSE framework in both single-user model and single-writer multiple-reader model, which provides verifiability for any SSE schemes and further supports data updates. In summary, our contributions are three-fold:

- We proposed a verifiable SSE framework in single-user model and designed the result verification algorithms. The framework seperate the verification index from the SSE construction and can provides generic verification for any SSE schemes. The algorithms guaranteed both data freshness and data integrity with support of data updates.
- We proposed the first verifiable SSE framework in the single-writer and multiple-reader model, which ensures data freshness across multiple users and provides result verification under data sharing scenario.

- Security analysis shows that our shceme will not leak any private information and experimental result shows that our scheme introduces negligible overhead for result verification.

Key words: Searchable Symmetric Encryption; Result Verification; Cloud Storage

目 录

主要符号对照表

SSE	对称加密搜索 (Searchable Symmetric Encryption, SSE)
VSSE	可验证对称加密搜索 (Verifiable Searchable Symmetric Encryption, VSSE)
MPT	默克尔帕特里夏树 (Merkle Patricia Tree, MPT)
IH	增量哈希 (Incremental Hash, IH)
$\mathcal{W}$	关键字集合
$\|W\|$	关键字集合大小
w_i	关键字，其中 $i \in \{1, \cdots, \|W\|\}$
$\mathcal{D}$	明文文件集合
D_w	包含关键字 w 的明文文件集合
C	密文文件集合
C_w	包含关键字 w 的密文文件集合
d	明文文件
c	密文文件
W_d	文件 d 包含的关键字集合
$\|W_d\|$	文件 d 包含的关键字集合大小
τ	令牌
λ	验证索引
π	鉴别符

第 1 章　绪论

1.1　研究背景及选题意义

随着云计算技术的发展与日渐成熟，云存储技术也被广泛提及。对个人用户来说，云存储技术通过提高存储效率，为其节省了大量本地存储空间，也便于用户随时随地存储数据，同时云存储技术还为多人之间的数据共享提供了高效的解决方案。对于企业用户来说，云存储技术通过虚拟化技术进行了资源整合，提高了存储空间的利用率，同时云存储技术具备的负载均衡、故障冗余等功能也确保了企业级数据的安全。目前，云存储技术主要分为私有云、公有云和混合云三个类别。其中私有云存储技术主要面向企业用户，为其提供一个安全、高效的云存储环境。一般来说，私有云往往部署在企业内部的数据中心内，它将数据维护在企业防火墙之内，因此安全性较高。而公有云存储业务则向互联网开放，其计算资源相对私有云有很大的优势，但其安全性相对私有云大大降低。公有云存储业务主要面向个人消费者，目前几乎任何一家大型互联网公司都在为用户提供公有云存储服务，例如，Google Drive，iCloud，Dropbox，百度云等等。但公有云存储确实导致了许多安全性问题的产生，例如，数据丢失，数据隐私泄露等等。iCloud 曾在 2014 年出现过严重的数据泄露事件，导致用户对其信任度大大降低，Dropbox 也出现过类似事件。总体来说，云存储技术虽然使得用户可以随时随地地存取数据，方便了用户之间的数据共享，降低了维护数据的成本 **???????**，但其带来的安全问题也不容忽视。云存储带来的安全性问题可以分为以下两类：

- 可用性问题。要求云服务器保证数据不丢失，用户可以将云服务器作为数据中枢进行数据备份和同步。目前，一般的云服务提供商都采用了多副本的方式来保障数据的可用性，即将数据的多个副本分别写入其他的存储节点，当一个节点发生故障时，其他节点继续提供服务，同时通过其他节点中的数据副本，快速恢复故障节点上丢失的数据。目前，针对数据可用性的相关学术研究包括数据拥有证明 (Proof of Data Possession, PDP) **?????** ，数据可恢复性证明 (Proof of Retrievability, PoR) **???**，数据审计 (Data Auditing) **?????** 等等。
- 隐私性问题。要求云服务器保证数据的隐私并且不泄露数据。目前，云服务提供商一般采用数据加密方式对隐私数据进行保护，但数据加密往往会导致数据可用性的降低，例如数据失去可搜索性。因此加密搜索 (Searchable Encryption, SE) 随之产生。加密搜索方案按照采用的秘钥机制不同可以分为两类，一是对称加密搜索 (Searchable Symmetric Encryption, SSE)**?????**，二是

公钥加密搜索 (Public Key Encryption with Keyword Search, PEKS) **?**。

加密搜索的提出，使得用户可以在上传数据给云服务器之前，对其进行加密，并且使得云服务器可以在加密数据上进行搜索。从而既保证了数据的隐私性，又保证了数据的可搜索性。目前，由于效率问题，应用较为广泛的为对称加密搜索技术。然而，大部分的对称加密搜索方案都基于服务器是半可信的假设 **???**，即服务器会遵循协议但是可以从用户的查询中推断相关信息。这种假设在实际应用场景中往往是不成立的。因为云服务器可能会因为外部攻击，内部配置错误，软件错误等等问题而导致其违反原有协议 **??**。这种协议违反所导致的最常见问题就是服务器返回的搜索结果不完整。例如，云服务器有可能为了节省计算开销和通信开销而返回少量搜索结果给用户，甚至有可能不返回搜索结果给用户。

为了解决该问题，可验证对称加密搜索 (Verifiable Searchable Symmetric Encryption, VSSE) 技术也相应提出**????????**。可验证对称加密搜索技术允许用户对搜索结果进行验证，从而来检测云服务器的不诚信行为，保障加密搜索的正确性。然而，据我们所知，现有的可验证对称加密搜索方案都是不完善的。例如，有的方案 **????** 不支持数据更新，只能作用在静态数据库中，数据库若有变化则需要重建整个索引。有的方案 **???** 无法防止服务器故意返回空结果来规避结果验证。即，这些方案**???** 在用户提交的关键字不存在于数据库中时，是不返回任何搜索结果的。这就导致了服务器可以对任意关键字返回空结果来规避结果验证，除非用户在本地保留数据库的所有关键字集合。另外，大部分的可验证对称加密搜索方案 **????????** 仅仅支持在单用户场景下工作，即数据持有者自己写自己读的场景，而现实情况中，数据往往有共享需求，即一方写多方读的多用户场景[①]。表格 **??**比较了现有可验证对称加密搜索方案的功能。

1.2　研究现状

1.2.1　安全云存储方案

可验证的云存储服务已经被广泛的研究过，例如，数据拥有性证明 **?????**，数据可取回证明 **???**，数据审计 **?????** 等等。这些方案主要侧重于云端存储数据的完整性验证，并支持丢失数据的恢复。注意，这些方案与加密搜索场景下的结果验证是不同的。因为加密搜索的结果验证不仅需要验证某个文件本身的完整性，还需要验证整个搜索结果集合是否完整。而这些方案只能单纯验证数据块的完整性，不支持对搜索结果完整性的验证。

① 本文所述的多用户场景指一方写入，多方读取的场景，下文中若不做特别说明，均指这种情况。

表 1.1　现有可验证对称加密搜索方案比较

	动态 [1]	新鲜性 [2]	完整性 [3]	验证效率 [4]	通用 [5]	多用户
KPR11 **?**	✓	✓	×	$O(\|W\|)$	✓	×
KO12 **?**	×	-	×	$O(n)$	×	×
CG12 **?**	×	-	✓	$O(log(\|W\|))$	×	×
KO13 **?**	✓	✓	×	$O(n)$	×	×
SPS14 **?**	✓	✓	×	$min\{\alpha + log(N), rlog^3(N)\}$	×	×
CYG15**?**	×	-	×	$O(\|W\|) + O(r)$	×	×
BFP16 **?**	✓	✓	✓	$O(r)$	✓	×
OK16 **?**	×	-	✓	$O(r)$	✓	×

[1] 注意，动态是指方案是否支持用户数据动态更新，由此可将可验证对称加密搜索方案分为静态和动态两种类型，后者在功能性上更完善。

[2] 注意，'×' 表示有实现的需求但是该方案没有实现, 而'-' 表示没有实现的需求。具体而言，静态的可验证对称加密搜索方案不存在数据新鲜性问题，因此方案 **????** 也没有进行数据新鲜性验证的需求。

[3] 我们考虑各种数据完整性攻击，尤其包括服务器故意返回空结果来规避结果验证的场景。

[4] 验证效率是指服务器进行结果验证支持所需要的计算开销。对于表格中的非通用型方案 **?????** 来说，由于他们的方案并没有将验证索引从加密搜索方案中解耦，因此他们的验证效率和服务器进行加密搜索所需的计算开销是等价的。这里，n 代表所有文件的数量, $|W|$ 表示所有关键字的数量, r 表示包含某一特定关键字的文件数量, α 表示某一关键字历史上被加入到集合中的次数 **?**, N 表示键值对 (文件，关键字) 对的数量。

[5] 一个通用的可验证对称加密搜索方案是指该方案可以为任何加密搜索方案提供结果验证的功能，而非通用的可验证对称加密搜索方案表示该方案仅支持在特定的加密搜索方案下工作。

1.2.2　安全加密搜索方案

加密搜索的概念首次由 Song 等人 **?** 在 2000 年提出，他们的方案允许用户将加密后的数据集存储到云端，并同时保证用户在该加密数据集上进行搜索的能力。随后，加密搜索方案被广泛的研究，总体来说可以分为以下两个分支：对称加密搜索 (Searchable Symmetric Encryption, SSE) 和公钥加密搜索 (Public Key Encryption with Keyword Search, PEKS)。其中，最经典的对称加密搜索方案 **?** 由 Curtmola 等人提出，他们对加密搜索的安全性进行了严格的定义，提出加密搜索方案至少要在

面对一个被动敌手 (Passive Adversary) 的情况下是安全可靠的。他们的方案利用了明文搜索中的倒排索引 (Inverted Index) 思想，在效率和安全性上较已有的方案都有较大提升。目前还有许多不同的对称加密搜索方案实现了不同的搜索功能。例如，动态对称加密搜索 (Dynamic SSE) 方案 **???** 允许用户更新其数据集，支持关键字排序 (Ranked Keyword Search) 的对称加密搜索方案 **?** 允许用户获取根据某一影响因子排序后的搜索结果。最经典的公钥加密搜索方案 **?** 由 Boneh 等人提出，他们的方案利用了双线性映射技术。总体来说，公钥加密搜索方案的性能是远远低于对称加密搜索方案的。

1.2.3　可验证数据结构

可验证数据结构 (Authenticated Data Structure, ADS) 在不可信的云存储环境中，主要被用于验证数据块的完整性。典型的可验证数据结构包括：默克尔树 (Merkle Tree, MT) **?**, 可验证哈希表 (Authenticated Hash Table, AHT) **?** 以及可验证跳表 (Authenticated Skip List, ASL) **??**。其中，默克尔树是最常见的用于验证数据完整性的数据结构，但是默克尔树对数据更新的支持不够灵活。采用默克尔树实现的可验证对称加密搜索方案 **?** 由于没法在中间节点存储关键字信息，因此也不支持与关键字相关的搜索。可验证哈希表采用了 RSA 累加器 (RSA Accumulator) [①] 方法来实现数据验证，但是它的搜索和更新性能都较低。具体而言，可验证哈希表的搜索与更新速度在毫秒级别，而我们采用的默克尔帕特里夏树 (Merkle Patricia Tree, MPT) 的搜索更新速度在微秒级别。可验证跳表采用了类似多级链表的方式来实现，一定程度上提升了搜索性能，但如果它将关键字信息存储于搜索路径上，存储空间将比 MPT 大很多。

1.2.4　可验证对称加密搜索方案

由 Kamara 等人提出的 CS2 方案 **?** 通过使用默克尔树构建验证索引来支持用户对搜索结果的验证。具体的做法是，以加密的关键字作为“键”，以该关键字对应的加密文件集合作为“值”，将该“键值对”存储在默克尔树的叶子结点上。用户在本地需要保留默克尔树的根哈希作为一个指纹信息。在进行结果验证时，用户需要通过其搜索的关键字本身及服务器返回的该关键字对应在默克尔树上的路径来重构出该根哈希，并与保留的根哈希进行比对，从而来进行结果验证。但是他们的方案无法检测服务器恶意返回空结果的情况。关键的原因是，当用户搜索的关键字不存在时，默克尔树上不会存在该关键字对应的路径，因此服务器无法

① RSA 为提出该算法的三个密码学家名字的首字母，分别为 Ron Rivest, Adi Shamir, 和 Leonard Adleman

返回任何信息给用户。解决该问题的一个简单的方法是在构建默克尔树时，将整个字典空间中所有可能的关键字集合都存储在默克尔树中，但这样做会导致大量的空间浪费。

近期，Kurosawa 等人提出了一系列可验证对称加密搜索方案 **???**。但是他们的方案要么效率很低 **??**，要么不支持用户数据动态更新 **??**。其中方案 **?** 需要线性搜索时间并且不支持数据动态更新。他们的扩展方案 **?** 支持了用户数据更新，该方案通过消息验证码 (Message Authenticated Code, MAC) 来确保了数据完整性，通过 RSA 累计器确保了数据新鲜性，但是方案的搜索复杂度超过了线性时间，并且该方案需要用户在本地维护一个关键字集合来探测服务器故意返回空结果的情况，这将引入较大的空间开销。Ogata 等人也提出了一个通用的可验证对称加密搜索方案 **?**，该方案可以为任何对称加密搜索方案提供结果验证服务，并且不需要用户自己在本地维护一个关键字集合，但是他们的方案仍然是一个静态的方案，即不支持用户数据更新。同样，方案 **? ?** 也只是静态方案。

由 Stefanov 等人提出的方案 **?** 采用了时间戳和消息验证码机制来实现了结果验证，但是他们的方案没法防御服务器故意返回空结果来规避结果验证的情况。Bost 等人提出的方案 **?** 是目前为止最完善的通用可验证对称加密搜索方案，但他们的方案在搜索时需要与服务器进行两轮通信，即用户需要在拿到加密搜索的结果后再与服务器进行通信来验证搜索结果。加密搜索和结果验证过程在云服务器端无法并行进行，这将导致较大的验证时延和通信开销，并且他们的方案同样也不支持多用户场景下的结果验证。

总体来说，一个完善的通用可验证对称加密搜索方案首先应该支持数据新鲜性和数据完整性验证，尤其要关注搜索结果为空时的验证，这是一个较大的安全性漏洞，但被大部分的方案忽略。其次该方案应该在支持结果验证的同时，尽量降低用户本身的存储和计算开销，例如不需要用户在本地维护一个关键字集合。另外，该方案还应该支持用户数据的更新，并且能够支持多用户场景下的结果验证。综上所述，现有的可验证加密搜索方案无法在保证验证效率的同时，完善地验证数据新鲜性和数据完整性。并且现有的方案都不能满足多用户场景下的对称加密搜索结果验证。这需要我们利用合理的数据结构，并设计合理的算法来设计一个通用的可验证对称加密搜索方案。

1.2.5　可验证公钥加密搜索方案

第一个可验证的公钥加密搜索方案 **?** 由 Zheng 等人提出，他们的方案采用了基于属性的关键字 (Attribute-based keyword, ABK) 技术，但是他们的方案也只适用

于数据库静态的情况。基于他们的工作，Liu 等人又提出了一个更高效的可验证公钥加密搜索方案 **?**，Sun 等人也提出了一个支持多关键字搜索的可验证公钥加密搜索方案 **?**。然而，由于公钥加密本身的限制，他们的方案必不可少地需要引入一个可信第三方，并且搜索的性能大大低于可验证对称加密搜索方案。

1.2.6　多用户加密搜索方案

目前有一些多用户场景下的加密搜索方案 **????**，但这些方案都不支持结果验证。Curtmola 等人在 2006 年即提出了一个基于广播加密的多用户加密搜索方案 **?**，该方案允许数据持有者将数据分享给其他用户，并且数据持有者可以设定其他用户的访问控制权限，可以随时撤销或者新增用户。Yang 等人也通过双线性映射 (Bilinear Mapping) 技术提出了一种支持多用户读多用户写的方案 **?**，但是该方案的搜索效率与数据集合的大小成正比，无法应用于数据量很大的场景中。Jerecki 等人随后又提出了一个多用户加密搜索方案 **?**，但是该方案需要数据持有者和其合法数据搜索用户产生频繁交互，给数据持有者带来了很大的通信开销。近期，Sun 等人提出了一个非交互式的多用户加密搜索方案 **?**，该方案降低了数据持有者的通信开销，但他们的方案不支持用户数据更新。据我们目前所知，现有的可验证对称加密搜索方案都只支持单用户场景下的结果验证，而不支持多用户场景下的结果验证，因为多用户场景下的结果验证会面临更多的挑战。例如，当数据在不同用户之间共享时，由于数据搜索用户无法探知数据持有者是否对数据集合进行了更新，因此一个恶意的服务器可以返回旧数据集合的搜索结果。除非数据持有者在每次更新时都通知所有的数据搜索用户，但这将会带来很大的通信开销。我们将在第 **??** 章具体讲述我们的多用户方案。

1.3　本文的主要内容

本文首先对目前可验证对称加密搜索方案中存在的问题进行了正式的定义，对论文需要实现的安全性目标和性能目标进行了明确说明，并对本文需要用到的先验知识进行了简要阐述。

随后，本文基于 MPT 和增量哈希 (Incremental Hash, IH) 技术，提出了一种单用户场景下的通用可验证对称加密搜索方案，解决了当前可验证对称加密搜索中常见的弊病。该方案将验证索引从对称加密搜索方案中解耦，使其可以与任何对称加密搜索方案结合，包括但不限于论文 **???** 中的方案，甚至包括公钥加密搜索方案。该验证索引基于支持动态更新的数据结构 MPT 构建，因此支持用户动态更新其数据集，而不需要重新构建验证索引。该验证索引将加密后的关键字和其对应

的文件存储于叶子节点中，从而使得 MPT 的根节点成为用户数据完整性的见证，用于后续支持结果验证。同时，本文还提出了基于该验证索引的一系列验证机制，来确保数据完整性和数据新鲜性的验证。方案要求云服务器在更新验证索引或搜索验证索引后都需要向搜索用户返回“证明”，更新时返回的“证明”确保了数据的新鲜性，搜索时返回的“证明”确保了数据的完整性。与以往的方案不同 **???**，我们的方案要求服务器不管搜索关键字存在与否，都需要给用户返回一个“证明”，用于让用户验证服务器是故意返回了空结果还是搜索关键字的确不存在与现有数据集中。此外，我们的方案不需要用户在本地维护文件集对应的关键字集合。

此外，基于以上方案，本文利用时间戳链 (Timestamp Chain) 和公钥加密机制，首次提出了一种多用户场景下的通用可验证加密搜索方案。该方案基于单用户方案构建而成，利用了验证索引中的根哈希。数据的持有者需要将根哈希和时间戳绑定加密并进行签名，以此生成一个鉴别符并上传给云服务器。数据搜索用户在搜索时会从云服务器端获取到该鉴别符，并通过解密得到鉴别符中包含的根哈希和时间戳。数据搜索用户通过鉴别符中的时间戳来判断收到的根哈希是否为最新鲜的根哈希，如果验证通过，数据搜索用户即可采用单用户方案中的验证算法来对数据完整性进行验证。该方案通过时间戳链和公钥加密机制，解决了多用户共享数据情况下的数据新鲜性验证问题，实现了多用户下的结果验证。

最后，本文针对以上的两个方案都进行了严格的安全性分析，证明了方案不泄露用户的数据隐私信息。另外，本文通过实验表明，单用户场景和多用户场景下的可验证加密搜索方案效率很高，与加密搜索法方案结合时，给加密搜索引入的额外开销很小。

1.4　本文的结构安排

本文的结构如下，第 **??** 章为绪论，介绍了加密搜索的研究背景、选题意义，并介绍了可验证对称加密搜索方案凡人研究现状以及本文的主要工作内容；第 **??** 章为问题定义及先验知识，对本文的攻击模型，需要解决的问题和预计实现的目标进行了正式定义与描述，并对本文用到的相关概念和先验知识进行了介绍；第 **??** 章为单用户下的可验证对称加密搜索方案研究，首先介绍了单用户场景下的系统框架，对该场景下涉及到的参与方和其分别需要执行的算法进行了明确的定义。随后对单用户场景下方案的整体工作流程进行了精确定义，并对定义中包含的每一个算法进行了详细的阐述和分析，利用一个简单的实例对算法的工作流程进行了梳理，最后通过安全性证明和实验验证，证明了单用户场景下的可验证对称加密搜索方案达到了第 **??** 章提出的安全性要求和性能要求；第 **??** 章为多用户下的可

验证对称加密搜索方案研究，整体结构与第 **??** 章类似，首先对多用户下数据持有者和数据搜索用户产生分离的系统框架进行了说明，并通过一个精确定义对多用户下的每一个参与方需要执行的算法进行了定义，随后对其算法进行了详细阐述。由于多用户场景下的方案基于第 **??** 章中的单用户方案实现，因此对于第 **??** 章中已阐述过的方案，这里不再赘述。最后，同样通过安全性证明和实验验证证明了多用户场景下方案的安全性和高效性；第 **??** 章总结了全文，对本文做出的主要贡献进行了总结，并对可验证加密搜索领域未来可能的发展方向进行了分析。

第 2 章　问题描述与背景知识

本章将首先对论文的攻击模型，论文需要解决的问题和论文预计达到的目标进行定义，然后对论文中需要用到的先验知识进行介绍。

2.1　问题定义

在本节中，我们将正式定义方案的攻击模型，方案需要解决的问题以及方案需要实现的目标。

2.1.1　攻击模型

在单用户场景中，数据持有者和数据搜索用户是同一人，而在多用户场景中，这两者是分开的。我们假定数据持有者本身和数据搜索用户是可信的，而不可信，而提供存储和搜索服务的云服务器是不可信的，即 1) 云服务器会试图从用户的加密数据和搜索请求中推断出一些隐私信息; 2) 云服务器有可能会因为外部攻击、配置错误、软件错误等原因背离原有协议，从而导致产生数据新鲜性攻击和数据完整性攻击，用以节省其自身的计算开销和通信开销。数据新鲜性攻击和数据完整性攻击的正式定义如下：

定义 2.1 (数据新鲜性攻击)： *在对称加密搜索中，数据新鲜性攻击是指一个恶意的云服务器试图从旧数据集中返回搜索结果，而不从最新的数据集中返回搜索结果。正式地，让 $\Delta_{n-1} = \{\delta_1, \delta_2, \cdots, \delta_{n-1}\}$ 代表用户数据集的历史版本，δ_n 代表用户的最新数据集，云服务器返回的搜索结果为 δ_i 的子集，其中 $1 \leq i \leq n-1$。*

定义 2.2 (数据完整性攻击)： *在对称加密搜索中，数据完整性攻击是指一个恶意的云服务器试图篡改搜索结果，阻止数据搜索用户获取到完整的搜索结果。正式地，让 τ 代表对称加密搜索方案中的搜索令牌，δ_i 代表数据集，其中 $1 \leq i \leq n$。对应的搜索结果应为 $\mathcal{F}(\delta_i, \tau)$，但云服务器返回的搜索结果为 $\mathcal{G}(\delta_i, \tau)$，其中 $\mathcal{G}(\delta_i, \tau) \neq \mathcal{F}(\delta_i, \tau)$。*

2.1.2　设计目标

本论文旨在设计一种通用的可验证加密搜索方案，即该方案可以和任意加密搜索方案相结合，甚至包括公钥加密搜索方案，使其能够完成结果验证的功能。本方案将现有的加密搜索方案当做黑盒，总体来说，需要满足以下几个需求：

1. **机密性：**数据和关键字的机密性是加密搜索最基本的安全需求。它保证了用户的明文数据和关键字信息无法被其他不可信第三方所得到。并且保证了敌手无法从方案的加密数据集，验证索引以及搜索关键字中推断出任何有用的隐私信息。
2. **可验证性：**一个可验证的对称加密搜索方案应该能够验证搜索结果的正确性和完整性，即防止数据新鲜性攻击和数据完整性攻击。
3. **高效性：**一个可验证对称加密搜索方案应该达到次线性的计算复杂度，即对数复杂度 $O(log(|W|))$，其中 $|W|$ 是关键字的总数，并且应该在支持用户数据更新的情况下仍然能达到该复杂度。注意，这里的计算复杂度仅仅指服务器提供结果验证服务时所需的额外计算复杂度，不包括加密搜索方案本身带来的计算复杂度。

2.2　背景知识

2.2.1　增量哈希

增量哈希 (Incremental Hash, IH) 由 Bellare 等人提出 **?**，并被已有的加密搜索方案 **?** 所使用。增量哈希函数是一个抗碰撞的函数：$IH : \{0,1\}^* \rightarrow \{0,1\}^l$，两个随机字符串通过增量哈希函数相加或相减后，生成的哈希值不会产生碰撞。举例来说，假设 D_w 是一个包含关键字 w 的数据集合，它的增量哈希值为 H。当一个新数据 d 加入到 D_w 中后，新的数据集合变为 D'_w，即 $D_w + d$。对于原有数据集 D_w 来说，数据 d 的加入只是微小的变动。增量哈希函数可以基于数据 d 和现有哈希值 H，并通过“加法”操作快速的计算出新文件集 D'_w 的一个抗碰撞哈希值，而不需要基于新文件集 D'_w 重新计算哈希值，这使得哈希操作的性能得到了较大的提升。

2.2.2　默克尔帕特里夏树

默克尔帕特里夏树，即 MPT，最早在以太坊 **??** (Ethereum) 中提出，它将传统的字典树 (Trie Tree) 和默克尔树结合，使得该树同时具有查找和验证的功能。MPT 具有三种类型的节点，分别为叶子节点（Leaf Node，LN），分支节点（Branch Node，BN）和扩展节点（Extension Node，EN）。其中叶子节点存储了键值对，扩展节点也存储了键值对，但扩展节点的“键值”分别为其子节点的公共前缀和子节点的哈希值。分支节点有 17 个元素，其中前 16 个元素代表了该节点上有可能的分支，即 16 个十六进制数字，第 17 个元素为值。当某一个“键”在该分支节点匹配完成时，该“键”对应的“值”就存储该元素中。

图 **??** 通过四个简单的例子展示了 MPT 的插入过程。首先是将一个“键值对”

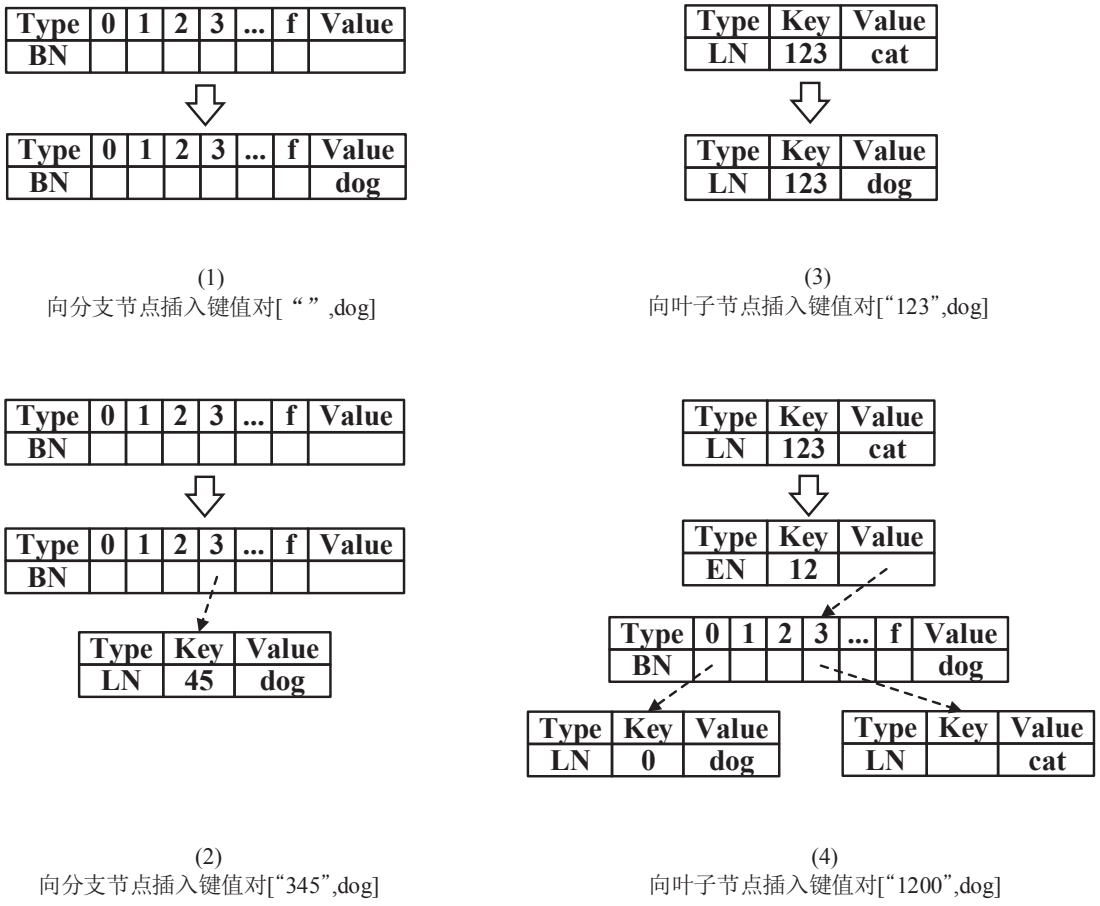

图 2.1　MPT 的更新过程

插入到分支节点的情况，有两种可能。如果当前的键空间已经为空，我们可以直接将“值”插入到分支节点的第 17 个位置，如图**??**(1) 所示。否则，在经过了分支节点匹配后，键空间中“剩余键”和“值”将会存储在分支节点指向的一个新的叶子节点中，如图**??**(2) 所示。其次是将“键值对”插入到叶子节点的情况，也有两种可能。如果当前键空间中“剩余键”与叶子节点中的“键”正好匹配，直接将叶子节点中的“值”修改为新的“值”即可，如图**??**(3) 所示。否则，我们将找到当前键空间“剩余键”和叶子节点“键”的共同前缀，将其作为一个新建的扩展节点的“键”，并新建一个分支节点，指向该扩展节点。我们将匹配完扩展节点和分支节点后的“剩余键”作为一个新建叶子节点的键，并将原有的叶子节点和这个新建的叶子节点作为子节点插入到分支节点对应的元素空间中，如图**??**(4) 所示。

注意，MPT 中的每一个节点都通过可递归长度前缀法 **?**(Recursive Length Prefix, RLP) 进行了编码并对编码值再进行了哈希。数据库中存储了每个节点的“键值对”，其中“键”为该节点 RLP 编码的哈希，“值”为该节点的 RLP 编码。这样每个节点

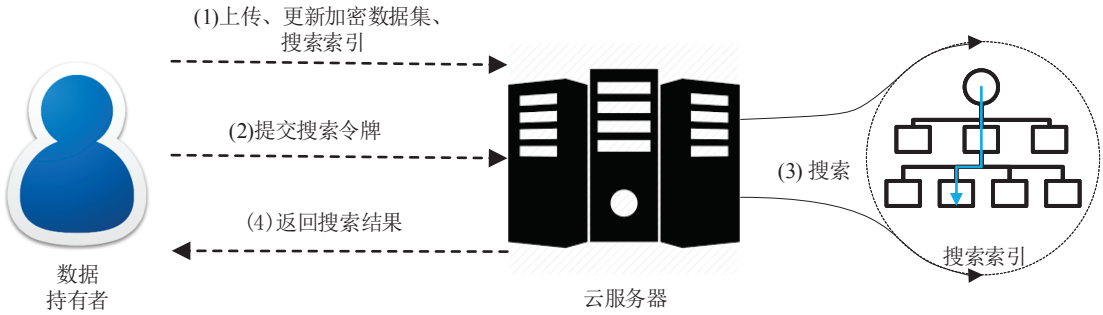

图 2.2　对称加密搜索方案的框架

可以通过他的哈希值被引用，同时保证了 MPT 的可搜索性和可验证性。通过这种方式，MPT 的根哈希成为了整棵树的指纹信息，根哈希的值由所有下层节点的哈希值所决定，任何节点的微小改变都会导致根哈希的值发生变化。此外，MPT 与默克尔树不同，MPT 是完全确定性的，即一组相同的“键值对”采用不同的顺序插入到 MPT 中，最终得到的根节点哈希值是相同的，而默克尔树不具有这个性质。

2.2.3　对称加密搜索

云存储的出现使得用户可以在云端对数据进行备份，并且节省了用户本地存储的开销。因此越来越多的用户选择将他们的数据上传给云服务器。但由于云服务存在诸多安全风险，数据应该加密后再上传，但这使得用户失去了对其数据的搜索能力。一个简单的解决方案是，用户将密文数据全部下载下来以后，对其进行解密，然后在明文数据上进行搜索。但这种解决方案需要用户下载整个数据库，带来的通信开销和存储开销很大，在许多设备上势必是不可能实现的，例如移动设备由于流量和存储空间都有限，就不可能采用这种简单的方案。加密搜索的提出将这种搜索功能转移到了云端，并且同时不造成明文数据泄露。目前，大部分的对称加密搜索方案都采用了建立一个加密索引 γ 的方式来解决此问题。该加密索引根据用户数据集 $\mathcal{D}$ 来构建。用户首先需要从该数据集中提取出所有的关键字集合 $\mathcal{W}$，并将 $\mathcal{W}$ 中的每一个关键字 w_i 与包含该关键字的文件 D_w 相关联，其中 $i \in \{1, \cdots, |W|\}$。最后用户通过加密每一个 (w_i, D_w) 对，建立搜索索引 γ，而加密后的关键字 w_i 将会被用户用于后续搜索。一个对称加密搜索方案的常见框架如图 **??**所示。用户需要对自身持有的明文数据集进行加密并上传到云端，与此同时，用户还需要根据该数据集生成一个加密索引。该加密索引保证了云服务器可以在不解密加密索引和密文数据集的情况下，对用户的密文数据进行搜索。当用户需

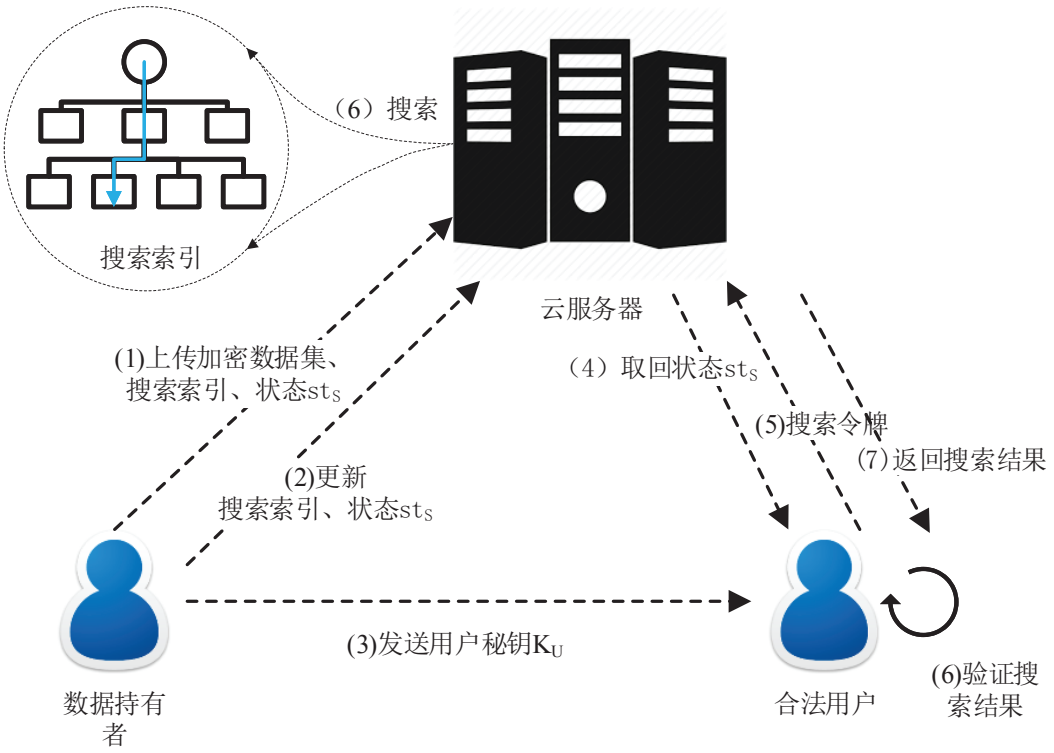

图 2.3　多用户下的对称加密搜索方案的框架

要搜索数据时，他将生成一个令牌，该令牌与搜索关键字相关，使得用户可以在不暴露关键字内容的情况下进行内容搜索。

2.2.4　多用户对称加密搜索

根据 Bosch 等人提出的综述 **?**，现有的加密搜索方案可以分为四种类别：单方写入/单方读取 (single writer/single reader, S/S)，单方写入/多方读取 (single writer/multi-reader, S/M)，多方写入/单方读取 (multi- writer/single reader, M/S) 以及多方写入/多方读取 (multi-writer/multi-reader, M/M)。在本文中我们主要针对单方写入/多方读取这种多用户场景，因为该场景多用于个人用户的存储与数据共享场景中，而多方写入/单方读取的场景多用于邮件数据存储，多方写入/多方读取场景主要用户企业级云存储方案中。

本文并不致力于研究一种访问控制方案来实现多用户的加密搜索方案，因为针对用户的访问控制问题目前已有许多研究，例如基于角色的访问控制策略 (Role-based Access Control Policies) **??** 等等细粒度的访问控制方案。数据持有者可以根据每个用户的不同职责来为其分配不同的访问权限，每个角色对应于一种访问权

限，例如只读等等。本方案可以为实现了这种数据访问控制的多用户加密搜索方案提供结果验证功能。目前已有相关研究提出了这种支持单方写入/多方读取的多用户加密搜索方案 **???**，例如广播加密方案 (Broadcase Encryption, BE) 和基于属性加密搜索方案 (Attribute-Based Encryption, ABE) 等等。

本文以广播加密方案 **?** 为例，阐述数据持有者如何对数据搜索用户进行访问控制。如图 **??**所示，数据持有者通过私钥 K_O 和用户身份信息生成用户私钥 K_U，他将用户私钥 K_U 发送给该用户 U，并将一个状态 st_S 更新给云服务器，其中 st_S 由数据持有者的合法用户群 G 和一个对称秘钥 r 计算得到。为了搜索一个关键字 w，用户需要从服务器上获取状态 st_S，并通过用户私钥 K_U 解密出其中包含的对称秘钥 r，随后用户通过计算 $\Phi_r(\tau_w)$ 并上传给服务器来进行访问控制验证。云服务器在收到 $\Phi_r(\tau_w)$，可以通过计算 $\tau_w = \Phi_r^{-1}(\Phi_r(\tau_w))$ 提取出其中包含的搜索令牌 τ_w。注意，如果数据持有者需要剔除一个用户 U，他仅需要重新选择一个对称秘钥 r'，然后根据新的用户群 $G' = G \backslash U$ 重新生成一个状态 st_S' 即可。只要云服务器和被剔除的用户不存在共谋，一个被剔除的用户不再属于用户群 G'，因此他将无法从从状态 st_S' 中解密出对称秘钥 r'，因此也无法生成一个合法的搜索令牌。通过这种广播加密方案，数据持有者可以方便的控制他的合法搜索用户，并且不需要再每次剔除一个用户时重新生成验证索引。

第 3 章　单用户下的可验证对称加密搜索方案研究

3.1　引言

本章提出了一种通用的可验证对称加密搜索方案 GS-VSSE，该方案可以在单用户场景下工作，与任意对称加密搜索方案结合后，可以为用户提供加密搜索的结果验证服务。本章的主要内容如下：首先介绍了单用户场景下 (即数据持有者为数据搜索用户本身) 的系统架构，介绍了该框架的参与方及其所承担的计算任务；接着，通过一个正式定义从抽象层面介绍了该框架工作的流程和每个参与方涉及到的算法。随后，对 GS-VSSE 方案涉及到的算法进行了详细分析，包括数据持有者构建和更新验证索引的算法，云服务器搜索验证索引并生成结果证明的算法，数据持有者进行结果验证的算法。随后通过一个简单的例子对这几个算法进行了详细的阐述。最后，通过安全性分析和实验结果分析证明 GS-VSSE 方案可以达到设计目标中的安全性要求和性能要求。

3.2　系统架构

单用户场景下的可验证对称加密搜索方案系统架构如图 **??**所示，数据持有者即为数据搜索用户本身。初始化时需要数据持有者对自身的数据集进行加密，并对该数据集构建加密的验证索引，用于后续结果验证。数据持有者将该验证索引上传给云服务器存储，并在需要时更新验证索引。当用户需要进行关键字搜索时，他将会构建出一个与关键字相关的搜索令牌，提交给云服务器进行搜索。云服务器接收到该搜索令牌后，通过某个加密搜索方案取得加密搜索结果，同时通过搜索验证索引取得一个针对于该搜索结果的结果证明，最后云服务器将结果证明返回给用户。用户在收到该结果证明和加密搜索方案提供的搜索结果后，对搜索结果进行结果验证，若验证失败，则丢弃该结果。

3.3　方案流程

由于 GS-VSSE 方案的主要目标是为对称加密搜索方案提供结果验证功能，因此在开始讲述 GS-VSSE 方案的流程前，我们先回顾对称加密搜索方案的常见定义。

定义 3.1 (SSE 方案)： 一个支持文件更新的 *SSE* 方案，参与方包括两个，分别为数据持有者本身和半可信的云服务器。数据持有者向云服务器提供加密数据集和

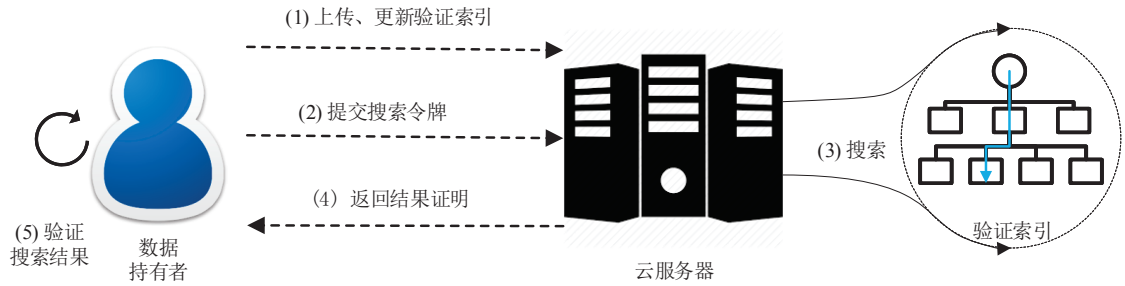

图 3.1　单用户场景下的可验证对称加密搜索框架 GS-VSSE

搜索索引，使得云服务器可以在密文上进行搜索，从而同时确保数据的可搜索性和隐私性。一个 *SSE* 方案可以定义为如下六个算法的集合：

- $KGen_{SSE}(1^k) \rightarrow \{\mathcal{K}\}$: 是由数据持有者执行的秘钥生成算法。它将一个安全参数作为输入，输出一系列对称秘钥 $\mathcal{K}$。
- $Init_{SSE}(\mathcal{K}, \mathcal{D}) \rightarrow \{\gamma, C\}$: 是由数据持有者执行的初始化算法。它将对称秘钥 $\mathcal{K}$ 和明文文件集 $\mathcal{D}$ 作为输入，输出搜索索引 γ 和密文文件集 C。数据持有者将搜索索引 γ 和密文文件集 C 上传给云服务器。
- $UpdateToken_{SSE}(\mathcal{K}, d) \rightarrow \{\tau_u\}$: 是由数据持有者执行的更新令牌生成算法。它将对称秘钥 $\mathcal{K}$ 和需要更新的文件 d 作为输入，输出一系列更新令牌 τ_u。数据持有者将更新令牌 τ_u 上传给云服务器。
- $Update_{SSE}(\gamma, \tau_u) \rightarrow \{\gamma'\}$: 是由云服务器执行的更新算法。它将搜索索引 γ 和更新令牌 τ_u 作为输入，输出更新后的搜索索引 γ'。
- $SearchToken_{SSE}(\mathcal{K}, w) \rightarrow \{\tau_w\}$: 是由数据持有者执行的搜索令牌生成算法。它将对称秘钥 $\mathcal{K}$ 和某一关键字 w 作为输入，输出与该关键字相关搜索令牌 τ_w。数据持有者将该搜索令牌 τ_w 上传给云服务器进行搜索。
- $Search_{SSE}(\gamma, \tau_w) \rightarrow \{C_w\}$: 是由云服务器执行的搜索算法。它将搜索索引 γ 和搜索令牌 τ_w 作为输入，输出搜索结果 C_w。云服务器将搜索结果 C_w 返回给数据持有者。

如定义**??**所示，*SSE* 算法对云服务器的假定为半可信的，即云服务器不会违背协议本身。因此在 *SSE* 方案中，云服务器在通过 *Search* 算法将搜索结果返回给用户后，用户无条件的接受该结果作为正确结果。然而在现实情况中，这种对云服务器半可信的假定往往是不成立的，云服务器往往会因为一些外部攻击，甚至配置错误、软件错误等原因背离原有协议，产生恶意行为。因此我们需要设计一

个方案来对搜索结果进行验证，以确保云服务器没有恶意行为。GS-VSSE 方案即在上述 *SSE* 的基础上，为其提供了一套结果验证机制。该机制可以与 *SSE* 方案中的每一个算法并行操作，在为其进行结果验证的同时，保证了整体方案的高效性。GS-VSSE 方案的具体定义如下。

定义 3.2 (GS-VSSE 方案)： *在 GS-VSSE 方案中，参与方有两个，分别为数据持有者本身和不可信的云服务器。数据持有者向云服务器提供验证索引，使得云服务器在用户搜索时可以向其返回结果证明，用于确保加密搜索结果的新鲜性和完整性。一个 GS-VSSE 方案是以下八个算法的集合:*

- $KGen(1^k) \rightarrow \{K_1, K_2\}$*: 是由数据持有者执行的秘钥生成算法。它将一个安全参数作为输入，输出对称秘钥* K_1, K_2。
- $Init(K_1, K_2, \mathcal{D}) \rightarrow \{\lambda\}$*: 是由数据持有者执行的初始化算法。它将对称秘钥* K_1, K_2 *和明文文件集* $\mathcal{D}$ *作为输入，输出验证索引* λ。*数据持有者在本地保存验证索引* λ *的根节点哈希* rt，*并将验证索引* λ *上传给云服务器。*
- $UpdateToken(K_1, K_2, d) \rightarrow \{\tau_u\}$*: 是由数据持有者执行的更新令牌生成算法。它将对称秘钥* K_1, K_2 *和需要更新的文件* d *作为输入，输出一系列更新令牌* τ_u。*数据持有者将更新令牌* τ_u *上传给云服务器。*
- $Update(\lambda, \tau_u) \rightarrow \{\lambda', \rho_u\}$*: 是由云服务器执行的更新算法。它将验证索引* λ *和更新令牌* τ_u *作为输入，输出更新后的验证索引* λ' *和更新证明* ρ_u。*云服务器将更新证明* ρ_u *返回给用户。*
- $VerifyUpdate(rt, \tau_u, \rho_u) \rightarrow \{rt'\}$: *是由数据持有者执行的更新算法。它将验证索引* λ *的根哈希* rt，*更新令牌* τ_u *和服务器返回的更新证明* ρ_u *作为输入，输出新的根哈希* rt'。*若更新证明* ρ_u *验证通过，则输出更新后的根哈希* rt'，*若更新证明验证失败，则输出的根哈希* rt' *与原始根哈希* rt *相同。*
- $SearchToken(K_1, w) \rightarrow \{\tau_w\}$*: 是由数据持有者执行的搜索令牌生成算法。它将对称秘钥* K_1 *和某一关键字* w *作为输入，输出与该关键字相关搜索令牌* τ_w。*数据持有者将该搜索令牌* τ_w *上传给云服务器进行搜索。*
- $Prove(\lambda, \tau_w) \rightarrow \{\rho_s\}$*: 是由云服务器执行的结果证明生成算法。它将验证索引* λ *和搜索令牌* τ_w *作为输入，输出结果证明* ρ。*云服务器将结果证明* ρ *返回给数据持有者。*
- $Verify(K_1, K_2, C_w, \rho_s, \tau_w, rt) \rightarrow \{b\}$*: 是由数据持有者执行的验证算法。它将对称秘钥* K_1, K_2，*加密搜索结果* C_w，*结果证明* ρ_s，*搜索令牌* τ_w *和保留的验证索引根哈希* rt *作为输入，输出一个比特* b，*代表接受或者拒绝该搜索结果。*

注意，上述流程中的每一个算法 (除了 $VerifyUpdate$，$Verify$ 算法)，都与加密搜

索流程中的算法一一对应。例如 $KGen$，$Init$，$UpdateToken$，$Update$，$SearchToken$ 算法都可以与加密搜索中的秘钥生成，初始化，更新令牌生成，更新操作以及搜索令牌生成同时进行，而 $Prove$ 算法则可以与加密搜索方案中的 $Search$ 搜索操作同时进行。该可验证加密搜索方案带来的额外算法是 $VerifyUpdate$ 和 $Verify$ 算法，它们分别用于用户收到更新结果和搜索结果后的验证操作。正是因为 GS-VSSE 方案的每一个算法都从加密搜索方案中解耦了出来，才使得该方案可以将加密搜索方案当做黑盒，并为任意加密搜索方案提供结果验证服务。

3.4　方案设计

在本节，我们将具体阐述 GS-VSSE 方案，即单用户场景下的可验证加密搜索方案。我们将详细讲解定义**??**中的算法。首先我们将描述如何通过 $Init$ 算法来建立验证索引，并通过 $Update$ 算法和 $VerifyUpdate$ 算法来阐述如何更新验证索引。然后我们将通过 $Prove$ 算法给出服务器生成结果证明的方法，并通过 $Verify$ 算法详细解释用户如何利用结果证明来确保搜索结果的正确性。最后，我们将通过一个简单的例子来详细说明上述算法的执行流程，以便于读者理解。

3.4.1　构建及更新验证索引

算法 1 *Init* 算法

输入： K_1, K_2: 对称秘钥; $\mathcal{D}$: 明文文件集合; $F, G : \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^*$ 伪随机函数; $IH : \{0,1\}^* \to \{0,1\}^k$ 增量哈希函数; $H : \{0,1\}^* \to \{0,1\}^k$ 哈希函数

输出： λ: 通过 MPT 构建的验证索引。

1: **for** each $w_i \in \Delta$, 其中 Δ 是包括了 $< w_i, D_{w_i} >$ 的明文倒排索引, $i \in \{1, \cdots, |W|\}$。**do**
2: 　生成“键”：$\tau_{w_i} = F_{K_1}(w_i)$。
3: 　生成“值”：$V_{w_i} = \sum_{d \in D_{w_i}} IH(G_{K_2}(d))$。
4: 　向 MPT 中插入键值对 (τ_{w_i}, V_{w_i})。
5: **end for**
6: **return** 返回从 MPT 构建得到的验证索引 λ。

算法 **??** 给出了建立验证索引的伪代码。首先数据持有者根据明文文件集 $\mathcal{D}$ 计算出倒排索引 Δ，其中倒排索引 Δ 是指关键字 w_i 与包含该关键字的文件 D_{w_i} 组成的索引。对倒排索引中的每一个关键字 w_i，我们计算他的" 键值对"，其中“键”

是每一个关键字通过伪随机函数生成的令牌，而“值”是包含该关键字的文件的增量哈希和。我们通过将这些“键值对”插入 MPT 中来形成验证索引。

算法 2 *Update* 算法

输入： λ: 验证索引；τ_u：更新令牌；

输出： λ': 更新后的验证索引；ρ_u: 更新证明；

1: 将更新令牌 τ_u 解析为键值对 $(\tau_{w_i}, G_{K_2}(d))$，其中 $i \in \{1, \cdots, |W_d|\}$，$d$ 为待更新的文件。
2: **for** each $\tau_{w_i} \in \tau_u$ **do**
3: 　将 τ_{w_i} 在 λ 中进行路径匹配，将 τ_{w_i} 匹配后剩余的后缀作为 $remain\_key$。
4: 　**if** $remain\_key = \emptyset$，即 $\tau_{w_i} \in \lambda$ **then**
5: 　　将 $IH(G_{K_2}(d))$ 更新到 τ_{w_i} 到对应的叶子节点。
6: 　　将 τ_{w_i} 在 λ 上的搜索路径 σ_{w_i} 添加到更新证明 ρ_u 中。
7: 　**else if** $remain\_key != \emptyset$，即 $\tau_{w_i} \notin \lambda$ **then**
8: 　　根据 MPT 更新规则，将 $(remain\_key, IH(G_{K_2}(d)))$ 作为键值对插入到新建的叶子节点中。
9: 　　将 τ_{w_i} 在 λ 上的搜索路径 σ_{w_i} 添加到更新证明 ρ_u 中。
10: 　**end if**
11: **end for**
12: **return** 返回更新证明 ρ_u。

对验证索引的更新操作支持三种方式，即插入、删除和编辑文件，其中编辑文件相当于删除一个文件后再新增一个文件。对于插入新文件操作，我们首先解析该文件 d，得到该文件包含的关键字集合 W_d，对每一个关键字 $w_i \in W_d$，我们都用伪随机函数生成他的令牌 τ_{w_i}，并将文件的伪随机结果 $G_{K_2}(d)$ 同时上传给云。如算法 **??**所示，云服务器在收到后更新令牌 τ_u 后，首先需要将更新令牌 τ_u 解析为键值对 $(\tau_{w_i}, G_{K_2}(d))$，并通过更新令牌 τ_{w_i} 找到对应的叶子节点，将 $IH(G_{K_2}(d))$ 与原有的叶子节点的值相加。如果该令牌对应的叶子节点尚不存在，则需要为其创建一个新的叶子节点，并将剩余键 $remain\_key$ 和对应的值 $IH(G_{K_2}(d))$ 作为键值对插入其中。删除操作同样，只是将原有的叶子节点的值减去 $IH(G_{K_2}(d))$。此外，云服务器在更新每一个令牌时，都需要将该令牌对应的搜索路径 σ_{w_i} 保存在更新证明 ρ_u 中，用于后续发回给用户进行更新验证。

算法 **??**展示了数据持有者在收到云服务器返回的更新证明 ρ_u 后，执行的更新验证操作，该操作确保了数据的新鲜性。由于数据持有者本身在本地并不保留验

算法 3 $VerifyUpdate$ 算法

输入： rt: 验证索引的根哈希；τ_u：更新令牌；ρ_u: 更新证明；

输出： rt': 更新后的根哈希。

1: 将更新令牌 τ_u 解析为键值对 $(\tau_{w_i}, G_{K_2}(d))$，其中 $i \in \{1, \cdots, |W_d|\}$，$d$ 为待更新的文件。
2: 将更新证明 ρ_u 解析为键值对 σ_{w_i}，其中 $i \in \{1, \cdots, |W_d|\}$。
3: **for** each $\tau_{w_i} \in \tau_u$ **do**
4:　根据更新令牌 τ_{w_i} 和更新证明 σ_{w_i} 重构出根哈希 rt_t。
5:　**if** $rt_t \neq rt$ **then**
6:　　**return** 验证失败，返回原有根哈希 rt。
7:　**end if**
8: **end for**
9: 根据更新令牌 τ_u 和更新证明 ρ_u 重构出根哈希 rt'
10: **return** 更新后的根哈希 rt'。

证索引 λ，只保留验证索引的根哈希 rt，因此在数据产生更新时，如何更新该根哈希 rt 十分重要。因为云服务器是不可信的，数据持有者在提交了更新令牌 τ_u 后，无法确保服务器执行了正确的更新操作，因此他需要云服务器返回更新证明 ρ_u 来进行验证，并且根据更新证明来更新根哈希 rt。服务器返回的更新证明 ρ_u 包含了更新令牌中每一个关键字令牌 τ_{w_i} 对应在验证索引 λ 上的路径 σ_{w_i}。用户在接受到该更新证明后，首先将自身生成的更新令牌 τ_u 解析为 $(\tau_{w_i}, G_{K_2}(d))$，将更新证明 ρ_u 解析为键值对 σ_{w_i}。随后对每一个令牌 τ_{w_i}，验证是否能根据 σ_{w_i} 生成原始根哈希 rt。若每个令牌都能验证成功，则用户通过更新令牌 τ_u 和更新验证 ρ_u 构建新的根哈希 rt'，否则验证失败，用户保留原有根哈希 rt。在本章第**??**节，我们将通过一个例子来说明建立和更新验证索引的过程。

3.4.2　生成结果证明

如算法 **??**所示，服务器根据用户提交的搜索令牌 τ_{w_i} 和验证索引 λ 来生成结果证明 ρ_s。首先服务器根据搜索令牌 τ_{w_i} 来寻找搜索路径 σ_{w_i}。如果搜索令牌 τ_{w_i} 对应的叶子节点存在，即用户查询的关键字存在，则服务器从叶子节点的上一层节点开始，返回搜索路径上的“键”作为结果证明。注意对于分支节点，服务器还需要返回不在搜索路径上的“键值对”。如果搜索令牌 τ_{w_i} 对应的叶子节点不存在，即用户查询的关键字不存在，则服务器需要从搜索终结的节点开始自底向上返回搜索路径中的“键”作为结果证明，而对于搜索的终结节点，服务器需要返回完整

的键值对。我们将在本章第**??**节，通过一个具体的例子来说明该过程。

算法 4 *Prove* 算法

输入: λ: 云服务器维护的验证索引；τ_{w_i}: 用户提交的搜索令牌；

输出: ρ_s: 搜索结果的结果证明；

查找搜索令牌 τ_{w_i} 在验证索引 λ 上的对应路径 $\sigma_{w_i} = (n_0, \cdots, n_i, \cdots, n_m)$，直到匹配到叶子节点或者匹配失败，其中 $n_i \in \{EN, BN, LN\}$, n_0 为根节点。
if t_{w_i} 在验证索引 λ 中存在 **then**
    for $i = m - 1$ to 0 **do**
        if $n_i = BN$ **then**
            $\rho_s = \rho_s \cup C_{n_i}$，其中 C_{n_i} 包括分支节点中在搜索路径 σ 上的" 键" 和不在搜索路径上的“键值对”。
        else if $n_i = EN$ **then**
            $\rho_s = \rho_s \cup C_{n_i}$，其中 C_{n_i} 包括扩展节点中的“键”。
        else
            $\rho_s = \rho_s \cup C_{n_i}$，其中 C_{n_i} 包括叶子节点中的“键值对”。
        end if
    end for
else
    for $i = m$ to 0 **do**
        重复步骤 4-10
    end for
end if
return ρ_s

3.4.3　结果验证

如算法 **??** 所示，当用户收到了结果证明 ρ_s 时，就可以开始验证数据的完整性。首先用户通过搜索令牌 τ_{w_i} 与结果证明 ρ_s 中的“键”进行匹配。如果结果证明 ρ_s 中的“键”是搜索令牌 τ_{w_i} 的前缀，则 $remain\_key$ 存储搜索令牌 τ_{w_i} 与结果证明匹配完后剩余的键。如果结果证明 ρ_s 中的“键”不是搜索令牌 τ_{w_i} 的前缀，那么 $remain\_key$ 就置为 $\emptyset$。如果搜索结果 C_w 和 $remain\_key$ 都为空集，即云服务器返回了空搜索结果，则我们通过结果证明 ρ_s 直接计算出根哈希值 rt_t。如果二者都不为空，则我们首先通过搜索结果 C_w 和 $remain\_key$ 生成叶子节点的哈希值，再

算法 5 Verify

输入： K_1, K_2: 对称秘钥; C_w: 加密搜索结果; ρ_s: 加密搜索结果证明; τ_w: 用户提交的搜索令牌;rt：用户本身保留的根哈希；

输出： $b \in \{0,1\}$, 如果 $b = 1$, 表示结果验证成功，否则表示结果验证失败；

1: 将 τ_{w_i} 与结果证明 ρ_s 中的键进行匹配，得到剩余键 $remain\_key$。
2: 如果 ρ_s 中的某条路径是 τ_{w_i} 的前缀，则 $remain\_key$ 为 τ_{w_i} 匹配后的剩余键；否则 $remain\_key = \emptyset$。
3: **if** $C_w = \emptyset$ && $remain\_key = \emptyset$ **then**
4:　　根据结果证明 ρ_s 自底向上计算根哈希 rt_t；
5: **else if** $C_w \neq \emptyset$ && $remain\_key \neq \emptyset$ **then**
6:　　计算 $\varphi = \sum_{d \in D_w} IH(G_{K_2}(d))$, 其中 D_w 是 C_w 对应的明文信息；
7:　　计算叶子节点 $LN = Compute(\varphi, remain\_key)$
8:　　根据结果证明 ρ 和叶子节点 LN 自底向上计算根哈希 rt_t.
9: **else**
10:　　**return** 0
11: **end if**
12: **if** $rt = rt_t$ **then**
13:　　**return** 1
14: **else**
15:　　**return** 0
16: **end if**

通过结果证明 ρ_s 重建出根哈希值 rt_t。除了这两种情况以外，我们就认为服务器故意返回了空结果或服务器篡改了结果证明的内容。最后，用户通过对比重建得到的根哈希 rt_t 和用户本身保留的根哈希 rt 是否相等来判断数据完整性。如果二者相等，则验证通过，如果二者不相等，则说明服务器少返回了搜索结果或者服务器篡改了结果证明。

3.4.4　实例分析

如图 **??**，图**??**，图**??**和图**??**所示，我们将通过一个解释性的实例来说明建立和更新验证索引 λ，生成更新证明 ρ_u 和验证更新，以及生成结果证明 ρ_s 和验证搜索结果的过程。

建立并更新验证索引：首先，我们假设数据持有者拥有四个文件，分别为 d_1, d_2, d_3, d_4，他们包含了四个关键字 w_1, w_2, w_3, w_4，其对应关系如图**??**中的倒排索

The Inverted List

keyword	file	token	value
w_1	d_1,d_2,d_3,d_4	'43779'	H_1
w_2	d_2,d_5	'a5432'	$H_2+IH(G_{K_2}(d_5))$
w_3	d_1,d_2,d_3	'a5cc1'	H_3
w_4	d_1,d_2,d_4	'ff48e'	H_4
w_5	d_5	'a5fab'	$H_5=IH(G_{K_2}(d_5))$

图 3.2　一个简单的倒排索引

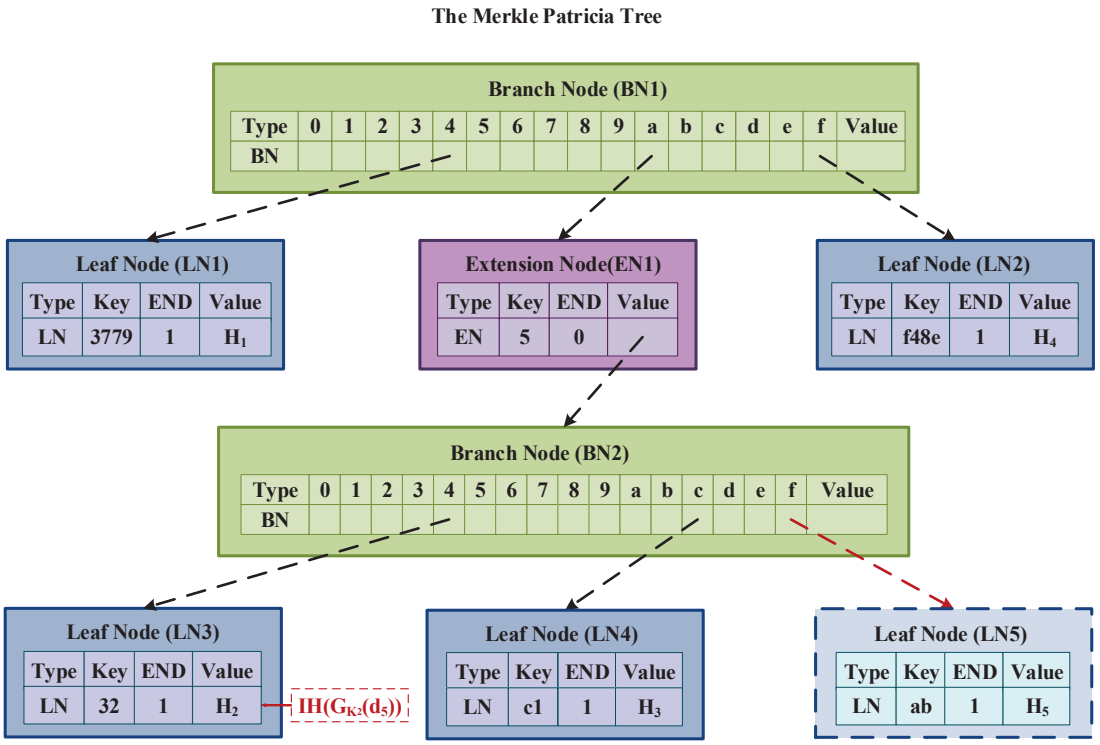

图 3.3　由倒排索引和 MPT 构建的验证索引

引所示。由这四个文件构建的验证索引如图**??**所示。当包含关键字 w_2 和 w_5 的文件 d_5 新增时，对于已经存在的关键字 w_2，云服务器只需将 $IH(G_{K_2}(d_5))$ 添加到原有的叶子节点上。而对于不存在的关键字 w_5，云服务器则需要创建一个新的叶子节点，并将 $IH(G_{K_2}(D_5))$ 作为他的节点值。

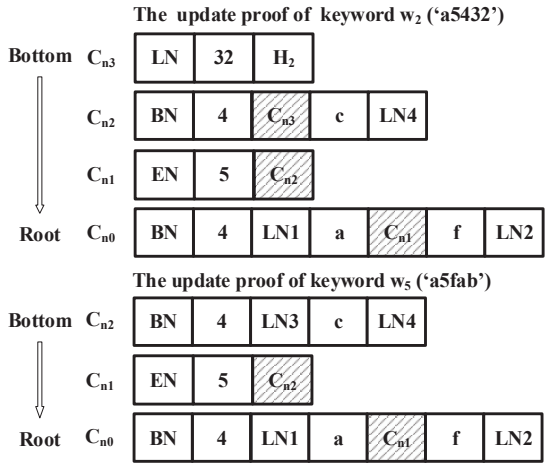

图 3.4　更新证明

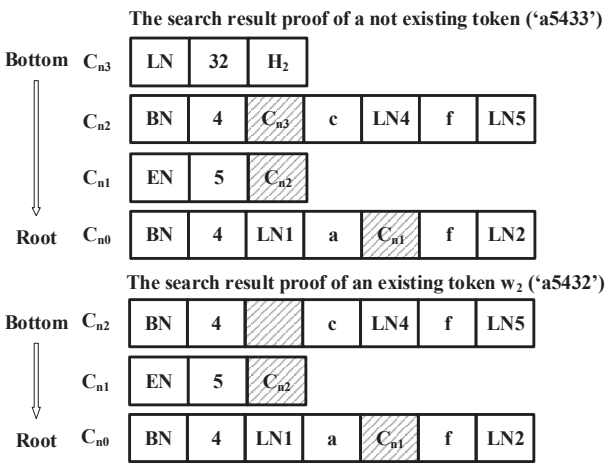

图 3.5　结果证明

生成更新证明和验证更新：云服务器需要将两个涉及到更新的关键字 w_2,w_5 对应的路径返回给用户，用以作为更新证明 ρ_u。如图 **??**所示，当用户拿到该更新证明后，对于每一个待更新关键字，首先需要确保该关键字对应的令牌或其前缀出现在更新证明中。例如，对于关键字 w_2，它对应的更新令牌为'a5432'，在其更新证明 $\sigma_{w_2} = \{C_n3, C_n2, C_n1, C_n0\}$ 中，我们可以自底向上找到一条路径与该令牌相同。而对于关键字 w_5，它对应的更新令牌'a5fab' 并没有出现在更新证明 $\sigma_{w_5} = \{C_n2, C_n1, C_n0\}$ 中，但他的前缀'a5' 出现在了更新证明中，这说明该关键字在验证索引 λ 中尚不存在。用户需要首先根据更新证明重构出根节点哈希，用于跟用户持有的根哈希进行对比。只有当每一个待更新关键字对应的更新证明重构出的根哈希与原根哈希相同时，更新验证才通过。验证通过后，用户通过更新令牌 τ_u 和更新证明 ρ_u 构造出更新后的根哈希 rt'。对于已存在于 λ 中的关键字 w_2，用户只需要将其对应的哈希值 H_2 更新到 $H_2 + IH(G_{K_2(d_5)})$，而对于尚不存在于验证索引 λ 中的关键字 w_5，用户需要根据更新证明为其新建叶子节点，即在 C_{n2} 这个分支节点中，添加一个新的分支 f，使其指向一个新的叶子节点 $('ab', IH(G_{K_2(d_5)}))$。最后用户通过联合两个更新证明更新验证索引的根哈希 rt'，从而确保后续结果验证时的数据新鲜性。

生成结果证明和验证搜索结果：结果证明 ρ_s 的生成可以分为两种情况来讨论。第一种情况，假设用户想要搜索的关键字为 w_2，他提交的对应该关键字的挑战令牌为“a5432”。如图 **??**所示，由于该关键字令牌在验证索引 λ 中已经存在，云服务器可以找到与该令牌对应的搜索路径 BN1,EN1,BN2,LN3，根据 *Prove* 算法，服务器会自底向上返回除 $LN3$ 以外的搜索路径上的“键值对”作为结果证明，如 C_{n2}, C_{n1}, C_{n0} 所示。用户在收到结果证明 C_{n2}, C_{n1}, C_{n0} 以后，可以根据该证明 ρ_s 和

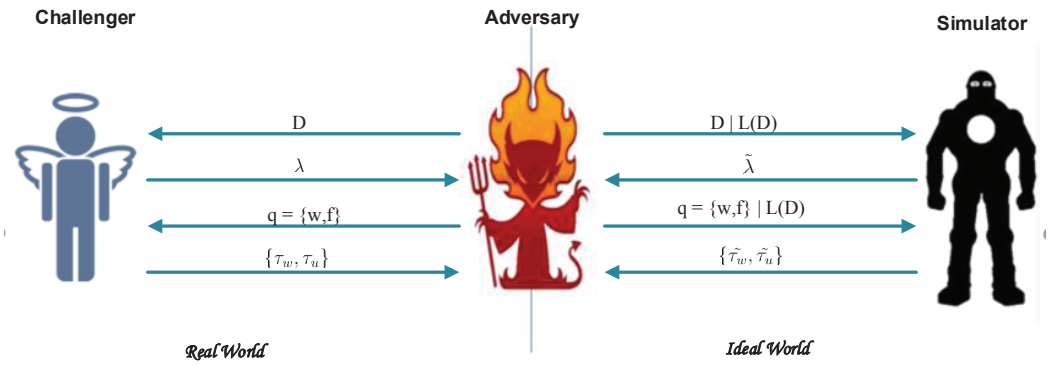

图 3.6　基于仿真博弈的安全性证明方案

搜索结果 C_w 重新构建根哈希。具体过程如下：首先用户将令牌“a5432”与结果证明中 ρ_s 的“键”进行匹配，发现“a54”为令牌“a5432”的前缀，因此剩余键 $remain\_key$ 为“32”。随后用户根据“32”以及搜索结果 d_2, d_5 重新生成叶子节点 $LN3$，并通过结果证明 ρ_s 自底向上构建冲根哈希的值。最后用户通过比较重构得到的根哈希和自身持有的根哈希，来判断数据是否完整。例如，假设云服务器只返回了文件 d_2，那么重构得到的根哈希将与正确的根哈希不匹配。第二种情况：假设用户搜索的关键字令牌为“a5433”，该令牌在验证索引 λ 中不存在。根据云服务器的查找方法，其搜索路径与“a5432”相同，但不同的是，该令牌在叶子节点 $LN3$ 处发生了不匹配。因此云服务器需要从叶子节点 $LN3$ 开始 (包括 $LN3$) 自底向上生成结果证明，如图 **??**中的 C_n3, C_n2, C_n1, C_n0 所示。用户在收到该结果证明以后，由于发现结果证明 ρ_s 中的“键”都不是搜索令牌 τ_s 的前缀，因此 $remain\_key$ 被置空。用户将直接根据结果证明 ρ_s 重构根哈希。同样，用户将其与正确的根哈希进行对比，如果不相同，则说明服务器篡改了搜索结果或是结果证明，产生了恶意行为。

3.5　安全性分析

在本节中，我们将对方案的安全性进行证明。方案的安全性主要分为两个部分，一个是机密性，另一个是可验证性。机密性是指敌手无法从验证索引 λ 和用户发送的令牌 τ 中获取文件和关键字的明文信息。可验证性是指当服务器返回不完整或者不新鲜的结果时，用户不会验证通过。

首先，我们采用基于仿真博弈 (Simulation-based Game) 的安全性方案来证明方案的机密性。该方案的流程解释如图 **??**所示，我们需要确保一个敌手无法区分出

Real World 和 **Ideal** World 的输出。具体的定义如下：

定义 3.3 (GS-VSSE **机密性**)： 令方案 *GS-VSSE* 是一个通用的、支持数据更新的可验证对称加密搜索方案，考虑以下概率性实验，其中 $\mathcal{A}$ 是一个有状态的敌手 *(state-fule adversary)*，$\mathcal{S}$ 是一个有状态的仿真者 *(stateful simulator)*，$\mathcal{L}$ 是有状态的泄露函数 *(leakage algorithms)*:

$\mathbf{Real}_{\mathcal{A}}(k)$: 一个挑战者采用 $KGen(1^k)$ 生成了对称秘钥 K_1, K_2。敌手 $\mathcal{A}$ 选择了一个文件集 $\mathcal{D}$ 让挑战者通过 $\{\lambda\} \leftarrow Init(K_1, K_2, \mathcal{D})$ 算法生成验证索引 λ。同时，敌手 $\mathcal{A}$ 生成了多项式数量级的自适应查询 $q = \{w, f\}$。对于每一个查询 q，敌手 $\mathcal{A}$ 从挑战者处收到一个搜索令牌 τ_w 和更新令牌 τ_u，其中 $\tau_w \leftarrow SearchToken(K_1, w)$，$(\tau_u) \leftarrow UpdateToken(K_1, K_2, f)$。最后，敌手 $\mathcal{A}$ 返回一个比特 b。

$\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$: 一个敌手 $\mathcal{A}$ 选择了一个文件集 $\mathcal{D}$。给定 $\mathcal{L}(\mathcal{D})$，仿真者 $\mathcal{S}$ 生成验证索引 λ 发送给敌手 $\mathcal{A}$。敌手 $\mathcal{A}$ 生成了多项式数量级的自适应查询 $q = \{w, f\}$。对于每一个查询 q，仿真者 $\mathcal{S}$ 向其返回一个恰当的令牌 τ。最后，敌手 $\mathcal{A}$ 返回一个比特 b。

如果对于任何概率多项式时间 *(Probabilistic Polynomial-Time, PPT)* 的敌手 $\mathcal{A}$, 始终存在一个概率多项式时间的仿真者 $\mathcal{S}$，使得，

$$|Pr[\mathbf{Real}_A(k) = 1] - Pr[\mathbf{Ideal}_{A,S}(k) = 1]| \leq negl(k). \tag{3-1}$$

则我们认为 *GS-VSSE* 是 $\mathcal{L}$-机密的。

在具体证明之前，我们首先给出敌手 $\mathcal{A}$ 能看到的内容，即泄露函数 $\mathcal{L}$ 的内容。$\mathcal{L}$ 定义如下：

$$\mathcal{L}(\mathcal{D}) = (|\lambda|, \tau_q, \sigma) \tag{3-2}$$

其中 $|\lambda|$ 表示验证索引的大小，以叶子节点的数目来衡量。$\{\tau_q\}$ 表示由 q 个查询产生的令牌，σ 表示验证索引中的搜索路径。我们有以下的定理：

定理 3.1： 如果 F,G 都是伪随机函数，那么方案 GS-VSSE 就是 $\mathcal{L}$-机密的。

证明 我们将证明，对于任何概率多项式时间内的敌手 $\mathcal{A}$，都存在一个概率多项式时间内的仿真者 $\mathcal{S}$，使得真实游戏 $\mathbf{Real}_{\mathcal{A}}(k)$ 和仿真游戏 $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$ 在计算上是无

法区分的 (Computationally Indistinguishable)，而在现代密码学中，不可区分性等同于机密性。

首先，给定 $\mathcal{L}(\mathcal{D}) = (|\lambda|, \tau_q, \sigma)$，$\mathcal{S}$ 通过选择 $|\lambda|$ 个随机“键值对”插入 MPT 中生成一个仿真的验证索引 $\tilde{\lambda}$。由于在真实的验证索引中，“键值对”都是采用了伪随机函数 F,G 进行了伪随机化的，因此敌手 $\mathcal{A}$ 将无法区分出真实的 λ 和仿真的 $\tilde{\lambda}$。

模拟搜索令牌时，对于挑战者生成的第一个令牌 τ_w ，如果它与 $\{\sigma\}$ 中的某一路径匹配，那么仿真者 $\mathcal{S}$ 就在 $\tilde{\lambda}$ 中选择任意一条路径作为令牌 $\tilde{\tau_w}$ 发送给敌手 $\mathcal{A}$，否则 $\mathcal{S}$ 就选择不在 $\tilde{\lambda}$ 路径中的随机字符串作为令牌 $\tilde{\tau_w}$ 发送给敌手 $\mathcal{A}$。对于后续的令牌，如果 τ_w 之前出现过，那么令牌 $\tilde{\tau_w}$ 就和之前发送给敌手 $\mathcal{A}$ 的维持一样。如果 τ_w 没出现过，那么令牌的生成方式就和第一个令牌的生成方式一样。由于令牌采用了伪随机函数 F 进行了加密，因此敌手 $\mathcal{A}$ 也无法区分真实的令牌和仿真的令牌。

模拟更新令牌时，更新令牌被设置为 $\tilde{\tau_u} = (\tilde{\tau_{w_1}}, \cdots, \tau_{\tilde{w_{|W_d|}}}, \tilde{\tau_d})$。对每一个更新令牌 $\tilde{\tau_{w_i}}$，其中 $i \in \{1, \cdots, |W_d|\}$，$\mathcal{S}$ 的模拟方法与模拟搜索令牌方法相同。由于每一个更新令牌都采用了伪随机函数 F 进行了加密，并且文件 d 采用了伪随机函数 G 进行了加密，因此敌手 $\mathcal{A}$ 无法区分真实的令牌、文件与仿真的令牌、文件。

因此我们可以得到结论：真实实验 $\mathbf{Real}_{\mathcal{A}}(k)$ 和仿真实验 $\mathbf{Ideal}_{\mathcal{A},\mathcal{S}}(k)$ 的输出结果是不可区分的。 □

GS-VSSE 方案的可验证性意味着该方案可以验证数据的新鲜性和完整性，即防御定义 **??**和 **??**提出的两种攻击。这里我们采用了一个基于游戏 (game-based) 的安全性定义来证明 GS-VSSE 方案的可验证性。

定义 3.4 (GS-VSSE **可验证性**)： *令方案 GS-VSSE 是一个通用的、支持数据更新的可验证对称加密搜索方案，考虑以下概率性实验，其中 $\mathcal{A}$ 是一个有状态的敌手:*

$\mathbf{Vrf}_{\mathcal{A}}(k)$*:*

1. *挑战者通过 $KGen(1^k)$ 生成对称秘钥 K_1, K_2。*
2. *敌手 $\mathcal{A}$ 给挑战者选择一个文件集合 $\mathcal{D}$。*
3. *挑战者通过 $\{\lambda\} \leftarrow Init(K_1, K_2, \mathcal{D})$ 生成一个验证索引 λ。*
4. *给定 λ 和对算法 $SearchToken(K_1, w)$, $UpdateToken(K_1, K_2, d)$ 的预言权限, 敌手 $\mathcal{A}$ 选择一个关键字令牌 τ_w，和一系列加密文件 $\tilde{C_w}$，其中 $\tilde{C_w} \neq C_w$，同时输出结果证明 $\tilde{rho_s}$.*
5. *挑战者计算 $b := Verify(K_1, K_2, \tilde{C_w}, \tilde{rho_s}, \tau_w)$.*
6. *实验的输出为一个比特 b.*

如果对于任何概率多项式时间 *(Probabilistic Polynomial-Time, PPT)* 的敌手 $\mathcal{A}$，

$$Pr[\mathbf{Vrf}_{\mathcal{A}}(k) = 1] \leq negl(k). \tag{3-3}$$

成立，则我们认为 *GS-VSSE* 方案是可验证的。

定理 3.2： 如果哈希函数 H 和增量哈希函数 IH 是抗碰撞的，并且 G 是伪随机函数，那么 GS-VSSE 方案就是可验证的。

证明 考虑服务器返回的搜索结果为 $\tilde{C_w}$，而正确的搜索结果为 C_w，其中 $\tilde{C_w} \neq C_w$。但是用户的 $Verify$ 算法通过了 $\tilde{C_w}$ 作为正确搜索结果的情况。对于 $Verify$ 算法，这种情况的产生存在两种可能。第一种可能是在计算 $\tilde{C_w}$ 和 C_w 对应的伪随机值 $G_{K_2}(d)$ 时产生了碰撞，其中 $d \in D_w$，D_w 为 C_w 的明文信息，或者是在计算对应的增量哈希值 $\sum_{d \in D_w} IH(G_{K_2}(d))$ 时产生了碰撞。第二种是在生成根哈希的路径中产生了哈希碰撞。不管是哪一种，都可以推出哈希函数产生了碰撞或者伪随机函数产生了碰撞，但是哈希函数产生碰撞或者伪随机函数产生碰撞的可能性是小于一个可忽略的值的，因此，我们的方案是可验证的。 □

3.6　性能评价

3.6.1　实验设置

为了证明方案 GS-VSSE 的有效性，我们通过 Crypto++ 5.6.5 库实现了方案的原型。原型系统包含大约 2200 行代码。我们使用 HMAC-SHA256 作为两个随机预言 (random-oracle)，使用 SHA3-256 作为哈希函数，使用 MuHash 作为增量哈希函数。我们的实验在一台处理器为 Intel Core i5 2.5GHz，内存为 4G 的笔记本上进行，使用单线程实现。

我们使用一个开源数据集 Enron email dataset **?** 作为实验的测试数据集，使用了其中从“allen-p”到“kaminski-v”之间的数据。我们从该数据集中提取出了大量的“文件-关键字”对 (document-keyword pairs)，并通过 python 脚本为他们构建出了明文的倒排索引。注意，从文件中提取关键字的时延并没有被考虑在实验评估中，因为该问题与我们研究的 GS-VSSE 方案是一个独立的问题。

下文中，我们首先评估了 GS-VSSE 方案的算法效率，然后将 GS-VSSE 方案与一个著名的对称加密搜索方案 **?** 进行了结合，以此来证明 GS-VSSE 方案引入的结果验证的开销并不大。注意，如无特别说明，下文中的每一个实验结果都是十次实验的平均值。

3.6.2　实验结果

首先，我们评估了 *Init* 算法的时延，即数据持有者生成验证索引 λ 所需的时间。*Init* 算法需要数据持有者对每一个“文件-关键字”对生成键值对，其中“键”为关键字的伪随机值，“值”为包含该关键字的所有文件的增量哈希和。如图 **??**所示，由于验证索引 λ 的生成需要对每一个“文件-关键字”操作，即加密后向 MPT 中执行的插入操作，因此其所需时间与“文件-关键字”对的数量大小成正比。总体来说，*Init* 算法在“文件-关键字”对达到 400 万的时候，可以在 25 秒内执行完毕。由于 *Init* 算法仅需要在数据持有者在初始化时执行一次，因此这个开销是可以接受的。

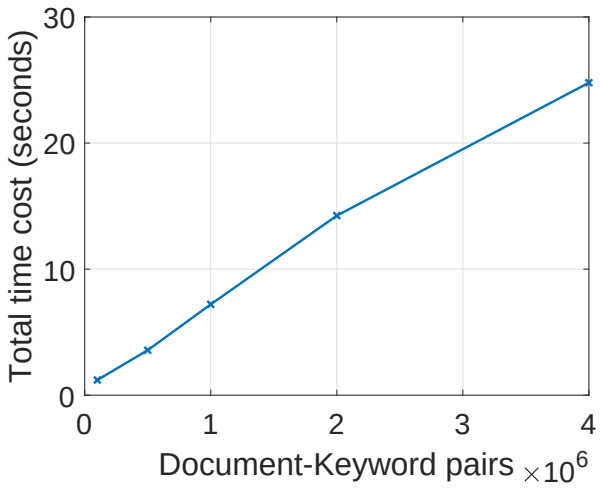

图 3.7　*Init* 算法的开销

云服务器更新验证索引的时间如图 **??**所示，更新算法 *Update* 的时延与数据库的大小有关，即与验证索引的大小有关，而验证索引的大小由它包含的关键字数量来衡量。严格意义上来说，一次更新的时延与 MPT 树的层数有关。为了更好的展示更新时延与验证索引大小的关系，我们使用了不同大小的数据库大小来评估更新时延。由于每一个文件包含的关键字个数不同，这里我们采用吞吐量 (throughput) 来衡量每秒钟云服务器可以更新的“文件-关键字”对。需要注意的是，*Update* 操作包括 *Add* 和 *Del* 两种操作，每一个关键字的更新不仅需要服务器更新相应的叶子节点，还需要服务器返回相应的搜索路径作为更新证明。从图中可以看到，*Add* 和 *Del* 操作的性能几乎相同。当数据库的大小增大时，吞吐量将会降低。当数据库的大小为 100 万个关键字时，云服务器每秒钟可以同时支持 30,000 次更新操作。同时，我们也测量了用户端 *UpdateToken* 算法引入的带宽开销，更新令牌 τ_u 中每一个关键字对应的密文带来的平均开销大小在 32 字节左右，而更新令牌的总大小

与待更新文件 d 包含的关键字个数有关。

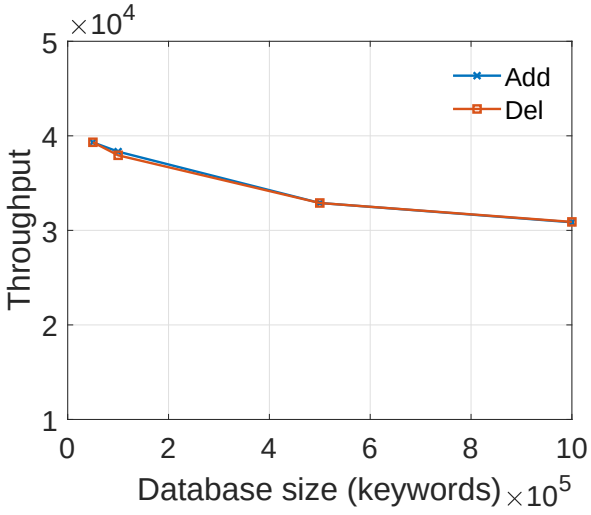

图 3.8　*Update* 算法的吞吐量

更新验证算法 $VerifyUpdate$ 的执行时间与用户更新文件所包含的关键字个数相关，因此我们采用了单个关键字更新验证需要的平均时间来进行评估。注意，数据持有者验证更新的操作包括对每一个更新令牌生成根哈希进行比对，当所有的更新令牌都验证通过后，最后在进行根哈希的生成。这里，我们只评估了验证每一个根哈希所需要的开销，因为相对生成更新后的根哈希，验证每一个更新令牌是否准确是该算法的主要开销。我们采用了数据库大小分别为 5 万，10 万，50 万和 100 万来衡量数据持有者验证更新证明的时间开销。如图 **??**所示，数据持有者验证每一个搜索令牌的开销随着数据库的增大而逐渐增长，并且逐渐趋于平缓。当数据库的大小为 50 万个关键字时，一个关键字的更新验证操作平均需要 55 微秒的时间。即，在验证索引包含 50 万个关键字时，数据持有者更新一个包含 1 千个关键字的文件，也只需要 55 毫秒的时间。

Prove 算法由云服务器执行，在数据持有者提交搜索请求时，云服务器需要根据搜索令牌在验证索引中查找到该关键字，并将其搜索路径作为结果证明返回。图 **??**中显示了该算法的开销，包括计算开销和通信开销。其中红色的线表示云服务器执行 *Prove* 算法的计算开销，而蓝色的线表示云服务器生成的结果证明的大小，即通信开销。从图中可以看到，云服务器可以在数据库大小在 100 万关键字时，每秒钟进行 43,000 次 *Prove* 操作，这意味着云服务器可以同时支持 43,000 个来自用户的查询请求。由 *Prove* 操作产生的结果证明 ρ_s 的大小也可以在图中看到，结果证明的大小随着数据库的增长而逐渐增长，趋于平缓。但总体而言，当数据库的大小在 100 万关键字时，结果证明的大小也只需要 2.5KB 左右。

Verify 算法由数据持有者执行，数据持有者在收到云服务器返回的搜索结果

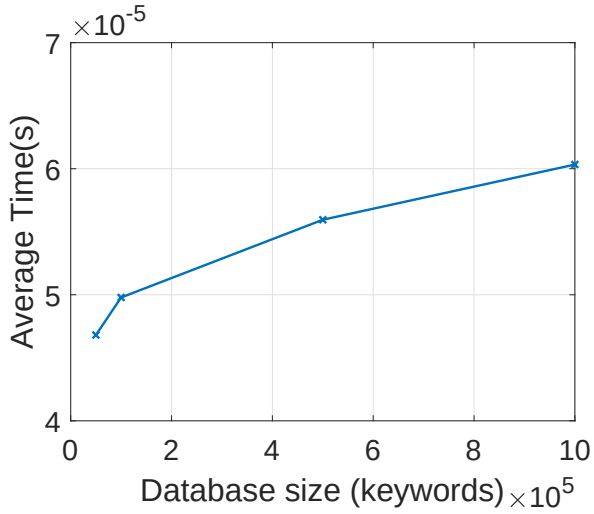

图 3.9　*VerifyUpdate* 算法的开销

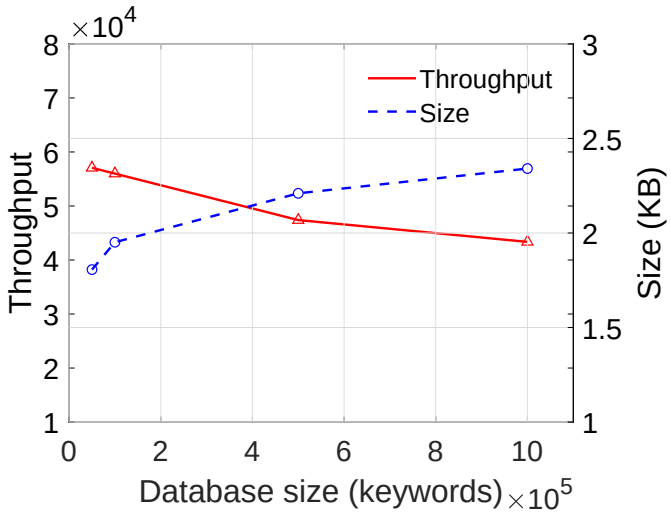

图 3.10　*Prove* 算法的吞吐量

和结果证明以后，需要重构出根哈希，并通过与自身保留的根哈希进行对比来确保搜索结果完整性。这里，我们通过累计分布函数图 (Cumulative Distribution Function, CDF) 来对用户端进行的 *Verify* 算法进行测试。这里采用的数据库大小为 10 万个关键字，即在验证索引大小为 10 万个关键字的情况下，数据持有者最快可以在 0.05 毫秒内完成验证操作，最长需要 0.1 毫秒以上的时间。总体来说，该算法需要的验证时间分布较为集中，在 99.7% 的概率下，数据持有者可以在 0.1 毫秒内完成结果验证，如图所示。对于一个非频繁搜索的数据持有者来说，0.1 毫秒的时间是完全可以接受的。

除了计算开销以外，我们还对算法引入的通信开销进行了评估，即由 *Update* 算法和 *Prove* 算法引入的更新证明和结果证明的开销。需要注意的是，由于更新证明和结果证明本质上都是搜索令牌在验证索引上的搜索路径，因此其大小与验

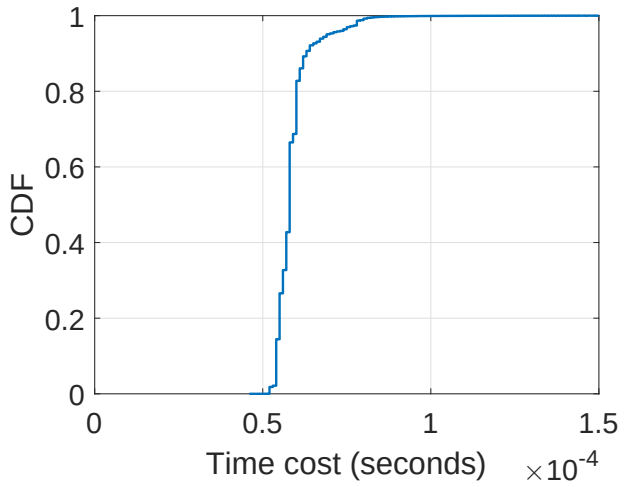

图 3.11　$Verify$ 算法的开销

证索引的层数直接相关，即与 MPT 的层数相关。如图 **??**所示，这里我们将 MPT 的层数作为参数来对其进行了衡量。另外，由于更新证明和结果证明的结构相同，因此我们采用了一个实验来衡量单个关键字产生的更新证明 (结果证明) 的通信开销。从图中可以看到，随着关键字个数的增长，结果证明 (更新证明) 的开销呈现对数形式增长。这是因为当验证索引达到 10 层时，并不是每一个“文件-关键字”对都存储于验证索引的第 10 层，而是根据关键字令牌的分布情况，散落在验证索引的各个层次。因此虽然验证索引的层数逐渐增长，但平均情况下，每一个关键字令牌产生的更新证明 (结果证明) 增长较为缓慢。从图中可以看到，一个关键字带来的平均更新证明开销为 2.5KB 左右，即如果用户更新的文件包含 1 千个关键字，那么更新证明的开销就为 2.5MB 左右。而对于结果证明，由于每次用户只搜索一个关键字，因此结果证明的开销即为 2.5KB。

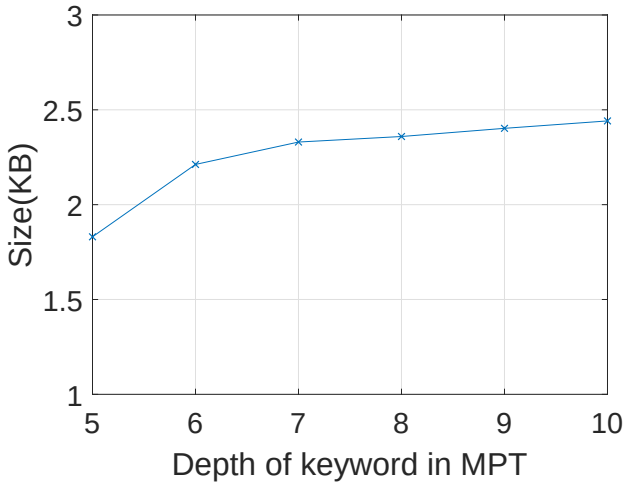

图 3.12　更新证明和结果证明的通信开销

除此以外，我们还对基于 MPT 构建的验证索引的的存储空间进行了评估。由 *Init* 算法可知，存储在 MPT 中的数据为每一个“文件-关键字”对的加密形式，其中“键”为每一个关键字的伪随机值，而“值”为包含该关键字的所有文件的增量哈希和。如图 **??**所示，如果我们使用一个 100 万个关键字的数据库，验证索引的存储空间大小大约为 82MB，而该数据库本身所占用的空间大小为 590MB，相对而言，基于 MPT 构建的验证索引带来的额外存储开销不算特别大。特别需要说明的是，这里我们评估验证索引的大小所采用数据集为关键字密集型的数据，即邮件数据。如果数据持有者需要加密的是媒体类型的数据，例如图片或是音乐文件等等，这些文件只包含少量的关键字和属性，因此为这些类型的数据构建验证索引时，验证索引占原数据集的比例将会很小，甚至可以忽略。

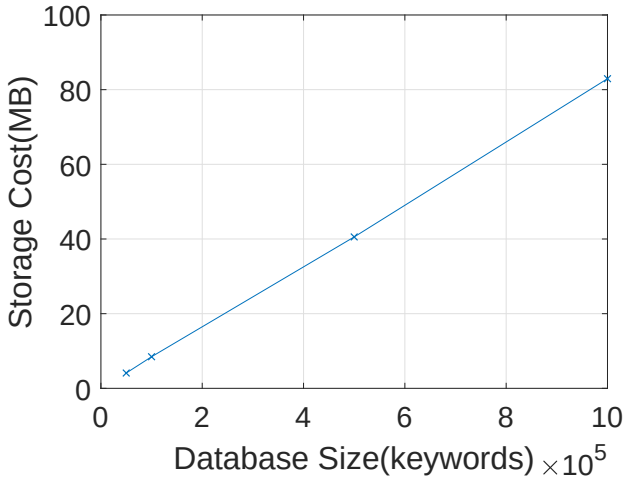

图 3.13　验证索引 λ 的存储开销

3.6.3　与 SSE 方案的对比

我们将我们的通用可验证对称加密搜索方案 GS-VSSE，与 Cash 等人提出的一个较为知名的动态对称加密搜索方案 (Dynamic Symmetric Searchable Encryption, DSSE) **?** 进行了结合，并展示了 GS-VSSE 为其提供结果验证服务带来的额外开销，如图 **??**所示。为了公平的进行性能比较，我们使用了同样的数据集，并在同样的设备参数下对两个方案联合进行了实验。如图 **??**所示，我们测量了 *Init* 阶段，*Search*(*Prove*) 阶段和 *Update* 阶段的性能开销。图中，*Init* 操作使用了 200 万个“文件-关键字”来分别构建 DSSE 方案 **?** 和方案 GS-VSSE 用到的索引，时间单位为秒。而其他三个操作 $Search(Prove), Add, Delete$ 的评估采用的数据库大小为 10,000 个关键字，时间单位为微秒。注意，DSSE 方案中的 *Search* 操作与 GS-VSSE

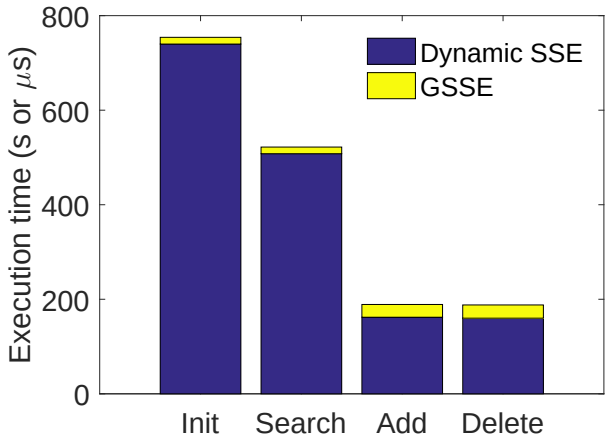

图 3.14　与 DSSE 方案 **?** 的对比

方案中的 *Prove* 操作对应。从图中可以看出，我们的 GS-VSSE 方案引入的额外开销非常小。其中，相较于 DSSE 方案而言，方案 GS-VSSE 在 *Init* 阶段引入的开销非常小，仅仅额外引入了 1.9% 的额外开销。而对于一次 *Search*(*Prove*) 操作来说，GS-VSSE 方案给云服务器引入的额外开销为 14 微秒，仅仅给 DSSE 方案带来了 2% 的额外开销。同样的，对于一次 *Add* 或 *Delete* 操作来说，我们的 GS-VSSE 方案仅仅引入了 27 微秒的时间，仅占方案 **?** 的 17%。

在表 **??**中，我们还比较了二者的通信开销。由于数据的方差较大，每一个实验结果都是 50,000 次实验的平均值。结果显示，DSSE 方案的搜索结果大小平均为 53KB 左右，而我们的结果证明大小仅仅为 3KB 左右，即由 GS-VSSE 方案引入的额外开销低于 6%。此外，DSSE 方案 **?** 生成的令牌大小平均为 390B，而 GS-VSSE 方案生成的令牌仅为 32B，即由 GS-VSSE 方案引入的额外开销低于 9%。这些实验结果充分表明了，GS-VSSE 方案是实用且高效的。

表 3.1　与 DSSE 方案 **?** 结合后的通信开销对比

通信开销	DSSE **?**	GS-VSSE	占比
搜索令牌	390 B	32 B	8.2%
搜索结果/结果证明	53 KB	3 KB	5.7%

3.7　本章总结

本章从定义、算法，安全性分析和实验几个角度，对 GS-VSSE 方案进行了充分的研究与分析。GS-VSSE 方案为用户在不可信云存储环境下提供了加密搜索解

决方案，即为传统加密搜索方案提供了结果验证功能。总体来说，GS-VSSE 方案完成了以下几点主要贡献。

- 功能完善。GS-VSSE 方案同时解决了数据新鲜性和数据完整性的验证。特别地，对于不可信云服务器故意返回空结果来规避结果验证的情况，GS-VSSE 也提出了解决方案，并且不需要用户在本地维护关键字集合。该问题是一个严重的安全漏洞，但现有的许多工作都忽略了该问题。GS-VSSE 方案首次对该问题进行了强调并进行了解决。
- 安全性。传统加密搜索方案以云服务器半可信为前提来提出解决方案，而 GS-VSSE 方案以云服务器不可信为前提。这种更真实的前提假定使 GS-VSSE 方案可以面对一个更强的敌手，因此也使得基于该假定设计的 GS-VSSE 方案具有更强的安全性。
- 高效性。GS-VSSE 方案的高效性体现在两个方面，一方面 GS-VSSE 方案的算法操作可与加密搜索方案的算法操作并行执行，在与加密搜索方案解耦的前提下，又在流程上与其紧密结合，提升了整体的执行效率。另一方面，从与加密搜索方案的对比实验中可以看出，GS-VSSE 方案的效率很高，每一步操作给整体方案带来的额外开销很小，并且 GS-VSSE 方案执行结果验证的效率也很高，几乎可以忽略不计。
- 通用性。GS-VSSE 方案通过索引解耦，将结果验证功能从加密搜索方案中独立出来，使得 GS-VSSE 方案可以为任何加密搜索方案提供结果验证功能，包括对称加密搜索方案甚至是公钥加密搜索方案。

第 4 章　多用户下的可验证对称加密搜索方案研究

4.1　引言

本章在 GS-VSSE 的基础上，提出了一种适用于多用户场景的可验证对称加密搜索方案 GM-VSSE。该方案同样可以与任意多用户场景下的加密搜索方案结合，来为其提供结果验证功能。与 GS-VSSE 方案不同的是，在多用户的场景下，即数据共享的情况下，数据持有者与数据搜索者产生了分离。数据持有者一方面需要对数据搜索者进行访问控制，以确保数据搜索者的合法性，并且确保数据搜索者只能读取数据而无法写入数据。另一方面，为了保证数据新鲜性，数据持有者还需要在数据发生了更新时，告知数据搜索者。我们将通过公私钥机制解决访问控制问题，并通过时间戳链机制解决数据新鲜性问题。本章的主要内容安排如下：首先介绍了多用户场景下（即数据持有者和数据搜索用户分离）的系统框架，明确了该框架的参与方及所承担的计算任务；随后通过一个抽象定义对该方案工作的流程进行了说明；算法分析部分对抽象定义中的具体算法进行了详细分析；最后，通过安全性分析和实验结果验证了本方案的安全性和有效性。

4.2　系统架构

多用户场景下的可验证对称加密搜索方案 GM-VSSE 如图 **??**所示，数据持有者和数据搜索用户产生了分离。首先数据持有者仍然需要对文件集进行处理，得到验证索引，同时他还需要生成一个鉴别符，将二者同时上传给云服务器。数据持有者在更新数据时，需要同时更新云端的验证索引和鉴别符。数据持有者可以将数据搜索者授权为一个合法用户。合法用户可以通过提交搜索令牌来对数据持有者的加密文件集进行搜索。云服务器在收到该搜索令牌以后，需要根据某一加密搜索方案向其返回加密搜索结果，同时返回结果证明以及鉴别符。通过这种方式，合法用户可以按需获取搜索结果并进行结果验证。

4.3　方案流程

在描述 GM-VSSE 方案的工作流程前，我们先回顾多用户对称加密搜索方案的常见定义。

定义 4.1 (**MSSE 方案**)：*一个支持文件更新的 MSSE 方案，参与方有三个，分别为数据持有者，合法用户以及半可信的云服务器。数据持有者向云服务器提供加密*

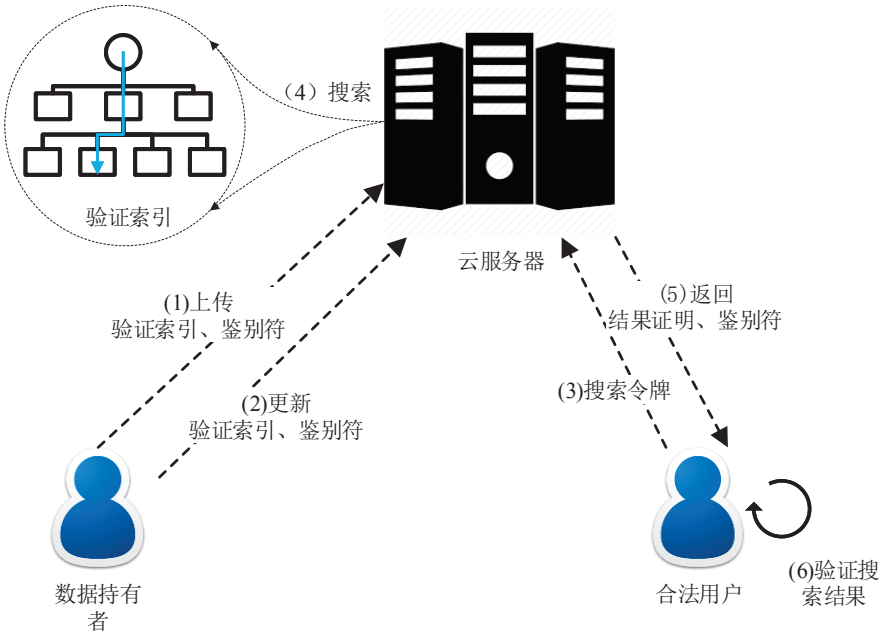

图 4.1　多用户场景下的可验证对称加密搜索框架 GM-VSSE

数据集和搜索索引，同时数据持有者可以对合法用户进行注册和撤销。云服务器可以为合法用户提供搜索功能。一个 $MSSE$ 方案是以下八个算法的集合:

- $KGen_{MSSE}(1^k) \rightarrow \{\mathcal{K}, K_O\}$: 是由数据持有者执行的秘钥生成算法。它将一个安全参数作为输入，输出一系列对称秘钥 $\mathcal{K}$ 和一个用户私钥 K_O。
- $EnrollUser_{MSSE}(K_O, U) \rightarrow \{K_U\}$: 是由数据持有者执行的用户注册算法。它将数据持有者的私钥 K_O 和待注册用户的身份证明 U 作为输入，输出该用户的私钥 K_U。数据持有者将私钥 K_U 发送给用户 U。该算法将一个用户添加到了数据持有者的合法用户集合中，即授权该用户对数据持有者的数据进行搜索。
- $RevokeUser_{MSSE}(K_O, U) \rightarrow \{b\}$: 是由数据持有者执行的用户撤销算法。它将数据持有者的私钥 K_O 和待注册用户的身份证明 U, 输出一个比特 b。如果 $b = 1$ 表示撤销成功，反之。该算法将一个用户从数据持有者的合法用户集中移除，即取消该用户对数据持有者云端加密数据的搜索权限。
- $Init_{MSSE}(\mathcal{K}, \mathcal{D}) \rightarrow \{\gamma, C\}$: 是由数据持有者执行的初始化算法。它将对称秘钥 $\mathcal{K}$ 和明文文件集 $\mathcal{D}$ 作为输入，输出搜索索引 γ 和密文文件集 C。数据持有者将搜索索引 γ 和密文文件集 C 上传给云服务器。
- $UpdateToken_{MSSE}(\mathcal{K}, d) \rightarrow \{\tau_u\}$: 是由数据持有者执行的更新令牌生成算法。它将对称秘钥 $\mathcal{K}$ 和需要更新的文件 d 作为输入，输出一系列更新令牌

τ_u。数据持有者将更新令牌 τ_u 上传给云服务器。

- $Update_{MSSE}(\gamma, \tau_u) \rightarrow \{\gamma'\}$: 是由云服务器执行的更新算法。它将搜索索引 γ 和更新令牌 τ_u 作为输入，输出更新后的搜索索引 γ'。
- $SearchToken_{MSSE}(\mathcal{K}, w) \rightarrow \{\tau_w\}$: 是由合法用户执行的搜索令牌生成算法。它将对称秘钥 $\mathcal{K}$ 和某一关键字 w 作为输入，输出与该关键字相关搜索令牌 τ_w。合法用户将该搜索令牌 τ_w 上传给云服务器进行搜索。
- $Search_{MSSE}(\gamma, \tau_w) \rightarrow \{C_w\}$: 是由云服务器执行的搜索算法。它将搜索索引 γ 和搜索令牌 τ_w 作为输入，输出搜索结果 C_w。云服务器将搜索结果 C_w 返回给合法用户。

注意，这里需要说明的是，本方案的目的不是设计多用户的对称加密搜索方案，而是为这些已有的多用户加密搜索方案提供结果验证功能。本方案在 GS-VSSE 的基础上进行了改进，验证索引的构建仍然需要利用 MPT 和增量哈希技术，同时 GM-VSSE 方案还使用了时间戳链和公私钥加密机制来实现跨用户的数据新鲜性和数据完整性验证。GM-VSSE 方案的具体定义如下。

定义 4.2 (GM-VSSE 方案)： *在 GM-VSSE 方案中，参与方有三个, 分别为数据持有者，合法用户以及不可信的云服务器。数据持有者向云服务器提供一个验证索引和鉴别符，使得云服务器可以为合法用户提供搜索结果证明，来确保加密搜索结果的新鲜性和完整性。一个 GM-VSSE 方案是以下八个算法的集合:*

- $KGen(1^k) \rightarrow \{K_1, K_2, K_3, K_O, (ssk, spk)\}$: 是由数据持有者执行的秘钥生成算法。它将一个安全参数作为输入，输出对称秘钥 K_1, K_2, K_3，数据持有者私钥 K_O 和一对签名公私钥对 (ssk, spk)。
- $EnrollUser(K_O, U) \rightarrow \{K_U\}$: 是由数据持有者执行的用户注册算法。它将数据持有者的私钥 K_O 和待注册用户的身份证明 U 作为输入，输出该用户的私钥 K_U。数据持有者将私钥 K_U 以及对称秘钥 K_1, K_2, K_3 和签名公钥 spk 发送给用户 U。该算法将一个用户添加到了数据持有者的合法用户集合中，即授权该用户对数据持有者的数据进行搜索。
- $RevokeUser(K_O, U) \rightarrow \{b\}$: 是由数据持有者执行的用户撤销算法。它将数据持有者的私钥 K_O 和待注册用户的身份证明 U, 输出一个比特 b。如果 $b = 1$ 表示撤销成功，反之。该算法将一个用户从数据持有者的合法用户集中移除，即取消该用户对数据持有者云端加密数据的搜索权限。
- $Init(K_1, K_2, K_3, ssk, \mathcal{D}) \rightarrow \{\lambda, \pi\}$: 是由数据持有者执行的初始化算法。它将对称秘钥 K_1, K_2, K_3，签名私钥 ssk 和明文文件集 $\mathcal{D}$ 作为输入，输出验证索引 λ 和鉴别符 π。数据持有者在本地保存验证索引 λ 的根节点哈希 rt，并将验证

索引 λ 和鉴别符 π 上传给云服务器。

- $UpdateToken(K_1, K_2, K_3, ssk, d) \rightarrow \{\tau_u, \pi\}$: 是由数据持有者执行的更新令牌生成算法。它将对称秘钥 K_1, K_2, K_3，签名私钥 ssk 和需要更新的文件 d 作为输入，输出一系列更新令牌 τ_u 和鉴别符 π。数据持有者将更新令牌 τ_u 和鉴别符 π 上传给云服务器。
- $Update(\lambda, \tau_u) \rightarrow \{\lambda', \rho_u\}$: 是由云服务器执行的更新算法。它将验证索引 λ 和更新令牌 τ_u 作为输入，输出更新后的验证索引 λ' 和更新证明 ρ_u。云服务器将更新证明 ρ_u 返回给数据持有者。
- $VerifyUpdate(rt, \tau_u, \rho_u) \rightarrow \{rt'\}$: 是由数据持有者执行的更新算法。它将验证索引的根哈希 rt，更新令牌 τ_u 和服务器返回的更新证明 ρ_u 作为输入，输出新的根哈希 rt'。若更新证明 ρ_u 验证通过，则输出更新后的根哈希 rt'，若更新证明验证失败，则输出的根哈希 rt' 与原始根哈希 rt 相同。
- $SearchToken(K_1, w) \rightarrow \{\tau_w\}$: 是由合法用户执行的搜索令牌生成算法。它将对称秘钥 K_1 和某一关键字 w 作为输入，输出与该关键字相关搜索令牌 τ_w。合法用户将该搜索令牌 τ_w 上传给云服务器进行搜索。
- $Prove(\lambda, \tau_w, t_q) \rightarrow \{\rho, \pi_q^t, \pi_c\}$: 是由云服务器执行的结果证明生成算法。它将验证索引 λ，搜索令牌 τ_w 以及合法用户提交查询的时间 t_q 作为输入，输出 ρ 和两个鉴别符 π_q^t，π_c 作为结果证明。云服务器将结果证明 ρ，π_q^t，π_c 返回给合法用户。
- $Verify(K_1, K_2, K_3, spk, C_w, \rho_s, \pi_q^t, \pi_c, \tau_w, rt) \rightarrow \{b\}$: 是由合法用户执行的验证算法。它将对称秘钥 K_1, K_2, K_3，签名公钥 spk，加密搜索结果 C_w，结果证明 ρ_s，鉴别符 π_q^t，π_c，搜索令牌 τ_w 和保留的验证索引根哈希 rt 作为输入，输出一个比特 b，代表接受或者拒绝该搜索结果。其中，该算法包括了两个子算法，分别为 $Check$ 算法和 $Generate$ 算法，它们分别可以定义为 $Check(K_3, spk, \pi_q^t, \pi_c) \rightarrow \{b\}$ 和 $Generate(K_1, K_2, K_3, C_w, \rho, \tau_w, \pi_q^t) \rightarrow \{b\}$.

4.4 方案设计

本节中，我们将阐述 GM-VSSE 方案，即多用户场景下的可验证加密搜索方案。注意，由于数据在多个用户之间共享，当数据持有者的数据需要更新时，如何将更新信息同步给多个用户，从而确保数据的新鲜性是本方案的重要挑战。一个简单的想法是，由于验证索引的根哈希是数据新鲜性和数据完整性的保证，在数据持有者需要更新数据时，他只需将更新后的验证索引根哈希值发送给所有的合法用户即可。但这种方式在数据需要频繁更新的场景下，势必给数据持有者带来

巨大的通信开销。并且，这种推送的方法存在带宽浪费，即某些合法用户搜索数据的频率可能较低，在两次推送时间间隔内，并没有数据搜索需求。考虑到该问题，我们提出了一种由合法用户按需获取验证索引根哈希的方式。该方法引入了由时间戳链构造的鉴别符，将验证索引的根哈希值与时间戳绑定，使得用户在搜索数据时，能够获取到最新的根哈希值，用以对数据的新鲜性进行验证。本节的内容安排如下，首先我们将描述如何建立和更新鉴别符，然后我们给出了通过鉴别符验证数据新鲜性的方法，最后通过一个简单的例子介绍了本方案的工作细节。

4.4.1　构建及更新时间戳链

由于云服务器不可信，当数据持有者向云服务器发送了多个鉴别符以后，云服务器有可能会向合法用户重放之前收到的鉴别符，以此来破坏数据新鲜性。针对该问题，本方案采用了时间戳链机制来构造鉴别符，而不仅仅是将验证索引根哈希与时间戳绑定。它使得合法用户可以通过该时间戳链来追踪其中包含的鉴别符，从而确保自身拿到的根哈希值为最新的数据。注意，我们采用的时间戳链方案与可验证对称加密搜索方案 **?** 不同。他们的方案仅仅能在单用户的场景下工作，即在用户持有更新信息的场景下工作。因此，他们的方案不能用户多用户场景下的数据新鲜性验证。

在方案开始工作前，我们先对方案的设定进行描述，首先数据持有者需要为鉴别符设定一个更新周期 (Update Interval)[①]，即设置固定更新时间点为 $\{up_1, up_2, \cdots, up_i, \cdots, up_m\}$。如果更新周期内数据持有者的数据产生了更新，则数据持有者需要生成鉴别符进行上传，否则，数据持有者只需要在固定更新时间点对鉴别符进行上传。注意，更新周期可以由数据持有者根据其更新频率来控制。一般来说，如果数据持有者的更新频率较高，则更新周期可以设置短一些，以保证一个更新周期内的鉴别符不会过长，并且验证的时延也会减少。但这会给数据持有者带来相对较高的更新负载。在我们的实验部分**??**，我们将会展示用户验证时延和数据持有者带宽开销之间的关系。这里我们将使用网络时间协议 (Network Time Protocol, NTP) **??** 来对云服务器，数据持有者以及合法用户进行时间同步。注意，时钟同步的精确度目前已经可以达到几毫秒甚至是几十微秒 **???**，足以保证 GM-VSSE 方案的验证正确性。此外，一个恶意的云服务器有可能会伪造时钟信息，但他永远无法伪造时间戳链信息，因为时间戳链的信息由数据持有者的签名私钥进行了保证。接下来我们将阐释方案细节。

首先，我们将阐述数据持有者如何通过时间戳链构造鉴别符。为了防止云服

① 在实验验证环节，我们将展示更新周期与结果验证时延的相关性 **??**

务器重放鉴别符，我们设计了一种基于时间戳链的机制来检测云服务器的恶意行为。我们通过将旧的鉴别符嵌套进新的鉴别符的方式，使得鉴别符串联起来形成链。如公式**??**所示，我们将旧的鉴别符 π 与时间戳 tp 和验证索引的根哈希 rt 进行连接，并将联合后的三者用对称秘钥 K_3 加密，同时使用数据持有者的签名私钥 ssk 进行签名，从而生成新的鉴别符。如果验证索引在一个更新周期内没有更新，则数据持有者只需要在下一个固定更新时间点对鉴别符中的时间戳进行更新。如果数据持有者的文件集在更新周期内产生了更新，即验证索引的根哈希产生了更新，则数据持有者需要用更新后的根哈希和最新的时间戳来构建一个新的鉴别符，并上传给云服务器。注意，在每一个更新周期内会形成一个时间戳链，每个时间戳链在下一个更新周期开始时结束，即每个时间戳链中的最后一个鉴别符是在下一个固定更新时间点生成的。换句话说，每一个更新周期内的鉴别符是连接在一起的，而不同更新周期内的鉴别符是不相关的。

$$
\begin{cases}
\pi_{i,0} = (\alpha_{i,0}, \mathrm{Sig}_{ssk}(\alpha_{i,0})), & up_i < tp_{i,0} \le up_{i+1} \\
\alpha_{i,0} = Enc_{K_3}(rt_{i,0} || tp_{i,0}) & \\
\cdots & \\
\pi_{i,j} = (\alpha_{i,j}, \mathrm{Sig}_{ssk}(\alpha_{i,j})), & tp_{i,j-1} < tp_{i,j} \le up_{i+1} \\
\alpha_{i,j} = Enc_{K_3}(rt_{i,j} || tp_{i,j} || \alpha_{i,j-1}) & \\
\cdots & \\
\pi_{i,n} = (\alpha_{i,n}, \mathrm{Sig}_{ssk}(\alpha_{i,n})), & tp_{i,n} = up_{i+1} \\
\alpha_{i,n} = Enc_{K_3}(rt_{i,n} || tp_{i,n} || \alpha_{i,n-1}) &
\end{cases}
\tag{4-1}
$$

这里 i 代表第 i 个更新周期，j 代表该更新周期内的第 j 个鉴别符.

在这种设置下，当一个合法用户发起了一次搜索时，云服务器需要将最新的鉴别符发送给用户。用户可以通过解密鉴别符从其中得到验证索引的根哈希值 rt 和时间戳 tp。如果时间戳 tp 在最近一次固定更新时间点之前，则可以认为服务器发回了旧的鉴别符，产生了恶意行为。这种方式保证了云服务器无法在当前更新周期外产生数据新鲜性攻击。

然而，如果云服务器只传送最新的鉴别符给用户，它仍然可以在当前更新周期内产生数据新鲜性攻击。具体而言，如果在当前更新周期内，数据持有者的文件集产生了一次或者多次的更新，即数据持有者在该更新周期内，上传了多个鉴别符，则云服务器可以发送该更新周期内的任意一个旧的鉴别符来发起数据新鲜性攻击。

当然，如果用户对数据新鲜性的要求不高，那么数据新鲜性的验证到此即可。

但对于某些必须保证数据实时同步的用户，我们还需要解决当前更新周期内发生数据新鲜性攻击的问题。为了解决该问题，云服务器除了需要将搜索时刻的鉴别符发送给用户以外，还需要将当前时间戳链中的最后一个鉴别符发送给用户。我们将当前更新周期内时间戳链的结束时间点成为检测点 (checkpoint)。通过这种方式，用户可以解密在检测点收到的鉴别符，并使用该鉴别符追踪在搜索时刻收到的鉴别符，以此来判定搜索时刻收到的鉴别符是否为搜索时刻最新的鉴别符，从而防止了云服务器在当前更新周期内发起数据新鲜性攻击。在下一小节，我们将详细描述用户如何通过两个鉴别符检测云服务器是否发起了数据新鲜性攻击。

4.4.2　结果验证

GM-VSSE 方案中，数据持有者的验证算法 $Verify$ 可以拆分为两个子算法 $Check$ 和 $Generate$，其中 $Check$ 算法主要用于对鉴别符的验证，保证鉴别符的最新就保证了用户收到的验证索引根哈希为最新，从而防止了数据新鲜性攻击。$Generate$ 算法在 $Check$ 算法后执行，它通过加密搜索结果，搜索令牌，结果证明和 $Check$ 算法解密得到的最新根哈希可以完成对数据完整性的验证。该算法的具体流程与 GS-VSSE 方案中的 $Verify$ 算法相同，只是 $Generate$ 算法在与重构出的根哈希进行对比时，使用的根哈希为从鉴别符中解密得到的根哈希，而不是自身保留的根哈希，因此，这里不再赘述。

算法 **??**展示了 $Check$ 算法的伪代码，该算法由合法用户执行，用以确保鉴别符没有被重放。假设 π_q^t 为用户在搜索时刻 t 收到的鉴别符，π_c 为用户在检测点收到的鉴别符。首先我们需要通过签名公钥 spk 确保两个鉴别符都来自于数据持有者。接着，我们需要通过鉴别符 π_q^t 中的 α_q^t 来确保该鉴别符的时间戳不会早于该更新周期。随后，我们需要通过 π_c 中的 α_c 不断解密得到 $rt_k||tp_k||\alpha_{k-1}$，直到我们找到某个时间戳 $tp_k < t$，或者找到 $\alpha_k = \emptyset$。我们将 α_k 与 α_q^t 和 $\emptyset$ 进行对比。如果 α_k 与两者均不相等，则证明产生了数据新鲜性攻击。否则，α_q^t 就被认为是正确的，即其中包含的根哈希可以用作数据新鲜性和数据完整性的见证。

在合法用户完成了鉴别符的验证以后，即验证了数据新鲜性后，用户即可使用鉴别符 π_q^t 中的根哈希来对数据完整性进行验证。如算法 **??**所示，合法用户需要通过 $Check$ 算法确保鉴别符 π_q^t 的正确性，从而确保其中包含的根哈希的新鲜性。如果鉴别符 π_q^t 验证通过，则再通过 $Generate$ 算法比较从鉴别符 π_q^t 中得到的根哈希与通过搜索结果和结果证明恢复出来的根哈希是否相等。只有这两个算法都验证通过，即都输出 $b = 1$，$Verify$ 算法才验证通过，表示接受服务器返回的加密搜索结果。

算法 6 *Check* 算法

输入： K_3: 对称秘钥; spk: 验证签名的公钥; π_q^t: 在搜索时刻 t 返回给数据搜索用户的鉴别符; π_c: 在检测点返回给用户的鉴别符。

输出： $b \in \{0,1\}$, 如果 $b = 1$, *Check* 算法成功，否则失败。

令 $\pi_q^t = \{\alpha_q^t, Sig_q^t\}$，$\pi_c = \{\alpha_c, Sig_c\}$
if $\alpha_q^t \neq (Sig_q^t)_{spk}$ || $\alpha_c \neq (Sig_c)_{spk}$ **then**
    return $b = 0$
end if
$(rt_q^t, tp_q^t, \alpha) \leftarrow Dec_{K_3}(\alpha_q^t)$
if tp_q^t 早于最近一个更新时间点 **then**
    return $b = 0$
end if
令 $\alpha_k = \alpha_c$
for $\alpha_k \neq \emptyset$ **do**
    $(rt_k, tp_k, \alpha_{k-1}) \leftarrow Dec_{K_3}(\alpha_k)$
    if $tp_k < t$ **then**
        break
    end if
    令 $\alpha_k = \alpha_{k-1}$
end for
if $\alpha_k = \alpha_q^t || \alpha_k = \emptyset$ **then**
    return $b = 1$
else
    return $b = 0$
end if

4.4.3　实例分析

如图 **??**所示，横向为时间轴，红色的线表示的是数据持有者的更新时间点，蓝色的线表示的是合法用户的查询点。$(up_1, up_2, \cdots, up_i, up_{i+1})$ 是更新时间点，其中 $(up_i, up_{i+1}]$ 即为一个更新周期，up_{i+1} 为该更新周期的检测点。每一个周期中，鉴别符都链式相连，例如图中的 $\pi_{i,0}, \pi_{i,1}$，但是不同的更新周期内的鉴别符不相连。

让我们考虑如下几种合法用户在不同时刻发起请求的情况：(i) 第一种情况，合法用户在 t_1 时刻发起搜索请求，其中 $t_1 < tp_{i,0}$；(ii) 第二种情况，合法用户在 t_2 时

算法 7 Verify 算法

输入： K_1, K_2, K_3: 对称秘钥；spk: 签名所使用的公钥；C_w: 加密搜索结果；ρ_s: 加密搜索结果证明；τ_w: 搜索令牌；π_q^t: 在搜索时刻 t 收到的鉴别符；π_c: 在检测点收到的鉴别符；

输出： $b \in \{0, 1\}$, 如果 $b = 1$, 接受该结果；否则拒绝该结果。

1: $b \leftarrow Check(K_3, spk, \pi_q^t, \pi_c)$

2: **return** $b \leftarrow b \quad \&\& \quad Generate(K_1, K_2, K_3, C_w, \rho, \tau_w, \pi_q^t)$

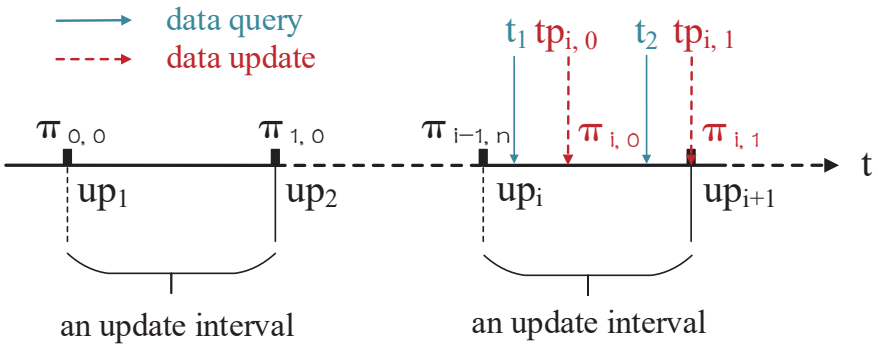

图 4.2　时间戳链的简单示例

刻发起搜索请求，该请求发生在一次数据更新时刻 $tp_{i,0}$ 之后，且云服务器发送给用户的鉴别符为 $\pi_{i,0}$；(iii) 第三种情况，合法用户仍然在 t_2 时刻发起搜索请求，但是云服务器发送给用户的鉴别符为 $\pi_{i-1,n}$。在最后一种情况中，一个数据新鲜性攻击产生，但是它会在检测点 up_{i+1} 被用户发现。用户将在检测点收到鉴别符 $\pi_{i,1}$，通过它用户可以验证用户在查询时刻收到的鉴别符是否正确。

我们仍然采用上述三种情况来描述合法用户验证鉴别符的过程。在上述第一种情况中，用户将收到鉴别符 $\pi_{i-1,n}$ 和 $\pi_{i,1}$，通过它们，用户可以提取出 $\alpha_{i-1,n}$ 和 $\alpha_{i,1}$,$\alpha_{i,0}$。通过 $Check$ 算法，我们可以发现在解密了 $\alpha_{i,0}$ 后，得到的 α 为 $\emptyset$，因此 $Check$ 算法将输出 $b = 1$，表示在搜索时刻收到的鉴别符 $\pi_{i-1,n}$ 是正确的。在第二种情况中，$\alpha_{i,0}$ 同样通过 $\alpha_{i,1}$ 被解密出，并且 $\alpha_{i,0}$ 的时间戳已经早于 t_2. 我们可以通过对比发现，通过检测点收到的鉴别符 $\pi_{i,1}$ 循环解密出来的 $\alpha_{i,0}$ 与在搜索时刻收到的鉴别符 $\pi_{i,0}$ 解密出来的 $\alpha_{i,0}$ 相等。因此 $\alpha_{i,0}$ 也被认为是正确的，即认为用户在搜索时刻收到的根哈希是正确的。然而，在最后一种情况中，我们将会发现数据新鲜性攻击，因为解密得到的 $\alpha_{i,0}$ 与用户在搜索时刻得到的 $\alpha_{i-1,n}$ 不相等。

4.5　安全性分析

在本节中，我们将对 GM-VSSE 方案的安全性进行证明。与 GS-VSSE 方案相同，我们需要从机密性和可验证性两个角度对 GM-VSSE 方案进行证明。在证明开始前，我们需要明确 GM-VSSE 方案与 GS-VSSE 方案的差别。GM-VSSE 方案通过引入鉴别符 π 的方式将根哈希和时间戳进行了绑定，搜索用户通过解密鉴别符，可以得到其中包含的最新根哈希，我们通过根哈希来对数据新鲜性和数据完整性进行验证。除此之外，GM-VSSE 方案的其他步骤均与 GS-VSSE 方案相同。因此我们只需证明鉴别符 π 的引入没有破坏数据的机密性和结果的可验证性，其他证明与**??**类似，故不再赘述。我们有以下的定理：

定理 4.1： 如果 (ssk, spk) 是签名公私钥对，K_3 是对称秘钥，那么 GM-VSSE 方案就是机密且可验证的。

证明 首先证明机密性。由于鉴别符 π 通过数据持有者的对称秘钥 K_3 进行了加密，因此敌手只要没有该对称秘钥，即无法获知鉴别符的内容。因此鉴别符的机密性是可以保证的，即 GM-VSSE 方案是机密的。其次证明可验证性。由于鉴别符 π 通过数据持有者的签名私钥 ssk 进行了签名，并且采用了对称秘钥 K_3 进行了加密。敌手没有签名私钥 ssk 和对称秘钥 K_3 就无法伪造鉴别符。因此也无法生成让合法用户验证通过的鉴别符，因此 GM-VSSE 方案是可验证的。 □

4.6　性能评价

4.6.1　实验设置

为了证明 GM-VSSE 方案的有效性，我们在一台处理器为 Inter Core i5 2.5GHz，内存为 4G 的笔记本上进行了实验，实验采用单线程执行。这里，我们的签名秘钥对采用了 RSA 签名，与 GS-VSSE 方案相同，我们同样使用了 Crypto++ 5.6.5 库来实现对称加密算法。实验的数据为十次实验的平均值。下文中，我们首先对 GM-VSSE 方案引入的数据持有者端的带宽开销进行了验证，随后对 GM-VSSE 方案所需的结果验证时间进行了评估。

4.6.2　实验结果

图 **??**和图 **??**评估了 GM-VSSE 方案的开销，包括数据持有者的通信开销和合法用户的验证开销。图中的 η 表示数据持有者的数据更新频率。首先，我们考虑数据持有者端的通信开销。如图 **??**所示，我们主要考虑由鉴别符带来的开销。这

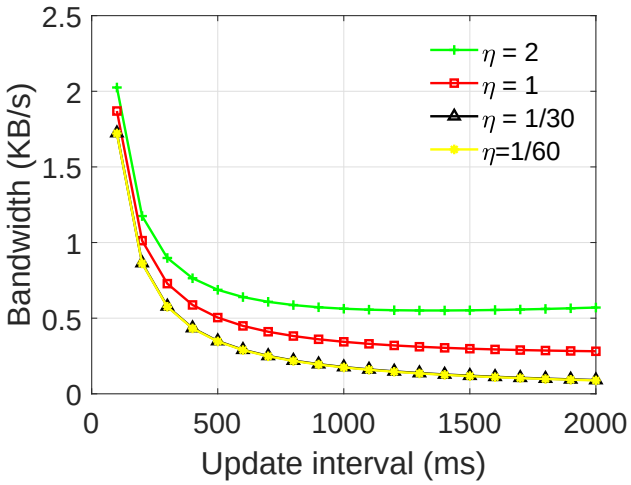

图 4.3　带宽开销

里，每一个更新周期内的第一个鉴别符大小约为 176 字节，包括了 32 字节的验证索引根哈希开销，8 字节的时间戳开销，8 字节的 AES-CBC 扩展开销，以及 128 字节的 RSA 签名开销。总体来说，鉴别符引起的带宽开销包括以下两个部分：由固定更新点导致的鉴别符更新开销和数据更新导致的鉴别符更新开销。图 **??**中的曲线充分体现了这两种更新带来的开销。我们可以发现当更新周期接近 0 时，鉴别符到来的带宽开销接近每秒 2KB，这是由固定更新点导致的鉴别符开销，因为数据持有者需要在每一个固定更新点到来时，选取当前的时间戳来重新构建鉴别符并上传。它是与带宽开销成反比的，即更新周期越小，固定更新点越密集，带宽开销越大。此外，带宽开销又随着更新周期的增长缓慢上升，这是由鉴别符自身长度增长带来的带宽开销增长。因为在数据持有者的更新频率一定的情况下，更新周期越长，一个更新周期内产生的数据更新次数将会更多。而每一次数据更新都会导致验证索引根哈希的变化，因此数据持有者需要重新选取时间戳来对更新后的根哈希进行加密上传，而一个更新周期内的鉴别符都是以链式生成的。因此更新周期越长，该更新周期内生成的鉴别符的数量也会越多，而鉴别符的长度也随着时间戳链的增长而逐渐变大。

其次，我们考虑合法用户端的验证开销。这部分开销包括用户等待检测点的时间 (wait time) 以及执行 *Check* 算法和 *Generate* 算法的时间，即 $Verify$ 算法的时间。根据实验测量，*Generate* 算法的开销为 0.1 毫秒，几乎可以忽略。因此图 **??**中并未标注该部分开销。这里，η 表示数据持有者的更新频率，我们假设合法用户在一个更新周期内发起搜索请求的时刻是均匀分布的，则该用户等待检测点的时间也呈均匀分布。即等待检测点的平均时间为半个更新周期的长度，这将占整个验证时延的大部分。*Check* 算法的开销与更新周期成正比，这主要由验证鉴别符的

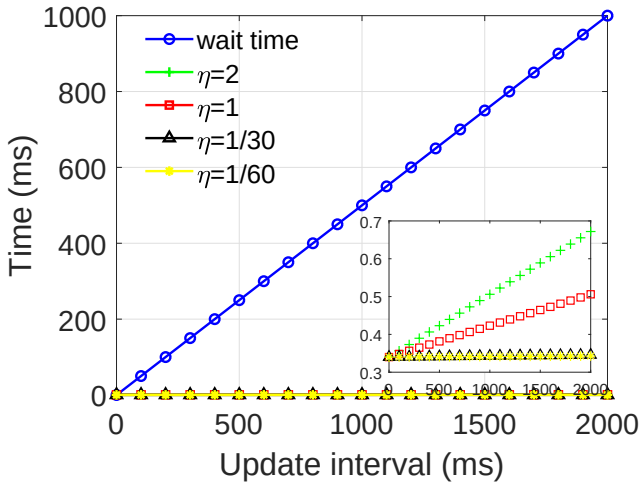

图 4.4　总验证时间开销

签名和解密鉴别符这两种开销导致，但是它相对等待检测点的时间是可以忽略的。需要特别说明的是，在以上的实验中，我们没有将网络的传输时间考虑在内，因为网络传输时间在不同的环境中差异非常大，并且根据我们的方案，传输时延对方案的影响不会很大。从图**??**中我们可以看到，当更新频率在 2Hz 到 1/60Hz 不等时，即数据持有者的更新频率在每秒钟两次至每一分钟一次时，合法用户端的验证开销主要取决于等待检测点所需的时间，即和更新间隔的设置有关，而 *Check* 算法的开销几乎可以忽略不计。

综合以上的实验结果，GM-VSSE 方案引入的带宽开销和验证时延都是可以接受的。总体来说，为了在验证时延和带宽开销之间寻找一个平衡点，我们建议可以将更新周期设置在 500 毫秒置 1500 毫秒之间。在该更新周期下，数据持有者的平均带宽开销在 1KB 以内，而合法用户的验证时延也在 200ms 和 800ms 之间，总体来说都是可以接受的。

4.7　本章总结

本章通过方案定义，方案描述，安全性分析和实验分析等方面对 GM-VSSE 方案进行了分析与阐述，GM-VSSE 方案基于不可信云存储环境，为用户提供了一种数据共享场景下的可验证对称加密搜索解决方案，总体来说，GM-VSSE 方案的贡献有以下几点：

- 功能完善。GM-VSSE 方案提供了一种多用户场景下的可验证对称加密搜索方案，该方案支持在单方写/多方读的场景下工作，为数据共享场景下的可验证加密搜索提出了解决方案，使得可验证加密搜索方案的功能性更加完善。
- 通用性。GM-VSSE 方案可以为实现了多用户场景的对称加密搜索方案提供

结果验证功能，同时确保数据新鲜性和数据完整性，使其通用性得到了进一步提升。

- 高效性。通过实验验证，GM-VSSE 方案为数据持有者和合法用户带来的通信开销和计算开销都很小，在方案有效的前提下保证了其高效性。

第 5 章　总结与展望

5.1　论文工作总结

本文主要研究了云存储环境下的可验证加密搜索问题，针对现有可验证加密搜索方案的缺点，本文首先提出了一种通用的、支持用户数据更新的可验证对称加密搜索方案 GS-VSSE，该方案支持单用户情况下的加密搜索结果验证。本文通过严格的方案定义和算法描述对该方案进行了详细的分析和描述，并通过安全性分析和实验结果证明了 GS-VSSE 方案的安全性和高效性。随后，基于 GS-VSSE 方案，本文又提出了一种支持多用户场景下的可验证对称加密搜索方案 GM-VSSE，该方案的引入使得数据共享场景下的结果验证成为可能，进一步提升了方案的通用性。

首先，本文介绍了云存储环境下的对称加密搜索、可验证对称加密搜索问题，并分析了几个主流方案运用的算法，比较了不同方案之间的优缺点。本文总结了这些方案普遍存在的问题，对可验证对称加密搜索问题中存在的数据新鲜性攻击和数据完整性攻击进行了正式定义，并对论文需要达到的几个目标进行了说明。

本文在单用户场景下的可验证对称加密搜索方案 GS-VSSE 中，提出了一种基于 MPT 和 IH 的验证索引，该验证索引独立于加密搜索方案中的索引，这种解耦使得该方案可以与任意的加密搜索方案结合，为其提供结果验证功能。同时该验证索引还支持用户数据更新，不需要在用户数据产生更新时重构该索引，只需要进行增量更改。在该验证索引的基础上，本文为云服务器和数据搜索用户提供了一套完善的结果验证方案：即由云服务器根据验证索引生成结果证明，由数据搜索用户根据该结果证明对加密搜索结果进行验证。该验证方案确保了云服务器的任何恶意行为都会被用户检测到，确保了加密搜索结果的新鲜性和完整性。特别需要说明的是，该方案能对云服务器返回的空搜索结果进行验证。以往的可验证加密搜索方案中，如果用户搜索的关键字不存在，云服务器不会返回任何证明。这使得一个恶意的云服务器可以对任何关键字声称不存在。本论文提出的验证方案不管关键字存在与否，都需要云服务器提供结果证明，这防止了该情况的产生。通过实验验证，本方案在计算开销和通信开销两个方面引入的开销都很小。与加密搜索方案结合后的实验表明，GS-VSSE 方案提供的结果验证服务带来的额外开销几乎可以忽略不计。

在 GS-VSSE 方案的基础上，本文又提出了一种多用户场景下的可验证对称加密方案 GM-VSSE。相对于单用户场景，多用户场景下的数据持有者和数据搜索用

户产生了分离，对数据新鲜性验证来说增加了其复杂性。本方案通过一种基于时间戳链的鉴别符来解决该问题，该鉴别符将验证索引的根哈希与时间戳进行了绑定，使得用户可以通过时间戳验证根哈希的新鲜性。该方案由数据持有者生成该鉴别符并上传给云服务器，数据搜索用户可以通过云服务器以拉取的方式来获得该鉴别符，降低了数据持有者的通信开销。通过实验证明，GM-VSSE 方案给数据持有者带来的通信开销很小，并且给数据搜索用户引入的验证延迟也可接受。

此外，本文还通过严格的安全性证明对上述两个方案进行了分析，证明了两个方案的机密性和可验证性，即不泄露用户数据和关键字明文信息，同时还保证了可以验证论文定义的数据新鲜性攻击和数据完整性攻击。

综上所述，本文提供了一种云存储环境下的安全加密搜索方案。该方案可以为各种各样的加密搜索方案提供结果验证功能，支持用户数据更新，支持单用户和多用户等多种场景，并且可以抵抗多种可能的安全性攻击，通用性非常高。

5.2　未来工作展望

本文提出的两种可验证对称加密搜索方案已具有较高的实用性，但未来仍有以下几个可以提升的点：

- 第一，本文提出的结果验证方案仅验证加密搜索结果在数量上的完整性以及数据本身的完整性。一些具有不同功能的加密搜索方案，例如支持多关键字查询的加密搜索方案，支持结果排序的加密搜索方案，这些方案对结果验证的功能需求更复杂。针对支持多关键字查询的加密搜索方案，验证方案需要支持对搜索结果数量的交并运算，而支持结果排序的加密搜索方案，验证方案需要支持对搜索结果顺序的验证。这些问题目前尚没有相关研究工作，今后可持续展开研究。
- 第二，本文提出的多用户对称加密搜索方案 GM-VSSE 基于一方写入多方读取的情况。在实际的云存储场景中，还存在多方写入多方读取，多方写入单方读取等情况，针对这些场景的研究目前尚没有。
- 第三，本文提出的两种可验证方案，云服务器的计算开销都在次线性级别。而目前已有常数级别计算开销的加密搜索方案，这促使我们寻找常数级别的可验证方案，以在性能上和常数级别的加密搜索方案进行匹配。

参考文献

Juels A, Kaliski Jr B S. Pors: Proofs of retrievability for large files[C]. Proc. of Computer and Communications Security(CCS) , 2007: 584–597.

Ateniese G, Di Pietro R, Mancini L V, et al. Scalable and efficient provable data possession[C]. Proc. of the International Conference on Security and privacy in communication netowrks (SecureComm) , 2008: 9.

Kamara S, Papamanthou C, Roeder T. Cs2: A semantic cryptographic cloud storage system [R] : Tech. Rep. MSR-TR-2011-58, Microsoft Technical Report (May 2011), http://research. microsoft. com/apps/pubs, 2011.

Wang Q, Wang C, Ren K, et al. Enabling public auditability and data dynamics for storage security in cloud computing[J]. IEEE Transactions on Parallel and Distributed Systems(TPDS), 2011, 22 (5): 847–859.

Stefanov E, van Dijk M, Juels A, et al. Iris: A scalable cloud file system with efficient integrity checks [C]. Proc. of Annual Computer Security Applications Conference (ACSAC) , 2012.

Kamara S, Papamanthou C. Parallel and dynamic searchable symmetric encryption[C]. Proc. of International Conference on Financial Cryptography and Data Security(FC) : Springer, 2013: 258–274.

Sun W, Liu X, Lou W, et al. Catch you if you lie to me: Efficient verifiable conjunctive keyword search over large dynamic encrypted cloud data[C]. Proc. of IEEE International Conference on Computer Communications(INFOCOM) : IEEE, 2015: 2110–2118.

Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores[C]. Proc. of Computer and Communications Security(CCS) , 2007: 598–609.

Curtmola R, Khan O, Burns R, et al. Mr-pdp: Multiple-replica provable data possession[C]. Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on : IEEE, 2008: 411–420.

Erway C C, Küpçü A, Papamanthou C, et al. Dynamic provable data possession[J]. ACM Transactions on Information and System Security (TISSEC), 2015, 17(4): 15.

Zhu Y, Hu H, Ahn G J, et al. Cooperative provable data possession for integrity verification in multicloud storage[J]. IEEE Transactions on Parallel and Distributed Systems(TPDS), 2012, 23 (12): 2231–2244.

Bowers K D, Juels A, Oprea A. Proofs of retrievability: Theory and implementation[C]. Proc. of the workshop on Cloud computing security (SCC) , 2009.

Wang C, Wang Q, Ren K, et al. Privacy-preserving public auditing for data storage security in cloud computing[M]. Proc. of IEEE International Conference on Computer Communications(INFOCOM) : IEEE, 2010a: 1–9.

Wang C, Ren K, Lou W, et al. Toward publicly auditable secure cloud data storage services[J]. IEEE Network, 2010, 24(4).

Wang C, Chow S S, Wang Q, et al. Privacy-preserving public auditing for secure cloud storage[J]. IEEE Transactions on Computers, 2013, 62(2): 362–375.

Zhu Y, Wang H, Hu Z, et al. Dynamic audit services for integrity verification of outsourced storages in clouds[C]. Proceedings of the 2011 ACM Symposium on Applied Computing : ACM, 2011: 1550–1557.

Song D X, Wagner D, Perrig A. Practical techniques for searches on encrypted data[C]. Proc. of Security and Privacy(S&P) : IEEE, 2000: 44–55.

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.

Kamara S, Papamanthou C, Roeder T. Dynamic searchable symmetric encryption[C]. Proc. of Computer and Communications Security(CCS) : ACM, 2012: 965–976.

Cash D, Jaeger J, Jarecki S, et al. Dynamic searchable encryption in very-large databases: Data structures and implementation[C]. Proc. of the Network and Distributed System Security Symposium (NDSS): volume 14 : Citeseer, 2014: 23–26.

Wang Q, He M, Du M, et al. Searchable encryption over feature-rich data[J]. IEEE Transactions on Dependable and Secure Computing(TDSC), 2016.

Boneh D, Di Crescenzo G, Ostrovsky R, et al. Public key encryption with keyword search[C]. International conference on the theory and applications of cryptographic techniques(EUROCRYPT) : Springer, 2004: 506–522.

Bost R, Fouque P A, Pointcheval D. Verifiable dynamic symmetric searchable encryption: Optimality and forward security.[J]. IACR Cryptology ePrint Archive, 2016, 2016: 62.

Kurosawa K, Ohtaki Y. Uc-secure searchable symmetric encryption[C]. Proc. of International Conference on Financial Cryptography and Data Security (FC) , 2012.

Chai Q, Gong G. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers [C]. Proc. of International Conference on Communications (ICC) , 2012.

Kurosawa K, Ohtaki Y. How to update documents verifiably in searchable symmetric encryption[C]. Proc. of International Conference on Cryptology And Network Security (CANS) , 2013.

Stefanov E, Papamanthou C, Shi E. Practical dynamic searchable encryption with small leakage[C]. Proc. of the Network and Distributed System Security Symposium (NDSS): volume 71 , 2014: 72–75.

Cheng R, Yan J, Guan C, et al. Verifiable searchable symmetric encryption from indistinguishability obfuscation[C]. Proc. of the ACM Symposium on Information, Computer and Communications Security(AsiaCCS) : ACM, 2015: 621–626.

Ogata W, Kurosawa K. Efficient no-dictionary verifiable sse[J]. IACR Cryptology ePrint Archive, 2016, 2016: 981.

Wang C, Cao N, Li J, et al. Secure ranked keyword search over encrypted cloud data[C]. Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS) , 2010c.

Merkle R C. A digital signature based on a conventional encryption function[C]. International conference on the theory and applications of cryptographic techniques(EUROCRYPT) , 1987.

Papamanthou C, Tamassia R, Triandopoulos N. Authenticated hash tables[C]. Proc. of Computer and Communications Security(CCS) , 2008.

Pugh W. Skip lists: a probabilistic alternative to balanced trees[J]. Communications of the ACM, 1990, 33(6): 668–676.

Goodrich M T, Tamassia R, Schwerin A. Implementation of an authenticated dictionary with skip lists and commutative hashing[C]. Proc. of DARPA Information Survivability Conference & Exposition (DISCEX) , 2001.

Zheng Q, Xu S, Ateniese G. Vabks: verifiable attribute-based keyword search over outsourced encrypted data[C]. Proc. of IEEE International Conference on Computer Communications(INFOCOM) , 2014.

Liu P, Wang J, Ma H, et al. Efficient verifiable public key encryption with keyword search based on kp-abe[C]. Proc. of Broadband and Wireless Computing, Communication and Applications (BWCCA) , 2014.

Yang Y, Bao F, Ding X, et al. Multiuser private queries over encrypted databases[J]. International Journal of Applied Cryptography, 2009, 1(4): 309–319.

Jarecki S, Jutla C, Krawczyk H, et al. Outsourced symmetric private information retrieval[C]. Proc. of Computer and Communications Security(CCS) : ACM, 2013: 875–888.

Sun S F, Liu J K, Sakzad A, et al. An efficient non-interactive multi-client searchable encryption with support for boolean queries[C]. Proc. of European Symposium on Research in Computer Security(ESORICS) : Springer, 2016: 154–172.

Bellare M, Goldreich O, Goldwasser S. Incremental cryptography: The case of hashing and signing [C]. Annual International Cryptology Conference(CRYPTO) : Springer, 1994: 216–233.

Wood G. Ethereum: A secure decentralised generalised transaction ledger[J]. Ethereum Project Yellow Paper, 2014, 151: 1–32.

Merkle patricia tree[EB/OL]. https://github.com/ethereum/wiki/wiki/Patricia-Tree.

Rlp code[EB/OL]. https://github.com/ethereum/wiki/wiki/RLP.

Bösch C, Hartel P, Jonker W, et al. A survey of provably secure searchable encryption[J]. ACM Computing Surveys (CSUR), 2015, 47(2): 18.

Sandhu R S, Coyne E J, Feinstein H L, et al. Role-based access control models[J]. IEEE Transactions on Computer, 1996, 29(2): 38–47.

Ferraiolo D F, Sandhu R, Gavrila S, et al. Proposed nist standard for role-based access control[J]. ACM Transactions on Information and System Security (TISSEC), 2001, 4(3): 224–274.

Enron_email dataset[EB/OL]. https://www.cs.cmu.edu/~enron/.

Mills D L. Internet time synchronization: the network time protocol[J]. IEEE Transactions on Communications, 1991, 39(10): 1482–1493.

Mills D, Martin J, Burbank J, et al. Network time protocol version 4: Protocol and algorithms specification: number 5905 [R] : RFC Editor, 2010.

Kopetz H, Ochsenreiter W. Clock synchronization in distributed real-time systems[J]. IEEE Transactions on Computers, 1987, 100(8): 933–940.

Elson J, Girod L, Estrin D. Fine-grained network time synchronization using reference broadcasts[J]. ACM Transactions on Operating Systems Review (SIGOPS), 2002, 36(SI): 147–163.

Zhou D, Lai T H. An accurate and scalable clock synchronization protocol for ieee 802.11-based multihop ad hoc networks[J]. IEEE Transactions on Parallel and Distributed Systems(TPDS), 2007, 18(12): 1797–1808.

致　谢

三年的研究生生涯如同白驹过隙，一晃而过。从小学到研究生，将近二十年的求学生涯即将结束，越长大也越珍惜当学生的机会。在这三年的时间里，从各位老师和同学身上，我不仅学到了专业知识，还学到了许多做人做事的道理。

首先要感谢的是我的导师李琦老师，他不仅在学术上给予了我许多指引与帮助，还给我提供了许多平台和机会，让我能在这三年时间里充分把握好时间，发挥所长。其次十分感谢香港城市大学的王聪老师，其治学之严谨，待人之亲善，让我对科研保有着敬畏之心，对自身的不足也有了更深刻的认识。另外，莫纳什大学的袁星亮老师，武汉大学的王骞老师也对我的科研工作给与了许多帮助，在此衷心感谢。还要感谢夏树涛老师，江勇老师，郑海涛老师和肖喜老师。夏老师为我们每一位同学的毕业花了很多心力，江勇老师的计算机网络课程和肖喜老师的随机过程让人印象深刻，郑海涛老师的羽毛球技十分了得，有幸观摩了几场，让人叹服。

除此以外，还要感谢我身边的同学朋友亲人们。感谢宋奇阳同学为科研工作提供的帮助，感谢室友陈杨、张云在生活上的互相扶持和理解，感谢张晓丽、吴珺等同学在我遇到挫折时的安慰，感谢孙晓梅、任青妹、冯博、李秉峻等同学在学业生活各方面的帮助。感谢妞妞同学研究生三年的陪伴，永远是我低落时期的强心剂。感谢我的父母，对我这个二十五岁还在啃老的女儿不离不弃。

此外，对我二十多年的求学生涯进行总结。我刚接触计算机是在小学，进机房要穿鞋套，操作系统还是 windows98，当年不敢乱动怕弄坏的东西，没想到多年以后竟成为了我的专业与职业。选择这个行业更多的是机缘巧合，高考完填下志愿的那一刻起，冥冥中已经注定了这之后许多年的命运。在这七年的时间里，我除了学习成绩尚可以外，在自我学习与自我管理上都有许多欠缺。即将进入的互联网行业处在不断地进步与更新中，求学生涯虽止于此，但希望学习不止于此。

最后，本课题承蒙国家自然科学基金 (61572278) 及国家重点研发计划专项 (2016YFB0800102) 资助，特此致谢。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1993 年 02 月 02 日出生于浙江省海宁市。

2011 年 09 月考入北京邮电大学计算机学院计算机科学与技术专业。

2015 年 07 月本科毕业并获得工学学士学位。

2015 年 09 月免试进入清华大学计算机科学与技术系攻读工程硕士学位至今。

发表的学术论文

[1] Jie Zhu, Qi Li, Cong Wang, Xingliang Yuan, Qian Wang, Kui Ren. Enabling Generic,Verifiable, and Secure Data Search in Cloud Services. IEEE Transactions on Parallel and Distributed Systems(TPDS), DOI: 10.1109/TPDS.2018.2808283)

研究成果

[1] 李琦, 朱洁, 王骞. 一种可验证的加密搜索方法: 中国, 201711277295.7. (中国专利申请号)

[2] 李琦，朱洁，陈艳毓，李漓春. 检测软件定义网络 (SDN) 中的路由环路的系统和方法：中国, 201810437997.5. (中国专利申请号)

[3] Qi Li, Jie Zhu, Yanyu Chen, Lichun Li. System and Method for detecting routing loops in a Software Defined Network (SDN): SG, 10201703959R. (新加坡专利申请号)