学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

密级\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

（黑体5号）

武汉大学本科毕业论文

（1号宋体居中）

MC-JXT:

多用户场景下支持Join查询的

对称可搜索加密方案

（2号黑体居中，标题行间距为32磅）

院（系）名 称：XXX XXX

专 业 名 称 ：XXX XXX

学 生 姓 名 ：X X X

指 导 教 师 ：XXX 教授

（宋体小3）

二○二三年六月

**郑 重 声 明**

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。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，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名： 日期：

摘 要

近年来数据安全问题日益突出，Google、Sony、Yahoo等公司都发生过大规模的用户数据泄露事件。为缓解外包数据安全问题，研究人员提出了对称可搜索加密技术，旨在不损坏云端数据隐私的同时，为用户提供（密态）数据检索服务。

对称可搜索加密方案从用户场景上可以分为单用户和多用户两种场景。单用户场景中用户既是数据提供者，又是检索者，而多用户场景中数据检索者和数据提供者是不同的两方。目前虽已有诸多关于对称可搜索加密的研究进展，但大部分工作仅支持单用户场景下的简单查询，如何在多用户场景下设计支持复杂查询（例如Join查询）的对称可搜索加密方案已成为当前的研究热点。近期，Seny Kamara[1]、David Cash等人[2]陆续提出了支持Join查询的对称可搜索加密方案，然而他们的方案都依然需要较大的存储开销，难以支持大数据场景下的功能需要。

针对上述问题，本课题的研究目标是设计一个适用于多用户场景，且能够高效支持Join查询的对称可搜索加密方案，并提供相应的具体实现、性能测试和安全证明。

关键词：对称可搜索加密；多用户场景；Join查询；安全数据检索；数据隐私安全

**ABSTRACT**

With cloud computing technology’s rapid development, more individuals and organizations outsource large-scale data storage and processing to third-party servers to reduce local information data management overhead. However, untrusted third-party servers pose a threat to users’ data. To mitigate outsourced data security problems, researchers have proposed searchable symmetric encryption(SSE) techniques that provide (cryptographic) data retrieval services to users without compromising cloud data privacy.

Searchable encryption schemes can be divided into single-client and multi-client scenarios. The client in the single-client scenario is both the data provider and the querier, while the data querier and the data provider in the multi-user scenario are two different parties. Most of the work in SSE only supports simple queries in single-client scenarios, and how to design symmetric searchable encryption schemes that support complex queries (e.g., Join queries) in multi-client scenarios has become a current research hotspot. Recently, Seny Kamara，David Cash and others have proposed symmetric searchable encryption schemes supporting Join queries, however, all of their schemes still require large storage overhead and are difficult to support the needs in big data scenarios.

To address the above issues, the research goal of this article is to design a SSE scheme in multi-client scenarios that can efficiently support Join queries. We will also provide corresponding implementations and security proofs.

**Keywords:** searchable symmetric encryption; multi-client scenarios; Join queries; secure data retrieval; data privacy security

目 录

[1 绪论 1-7](#_Toc132295657)

[1.1 研究背景及意义 1-7](#_Toc132295658)

[1.1.1 研究背景 1-7](#_Toc132295659)

[1.1.2 研究意义 1-8](#_Toc132295660)

[1.2 国内外研究现状和发展趋势 1-8](#_Toc132295661)

[1.2.1 对称可搜索加密 1-8](#_Toc132295662)

[1.2.2 多用户对称可搜索加密 1-9](#_Toc132295663)

[1.3 研究内容 1-10](#_Toc132295664)

[1.4 章节安排 1-11](#_Toc132295665)

[2 相关工作 2-12](#_Toc132295666)

[2.1 对称可搜索加密 2-12](#_Toc132295667)

[2.1.1 关系型数据库 2-12](#_Toc132295668)

[2.1.2 对称可搜索加密定义 2-12](#_Toc132295669)

[2.1.3 对称可搜索加密正确性 2-13](#_Toc132295670)

[2.1.4 对称可搜索加密安全性 2-13](#_Toc132295671)

[2.1.5数据结构 2-14](#_Toc132295672)

[2.2 Join查询 2-14](#_Toc132295673)

[2.2.1 Join查询的概念 2-14](#_Toc132295674)

[2.2.2 Join属性与Join关键词集 2-15](#_Toc132295675)

[2.2.3 Join查询的形式化定义 2-15](#_Toc132295676)

[2.3 JXT协议 2-15](#_Toc132295677)

[2.3.1 实现Join查询的思路 2-16](#_Toc132295678)

[2.3.2 两个子查询的实现 2-17](#_Toc132295679)

[2.3.3 协议具体流程 2-17](#_Toc132295680)

[3 MC-JXT协议设计 3-20](#_Toc132295681)

[3.1 多用户对称可搜索加密定义 3-20](#_Toc132295682)

[3.2 MC-JXT协议构造 3-21](#_Toc132295683)

[3.2.1 MC-JXT——算法 3-21](#_Toc132295684)

[3.2.2 MC-JXT——算法 3-24](#_Toc132295685)

[3.2.3 MC-JXT——算法 3-26](#_Toc132295686)

[3.3 MC-JXT协议正确性分析 3-29](#_Toc132295687)

[3.4 MC-JXT协议安全性分析 3-30](#_Toc132295688)

[3.4.1 攻击者为服务端 时的安全性 3-30](#_Toc132295689)

[3.4.2攻击者为客户端 时的安全性 3-30](#_Toc132295690)

[3.5 MC-JXT协议理论开销分析 3-32](#_Toc132295691)

[3.5.1 存储开销 3-32](#_Toc132295692)

[3.5.2 计算开销 3-33](#_Toc132295693)

[3.5.3 通信开销 3-33](#_Toc132295694)

[4 协议实现以及性能测试 4-34](#_Toc132295695)

[4.1 协议实现 4-34](#_Toc132295696)

[4.2 性能测试 4-34](#_Toc132295697)

[4.2.1 数据集 4-34](#_Toc132295698)

[4.2.2 实验设计以及实验结果 4-35](#_Toc132295699)

[5 总结与展望 5-40](#_Toc132295700)

[5.1 本文总结 5-40](#_Toc132295701)

[5.2 工作展望 5-40](#_Toc132295702)

[附录 5-41](#_Toc132295703)

[参考文献 5-45](#_Toc132295704)

[致谢 5-48](#_Toc132295705)

# 绪论

## 研究背景及意义

### 研究背景

随着云计算技术的快速发展，越来越多的个人和组织将大规模数据的存储和处理工作外包给第三方服务器，以降低本地信息的数据管理开销以及系统维护成本。然而，数据安全威胁也随之而来，恶意的云端服务器管理员或者不具有访问权限的服务器用户可能会在执行正常存储和查询功能之外，试图获取数据库信息，从而导致用户隐私数据的泄露。近年来数据安全问题日益突出，Google、Sony、Yahoo等公司都发生过大规模的用户数据泄露事件。

为了解决云端数据隐私安全问题，我们必须要对上传到云端上的数据进行加密，这样才可以阻止不可信的服务器获取数据明文信息。但如果使用传统的加密算法进行加密，由于数据对服务器是不可见的，此时数据的可检索性就会消失，服务器则以提供其原有的服务，如接受客户端的请求，提供客户端请求的数据。此情况下一种保持数据可检索性的简单方法是，客户端将整个加密的数据库下载下来，本地将整个数据库进行解密，然后查询所需要的信息，但是这种做法开销太大，显然不现实。还有一种方法就是我们把密钥发送给服务端，让服务端解密数据并进行查询，但是显然此种方法仍然会造成安全性问题。因此我们需要能直接在密态数据上提供完整数据检索服务的技术。

能实现此种需求的技术有很多，包括对称可搜索加密（SSE, Searchable Symmetric Encryption），不经意随机存取（ORAM, Oblivious Random Access Memory），属性保护加密（OPE, Property-Preserving Encryption）等等，其中目前得到广泛研究和应用的是对称可搜索加密技术。对称可搜索加密是一种特殊的加密技术，可以实现在不泄露明文信息的前提下，对密文数据进行匹配和检索。在对称可搜索加密场景中，客户端首先用特殊加密算法将自己的数据库（或者数据库索引）加密并发送给服务端，服务端接收并存储下来；当客户端要发其查询的时候，其根据自己要查询的关键字生成一个对应的搜索凭证，并发送服务端；服务端接收到搜索凭证，将其和密态数据库中的记录进行匹配，如果达到预期匹配结果，就将此密态记录返回给客户端。客户端接收密态查询结果，并解密得到最后的查询结果。

最初的对称可搜索加密方案仅仅支持单用户场景下的单关键字查询，难以满足实际应用的需要，越来越多的对称可搜索加密方案支持更丰富的应用场景和查询类型。这里的应用场景主指客户端和服务端的架构，起初的方案仅仅支持单用户单服务端的场景，目前包括多用户单服务端、单用户多服务端、多用户多服务端等场景的可搜索加密方案正被广泛研究。而查询类型则是指数据库查询类型，起初的方案仅仅支持单关键字查询，后来支持包括多关键字联合查询、布尔查询、模糊查询、Join查询等更多查询类型的对称可搜索加密方案陆续出现。

### 研究意义

目前虽已有诸多关于对称可搜索加密的研究进展，但大部分工作近支持单用户场景下的简单查询，如何在多用户场景下设计支持复杂查询的对称可搜索加密方案已成为当前的研究热点。Join查询就属于上述待研究的“复杂查询”之中，也是关系型数据库中的一类基础查询，近期，Seny Kamara、David Cash等人陆续提出了支持Join查询的对称可搜索加密方案，然而他们的方案都需要较大的存储开销，难以支持大数据场景下的功能需要。

本课题同时解决了上述如何在多用户场景下支持复杂查询和如何提高Join查询的效率两个关键性问题，设计出了多用户场景下高效支持Join查询的对称可搜索加密方案MC-JXT（Multi-Client Join Cross-tag Protocol），进一步提高了对称可搜索加密在数据共享场景下的应用价值。

## 国内外研究现状和发展趋势

### 对称可搜索加密

Song等人在2000年首次提出了基于对称算法下的SSE方案实现。其加密方案使用确定性加密（Determined Encryption）方案将文档中的每一个关键字进行加密，同时在密文中附加一个特殊形式的哈希值；在检索过程中，服务端进行密文单词扫描，将文档中的每个关键字密文中的哈希值提取出来，并和客户端发送来的搜索凭证进行匹配计算得到查询结果，此方案缺陷在于查询效率低下。Goh等人在2003年提出了安全索引（Secure Index）的概念，其方案首先对明文数据库中的每个文档建立基于布隆滤波器的索引，然后将加密后的索引和数据库一并发送给服务端；服务端通过验证待检索关键字是否存在于文档的索引中来确定其在文档中的存在性。此方案相比传统方案大大降低了查询复杂度，后续许多对称可搜索加密方案均采用类似建立密态索引的思想。后续Curtmola等人延续了Goh方案中安全索引的思想，提出了CGK方案，此方案建立基于关键字的倒序索引，为每个关键字维护了一个加密链表，链表的每个node包含一个含有此关键字的文档ID以及解密下一个node的密钥。CGK实现了非线性（最优的）的查询时间开销，此后许多查询高效的对称可搜索加密方案均在其基础上改进而来。

越来越多支持复杂查询的对称可搜索加密方案陆续出现。联合查询方面， Golle等人提出了第一个支持联合查询的对称可搜索加密方案，其核心思想是为文档设定一个关键字域，在查询时，客户端根据自己要查询的关键字连同其域生成一组搜索凭证并发送给服务端，服务端将搜索凭证和密态文档进行对比，从而确定查询结果；Ballard等人提出了一个定义在标准模型下并支持联合查询的可搜索加密方案，该方案基于基于Shamir秘密分享以及关键字域技术；Wang等人使用双线性映射替代了关键字域方案，首次提出了不需要关键字域的，支持联合查询的可搜索加密方案。Cash等人提出了首个支持非线性查询时间复杂度，以及联合查询、布尔查询的可搜索加密方案OXT（OXT, Oblivious Cross-tag Protocol），该方案在大规模数据场景中表现优秀，并且支持任何结构的数据，是里程碑式的工作。Join查询方面，Seny Kamara、David Cash、Florian Hahn陆续提出了支持Join查询的对称可搜索加密方案；近日，Jutla等人提出了一种高效支持Join查询的可搜索加密方案(JXT, Join Cross-tag Protocol)，大大降低了传统方案的空间开销。

### 多用户对称可搜索加密

越来越多支持不同的应用场景可搜索对称方案被提出。为了支持数据共享的场景需求，多用户可搜索加密方案应运而生。在多用户场景（我们此处所说的多用户，默认是指多用户单服务端场景）中，用户被分离为数据提供者和数据检索者两种角色，数据提供者可以将其加密数据上传至服务端，以供其他数据检索者实体在通过其认证的前提下检索其存储在云端的加密数据，在这个过程中服务端不会得到任何额外的数据信息。

Curtmola等人在2006年提出了基于广播加密技术的多用户可搜索加密方案。在此方案中，数据提供者将密钥分发给准许检索群组中的用户，以便这些用户生产搜索凭证并进行数据检索； Raykova等人在2009年提出了基于重路由加密技术的多用户对称可搜索加密方案。此方案引入一个第三方实体查询路由来进行查询的认证，检索者将加密后的查询发送给查询路由，若认证通过，查询路由将查询进行转换并发送给服务端进行查询，以此实现检索者匿名地进行数据检索。以上工作都仅仅支持单关键字查询，Jarecki等人在2013年提出了支持联合查询和布尔查询的多用户对称可搜索加密方案，此工作对单用户场景下，支持上述两种查询功能的OXT协议进行了多用户扩展。

## 研究内容

如何在多用户场景下设计支持复杂查询的对称可搜索加密方案是当前的研究热点。本文致力于设计出多用户场景下高效支持Join查询的对称可搜索加密方案，以提高对称可搜索加密技术在数据共享场景下的应用价值。本文包含的主要内容有如下几项:

（1）密码学协议设计

第一部分内容主要介绍本文设计的多用户场景下支持Join查询功能的MC-JXT协议。这部分会给出MC-JXT协议的技术路线、解决以往瓶颈的方法以及具体算法流程。

（2）协议安全性、正确性和性能分析

第二部分内容主要对（1）中所涉及协议进行功能正确性、协议安全性和理论开销进行分析。正确性方面本文会给出协议的正确性定义和证明；安全性方面本文会给出协议的安全性定义，泄露函数以及基于模拟的安全性证明；性能方面本文会给出协议的存储开销、计算开销、通信开销三方面的分析。

（3）协议实现及性能测试

第三部分内容主要介绍协议的一个具体代码实现方式，并设计实验测试协议各方面性能的综合表现和性能主要制约因素，实际验证协议性能表现与理论分析的契合性。

（4）研究展望与开放性问题

第四部分内容主要介绍本文所设计的协议目前还有哪些没有解决或者涵盖到的问题，以及未来的改进方向。

## 章节安排

本文共分为六个章节，具体内容安排如下：

第一章绪论。本章主要介绍本文的研究背景，研究目的和研究意义。本章大致梳理了当前对称可搜索加密领域的研究进展和国内外研究现状，最后对本文的研究内容进行总结陈述。

第二章相关工作。本章主要介绍和本文研究问题相关的背景知识。首先介绍对称可搜索加密的语义、正确性、安全性定义。接着介绍Join查询的相关概念以及Join查询在关系型数据库的重要性。最后介绍目前支持Join查询的对称可搜索加密的研究方案JXT协议的构造。

第三章MC-JXT协议设计。本章主要阐述我们基于JXT方案，设计多用户场景下支持Join查询的对称可搜索加密方案的思路、方案的先进性以及所设计的协议的具体算法。我们这里会着重对协议的可行性，正确性、安全性以及协议的理论性能进行分析。

第四章协议实现和性能测试。本章将介绍MC-JXT协议的代码实现方式和其在选定数据集上测试协议的性能表现，并论证测量的实验数据和理论分析之间的一致性。

第五章总结与展望。本章将会对本文前面章节的工作内容进行总结，阐释协议的优势以及目前存在的局限性。同时还会提出未来可以优化改进的方向和开放性问题。

# 相关工作

## 对称可搜索加密

### 关系型数据库

一个关系型数据库是一个表的集合，即 。每张表 由若干记录组成，我们将关键词定义为关系型数据库中的属性以及对应值组成的键值对，那么每条记录就是一个关键词集合，故每张表可表示为 。此处 是一个在 上的字符串，代表着第 张表中第 条记录的标号； 代表着第 张表中第 条记录的关键词集合。为了方便描述，我们用 指代表 中所有关键词组成的空间。

上述的记录标号 是一个在数据提供者在建立数据库时，标识各个记录的值，其主要作用是为了方便数据库查询，减小通信开销。服务端（数据库存储方）在执行查询之后，得到满足查询的记录标号并返回查询者，而不是直接返回符合查询的记录。在对称可搜索加密的场景下， 通常是原标号置换加密后的值。在本文中，我们假定记录标号同时与记录和记录所在的表相关，也就是说不同的表之间没有重合的记录标号。

值得一提的是，我们在数据库上执行查询时，通常不会直接在数据库上查询满足条件的记录，因为这样做开销太大，而是会先在数据库上建立索引结构，然后在数据库索引上执行查询，以提升查询效率。数据库索引有很多种，其中一种常用的，查询效率较高的索引是倒序索引。我们令 表示第 张表中，包含关键词 的记录标号集合，即 ，则倒序索引中的每一项由一个数据库关键词索引，其值为包含此关键词的记录标识符集合，即：

### 对称可搜索加密定义

本节我们给出对称可搜索加密的形式化定义。一个对称可搜索加密方案主要包含涉及服务端和客户端的两个算法: 和 ，方案构造如图 2‑1所示，算法具体语法如下：

（1） ：此算法由数据提供者进行，其输入为明文数据库 ，输出为 对应的加密数据库 和其对应的私钥 。值得注意的是，此处数据提供者在单用户场景下与客户端为同一实体，而在多用户场景下为不同实体。

（2） ：此算法由客户端和服务端共同执行，客户端输入为私钥 以及要发起的查询 服务端输入为 中输出的密文数据库 。最终，在客户端和服务端进行交互之后，客户端输出满足查询的记录标识符集合 ，而服务端没有输出。

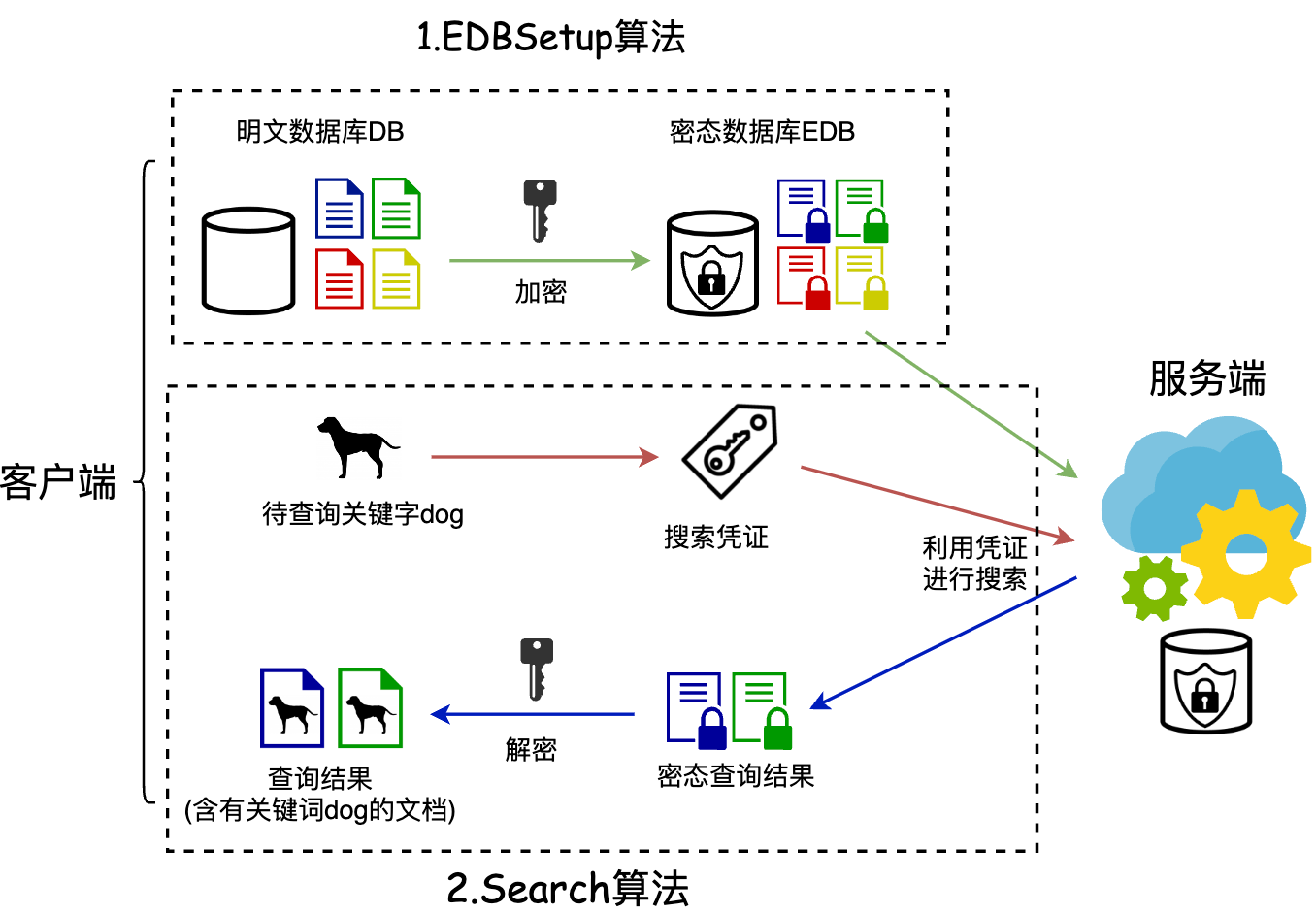


图 2‑1 对称可搜索加密

### 对称可搜索加密正确性

对称可搜索加密方案的正确性定义如下：对于所有输入的数据库 和查询 ，在协议执行完上述描述的 和 算法之后，如果客户端的输出等于 的话，那么我们就称此方案满足正确性定义，反之，则不满足。

### 对称可搜索加密安全性

对称可搜索加密主要考虑攻击者为服务端时的安全性，其语义安全定义在由泄露函数 参数化的理想/现实范式框架下。此处泄露函数 也就是指协议交互过程中客户端向攻击者（即服务端）泄露的信息上界。在攻击者可以自由选取数据库和查询的前提下，如果攻击者与协议交互的视角，都可以被一个模拟算法只通过泄露函数 模拟出来，那我们就说此协议是安全的。

### 数据结构

JXT协议以及本文设计的MC-JXT协议都用到了一种称为 的数据结构，其本质就是一个单关键字可搜索加密的实现方案，这里我们放到这一节一同介绍。实际上就是一种密态倒序索引实现，与明文倒序索引不同的是，中的键和相应的值都是以密态的方式存储的，其安全性和正确性满足对称对称可搜索加密的定义。

我们假设关系型数据库 中 对应的倒序索引为 ，即有：

则关于 有三个算法:

（1）。此算法接收基于数据库 的明文倒序索引 作为参数，并输出在此索引上建立的密态倒序索引 结构和用到的密钥，输出的 是一个数组，其中的元素对应于明文索引 。

（2） 。此算法用于生成 上的查询凭证，算法输入要查询的表的标号 ，关键字 和 密钥 ，输出一个相应的搜索凭证 。

（3） 。此算法接受 、要查询的表的标号 中生成的 , 最终返回包含 的记录标识符集合 。

## Join查询

### Join查询的概念

在数据库管理系统中，Join运算是指根据两个或多个表之间指定的共有属性进行组合，并生成一个新表的过程。其作用就是在表之间建立一种关系，反馈给用户多个表中的不同实体的信息，给数据库查询带来了极大的灵活性。Join查询就是指要需要查询来自多个表的字段的数据库查询，在数据库管理系统中，大多数复杂的查询都涉及Join查询。要实现查询不同表字段的功能，通常我们需要先计算出多个表之间的Join表，然后进一步在Join表进行字段的匹配和查询。

### Join属性与Join关键词集

我们称Join运算中所涉及的，表之间的共同属性为Join属性。具体地，我们将 和其他表共有属性表示为 ，为Join属性的个数。这个属性集合时建立数据库伊始就确定的。同时，我们将记录 其Join属性集 上的关键词集合称为其Join关键词集，记为：

### Join查询的形式化定义

对于一个涉及 和 的Join查询 ，其形式化定义为:

其中 , 是我们要查询的，分别来自 和 的关键词，而则是要查询的，和 之间的Join属性集合。我们用 指代Join查询的结果，实际上是一个记录标识符对的集合，即：

，且 (同时满足:

上述表达式的含义为：,分别为 和 中包含 的记录，且这两条记录之间在Join属性集 上有相同的关键词集合。

## JXT协议

JXT协议（Join Cross-Tag Protocol）由Jutla等人在2021年提出，是首个无需预运算过程，并且能高效支持Join查询的对称可搜索加密方案。学术界中传统的支持Join查询的可搜索加密方案在建立密态数据库时往往需要预先计算待查询表之间的Join表。而由Join运算的性质可知，这一步可能会造成存储空间的爆炸性增长，从而导致密态数据库增长规模很大。JXT协议的优势主要在于此方案不需要任何上述的预运算，但仍然能实现对称可搜索加密场景下的Join查询同时保持可靠的安全性，这使得JXT协议在大规模数据场景下有很大的性能优势。

### 实现Join查询的思路

JXT协议实现Join查询的主要思路是将一个Join查询拆分成多个其他查询类型的“非Join查询”子查询 ，然后分别实现子查询，从而实现在不预先计算表之间Join的前提下仍然支持高效Join查询。下面我们考虑一个在表A（tabA）和表B（tabB）之间，查询分别来自两个表的字段的Join查询示例。假设表A和表B之间有Join属性personID，表 A有待查询属性name，表 B有待查询属性city，考虑如下Join查询 ：

我们可以将此查询拆分为 和 两个子查询。 子查询在表A中查询满足（name, NN）字段的记录的personID值集合:

我们将子查询的结果集合记为 。而 子查询则是在表B中查询满足（city, CC）且personID属性上的值处于 查询结果之中的记录：

我们遍历 查询结果中的每个personID值，在表B中寻找personID匹配并且city字段也匹配的记录。最终的查询结果为 ，是 子查询结果中每个personID对应的记录，是表B中匹配(city, CC)，并且匹配 记录对应的personID字段的记录。

总结以上示例，我们可以将一个涉及两个表、两个字段的Join查询 拆分成 两部分查询。在 中查询包含 的记录的Join关键词集记为:

然后 在 中进行一个联合查询，查询包含 同时Join关键词集在 查询结果当中的记录。

满足

在执行完 之后，每一对匹配的 就是查询的结果。其正确性是显然的，和有着相同的Join关键词集，并且各自匹配两个查询字段，所以这两个记录Join得到的新记录必然是匹配查询 的。

### 两个子查询的实现

其实就是一个单关键字查询，为了增加 查询的效率，我们在数据库索引中的每个单元增加一个字段，事先就把每条记录对应的Join关键词与其记录标识符共同存储起来，即将倒序索引中每个 对应的 替换为

与此同时，倒序索引更新为 。通过这样做，我们仅仅查询表中包含 中的记录就可以直接获得所需的Join关键词集，以节省计算开销。

是一个联合查询，第二个字段是 查询得到的Join关键词，我们要完成 查询，只需先将满足 的记录查询出来，然后再逐个验证此记录的Join关键词是否在 查询的结果之中。此处我们需要大量的验证满足 的记录标号同查询结果中每一个Join关键词的匹配关系。为了高效地完成上述验证，我们建立了一个叫做 的查找表。在建立密态数据库时，我们为每一张表生成对应的 ，其每一单元存储着一个记录和其对应的Join关键词集，具体定义如下:

此处 是伪随机函数。这样通过 ，我们就可以快速确定一个给定的记录 和一个给定的Join关键词集 是否匹配，只需验证 是否成立即可。

### 协议具体流程

JXT协议总共包含两个算法，分别是 和 ，下面我们首先介绍一个初级版本的JXT协议，然后再引出最终的JXT协议构造。

#### 初级JXT协议构造

由数据提供者执行，接受明文数据库作为输入，输出密态数据库和一组密钥，后续被存储在服务端上以供查询。中，对于每张表，数据提供者首先建立其对应的 ，即将每条记录的标识符 和其对应的Join关键词集 的匹配关系存储下来，正如2.3.2。在JXT中 ， 和 分别是 和 连同每个Join属性t经过另一个伪随机函数计算之后的值。然后，数据提供者对每张表建立密态倒序索引，即 ，它存储着每个关键词 和包含该关键词的记录标识符集合。 由搜索凭证 进行索引，正如2.3.2所提到的那样，不仅是个记录标识符集合，还存储着各个标识符对应的Join关键词集(伪随机函数值形式)，其构造为:

为经过加密的 ，而 和 的含义就像前面所提到的那样，用于后续的 查询。

过程从更高的层面来看就是一个涉及客户端和服务端的两方计算，客户端的输入是Join查询，服务端的输入是密态数据库 。最终客户端输出符合Join查询的记录标识符集合，服务端不输出任何信息。客户端选择Join查询，首先进行 查询，客户端计算出 并发送给服务端，服务端用检索得到 ，得到满足 的记录对应的Join属性集 。然后就是查询的执行过程，客户端计算出 并发送给服务端，服务端用其检索出 ，得到满足的记录以及其对应的 。接着服务端遍历每一组 ，并验证对于每个 是否成立，也就是验证 中的记录 和 中的 是否有相同的Join关键词集，如果满足，则服务端返回 给客户端，客户端进一步解密得到一对查询结果 。

#### 最终的JXT协议构造

显然初级JXT协议的构造是不安全的。一方面，的构造直接向服务端暴露了 和 ，同时 过程中也向服务端暴露了 中的元素。为了解决此安全性问题，我们需要对原协议进行一些修改。首先在 过程中我们给 和 加上掩码，在 中存储掩码后的值，此时 中每个单元表示为：

其次在 中，我们使用掩码后的 和 去计算 并进行验证，这对掩码的设计提出了要求。

改进过后的 中 每个单元内容为 ，其中

这里针对 和 有两组掩码值，分别为 和 ，这两组掩码值都是关于 的伪随机函数值，唯一的区别就是使用了不同的密钥 。 在计算过程中加入了计数值 ， 中每个单元都有一个对应的 值， 存在的意义是使得 中不同单元的掩码值不相同。

在改进后的Search中，客户端额外向服务端发送两组辅助查询的token：

,

其中 ，。服务端利用这两组token就可以在 和 被掩盖的情况下仍然计算出正确的 。服务端用两个计数器分别遍历 和 ，并验证这每对记录是否匹配。对于查询中的每一个Join属性，即每个 ，分别计算

最后计算 是否属于，如果全部满足条件，则将对应的一对密态标识符返回给查询者，查询者进行解密得到查询结果。

# MC-JXT协议设计

本文对JXT协议进行了多用户场景拓展，提出了多用户场景下支持Join查询的对称可搜索加密方案MC-JXT。MC-JXT保持了JXT高效实现Join查询的特性，同时可以支持多个客户端向服务端提供数据和发起数据检索请求，提升了可搜索加密在数据共享场景下的应用价值。

## 多用户对称可搜索加密定义

不同于单用户场景中客户端既提供数据又检索数据，多用户对称可搜索加密方案（以下称MC-SSE）中的客户端被分离为数据提供者 和数据检索者 两种角色。持有明文数据库 以及其自己指定的查询政策 ；服务端 存放着供检索的密态数据库 ； 可以发起数据库查询 ，并得到相应的查询结果 ，具体过程为： 先将 发送给 进行审查，如果查询满足 ，则 将与 对应的搜索凭证分发给 ，然后 就可以使用搜索凭证与服务端进行交互，并最终获取到查询结果，如果查询不满足 ，则 告知 查询失败，协议终止。

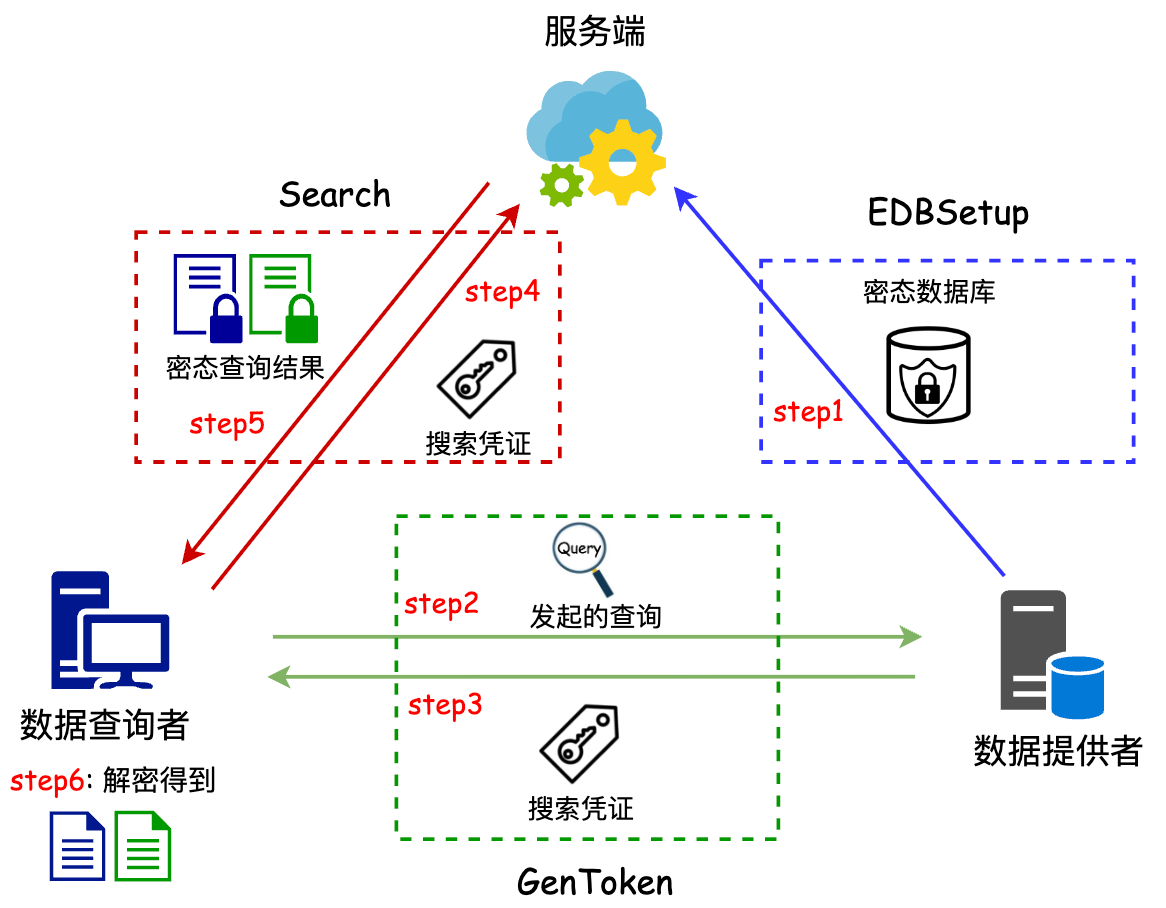


图 3‑1 多用户对称可搜索加密

总而言之，多用户场景下 把自己数据的查询权外包给了第三方实体 ，并要求 在从自己这里获得搜索凭证之后才能进行查询。 对于以上过程， 的安全性要求有两个，一方面是希望密钥数据库本身以及其检索过程尽可能地不泄露信息给，另一方面希望 只能发起它实现规定的查询。以上所描述的场景在数据共享方面有着很高的应用价值，用户希望把自己的数据上传到云服务器分享给其他用户，但不希望云服务器和其他数据使用者了解到超出规定的信息。

MC-SSE包含 三个算法。首先 运行 生成 和密钥 ；接着以 发送来的查询 和密钥 作为参数运行 算法，并产生搜索凭证 ；然后 和 共同运行 协议，输入 ，输入 ，验证 是否由 签发，如果是则继续进行查询，最终输出查询结果，不输出任何信息，否则协议终止，具体结构见图 3‑1。

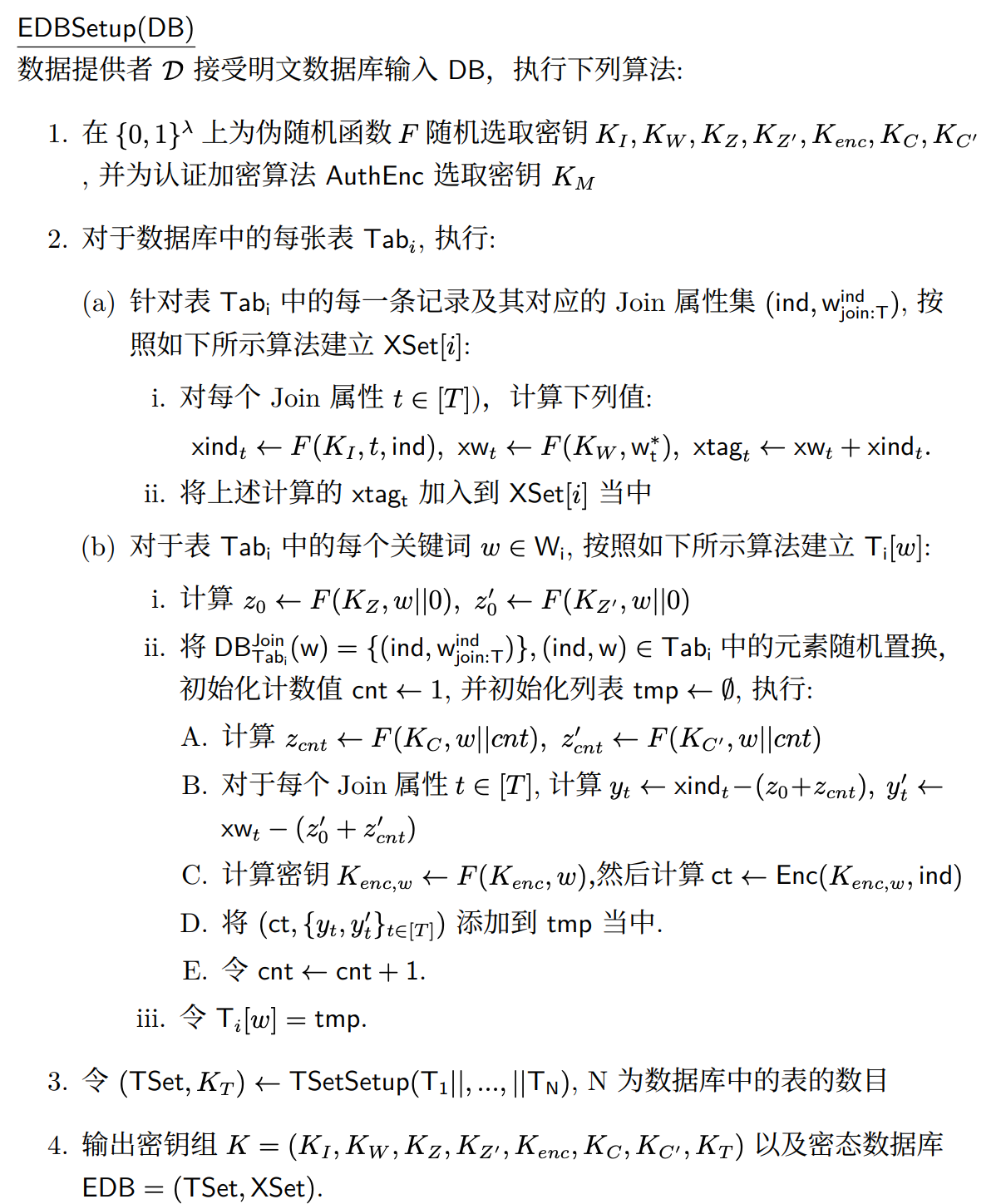
## MC-JXT协议构造

本节介绍MC-JXT协议的构造。MC-JXT协议是对2.3.3中介绍的JXT协议的多用户扩展，协议同JXT协议一样支持高效的Join查询，同时满足上述MC-SSE的定义。MC-JXT协议同样由 三部分算法构成，具体结构见图 3‑2。

### MC-JXT——算法

MC-JXT协议的 算法同JXT协议大致相同，作修改的地方主要有两处。首先是生成 和 所采用的掩码为了适应多用户可搜索加密场景而做出了改变；其次是算法输出的密钥组增加了密钥 ，此密钥由 和 共享，用于让 验证 发送过来的 是否由 签发。

MC-JXT协议对数据库中的每张表建立对应的 结构和 结构。我们引用前面的数学定义，假设考虑的表为 ，其关键词空间为 ，Join属性数目为 , 记录 对应的Join关键词集为，具体算法如算法 1所示。



算法 1 MC-JXT-EDBSetup

#### 生成过程

存储着每条记录和其对应的Join关键词集的匹配关系，MC-JXT协议的 构造同JXT协议基本相同。对于表 中的记录 ，针对每个Join属性 ，计算下列值:

是在上随机选取的密钥，是选定的伪随机函数。接着算法计算上面描述的“匹配关系”:

最后，算法将 加入到 。

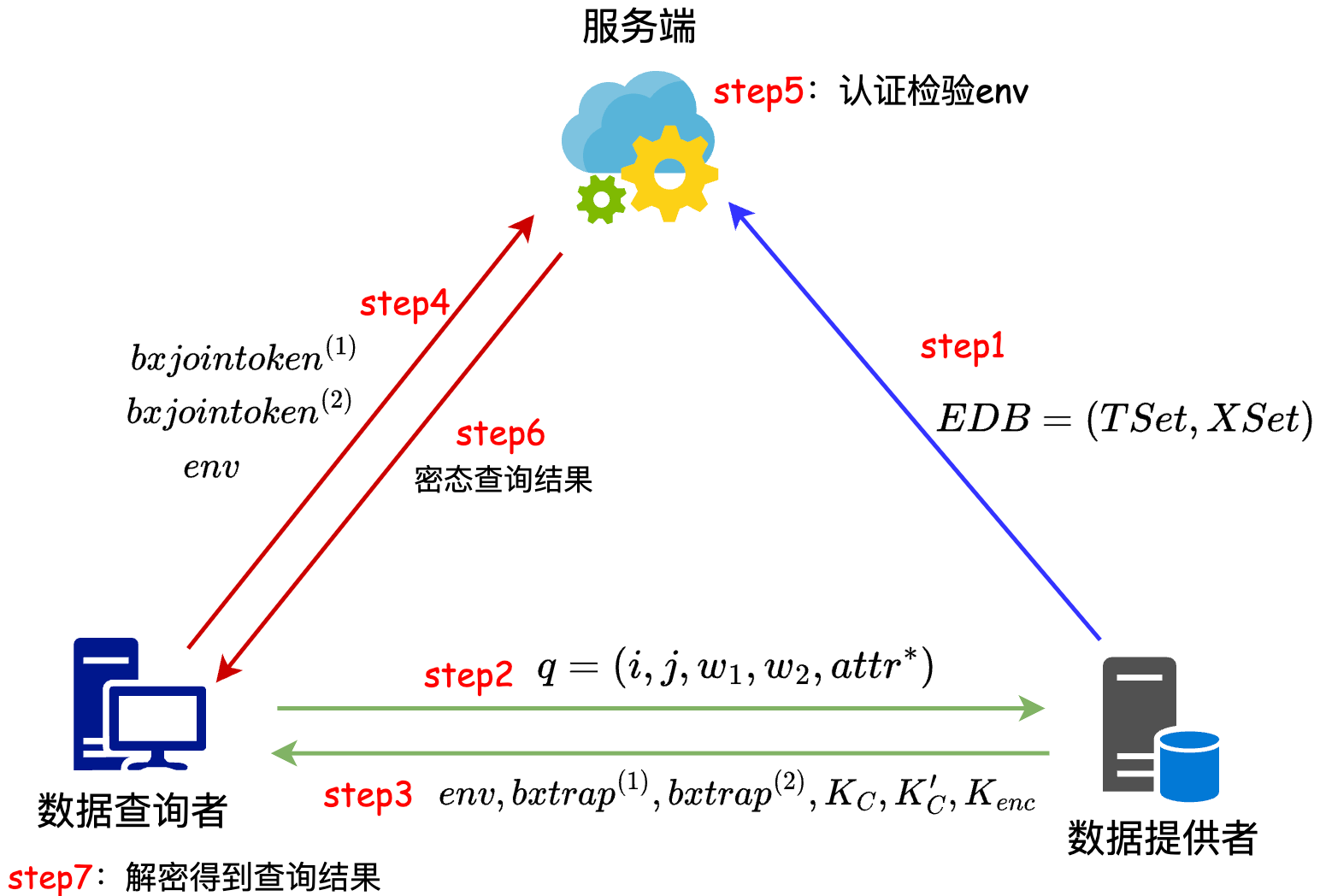


图 3‑2 MC-JXT协议结构图

#### 生成过程

对于每一个关键词 ， 存储着经过加密的 中包含 的记录标识符集合以及相应的支持Join查询功能的额外的字段。 算法建立 的步骤如下:

（1）生成如下一对掩码值:

此处 是在上随机选取的密钥，是选定的伪随机函数， 代表将 和0值拼接起来作为伪随机函数的输入。

（2）对 中的每一个元素，设为 ，都赋予一个不同的计数值 ，并针对 计算另一对额外的掩码值:

此处 是在上随机选取的密钥。

（3）接着对每个Join属性 ，算法计算 ：

（4）然后 计算 对应的加密记录标识符 :

此处 如同前面的密钥，在上随机选取。

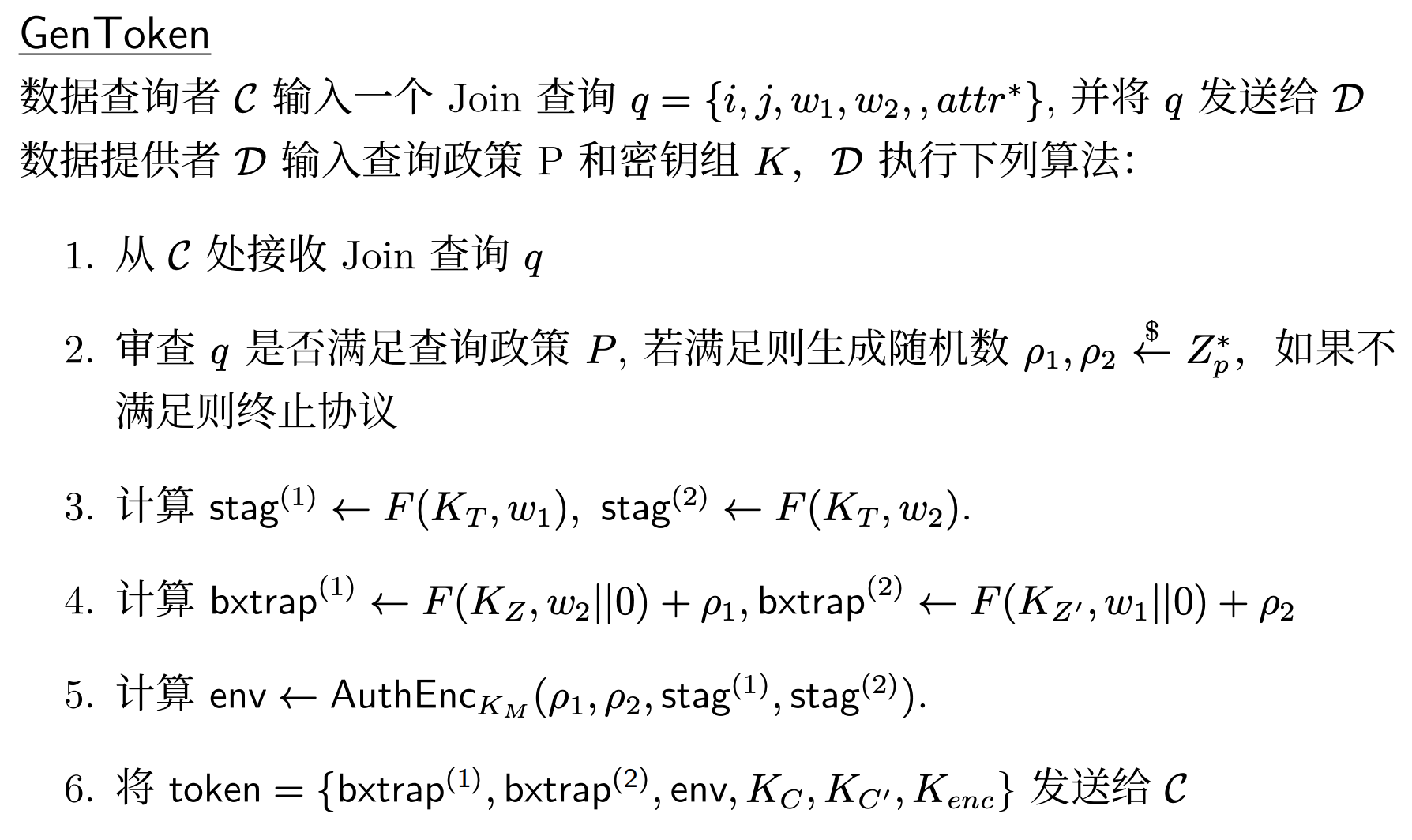
（5）最后算法构造出元组 加入 中。

这里计算掩码是不同于JXT改变的点在于，JXT中计算额外掩码值 的方法时使用的密钥和计算 时的相同，都是 ，而MC-JXT计算额外掩码值采用了另一组不同的密钥 。这样做的目的是为了降低后续 时和 之间的通信开销，具体原因见3.2.2.1。

以上就是 和 的建立过程，如本节开头所解释的那样，最终算法还要随机选取一个密钥 ，和 、以及其他密钥一同作为算法输出。

### MC-JXT——算法

算法是MC-JXT相比于JXT协议独有的算法，其具体过程就是 输入Join查询 ，输入查询审查政策 ，将 发送给 进行审查，如果 判断 满足 ，则将与 对应的搜索凭证 分发给 ，否则终止协议。接着 将搜索凭证发给 ， 核对 的合法性，如果确认 是由 生成的，则用其进行查询并返回结果，否则终止协议。具体算法见算法 2。



算法 2 MC-JXT-GenToken

这里我们主要关注两个问题的解决。第一个问题是 如何给 分发搜索凭证，第二个问题是 如何验证 是否由 生成。

#### 问题一—— 如何给 分发搜索凭证

2.3.3中提到对于Join查询 ，其对应的搜索凭证构造为:

,

其中 ，。 为 中各个元素对应的掩码值， 同理。最简单的分发方式，就是 接收到发送来的查询并审查通过后，直接将 全部发送给。但是这样做通信开销显然比较大，仅仅对于一个查询 ，就需要发送 单元的数据。

我们采用了另一种方案，以降低分发过程中的通信开销。我们注意到 和 数组中所有项有共同的部分，那就是 和 ，这两个值不随 改变而改变，因此我们可以将 ,拆分成两个部分，分别是 和 ：

我们可以让 部分由自己生成，让 部分只能由生成，并且在查询审查通过之后分发给，然后自己将两部分组合成完整的 和，再进行后续交互查询。此时对于一个查询，分发的通信开销只有两个单元，相比之前大大减小。为了实现上面的方案，我们需要让 和 用不同的密钥组生成，而不是像JXT协议中都由 生成。我们让 仍由生成，而 用另一个密钥组生成，这样，分发过程就是本地用 计算出，然后将连同发送给。

#### 问题二—— 如何验证 是否由 生成

最直接的思路就是让对 进行数字签名，然后服务端验证签名。但是 此时无法对直接进行签名，因为按照上一节中的方案，是由组合生成的而不是 ，对 是不可见的。我们设计了一个同态签名方案来解决上述的问题，虽然无法直接对 签名，但是可以对进行签名，如果我们设计的签名具有加法同态性，那么由已签名的 和未签名的 就可以计算出已签名的，即:

这样便达到了在不破坏问题一中分发凭证算法的情况，使得 完成对 的签名。我们称 中的两个元素 分别为 ,，计算出之后，再随机选取随机值 ，然后计算:

并将 连同发送给，然后利用 本地计算出 , 然后计算出:

,

, 。此时 就等于在 的基础上加上随机值 ，也就是对 的签名。之后 将 发送给 。

要使得能够验证签名，我们需要把 由传递给，将从接收到的 减去从接收到的 ，只有 是由生成的，这一步才可以产生正确有效的 也就才可以进行后续查询。此外 还需将查询所需要的 直接发送给，这两个值不能披露给。我们这里使用认证加密算法来进行上述值从 到的传递。和共享密钥 ，然后使用认证加密算法AuthEnc对上述值进行加密:

然后将 发送给，将其 一同发送给 。

### MC-JXT——算法

Search算法由 和 交互进行，由三个轮次组成，具体算法见算法 3。

**轮次1**： 生成查询需要的相关凭证，并发送给 。

**轮次2**： 利用从 接收到的凭证，在密态数据库中检索匹配的数据，作为初步结果并返回给 。

**轮次3**： 本地对从 接收到的初步结果进行解密，获取到最终查询结果。

#### 轮次1:

首先 接受 发送过来的数据:

接着 利用 中的 按照上一节的描述的方法生成并发送 ，直到收到来自 发送的 停止信号。此外 还会将 发送给 。

#### 轮次2:

首先 从 中取出 ，利用 查询出对应的 中 对应的索引 ，其具体形式为 ，接着我们对查询中的Join属性集合中的每个Join属性 计算：

遍历完 中所有的元素之后，向 发送 。

类似地，从 中取出 ，利用 查询出对应的 中 对应的索引 ，其具体形式为 ，接着对查询中的Join属性集合中的每个Join属性 计算：

遍历完 中所有的元素之后，向 发送 。

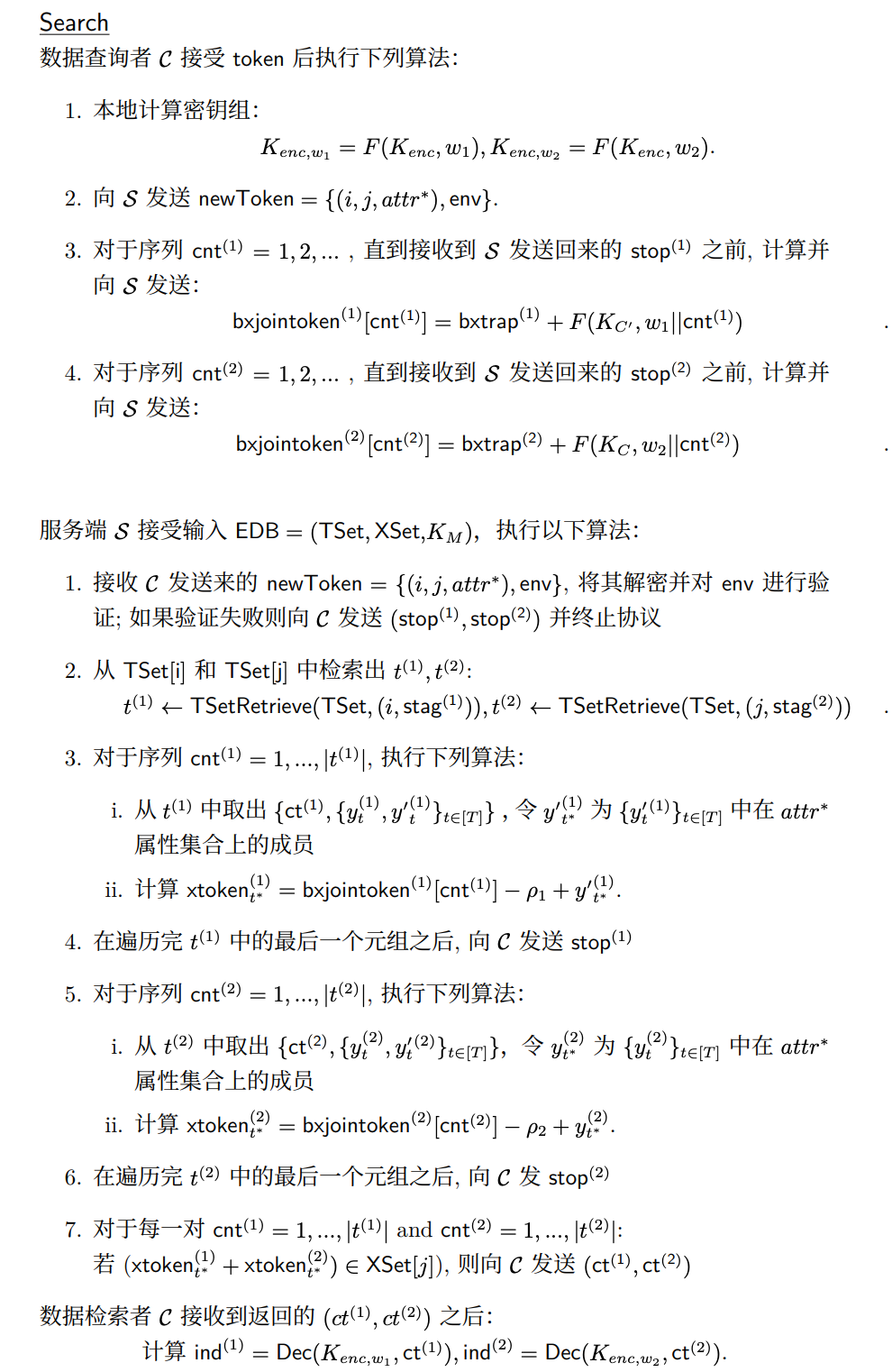
然后 针对每一组 ，计算

是否成立。如果所有的 ，上式均成立，则 将 给 。

#### 轮次3: 本地解密

从 中取出密钥 , 像 在 中那样，计算相应的：

然后对接收到的 进行解密得到最终的查询结果 ：



算法 3 MC-JXT-Search

## MC-JXT协议正确性分析

本节我们对MC-JXT的正确性进行形式化分析。MC-JXT协议正确性的定义是指，发起任意查询 如果其运行完MC-JXT协议之后其得到的查询结果同 一致，我们就说MC-JXT协议是正确的。本文下面就会对此MC-JXT正确性定义进行证明，我们的证明脉络是证明 中的任意元素都包含在一次MC-JXT实例最终的查询结果当中，相应的，不属于 的元素一定不会在MC-JXT实例最终的查询结果中。

考虑Join查询 ，假设存在一对记录 ，也即满足:

.

我们论证此协议的正确性，也就是论证 处于 之后 所获得的查询结果之中。由上一节可知，协议中 首先获取 中对应的项 ,，且有，此外又可知对于每个 ：

,

而在轮次一中 发送给 的token为:

,

其中 ，，所以在轮次二中对于每个 ，计算的 值为:

类似地有:

所以最后计算的 为：

由我们上面对 的相关定义可知，相应的有对于每个 ，有 ，所以有

而由 可知，，故可知 成立，所以 会将 对应的 返回给 ，最终获得的查询结果中会包含。

同理我们可以证明 的情况下，运行完整个协议之后得到的查询结果一定不包括 。由此我们便证明了运行完MC-JXT协议之后，得到的查询结果同 是相同的，也就证明了协议的正确性。

## MC-JXT协议安全性分析

在一般的单用户可搜索对称方案中（包括JXT），攻击者只包括服务端 ，对于MC-JXT方案，我们还要额外考虑客户端 作为攻击者的情况。总体来说，MC-JXT考虑如下情况下的安全性：（1）诚实且好奇的服务端；（2）恶意的客户端；（3）客户端和服务端之间不会勾结。MC-JXT所有的安全定义都定义在由泄露函数 参数化的理想/现实范式框架下。

### 攻击者为服务端 时的安全性

MC-JXT首先考虑诚实且好奇的服务端作为攻击者的情况，这也是大部分对称可搜索加密方案安全定义中的攻击者配置，包括JXT。JXT的安全性定义为抗自适应攻击的 -语义安全，MC-JXT在此情况下的安全性同JXT相同，我们可以直接采用。具体的安全性定义见JXT安全性定义。

### 攻击者为客户端 时的安全性

#### 泄露函数

由3.2中的协议算法可知，协议中服务端会泄露给客户端满足 和 的记录数目，即 和 。泄露的原因是 过程中，服务端分别在遍历完 和 之后会向客户端发送停止信号 ，因此客户端可以由自己发送的 数目来确定上述两个值。

在实际的应用中，我们通常会使用增加冗余的方式来对上述泄露信息做掩码。我们设置一对掩码值 ，。然后我们在阶段将 和建立时使用的计数值的范围扩展为 和 ，这样客户端在查询过程中就要生成掩码值数目的 ，原先的值就被掩盖起来了。故泄露函数定义为：

#### 安全性定义

此处的安全性定义针对的是在协议运行过程中，除了查询结果 之外，客户端获取到的其他额外信息。遵循Lindell 等人对恶意的攻击者的定义，MC-JXT安全定义中的理想世界中存在可信第三方运行一个理想算法 并同攻击者 和模拟者进行交互。此算法是对MC-JXT协议功能的一种理想模拟，其接受查询 作为参数，然后验证 是否满足政策 ，如果满足，则该算法返回，如果不满足，则返回空值。MC-JXT针对恶意的客户端的完整安全性定义如下:

**定义1**：令 为MC-JXT协议的一个实现。给定泄露函数 ，以及模拟者，定义如下两个实验：

(1) : 选择 ，接着实验运行 。攻击者 可以自由地调用 和 实例，在前者中 接受 和 并与数据提供者 进行交互；在后者中 与接受 的服务端 进行交互。在这些交互过程中可以任意表现，最终输出一个比特 ，且 就是实验最终的输出。

(2) : 同上述实验选择 ，实验首先运行 以初始化模拟者算法 。当 调用 时，它将于运行着 的实验进行交互；当 调用 时，将于运行着的实验进行交互。此处 和 在与 进行交互时，都可以更新全局模拟者状态变量 。两者也都可以向理想算法 发起查询并得到上述的查询结果。在这些交互过程中可以任意表现，最终输出一个比特 ，且 就是实验最终的输出。

如果对于攻击者 所能进行的任何多项式时间算法，总是存在一个运行算法 的模拟者，使得:

即使得两个实验输出结果的统计学差异为安全参数 的可忽略函数，我们就称协议 满足抗恶意客户端攻击的 -语义安全。

以上是攻击者为恶意客户端的情况下MC-JXT协议的安全性定义，对于一个实际的MC-JXT实现而言，当满足下列条件时，我们称此实现是安全的：

**结论1**：令泄露函数为 ，当满足以下条件时，一个MC-JXT实现满足抗恶意客户端攻击的 -语义安全:

（1） 是一个安全的伪随机函数。

（2） 满足计算正确性。

（3）认证加解密算法(AuthEnc, AuthDec)满足IND-CPA以及Strongly-UF-CMA定义。

结论1具体的证明见附录。

## MC-JXT协议理论开销分析

本节我们对MC-JXT的性能进行理论分析，后续第四章我们将对协议进行代码实现并对理论分析的结论进行实验验证。我们将从三个方面分析MC-JXT协议的性能，分别是存储开销，计算开销，和通信开销。为了便于分析，我们假设进行Join查询的两个表是大小相同的，这里的大小相同指两表有相同的记录数和属性数。我们设两表的记录数为 ，属性数为 ，两表Join属性的数目为 。

### 存储开销

本节我们讨论MC-JXT中每张表对应的密态数据所造成的存储开销，此开销主要分为两部分，分别是 和。由3.2.1可知，对于表中所有的关键词 ，中都有一个对应的列表 ，列表中元素的形式为 。显然每一个元素占用 个存储单位，分别对应于 和 。这样以来 占用的空间为 个单位，而这张表 整体占用的空间则为 个单位。又因为，故 。

接下来我们再考虑 的存储开销。由3.1.1可知，算法会对表中每条记录的每个Join属性计算一个 值，将其存储在 中，故知表的XSet由 个 构成，每个 由两个伪随机函数值相加得到，占用一个存储单元，所以 。

综上所述，MC-JXT中一张表的存储开销为:

### 计算开销

的计算开销：数据提供者 首先运行 算法建立密态数据库，首先考虑 和 的建立计算开销。正如3.4.1所提到的那样，的建立需要计算 个 ，每个 需要若干次计算，故其建立计算开销为 ；TSet建立的计算开销也就是生成TSet中所有单元的计算开销，由3.4.1可知，其为 。此外，还需要承担的计算任务有：生成、和 ，这些运算的计算复杂度都是所以整体计算开销为。

的计算开销：客户端 在协议中根据 发送而来的 和密钥，计算出搜索凭证 和 。根据3.1的协议描述可知，此过程需要计算的凭证数量为 |，所以其计算开销为 。

的计算开销：服务端 在协议中的任务分成两部分，分别是恢复出和检索密态数据库。在第一部分中服务端首先将 进行解密，得到 ，这一步的计算复杂度为 。接着 将收到的和 分别减去 以恢复出 ，此过程的计算复杂度为 。在第二部分中，服务端首先使用 检索出 和 ，接着其会遍历这两个列表笛卡尔积的每一个元组对，并针对每一个Join属性进行匹配运算，这一过程的计算复杂度为 。

### 通信开销

由上面描述的协议流程，和 之间的通信开销显然是 的，而 和 之间没有通信。对于 和 而言，从接受查询结果，大小为 个单位，而 从获得的数据有 个单位。总而言之，协议整体的通信复杂度为 。

# 协议实现以及性能测试

本章主要介绍了一种具体的MC-JXT代码实现，以及其在关系型数据集上的协议性能测试。性能测试的指标主要是运行时间，其目的是为前文中提出的理论协议加以实验上的功能和效率的验证。本章首先会介绍MC-JXT的包括 等组件的具体实现方案。之后会介绍实验的数据集以及相应的性能测试实验结果，并讨论实验结果同3.5中理论分析的契合度。

## 协议实现

我们使用了Java语言实现了MC-JXT协议，该实现由两部分组成，分别是数据预处理阶段和查询执行阶段。数据预处理阶段将读取数据集，并在其基础上执行 获得密态数据库；查询执行阶段主要包括生成所需的 以及查询的执行。这两阶段程序上的密码学原语都是依赖包JPBC-2.0.0（Java Pairing-Based Cryptography Library）实现的。具体而言，伪随机函数 的具体实现为:， 为JPBC库中对称质数双线性群TypeA（建立在圆锥曲线 之上）；加密算法实现为AES的ECB模式；认证加密算法AuthEnc实现采用Encrypt-then-MAC算法，其中加密算法和MAC算法分别选取AES-ECB算法和HMAC-SHA256算法。

数据预处理阶段生成 和结构，并存储在内存里。具体的实现是一个由 作为键的哈希表，每个键对应的值是一个固定大小的元组列表。此处我们使用布隆过滤器实现，布隆过滤器的插入和查询时间均为常数，此种实现方式可以大大提升判断 是否在 中的判定效率。然而布隆过滤器会造成“误判”（假阳性）的可能，我们此处将布隆过滤器的误判率设置为 ，以达到 准确性同占用存储空间之间的平衡。

## 性能测试

### 数据集

本文采用人造的美国人口普查数据（US-cencus data）作为测试数据集。美国人口普查局提供有关国家人口的数据，此数据公开在相关网站上。该数据集只有人口数据的统计信息，没有个人的具体信息，因此本文按照公开的US-cencus data中统计数据的比例，构造了一个大小为1000000条记录的美国公民个人信息数据库，包含（personID, firstname, lastname, sex, age, state, race）七个属性，数据库具体包含以下两个表：

（1）state\_census(personID, firstname, lastname, sex, age, state)

（2）race\_census(personID, firstname, lastname, sex, age, race)

表state\_census 和表race\_census分别存储着相同集合公民的住址信息（来自哪个州，包含50个州）以及种族信息（是什么种族，包含白人，拉丁裔，亚裔等七个种族），两表中公民的顺序是被随机打乱的，并且两表都有两个主键，分别是(personID)和(firstname,lastname)。我们执行以下Join查询来测试MC-JXT方案的性能表现：

### 实验设计以及实验结果

本节将设计具体实验，去验证3.4中分析的协议各方面性能的相关因素。实验环境信息如下:

表 4‑1实验环境介绍

|  |  |
| --- | --- |
| **处理器** | Apple M1 8 cores |
| **内存** | LPDDR4 16 GB |
| **操作系统** | macOS Ventura 13.1 |
| **硬盘** | APPLE SSD AP0256Q |

实验中会根据需求随机从上述构造的仿真的美国人口普查数据集中获取记录并构建实验数据集，然后对每一组选取的实验数据集建立密态数据库EDB并执行相应的查询，整个实验阶段使用到的记录从200到30000不等。本文实验中的性能评价指标是协议的运行时间，此运行时间在代码中相应语句测量，以下是具体的实验结果:

#### MC-JXT在实际应用场景中的一般性性能表现

本实验测量在不同查询规模下协议整体的运行时间。由于Join运算的特点（即两表的记录集合做笛卡尔积），此处的查询规模指满足查询中 和 的记录数量之积，即为 。本实验主要反映了MC-JXT协议在实际应用场景中的一般性性能表现，本实验所选取的查询规模从19840到7279584不等，实验结果如图 4‑1所示。



图 4‑1 MC-JXT运行时间随查询规模的变化

由上图可知，MC-JXT协议的性能随查询规模呈现线性或者非线性的关系，在大规模查询的场景下仍只需要较少的查询时间。

#### 阶段性能的影响因素

前面3.4.2我们分析出的结论是 计算复杂度为 ，下面我们通过两个实验验证这一结论。

（1）运行时间与数据库规模之间的关系

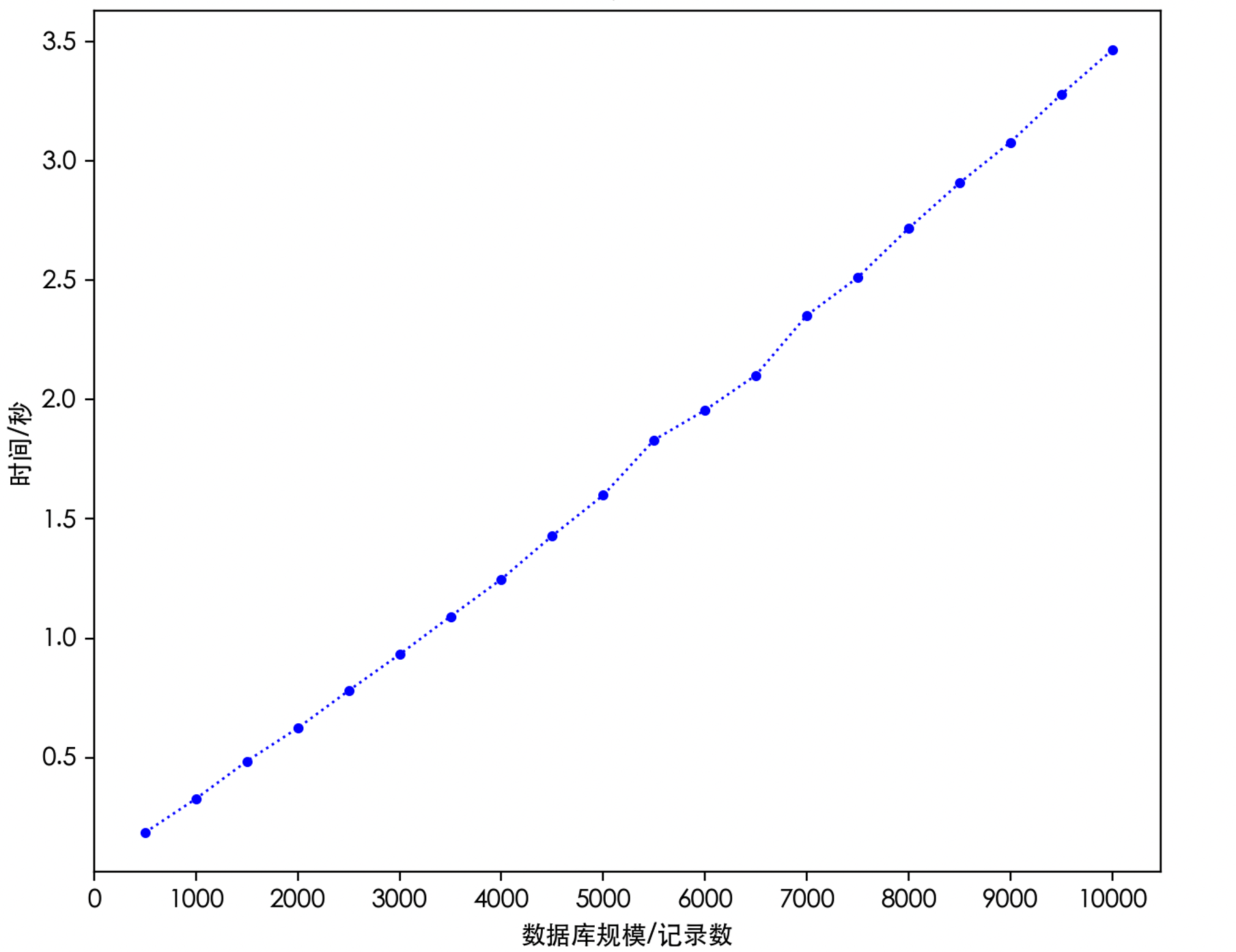


图 4‑2 EDBSetup运行时间与数据库规模的关系

此处我们改变数据库规模大小的方式是控制数据库中的记录数目，不改变数据库中表的属性，故数据库规模指表中的记录数目 ，图 4‑2为实验结果。

实验结果显示 的运行时间与数据库规模 成正比，与理论分析相符。

（2） 运行时间与表之间Join属性数目 的关系

我们在4.2.1中提到，实验的两表的主键有两组，分别为（personID）和（firstname, lastname），所以实验构造三种不同的Join属性集合，如表 4‑2所示。

表 4‑2 T1,T2,T3取值

|  |  |
| --- | --- |
| T1 | （personID） |
| T2 | （firstname, lastname） |
| T3 | （personID, firstname, lastname） |

实验测量三种场景各自的性能表现，实验结果如图 4‑3所示。

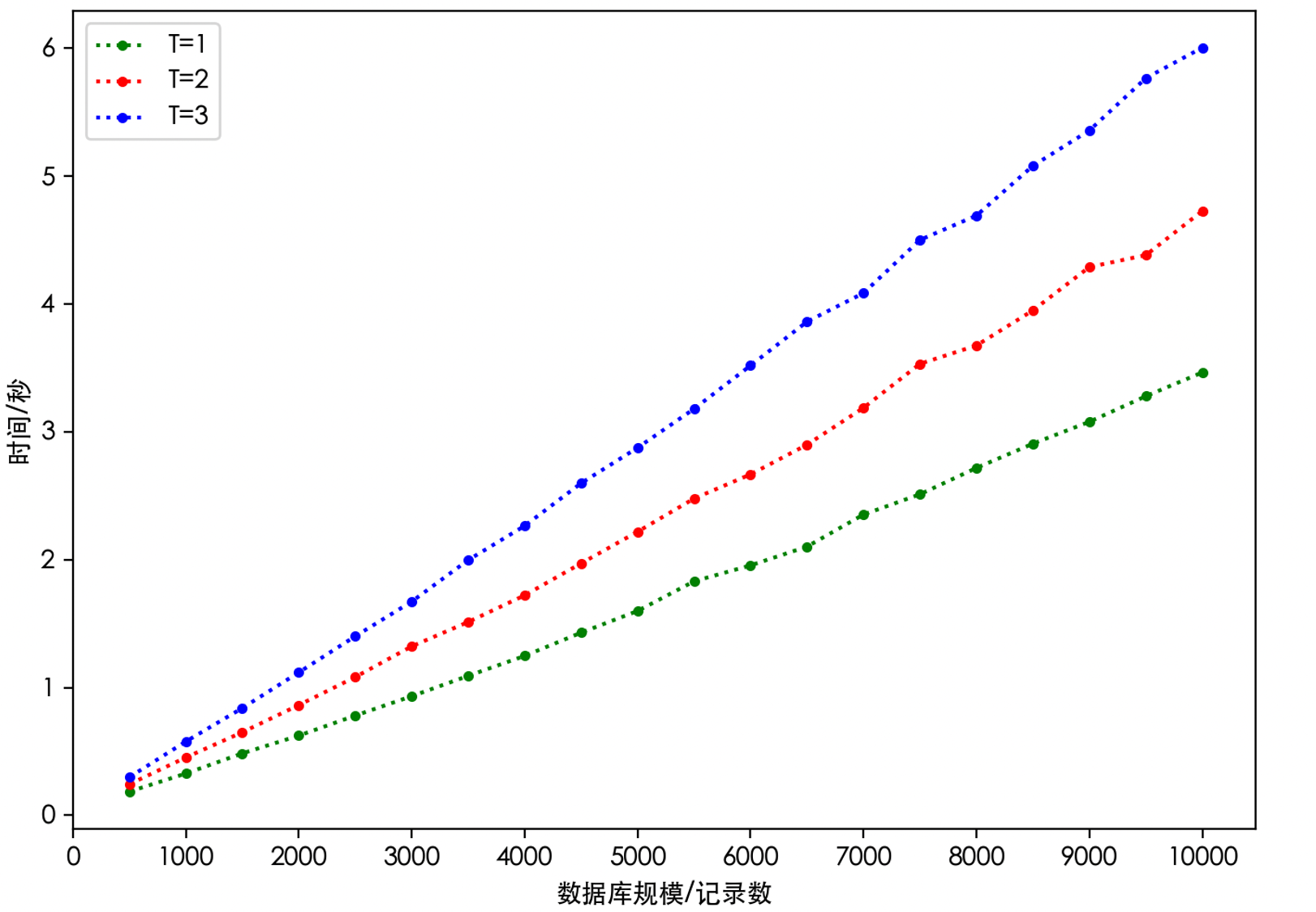


图 4‑3 EDBSetup运行时间与Join属性数目的关系

由图 4‑3可知，随着T的增大，运行时间也在增大。为了进一步探究具体的 运行时间的增大和T变化之间的关系，我们计算每一个数据库规模下，T=3比T=2增加的时延同T=2比T=1增加的时延的关系，结果如实验结果如图 4‑4所示。

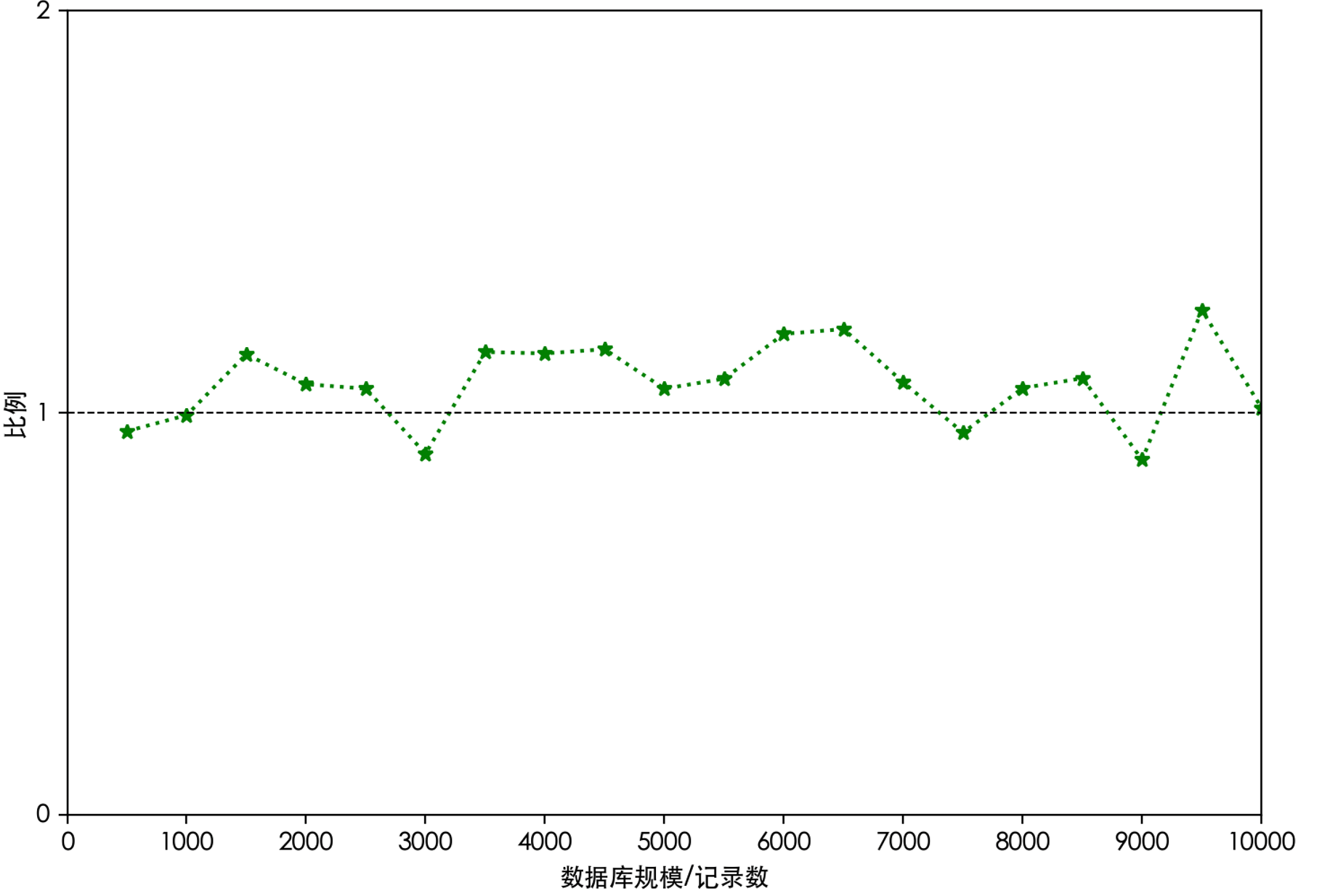


图 4‑4 不同T下时延增量的比例关系

由图 4‑4 可知，T=3相比T=2增加的时延与T=2相比T=1增加的时延基本相等，故 运行时间同 成线性关系，与理论分析结论相吻合。

#### 阶段性能的影响因素

前面3.4.2分析出的结论是 的计算复杂度为 ，下面我们通过两个实验验证这一结论。本实验选取的查询规模从19840到7279584不等，不同查询规模下测得的 运行时间如图 4‑5。

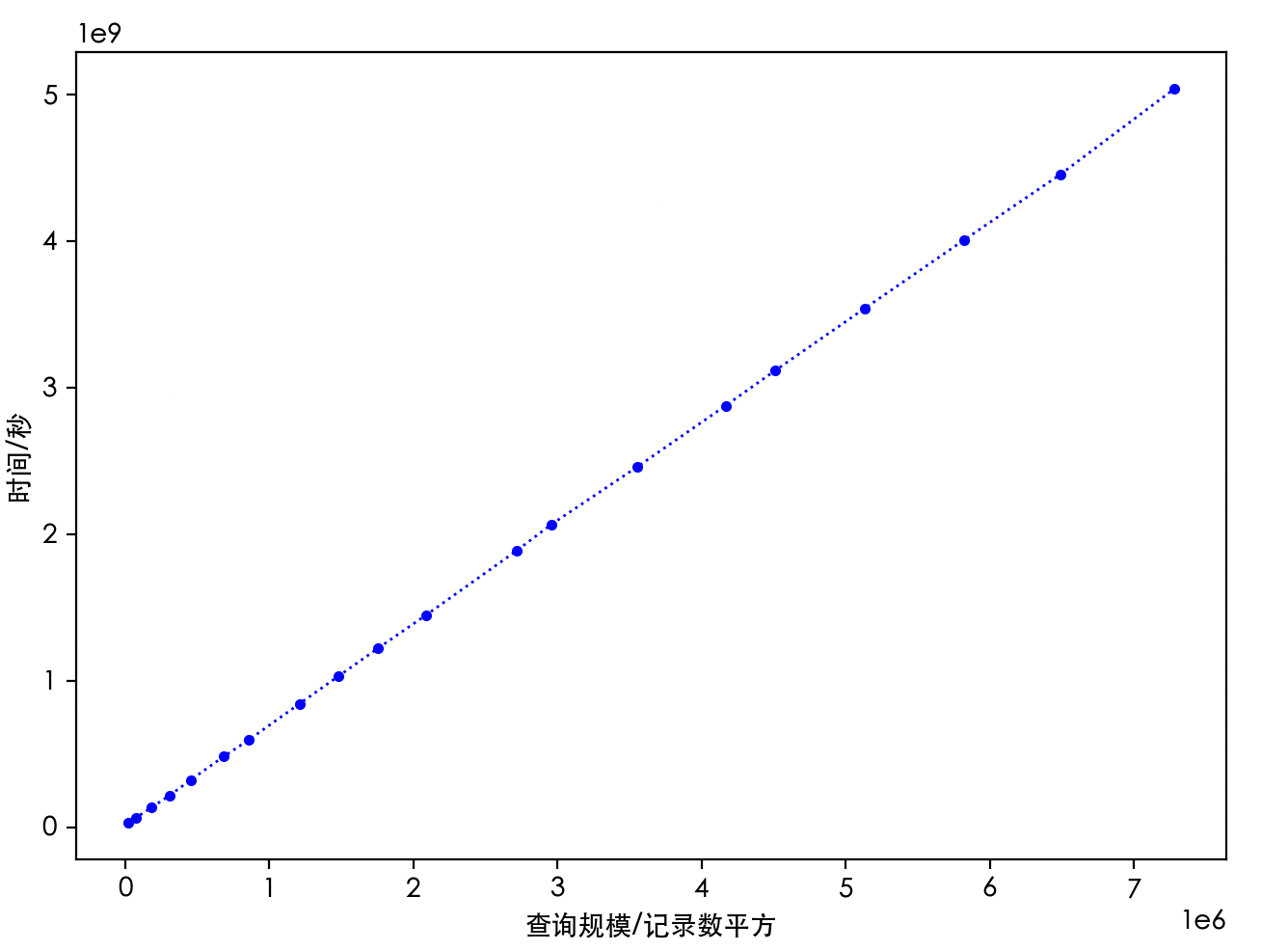


图 4‑5 Search运行时间与查询规模的关系

实验结果显示 运行时间同查询规模成正比，这一点与理论分析结论相符。此外为了进一步验证MC-JXT协议的性能优势，本文还设计实验探究了MC-JXT 阶段运行时间同数据库规模的关系，我们控制查询规模不变，让数据库规模不断增大（这一点的实现方式是在表中插入特定数量的，和执行的查询无关的冗余记录），然后测量不同数据库规模下的 运行时间，实验结果如图 4‑6所示。

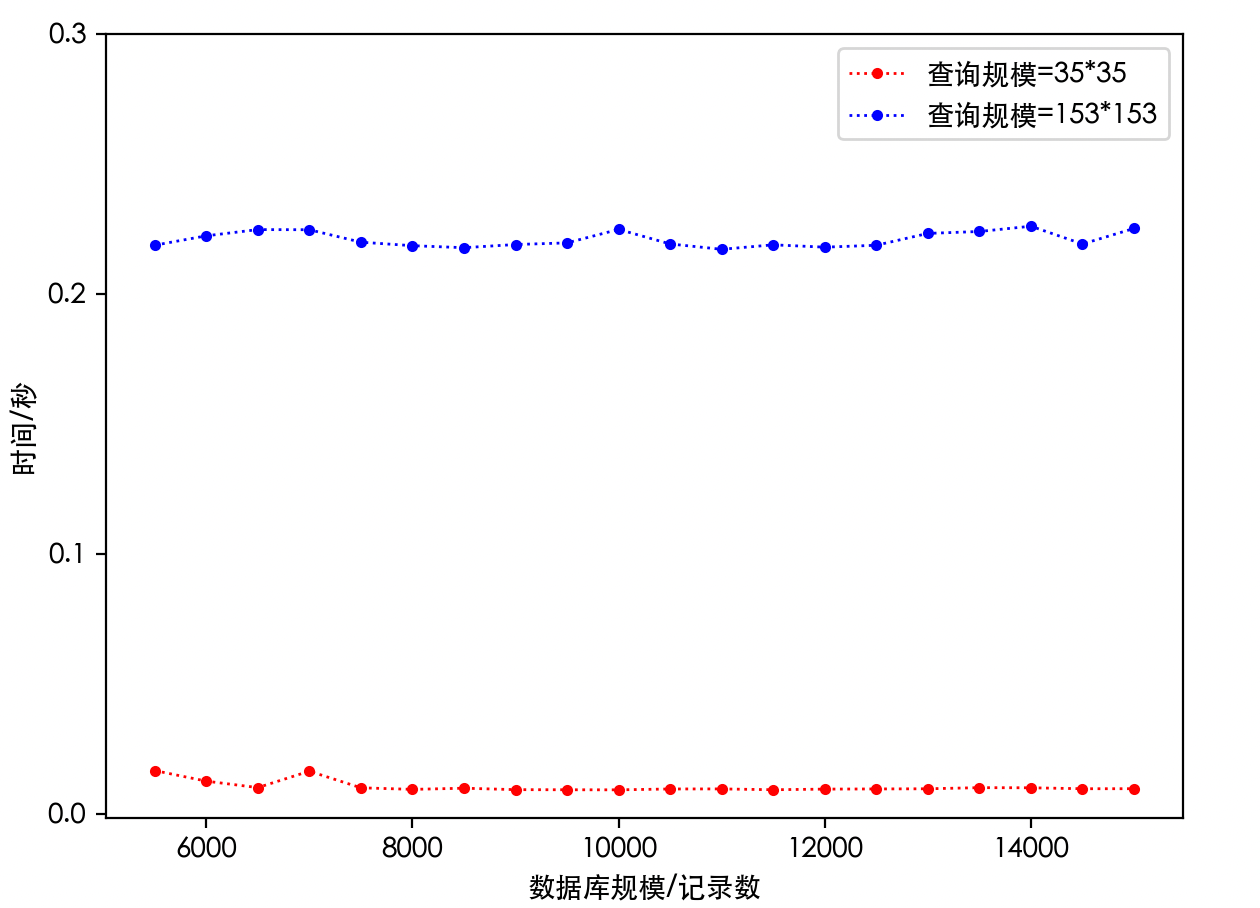


图 4‑6 固定查询规模的下Search运行时间与数据库规模的关系

实验结果显示MC-JXT协议 阶段的运行时间同数据库规模之间没有联系，只和查询规模相关，这说明MC-JXT协议不会因为数据库规模的增大而产生直接的性能下降，进一步显示出MC-JXT协议在大规模数据下的应用价值。

# 总结与展望

## 本文总结

在隐私泄露事件频发，数据安全挑战日益严峻的背景下，如何提升对称可搜索加密技术的实用价值对进一步保护云端用户隐私数据的安全有着重要的意义。如何设计支持更加多样的应用场景以及更加丰富的查询类型的对称可搜索加密方案，是当前此领域的重要发展方向。

本文首次设计提出了多用户应用场景下，高效支持Join查询的可搜索对称搜索加密方案，并针对协议的正确性、安全性给出了严格的定义和相关证明。协议在已知高效支持Join查询的JXT协议上做多用户扩展和改进，可以让第三方实体在经过数据原始提供方允许的情况下，对其存储在云端上的数据进行高效Join查询。本协议在实现Join查询时，不需要预先进行表之间的Join运算，从而实现高效查询的目的。此外，本文还设计实验测试了协议在实际应用场景下的性能表现。理论分析和实验结果都证明，MC-JXT协议有着较好的安全性，并且在大规模数据应用场景中有着较少的存储开销和计算开销。

## 工作展望

与此同时，本文也提出了一些未来值得研究的新问题，我们总结如下面几点。

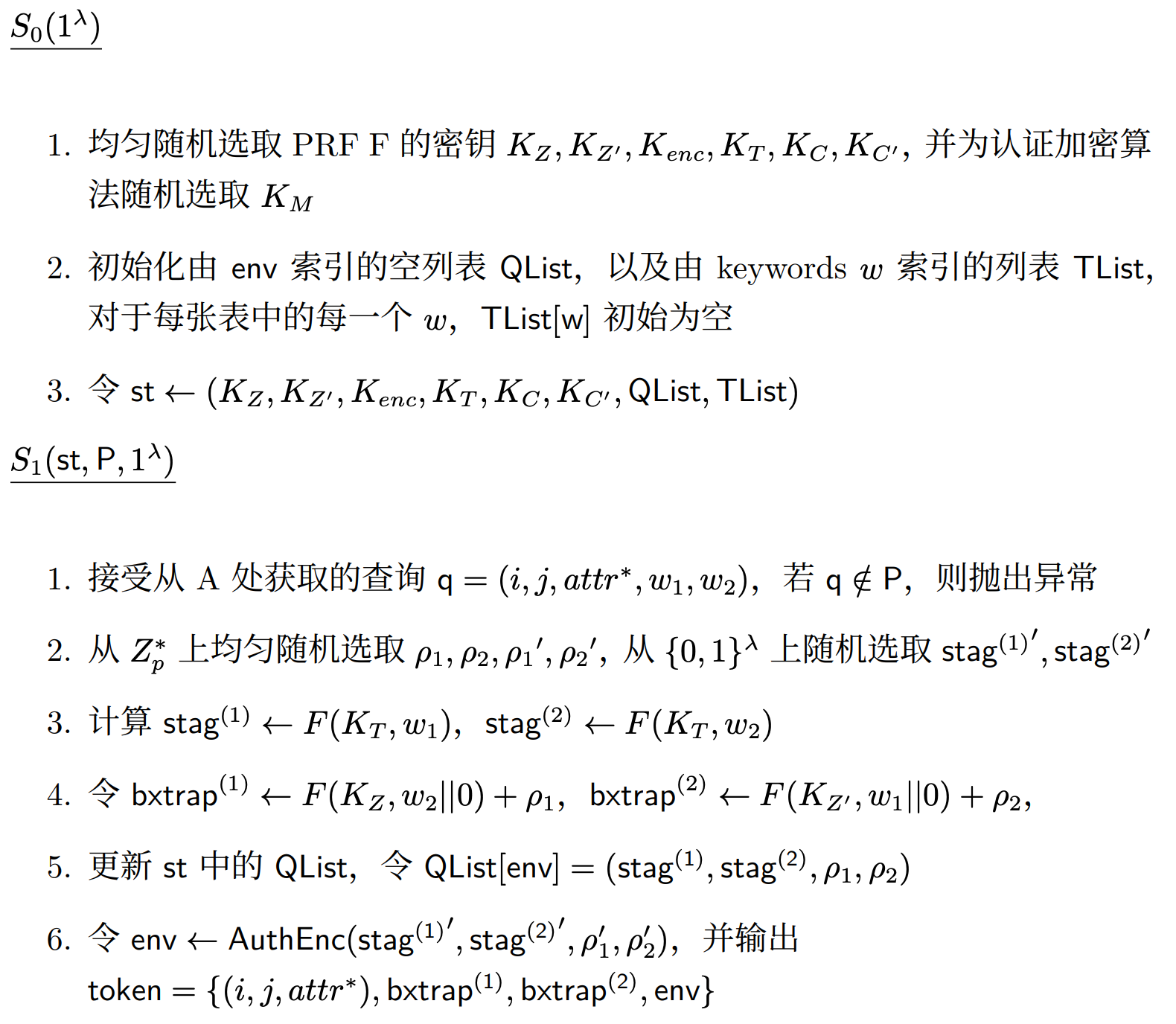
首先是如何设计支持两个以上表之间Join查询的对称可搜索加密方案。本文设计的协议主要考虑两表之间的Join查询，尚不能支持三个及三个以上表的Join查询。上述更多表之间的Join查询可以按照本文的思路进行拓展。

其次如何将MC-JXT协议扩展为支持动态增减的对称可搜索加密方案。目前MC-JXT协议建立的密态数据库是在 创立之初确定的，后续不支持对密态数据库添加或者删除记录。目前已有相关的文献将OXT协议扩展成为支持动态增减的方案，MC-JXT的相关扩展可以参考相同的思路。

最后是如何改进MC-JXT中的签名方案，使得 的签名成本更小。MC-JXT中要为每一个Join查询都生成一组随机值，这样还是会造成额外的通信开销。值得注意的是MC-JXT中的同态签名是理念上的同态签名，其与学术界严格定义的同态签名诸如不同，因此如何改进MC-JXT的签名方案，使之具有真正意义上的同态性是需要进一步研究的问题。

# 附录

此处我们给出对结论 1的具体证明。此证明的模拟者算法 如算法 4，算法 5所示。我们令游戏 指代定义 1中的 实验，该实验中攻击者同真实世界的MC-JXT协议进行交互。在下列的安全性证明当中，我们一步步地对 做修改和放缩，使得最终获得的 与定义 1中的 实验相同。我们设为是游戏 输出1的概率。如果我们能证明 和 的差异在安全参数 下是可忽略不计的，也就证明了结论 1，具体证明如下所示。



算法 4 模拟者算法S（a）

Game ：相比于，去除了以下不安全的情况：在 协议中，存在没有由 在 协议中生成并签名的密文（即伪造的 ）通过服务端认证。若认证加密算法满足Strong-UF-CMA不可伪造定义，则有 。

Game ：相比去除了以下不安全的情况：存在两个不同的 实例产生相同的 密文。由于协议中 是 上随机生成的，因此可知 对应的明文发生碰撞的概率可忽略不计；同时由Strong-UF-CMA定义可知，此认证加密算法出现不同明文对应密文相同的概率也是可忽略的，因此有。

Game ： 去除了以下不安全的情况：存在两个不同的关键字 ，，满足 或者 。此种情况代表着协议中 或者 发生了碰撞，由于 是一个伪随机函数，由其定义可知碰撞发生的概率可忽略不计，即有 。

Game ： 去除了 。在 中，我们不再调用 来通过明文索引 建立 ，而是仅仅保留 ; 在 中用 查询 时，我们也不用 来得到结果，而是直接令。只要保证 的正确性，对于一个合法的，我们使用上述方式得到的 和使用 得到的 一定是相等的，因此有 。

Game ：在 中，我们在 中计算 的方式是加密 ，将其替换为加密一组独立随机值 ，其中 从 上均匀随机选取，从 上随均匀随机选取。为了保证协议功能性，维护一个由 密文作为索引的数组 ，在调用 时，真正的被存储在 中。相对应的，在 中，模拟者不再解密 以获得 ，而是直接查询。如果认证加密算法满足IND-CCA定义，则有 。

Game ：我们将PRF 替换为上的随机函数 。由 PRF 的定义，有 。

Game ：去除了 ，修改后的 过程如下：

（1）服务端接收客户端发送的 , 并从 中取出 。

（2）对于 和 ，检验是否满足以下条件 ：

若不满足，则停止协议。

（3）继续检验是否满足 ：。这里 对应于 的第 个元组 ， 同理。若仍然满足此条件，则服务端返回相应的 ，否则停止协议。

对于任意的 ，中的查询结果 一定也在 的查询结果中。因为如果满足，那么保证了 和 是合法的，按照协议后续步骤一定可以计算出；而 则进一步说明了 。因此， 和 之间唯一的区别就是有以下特例发生：不满足，即 是不合法的，但是由 计算出的 ，即：

令 ，若不满足，则有 ，其中

而 可以说明 ，使得，即为：

由 是PRF可知，上式成立的概率可以忽略不计，故有 。

Game ： 考虑了 复用的安全性问题，去除了以下不安全的情况: 对于同一个 ，存在两对元组 , 有，使得如下等式成立：

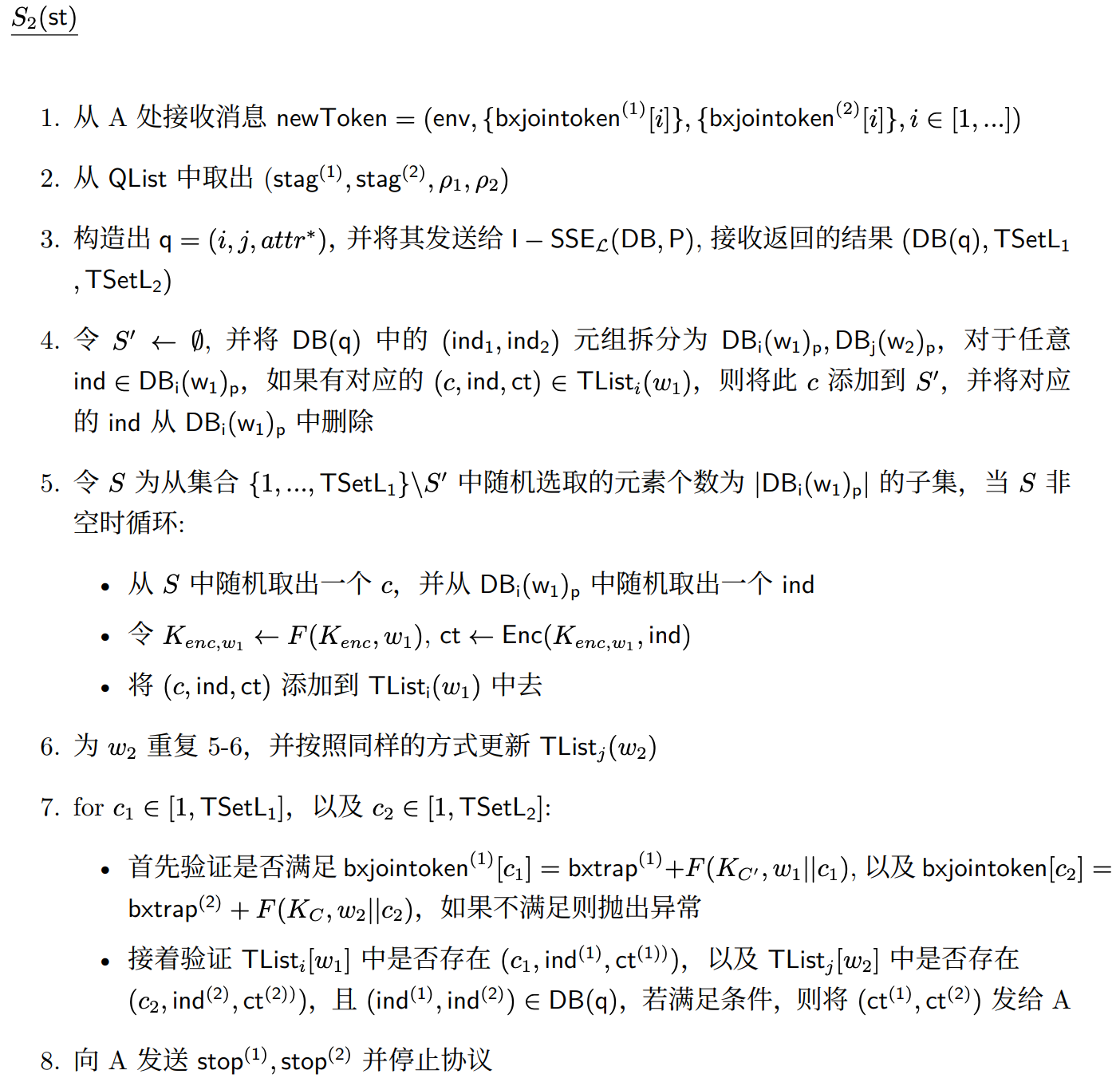
对于 同理。此种情况发生于攻击者将上一次 实例的 保存下来，在这一次 实例中重复使用。我们只考虑 ，此问题就等价于恶意的客户端 在 有效的情况下（用的是之前 实例的），猜测变量 的值等于 的成功概率问题。由于 是PRF，并且 是均匀随机选取的，故此处猜测成功的概率可忽略不计，即有 。

Game ：在上述游戏中，模拟者是知晓攻击者选取的Join查询 的，因为协议中客户端会将 直接发给模拟者进行审查。所以我们在 中，可直接将 发送给理想算法，并得到正确的查询结果，游戏具体流程如下:

（1）在 过程中抛弃了 和 ，仅仅选取密钥 。

（2）过程同相同。

（3）在 中，将 发送给 并得到 。接着 将 中的 元组进行拆分，拆分为 和 ，分别含有所有元组的第一维和第二维。然后 赋给 中元素1到 的非重复随机索引标号，对 也同样这样做。为了跟踪此索引标号，维护列表 ，其作用是模拟原先游戏中的 。以 为例， 中存储着元组集合 ，c是上述过程给此 的索引标号；是 对应的密文。在协议的开始， 将所有的 初始为空，每次在 中调用完 之后，将 中的不在 的 按照上述的方式赋索引值，并连同相应 加入 中。后续则如。



算法 5 模拟者算法S（b）

# 参考文献

[1] KAMARA S, MOATAZ T. SQL on Structurally-Encrypted Databases [Z]. Advances in Cryptology – ASIACRYPT 2018: 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2–6, 2018, Proceedings, Part I. Brisbane, QLD, Australia; Springer-Verlag. 2018: 149–80.10.1007/978-3-030-03326-2\_6

[2] CASH D, NG R, RIVKIN A. Improved Structured Encryption for SQL Databases via Hybrid Indexing; proceedings of the Applied Cryptography and Network Security, Cham, F 2021//, 2021 [C]. Springer International Publishing.