**编 号：**

**审定成绩：**

|  |  |  |  |
| --- | --- | --- | --- |
| **重庆邮电大学**  **毕业设计（论文）** | | | |
|  | | | |
|  | **中文题目** | **具有完整性验证功能的婚车租赁系统设** |  |
|  | **计与实现** |
| **英文题目** | **Design and Implementation of Wedding** |
|  | **Car Rental System with Integrity Verification** |
| **学院名称** | **计算机科学与技术学院** |
| **学生姓名** | **陈馨怡** |
| **专 业** | **信息安全** |
| **班 级** | **04041504** |
| **学 号** | **2015210519** |
| **指导教师** | **唐飞 副教授** |
| **答 辩 组**  **负 责 人** | **宋秀丽 副教授** |
|  | |

**2019 年 5 月**

**重庆邮电大学教务处制**

**计算机科学与技术**学院本科毕业设计(论文)诚信承诺书

本人郑重承诺：

我向学院呈交的论文《具有完整性验证功能的婚车租赁系统设计与实现》，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明并致谢。本人完全意识到本声明的法律结果由本人承担。

年级 大四

专业 信息安全

班级 040401504

承诺人签名 陈馨怡

2019 年 5 月 22 日

学位论文版权使用授权书

本人完全了解重庆邮电大学有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权重庆邮电大学可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

|  |  |
| --- | --- |
| 学生签名：陈馨怡 | 指导老师签名： |
| 日期： 2019 年 5 月22 日 | 日期： 年 月 日 |

摘要

随着互联网的飞速发展，越来越多的传统产业向线上互联网靠拢。婚车租赁行业历史悠久，在传统线下婚车租赁的过程中，租赁一辆婚车要几个小时甚至几天，巨大的时间耗费增加了婚车租赁的成本。所以，利用互联网技术优化婚车租赁管理，对婚车租赁管理的的整体业务需求进行深入的分析，开发一个婚车租赁系统，对婚车租赁行业的发展是有意义的。此外，鉴于婚车租赁系统涉及到价格，为了保证价格在传输过程中不被攻击者恶意篡改，此系统增加了对价格的完整性认证算法，增加了系统的可靠性。

论文内容包括：第一，调查了婚车租赁行业的现状；与从业人员进行沟通；获取婚车租赁系统的详细业务需求信息；梳理系统的总体流程；确定系统的功能性需求和非功能性需求；使得系统具有良好的用户体验。第二，研究婚车租赁系统。开发时候采用了维护方便，安全性高，效率高的Java语言。使用了SSM框架。Mysql作为后台数据库。前台采用了react-native技术，使得客户端具有兼容ios和Android两个平台的优势。第三，完成完整性认证算法。为了保证婚车价格不被篡改，增加了对价格的完整性认证算法。第四，完成婚车租赁系统的实现后，进行对系统各个模块的测试工作。,测试结果显示，各个功能模块达到了预期的设想,系统具备了运行的条件。

系统主要实现了登录、注册、婚车订单的发布、婚车租赁、婚车收藏、价格完整性认证、婚车信息搜索、热门婚车推荐、个人购买列表查看、个人收藏列表查看、个人发布列表查看。界面简单直观，操作方便，有较强的推广意义。

**关键词：**婚车租赁，完整性认证，react-native，Mysql，SSM

**Abstract**

With the rapid development of the Internet. More and more traditional industries are moving closer to the online Internet. The wedding car rental industry has a long history. In the process of traditional offline wedding car rental, it takes several hours or even a few hours to rent a wedding car. The huge time cost increases the cost of the wedding car rental. Therefore, the use of Internet technology to optimize the management of wedding car rental, in-depth analysis of the overall business needs of wedding car rental management, the development of a wedding car rental system, the development of the wedding car rental industry is meaningful. In addition, in view of the price involved in the wedding car rental system, in order to ensure that the price is not maliciously tampering by the attacker during the transmission process, the system increases the integrity authentication algorithm for the price and increases the reliability of the system.

The contents of the thesis include: First, investigate the current situation of the wedding car rental industry, communicate with the employees, obtain detailed business demand information of the wedding car rental system, and sort out the overall process of the system. Determine the functional and non-functional requirements of the system. Make the system have a good user experience. Second, study the wedding car rental system. Developed with a Java language that is easy to maintain, safe, and efficient. SSM framework are used. Mysql as a backend database. The front-end uses react-native technology, which makes the client have the advantage of being compatible with both iOS and Android platforms. Third, complete the integrity authentication algorithm. In order to ensure that the price of the wedding car is not falsified, an integrity authentication algorithm for the price is added. Fourth, after the implementation of the wedding car rental system is completed, the testing of each module of the system is carried out. The test results show that each functional module has reached the expected vision, and the system has the operating conditions.

The system mainly realizes registration, registration, release of wedding car rental information, vehicle rental, vehicle collection, price integrity certification, wedding car information search, popular wedding car recommendation, rental list, favorite list, and release list. The interface is simple and intuitive, easy to operate, and has a strong promotion significance.

**Keywords**: wedding car rental, integrity certification, react-native, Mysql, SSM

目录

[第1章 引言 1](#_Toc10066771)

[1.1 研究背景和意义 1](#_Toc10066772)

[1.2 主要内容及章节安排 2](#_Toc10066773)

[第2章 相关技术介绍 3](#_Toc10066774)

[2.1完整性验证技术 3](#_Toc10066775)

[2.1.1 CRC校验 3](#_Toc10066776)

[2.2系统开发技术 4](#_Toc10066777)

[2.2.1 react-native技术 4](#_Toc10066778)

[2.2.2 SSM框架 5](#_Toc10066779)

[2.2.3 MVC FrameWork 5](#_Toc10066780)

[2.2.4 Mysql 数据库 6](#_Toc10066781)

[2.3本章小结 7](#_Toc10066782)

[第3章 系统需求分析和概要设计 8](#_Toc10066783)

[3.1 需求分析 8](#_Toc10066784)

[3.1.1 系统需求分析 8](#_Toc10066785)

[3.1.2 系统非功能需求分析 8](#_Toc10066786)

[3.2 概要设计 10](#_Toc10066787)

[3.2.1系统体系结构 10](#_Toc10066788)

[3.2.2系统功能模块设计图 10](#_Toc10066789)

[3.2.3 数据库设计 11](#_Toc10066790)

[3.2.4 完整性验证算法设计 14](#_Toc10066791)

[3.3本章小结 15](#_Toc10066792)

[第4章 系统详细设计与实现 16](#_Toc10066793)

[4.1 系统实现环境的选择 16](#_Toc10066794)

[4.1.1 开发工具 16](#_Toc10066795)

[4.1.2 系统运行环境 16](#_Toc10066796)

[4.2 系统功能模块的实现 17](#_Toc10066797)

[4.2.1 注册/登录 17](#_Toc10066798)

[4.2.2 婚车发布 18](#_Toc10066799)

[4.2.3 婚车搜索 19](#_Toc10066800)

[4.2.4 婚车详情 20](#_Toc10066801)

[4.2.5 婚车租赁 20](#_Toc10066802)

[4.2.6 婚车收藏 21](#_Toc10066803)

[4.2.7 热门婚车推荐 21](#_Toc10066804)

[4.2.8 个人购买/收藏/发布列表查看 22](#_Toc10066805)

[4.2.8 价格验证 22](#_Toc10066806)

[4.3本章小结 23](#_Toc10066807)

[第5章 系统运行与测试 24](#_Toc10066808)

[5.1 系统测试环境 24](#_Toc10066809)

[5.2 系统功能测试 24](#_Toc10066810)

[5.2.1 注册/登录模块 24](#_Toc10066811)

[5.2.2 婚车管理模块 25](#_Toc10066812)

[5.2.3 用户信息管理模块 25](#_Toc10066813)

[5.2.4 价格验证模块 26](#_Toc10066814)

[5.3 本章小结 27](#_Toc10066815)

[第6章 总结与展望 28](#_Toc10066816)

[参考文献 29](#_Toc10066817)

[致谢 30](#_Toc10066818)

[附录A 英文翻译 31](#_Toc10066819)

[一、 原文 31](#_Toc10066820)

[二、 翻译 45](#_Toc10066821)

# 

# 第1章 引言

随着计算机技术和网络技术近几年的高速发展，互联网已经影响到人们学习、生活和工作的各个方面。作为一个具有良好发展前景的行业，婚车租赁行业也需要计算机技术和互联网的参与。

## 1.1 研究背景和意义

婚庆作为朝阳产业，它的年产值逐年增加。婚庆公司可以将摄影工作室，酒店，珠宝，汽车租赁行业与婚礼消费者全面整合。一方面为商家提供了一个平台，可以用来发布婚庆相关产品；另一方面，消费者可以通过婚庆公司获取这些产品信息。婚礼汽车服务在传统婚礼服务中起着举足轻重的作用。通常婚车租赁公司的经营方式是：公司不拥有车辆本身，但有许多私人车主和汽车俱乐部的汽车资源可以提供所需的服务。婚车服务车，当有婚车服务需求时，婚车租赁公司向车主支付合理的车辆使用费，租车然后进行车辆整合和包装，为客户提供婚车服务并收到合理的服务费。传统情况下，婚车租赁公司作为婚车服务过程中的平台连接到车主和客户，提供给客户的婚车服务，是在租用婚车基础上附加了租赁公司的增值服务，这些增值服务包括准时舒适，安全驾驶，顺畅沟通，服务热情等。婚礼汽车租赁公司可以提高增值服务的质量引入更多的婚礼客户，但婚车租赁公司提供这些服务费用昂贵，而且这些费用都转嫁给了顾客。而在传统的婚车租赁公司，业务流程仍然依赖于纸质的会计账户，缺乏可迅速查询的管理信息数据库，操作不灵活。 业务点在地理上分散，公司之间缺乏互操作性，更不用说统一管理了。严重拖延婚车租赁业务的发展。近几年出现了一些婚车租赁平台，但这些平台普遍缺少了对租赁价格数据的保护，在网络环境并不安全的今天，价格作为一个敏感数据，非常容易被不法分子篡改。本文的意义一方面在于减少婚车租赁的成本，跳过婚礼公司这一环节，为婚礼车主和婚车租赁者提供统一管理的租赁平台。另一方面本文提供了一个价格完整性认证方案，保护租赁价格数据。确保租赁者看到的价格数据是未被篡改的。

## 1.2 主要内容及章节安排

本文描述的婚车租赁系统是按照软件工程学理论，从系统的需求分析、系统设计、系统实现、系统测试到最后的运行维护等五个阶段进行系统的研究。

全文共六章，章节安排如下：

第一章为引言，该部分主要介绍了婚车租赁业的现状，分析了当前婚车租赁管理方式的落后和不足，从而引出本系统开发的意义和目的。并且对论文的章节安排和研究内容进行简单的介绍。

第二章为婚车租赁系统的相关理论介绍。该部分主要对完整性验证技术以及系统开发过程中用到的其它关键技术进行分析，为系统的进一步设计和实现奠定了技术基础。

第三章为婚车租赁系统的需求分析和概要设计。首先，根据业务场景对系统进行了功能需求分析和非功能需求分析。然后，根据系统需求分析结果，依据软件工程思想设计了婚车租赁系统的总体架构。最后，进行系统的功能模块设计和数据库设计。

第四章为婚车租赁系统的实现。根据需求分析和系统设计的结果，具体实现各功能模块，给出各功能模块的实现效果和核心代码。 完成了整个婚车租赁系统的开发。

第五章为婚车租赁系统的测试。针对用户的业务需求和系统的功能要求进行了测试。成功验证系统设计和开发结果的正确性。

第六章为论文的总结，对本文描述的婚车租赁系统进行了总结和展望。

# 第2章 相关技术介绍

## 2.1完整性验证技术

信息完整性验证，也称为“消息验证”，要求检测到对接收信息的任何更改。其主要方法是利用特殊的数学函数对信息的一些关键元素进行变换运算，得到一个固定长度的“摘要值”，相当于信息的“数字指纹”。发送信息时，摘要值同时发送给接收者。接收到信息后，接收者根据相同的算法再次转换信息，并将获得的摘要值与发送的摘要值进行比较。如果两个值相同，则确认该信息是真实有效的，否则可以确定该信息已被篡改。

### 2.1.1 CRC校验

循环冗余校验（CRC）是数据通信领域中最常用的错误校验码之一。 其特征在于可以任意选择信息域和检查域的长度。 循环冗余校验（CRC）是数据传输错误检测功能。 它以多项式计算数据，并将结果附加到帧的后面。 接收设备还执行类似的算法以确保数据传输的正确性和完整性。

循环冗余校验码（CRC）的基本原理是：在发送方要发送的K位数据码后，以一定的规则产生一个r位用于校验的监督码，附加在原数据后面，构成的信息码为n=k+r位，因此，这种编码又叫(n，k)码[13] 当已经给定了（n，k）码后，可以证明存在最大功率为n-k=r的多项式G（x）。得到G（x）后，可以根据G(x)生成K位的校验码，所以称G（x）为该CRC码的生成多项式。 校验码的具体生成过程如下：假设用多项式C(x)表示要发送的信息，将C(x)向左移动R位（可表示成C(x)\*2R），这样C(x)的右边就会多出有R位空缺，这就是校验码所在的位置。用 C(x)\*2R 除以生成多项式G(x)最后结果所得到的余数就是校验码。

## 2.2系统开发技术

### 2.2.1 react-native技术

React Native是开源的JavaScript框架。开发人员可以使用它开发跨平台移动应用程序。在不到一年的时间里，它已经成为手机端必不可少的发展模式之一。目前，react-native基本完成了对多终端的支持，实现了真正的面向配置的开发：开发者可以使用js开发项目。最终效果：android、ios两个端代码可以重用。它的核心设计理念：既有本机的用户体验，又保留了React的开发效率。

1）React Native的特点:

（1）使用了 Virtual DOM(虚拟DOM)。

（2）提供了响应式(Reactive)和组件化(Composable)的视图组件。

（3）专注于核心库，与此同时，有负责处理全局状态管理的库和配套的路由。

2）React Native的优势:

（1）快速编译：与编译需要很长时间的Xcode的本机代码相比，React Native使用了一种热加载，即时编译方法，使App UI开发体验更加完美，几乎与Web开发一样容易随时更改。

（2）快速发布：可以通过更新远端Js，直接更新App[8]。相比原来冗长的审核和上传过程，大幅提高了发布和测试新功能的效率。

（3）简单易学：不需要学习IOS和Android复杂知识系统。只需要学习React-Native相关知识。开发人员不难理解，实际操作很容易入门，易于上手。 如果是一个前端开发人员，对Javascript有相应的理解，使用React-Native开发移动应用程序是很自然的选择。

（4）代码重用度高 ：目前码农的成本是越来越高，因此代码的可 重用是很重要的一点，代码平台无关性也是每一个技术人员所喜欢 的。ReactNative的Android)~本和IOS版本的代码 可以80％以上共 用，这在当前主流的原生开发中是难以置信的。同时由于React Native技术是从PC端的React技术演变而来，因此ReactNative的 绝大多数代码也可以转化为React来使用，进行PC端的前端开发，因 此，一次开发就可以三次使用大大的降低了人力成本和时间成本， 对于企业来说是非常可喜的[7]。

### 2.2.2 SSM框架

SSM是 SpringMVC、 Spring、 Mybatis的简写。SSM作为一个轻量的框架，非常适用于小型网站的开发。Spring作为一个业务层的框架，能够很好地整合表现层的SpringMVC框架和持久层的Mybatis框架。整合以后，Spring就会充当整个应用的工厂[9]。

Spring是一个开源框架。它诞生的目的是解决复杂的企业应用程序开发。Spring使用基本的JavaBeans来执行以前只能使用EJB的事物。但是，使用Spring并不仅限于服务器端开发。从简单性、可测试性和灵活耦合的角度来看，任何Java应用程序都可以从Spring中受益。 简而言之，Spring是用于控制反转（IOC）和面向切面（AOP）的容器框架。

Spring MVC作为Spring Framework的后续产品，并已被集成到Spring Web Flow中。它支持原生的Spring功能，使开发变得更加容易。Spring MVC将模型对象、分派器、控制器以及处理程序对象分离了开来，以便于使用者自行定制。

Mybatis框架主要包括DAO组件与 SQLMap组件两大类，DAO组件主要将应用程序的数据访问层和持久层进行抽象化，从而帮助开发人员基于DAO设计的模式进行设计[10]。SQL Map组件作为 mybatis框架的重要组成部分之一，它使用xml或注解的方式将要执行的各种数据库查询配置起来，并通过java对象和xml中的sql进行映射生成最终执行sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回[11]。

### 2.2.3 MVC FrameWork

模型 - 视图 - 控制层（MVC）是一种目前正在软件工程中使用的软件架构。该模式从用户界面（输入和显示）分离“域逻辑”（用户应用程序逻辑），允许开发者独立开发每个测试和维护[1,2]。

模型层：处理基础数据，不包含任何有关用户界面信息。

视图层：用户界面，用户可以看到并响应，代表一个窗口到模型可以有很多的视图。

控制层：连接模型层和视图层[3]。

优势：

(1) 高可重用性和适用性：不改变模型层和控制层，只更改视图层，可以适应不同的客户端。

(2) 低耦合性：业务逻辑与用户界面之间的交融会影响设计进度。MVC模式可以将两者合理分离开来，为后续的系统设计开发工作奠定基础[12]。

(3) 可维护性：分为三层的分层模型使得项目更易于维护和修改。

(4) 快速开发：这种设计模式的优点是架构十分简单，开发速度非常快，特别适合业务需求简单的小型项目的开发，比如那些只需要对少量且小型数据库表进行增删改查操作的项目，有利于节省生产成本[15]。

(5) 清晰的软件架构：大多数应用系统都可以实现的MVC三层架构设计,正因如此,方便基于此设计模式建立一种可以通用框架结构,不局限于视图设计过程中,而且还可以用在系统总体布局或者局部功能设计[14]。

### 2.2.4 Mysql 数据库

MySQL 数据库是一种开源、性能出众且无平台相关性的轻量级数据库。它支持 多种语言的连接操作和不同的存储引擎，拥有强大的灵活性[4]。在进行读写操作时，MySQL使用表级锁，也就是说，当系统写入表时，其他线程将无法写入该表，因此MySQL中的整体读写机制是主表只负责写操作，并且读操作由多个其他从表执行，因此系统的读写速度非常令人印象深刻。 此外，它的内存小，稳定性好，管理和维护都非常简单，因此被广泛应用于互联网公司的一些小规模业务处理中。一些知名网站，比如淘宝的数据魔方和myfox的底层关系数据库，都由MySQL数据库存储。

## 2.3本章小结

本章主要介绍婚车租赁的相关理论知识。 本部分主要分析了完整性验证的相关理论和系统开发的相关技术。 通过本章的研究，为系统的进一步开发和设计提供了技术方案。

# 第3章 系统需求分析和概要设计

## 3.1 需求分析

### 3.1.1 系统需求分析

注释婚车租赁系统主要是由用户发布婚车出租服务。需要使用婚车的用户所搜索婚车，选择适合自己的婚车进行租赁。通过整理婚车租赁业务需求和客户实际需求，总结出系统的基本功能点。

1）注册登录：用户注册登录后才能进行发布婚车出租业务、租赁婚车业务、收藏他人发布的婚车出租信息等操作。

2）婚车搜索：用户输入关键字搜索自己想要租赁的婚车。

3）婚车租赁：用户选择想要租赁的婚车商品进行购买。

4）婚车收藏：如果用户看到符合心意的婚车商品，但又不想现在就进行购买，可以选择先收藏。

5）婚车出租信息发布：用户发布婚车出租信息，填写婚车的基本信息，比如价格，等待买家购买。

6）购买列表查看：用户在个人信息中可以看到以往自己的购买记录。

7）收藏列表查看：用户在个人信息中可以看到以往自己的收藏记录。

8）发布列表查看：用户在个人信息中可以看到以往自己的发布记录。

9）热门婚车推荐：系统会把点击率最高的10份婚车出租信息放在热门婚车推荐中，便于用户选择。

10）婚车价格验证：价格作为敏感数据，用户可以验证价格，以确保价格在网络传输过程中没有被恶意篡改。

### 3.1.2 系统非功能需求分析

1）系统的实用性

系统的实用性意味着系统设计和开发的功能应尽可能简单实用。在使用系统的过程中，用户可以感觉到系统快速方便，并且没有很多麻烦和冗余的操作或功能。

2）系统的灵活性

该系统应该能够适应婚车租赁管理需求的发展变化。

3）系统的安全性

婚礼汽车租赁系统记录重要数据，例如婚车出租信息，用户个人信息，用户租车信息等。一些信息必须具有高机密性，例如用户信息，营销数据等。因此，婚车租赁系统需要具有一定的安全性，并且可以确保重要的数据信息不容易被盗和破坏。

4）系统的稳定性

系统稳定性低表示系统在用户使用时可能存在数据操作错误,并且页面响应时间太长或无法响应。因此,系统稳定性是评估的重要指标之一。系统必须使用稳定的操作系统,数据库,中间件等。确保系统稳定。

5）系统的开放性

系统的开放性表现在系统具有良好的兼容性，比如，可以在大多数Android和IOS系统版本中流畅运行。

6）系统的可扩展性

婚车租赁系统的可扩展性是用户非常关注的问题。所谓的可扩展性是指系统在业务扩展时提供支持的能力。在实践中，用户将保留某些系统功能，以应对业务的未来发展。婚车租赁系统的未来仍有很大的发展空间。在后期进行业务转型和扩张是不可避免的。因此，系统必须具有良好的可扩展性。

7）系统的先进性

系统的先进性意味着系统应支持主流的尖端开发技术，以及在主流软件和硬件平台上运行文档的能力。未来，大规模多方核心架构服务器将成为主流。系统还必须支持多进程和多线程开发机制。

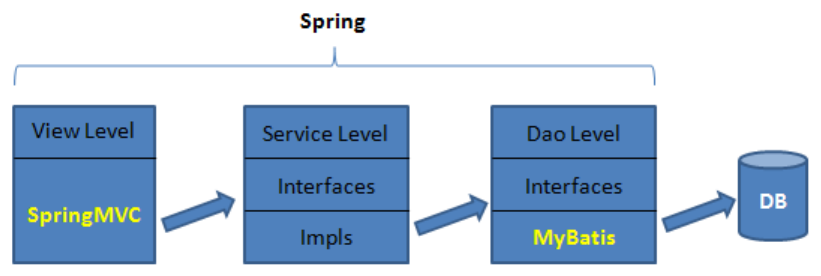
8）系统的适度超前性

对各种业务需求的分析和判断不仅要关注当前，还要科学地预见未来需要快速发展，并在规划和设计上适度推进。系统使用的开发技术应该是相对前沿的，并且在未来几年内不会很快被淘汰。

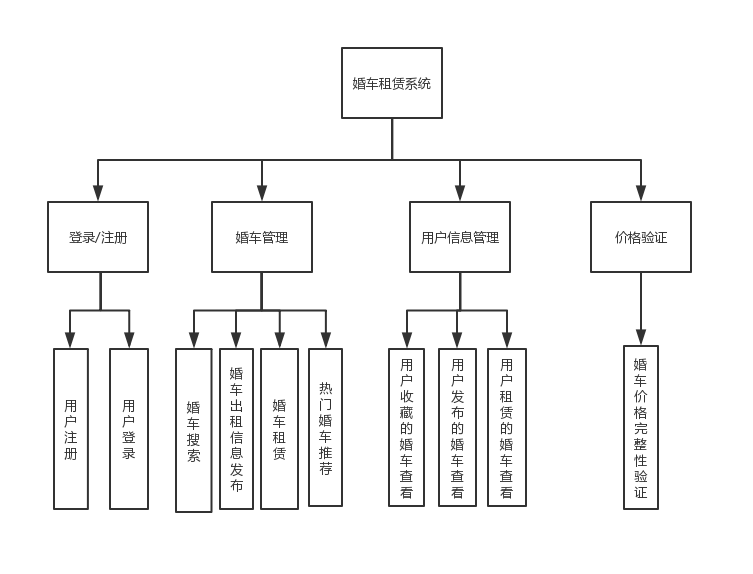
## 3.2 概要设计

### 3.2.1系统体系结构

系统采用SSM架构进行开发，通过Spring MVC框架，将系统分为表现层、业务层、数据层三层。第一层是把业务逻辑表现在 Web 页面上，实现用户与多媒体资源网络管理系统的交互。而第二层则是在数据访问层的基础之上，在逻辑上对业务流程进行描述，并根据相应情况调用数据访问层的对应功能[5]。第三层是对数据库中的数据进行增删改查基础操作。本系统采用目前最流行的 J2EE 技术实现，系统中主要应用了 SpringMVC 框架、Mybatis框架、react-native技术、Mysql数据库。



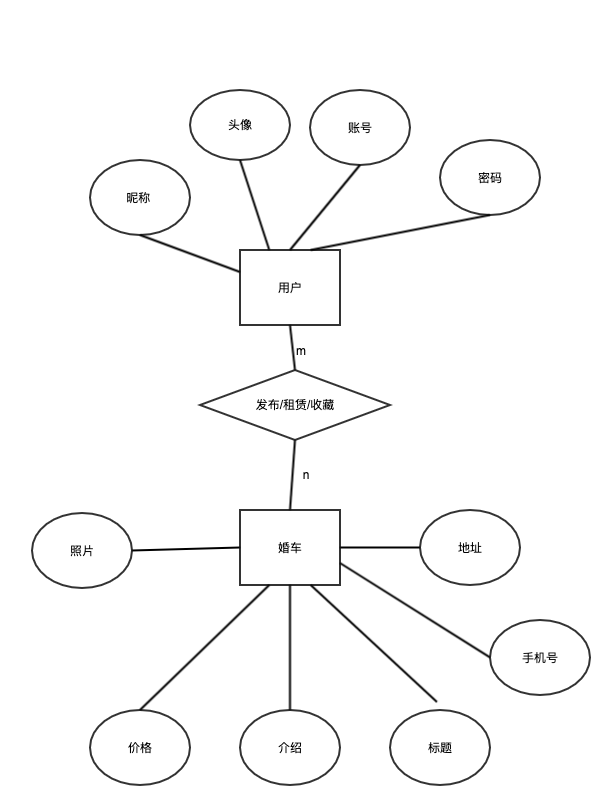
### 3.2.2系统功能模块设计图



### 3.2.3 数据库设计

#### 3.2.3.1 数据库逻辑设计

E-R图如下：



#### 3.2.3.2 数据库表的设计

1）用户表：记录用户的个人信息

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 键属性 |
| user\_id | 用户Id | 主键 |
| account | 账号 |  |
| password | 密码 |  |
| avatar\_path | 头像地址 |  |

2）婚车表：记录婚车的信息

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 键属性 |
| car\_id | 婚车订单Id | 主键 |
| title | 婚车订单标题 |  |
| description | 婚车介绍 |  |
| image\_path | 婚车照片地址 |  |
| address | 婚车所在地址 |  |
| phone | 联系电话 |  |
| price | 婚车价格 |  |
| buy\_status | 是否被购买 |  |
| brows\_times | 浏览次数 |  |

3）婚车租赁表：用户租赁婚车的记录

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 键属性 |
| buy\_id | 租赁Id | 主键 |
| car\_id | 婚车订单id |  |
| user\_id | 用户Id |  |

4）婚车订单发布表：用户发布婚车订单的记录

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 键属性 |
| release\_id | 发布Id | 主键 |
| car\_id | 婚车订单id |  |
| user\_id | 用户Id |  |

5）婚车订单收藏表：用户收藏婚车订单的记录

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 键属性 |
| collect\_id | 收藏Id | 主键 |
| car\_id | 婚车订单id |  |
| user\_id | 用户Id |  |

### 3.2.4 完整性验证算法设计

本项目中校验码的设计参考了CRC校验码的产生过程。前后端用相同的算法生成校验码。后端在传递数据的过程中，将生成的校验码也同时传递给前端，与前端生成的校验码进行比对。

算法设计如下：

1) 将16位寄存器预置为全1，将此寄存器称为寄存器A。

2) 将寄存器A右移8位。

3) 将信息帧第一个字节构成的8位二进制数据和寄存器A进行异或运算。

4) 将寄存器A的内容向右移动一位，并用0填充最高位，最后检查移出位是0或1。

5) 如果移出位是0：重复步骤4。如果移位是1：则将寄存器A和多项式A001(1010000000000001)进行异或运算。

6) 重复步骤4和5，直到右移8次，这样就可以处理完信息帧的第一个字节。

7) 对于信息帧的下一个字节，重复步骤2到6。

8) 最终寄存器A中存储的数据就是校验码。

## 3.3本章小结

本章主要对婚车租赁系统进行需求分析和概要设计。进行了详细的需求分析、总体框架设计、功能模块设计、数据库设计、完整性验证算法设计。介绍了各功能模块的具体功能和工作流程，探讨了实现该系统的技术基础。

# 第4章 系统详细设计与实现

## 4.1 系统实现环境的选择

### 4.1.1 开发工具

在充分考虑婚车租赁的实际情况和需求后，系统的运行环境将兼容Android和ios两个平台且尽可能提高运行效率。

|  |  |
| --- | --- |
| 项目 | 选用技术或平台工具 |
| 开发技术 | JavaEE, react-native |
| 开发工具 | IDEA, VS code |
| 编程语言 | Java, javascript |
| 数据库 | Mysql |
| Web 服务器 | Tomcat |
| 客户端 | Android, ios |
| 辅助工具 | Postman |

### 4.1.2 系统运行环境

1）服务器端 ：

操作系统：macOS

数据库管理系统：Mysql

Web 服务器：Tomcat

应用程序运行框架：JavaEE

2）客户端

操作系统：IOS/Android

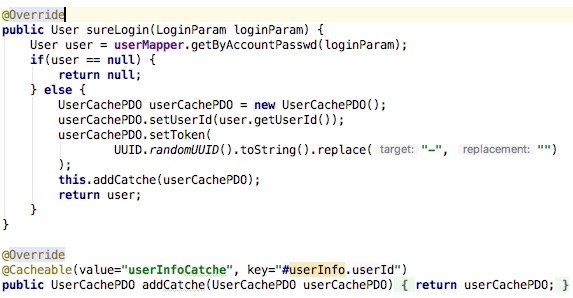
## 4.2 系统功能模块的实现

### 4.2.1 注册/登录

1）注册功能

用户填写账号、密码、昵称，上传头像后，后台查询数据库中是否已有相同账号存在，如果没有，则将用户信息存入数据库并告知用户注册成功自动跳转到登录页面。如果有相同账号存在，则告知用户账号已存在。

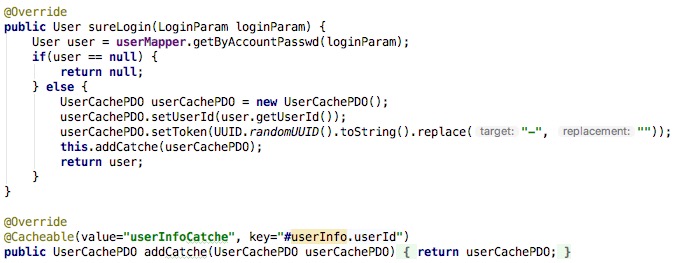
核心代码如下：



2）登录功能

在用户登录功能中，必须填写对用户名和密码才能正常登陆成功。当用户填写了账号和密码点击登录后，系统到数据库中查询是否有正确的账号密码信息。如果有，系统会生成一个token作为登录凭证，和用户信息一起存入后端缓存。后端会返回这个token和用户信息存入前端缓存，由此，用户可以正常使用系统了。如果数据库中没有正确的账号密码信息，则会提示用户账号或密码错误，用户不能正常进入系统。

核心代码如下：



### 4.2.2 婚车发布

用户既可以发布婚车订单也可以租赁婚车也就是购买其他用户发布的婚车订单。用户发布婚车订单，需要用户在app的订单发布页面填写订单标题、婚车具体描述、婚车租赁价格、联系电话、婚车所在地址、婚车照片。点击发布后，后台会将这些订单信息存入数据库，告知用户订单发布成功。

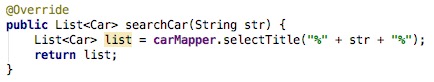
核心代码如下：



### 4.2.3 婚车搜索

为了方便用户找到自己想租赁的婚车，手机app提供了婚车搜索功能，用户可以通过已发布的婚车订单标题来找到自己心仪的婚车。流程如下：用户输入婚车关键词进行搜索，系统后台由用户输入的关键词在数据库中查找响应的婚车订单信息返回给用户。

核心代码如下：



### 4.2.4 婚车详情

该功能可以获取到婚车订单的详情。用户选择想要查看的婚车订单，点击后，后台系统在数据库查询到该订单的详细信息，将数据返回到客户端，供用户查看。具体详情信息有：订单标题、婚车具体描述、婚车租赁价格、联系电话、婚车所在地址、婚车照片。

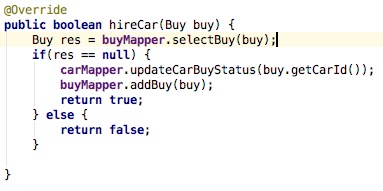
核心代码如下：



### 4.2.5 婚车租赁

用户查看了婚车详情后，如果觉得合适，可以选择租赁婚车。客户端请求通过fetch发送给后台系统后，后台系统在数据库中将此订单设置为已被租赁，其他用户就无法搜索到此订单了。

核心代码如下：



### 4.2.6 婚车收藏

用户查看某个婚车订单详情后，较为满意，但不想立即购买，则可选择先进行收藏，后续可以从个人收藏列表中快速找到此订单。流程如下：用户点击收藏后，后台系统会在数据库中将此订单和用户加入收藏表，建立收藏关系。

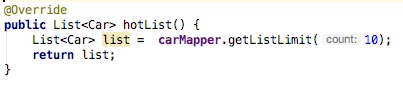
核心代码如下：



### 4.2.7 热门婚车推荐

为了给毫无头绪的用户一些建议。本APP提供了热门婚车推荐的功能。订单每次被查看一次详情，点击率会加一。用户点击率最高的10个婚车订单会在首页被推荐给用户。

核心代码如下：

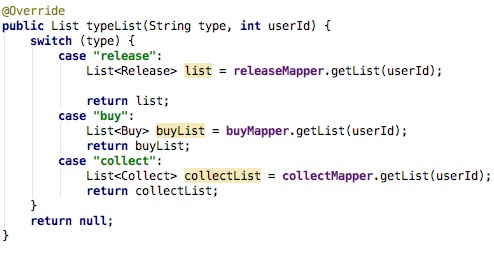


### 4.2.8 个人购买/收藏/发布列表查看

用户可以在个人信息页面下可以方便地查看到个人购买/收藏/发布的订单。

通过登录成功后缓存的用户信息和想要获取的列表类型（购买/收藏/发布列表），向后端发起fetch请求，后台系统根据用户ID，从数据库中查询到此用户的所有购买/收藏/发布订单信息，返回给客户端。

核心代码如下：



### 4.2.8 价格验证

为了保证敏感数据——价格在网络传输过程中不被恶意篡改。本APP提供了价格完整性验证功能。在前端和后端使用相同的价格完整性验证算法。后端从数据库中查询到价格后，会通过完整性算法计算出一个校验码A。后端将校验码随同价格一起传递给前端。前端接收到后端传递过来的价格和校验码A后。用相同的完整性算法，对价格得出校验码B。比较校验码A和B，如果相等，则价格在传递过程中未被篡改，如果不相等，则价格在传递过程中已被篡改。

核心代码如下：



## 4.3本章小结

本章介绍了实现婚车租赁系统的过程。功根模块严格依据系统功能设计实现。系统使用简单方便，满足用户使用要求，实现了项目目标。

# 第5章 系统运行与测试

## 5.1 系统测试环境

Web 应用服务器（Tomcat）配置：在 Tomcat 的安装目录下找到 conf 文件夹，找 到 server.xml 文件，使用文本编辑器打开进行设置[4]。

数据库配置：打开开始 --> 所有程序 -->PremiumSoft-->Navicat 8.0 for MySQL-->Navicat for MySQL->MySQL 的服务。然后打开内部 myconnector 里的 clms 数据库配置内部数据[6]。

## 5.2 系统功能测试

### 5.2.1 注册/登录模块

注册/登录模块的具体测试用例如下所示。

|  |  |
| --- | --- |
| 用例编号 | TestCase\_001 |
| 测试条件 | 测试环境搭建完成，系统能够正常运行。 |
| 步骤说明 | 1．进入系统注册页面，填写注册信息，注册成功后，页面自动跳转到登录页。  2. 在登录页填写账号密码，经过账户验证后进入首页。 |
| 预期结果 | 注册登录成功，能正常进入首页。 |
| 实际结果 | 与预期结果一致，测试通过。 |
| 总结说明 | 系统具有良好的容错能力，可以有效地完成数据库操作。 测试结果表明它满足用户的需求，测试通过。 |

### 5.2.2 婚车管理模块

婚车管理模块的具体测试用例如下所示。

|  |  |
| --- | --- |
| 用例编号 | TestCase\_002 |
| 测试条件 | 测试环境搭建完成，系统能够正常运行。 |
| 步骤说明 | 1．进入系统登录页面，经过帐户验证后，进入首页。  2. 确认首页推送了热门婚车订单。  3. 在搜索框输入婚车信息，搜索后显示相关的婚车订单列表。  4. 点击列表中的一项，显示婚车订单详情。  5. 婚车订单详情中，点击收藏/购买按钮，进行收藏/购买。  5. 点击发布按钮，填写婚车订单信息，点击确认发布婚车订单。 |
| 预期结果 | 首页推送了热门婚车订单。搜索婚车信息后，成功显示相关婚车订单，点击订单后，显示婚车详情，点击详情页的收藏/购买按钮，操作成功。点击发布，填写信息后，成功发布婚车订单。 |
| 实际结果 | 与预期结果一致，测试通过。 |
| 总结说明 | 系统具有良好的容错能力，可以有效地完成数据库操作。 测试结果表明它满足用户的需求，测试通过。 |

### 5.2.3 用户信息管理模块

用户信息管理模块的具体测试用例如下所示。

|  |  |
| --- | --- |
| 用例编号 | TestCase\_003 |
| 测试条件 | 测试环境搭建完成，系统能够正常运行。 |
| 步骤说明 | 1．进入系统登录页面，经过帐户验证后，进入首页。  2. 点击导航进入个人页。  3. 点击个人页的收藏列表/购买列表/发布列表。 |
| 预期结果 | 能看到个人收藏的所有婚车订单，个人购买的所有婚车订单，个人发布的所有婚车订单。。 |
| 实际结果 | 与预期结果一致，测试通过。 |
| 总结说明 | 系统具有良好的容错能力，可以有效地完成数据库操作。 测试结果表明它满足用户的需求，测试通过。 |

### 5.2.4 价格验证模块

价格验证模块的具体测试用例如下所示。

|  |  |
| --- | --- |
| 用例编号 | TestCase\_004 |
| 测试条件 | 测试环境搭建完成，系统能够正常运行。 |
| 步骤说明 | 1．进入系统登录页面，经过帐户验证后，进入首页。  2. 点击任意一个婚车订单查看详情。  3. 在婚车详情页中点击价格完整性验证按钮。  4. 用户手动修改价格后再次点击价格完整性验证按钮。 |
| 预期结果 | 在默认价格的情况下，点击价格完整性验证按钮，提示价格未被篡改。用户手动修改价格后再次点击价格完整性验证按钮，提示价格已被篡改。 |
| 实际结果 | 与预期结果一致，测试通过。 |
| 总结说明 | 系统具有良好的容错能力，可以有效地完成数据库操作。 测试结果表明它满足用户的需求，测试通过。 |

## 5.3 本章小结

本章主要测试了婚车租赁系统。主要介绍了测试计划的开展和测试用例的编写。在系统功能测试中，详细地分析了测试方法的设计和测试结果

# 第6章 总结与展望

通过这段时间的学习与努力，婚车租赁系统整体的开发已经基本完成，本系统的功能也已经可以满足用户的需求，并且打算在以后不断的学习和积累中对此系统进行改进。在婚车租赁系统的设计和开发过程中，已经实现了系统的基本需求。但是由于时间有限，所以系统还有一些不足之处。在毕业设计的过程中通过查询资料和请教老师同学，不仅使得自己的知识体系得到了完善，而且让自己解决问题的能力也有所提升，能够对知识进行灵活运用。

在本婚车租赁系统的设计到开发的过程中，自己学到了很多东西。比如更加熟练地掌握了使用IDEA进行有效率的后端项目开发，在react-native中如何通过数据绑定，完成数据与前台界面的完美交互。积累了对react-native项目开发中常见问题的解决办法。也让自己对Mysql数据库的使用有了更深层次的了解。最重要的是，意识到了软件设计在软件开发过程中是非常重要的，终于明白了课本中说的——需求分析是系统开发的第一步同时也是最重要的一步。

作为开发人员，只有在与用户充分沟通并了解要求后才能进行正确的系统设计; 否则，在编码过程中对需求定义的任何更改都可能导致设计和编码中的大量返工。 最后，要重视调试方法，使用有效的软件系统调试方法。在调试阶段，每个模块的调试过程比相应的开发工作复杂得多。所有相同或不同的操作以及提交给后端服务器的每个请求都可能包括许多小代码执行差异。本系统在调试阶段主要采用单步调试方法，通过查看每句代码在代码执行之前和之后变量值或其他参数值的更变，来定位BUG并最终解决问题。

此外，在该系统的设计和开发过程中，我对软件项目开发过程有了进一步的了解，也为今后的工作和学习积累了宝贵的经验。本项目使我不仅可以积累学习经验，更重要的是，积累经验并在实践中提高了自己。同时，我也意识到，在一个人同时开发前端和后端的情况下，项目中的个人经验不足，水平有限。许多方面仍需要继续学习和进一步提高自己。意识到自己应该在未来的学习和工作中更加努力，获得更多进步以提高自己的能力。

# 参考文献

1. 杨艳华. 探讨高质量JSP项目开发技巧[J]. 电子技术与软件工程, 2016, 02: 197.
2. Rachit Mohan Garg, Yamini Sood, Balaji Kottana, Pallavi Totlani. A Framework Based Approach for the Development of Web Based Applications Waknaghat [J]. Jaypee University of Information Technology, 2011, 1(1): 1-4.
3. 张俐, MVC模式在数据中间件中的应用[J]. 计算机工程, 2010(9): 70-72.
4. 林锦标. 基于B/S模式汽车租赁管理系统的设计与实现[D].
5. 李 敏. 使用Java语言实现数据的查询和导出 [J]. 电脑编程技巧与维 护, 2015, 24: 68-69.
6. 王飞. 基于B/S模式的企业信息文档管理系统的设计与实现[J]. 中国电子商务, 2014(19): 55-55.
7. 焦锋. react native在app开发中的应用研究[J]. 数字技术与应用, 2017(6): 109-109.
8. 严新巧. 基于移动开发现状探讨React Native[J]. 电脑知识与技术, 2016(32).
9. 李洋. SSM框架在Web应用开发中的设计与实现[J]. 计算机技术与发展, 2016(12): 190-194.
10. 陈欣. 基于java三层构架的管理信息系统中DAO层的构建探索[J]. 科技资讯, 2015, 13(11): 26-27.
11. 邹红霆. 基于SSM框架的Web系统研究与应用[J]. 湖南理工学院学报（自然科学版）, 2017(1).
12. 张行健. 探究ASP.NET平台中MVC设计模式的应用[J]. 电脑编程技巧与维护, 2017(4): 26-26.
13. 王玉玲, 王燕锋. CRC16校验码移位算法及VHDL实现[J]. 电子技术与软件工程, 2017(7): 189-190.
14. 张睿涵. 基于经典MVC架构的实现Web应用系统设计与实现[J]. 数码世界, 2017(4): 17-18.
15. 梅本祥. 基于MVC模式的车载服务平台监控管理系统研究[D].

# 致谢

在重庆邮电大学度过了四年的紧张学习时光，系统地学习了计算机相关的各方面知识，深深的佩服各位专业老师的学识，从中我不仅学习到计算机专业知识，而且学到很多做人、做事、做学问的道理，在此表示真挚的谢意。在论文即将完成之际，我要感谢我的导师唐飞老师。在论文撰写的整个过程中，从论文选题、到撰写开题报告、最后到正文撰写，唐老师都提出了很多宝贵意见。唐老师指出的每一个问题，指导的每一个思路，都使我有醍醐灌顶之感。

此外，还要感谢朋友以及同学们在论文编写中提供的大力支持和帮助，给我带来极大的启发。也要感谢参考文献中的作者们，通过他们的研究文章，使我对研究课题有了很好的出发点。

最后，谢谢论文评阅老师们的辛苦工作。衷心感谢我的家人、朋友，以及同学们，真是在他们的鼓励和支持下我才得以顺利完成此论文。

# 附录A 英文翻译

1. 原文

**Crowdsourced Data Integrity Verification**

**for Key-Value Stores in the Cloud**

Grisha Weintraub

*Dept. of Mathematics and Computer Science The Open University Raanana, Israel*

Ehud Gudes

*The Department of Computer Science Ben-Gurion University of the Negev Be’er-Sheva, Israel*

Abstract

Thanks to their high availability, scalability, and usability, cloud databases have become one of the dominant cloud services. However, since cloud users do not physically possess their data, data integrity may be at risk. In this paper, we present a novel protocol that utilizes crowdsourcing paradigm to provide practical data integrity assurance in key- value cloud databases. The main advantage of our protocol over previous work is its high applicability - as opposed to existing approaches, our scheme does not require any system changes on the cloud side and thus can be applied directly to any existing system. We demonstrate the feasibility of our scheme by a prototype implementation and its evaluation.

Keywords:cloud; NoSQL; key-value stores, data integrity; secure storage;

I. INTRODUCTION

Relational database management systems (RDBMS) are not the only solution for data storage any more. A large number of non-relational databases have been developed in recent years in order to manage continuously growing amounts of data. The main characteristics of these systems, often referred to as NoSQL [1,2], are flexible schema, horizontal scaling and relaxed consistency. They store and replicate data in distributed systems, commonly across dat- acenters, thereby achieving scalability and high availability. NoSQL databases are usually classified into three groups, according to their data model:

* Key-value stores: Data is stored as key-value pairs, such that the key is a unique identifier and a value is an arbitrary entry.
* Document-based stores: Data is stored in a document- like structure such as JSON or XML.  • Column-oriented stores: Data is organized in tables, which consists of row keys and column keys. Column keys are grouped into sets called column families.  In this paper, we will focus on the first group - key-value stores.  Key-value stores can be used either as internal database systems or as cloud databases - cloud services that provide users with access to data without the need for managing hardware or software. However, storing data in a cloud in- troduces several security concerns. In particular, since cloud users do not physically possess their data, data integrity may be at risk. Cloud providers (or some malicious entity) can change users’ data, omit some of the data from query results or return a version of the data which is not the latest. In other words, data correctness, completeness and freshness might be compromised.
* Data integrity in outsourced relational databases has been studied for several years [13-16, 18-21]. Nevertheless, exist- ing solutions are inappropriate for key-value stores for the following reasons:
* Data volumes in key-value stores are expected to be much higher than in RDBMS and therefore data is (usually) distributed across many different nodes.
* The query model of key-value stores is much simpler than in RDBMS.  These differences between RDBMS and key-value stores in- troduce both challenges and opportunities. On the one hand, data integrity assurance in key-value systems requires more sophisticated solutions due to its distributed architecture and high data volumes. On the other hand, their extremely simple query model may allow us to design much simpler and efficient protocols for data integrity verification. We are especially interested in developing a method that does not require server side modifications, so it may be seamlessly applied to existing real world systems.  The goal of this paper is to demonstrate that data integrity of key-value stores in the cloud can be verified without server side modifications and with reasonable performance overhead. Our main contributions are as follows:
* Development of a novel probabilistic method that al- lows users to verify data integrity of the data that resides in cloud key-value stores and its analysis.
* A demonstration of the feasibility of our method through a prototype implementation and its experimen- tal evaluation.  The rest of the paper is structured as follows: Section II outlines system and threat models. Section III presents our method for data integrity verification. Security analysis of our approach is presented in Section IV. Section V intro- duces our proof-of-concept implementation and provides an experimental evaluation thereof. Section VI reviews related work and Section VII concludes the paper.

II. SYSTEM AND THREAT MODELS

A. System Model and Assumptions

Database-as-a-service paradigm (DBaaS), firstly intro- duced more than a decade ago [3], has become a preva- lent service of today’s cloud providers. Micorosoft’s Azure SQL Database, Amazon’s DynamoDB and Google’s Cloud BigTable are just a few examples.

We assume that there are three entities in the DBaaS model:

* Data owner (DO): uploads the data to the cloud.
* Cloud provider (CP): stores and provides access to the  data.
* Clients: retrieve the data from the cloud.  There is only one instance of DO and CP in our model, whereas the number of clients is not limited. We will assume that data uploaded to the cloud is stored in a key-value database. We model key-value database as a set of tuples D = {(k, v)}, where k is the key and v is the value associ- ated with k. The DO uploads data to the cloud by sending to the cloud bulks of the predefined size B = {(k,v)}. By sending a key k to the cloud, client retrieves a tuple (k, v) ∈ D.  Our system model is both write and read intensive. We assume that the DO only appends new values to the database; existing tuples are not updated. While privacy in the cloud is a topic that received much attention (see for example [22-24]), our model focuses on data integrity. As already mentioned above, we are interested in a highly applicable solution and therefore we assume that no server changes can be performed on the cloud side.  B. Integrity and Attack Model  We assume that the CP is not trusted neither by the DO nor by the clients and that it can behave maliciously in any possible way to break data integrity. For example, the CP may modify some of the tuples, add or delete tuples, or return partial (or empty) results to the clients’ queries.  We focus on data integrity protection in the following two dimensions:
* Correctness: Data received by the clients was originally uploaded to the cloud by the DO and has not been modified maliciously or mistakenly in the cloud side.
* Completeness: The CP returns to the clients all the data that matches the query. In other words, no data is omitted from the result.  Freshness is another important dimension of data integrity, meaning that the clients get the most current version of the data that was uploaded to the cloud. However, since in our system model there are no updates, freshness is not an issue.

III. OUR APPROACH

In our model, clients query the cloud by “get value by key” queries. The result returned by the CP may be either empty or not. In case it is not empty, the client only needs to check its correctness, and that can be easily achieved by data authentication. The major focus of our work is to ensure that empty results are supposed to be empty. In other words, we want to ensure that the results of clients’ queries are *complete*.

Below we describe our protection techniques for correct- ness and completeness verification.

A. Preliminaries

1) Hash Function: We use collision-resistant hash func- tion that has a property that it is computationally hard to find two inputs that hash to the same output. SHA-256 and SHA- 512 [4] are examples of such functions. Hash operation on value x is denoted by H(x).

2) Secret Keys: We assume that the DO and the clients share two secret keys {Ke , Km }; Ke for data encryption and Km for data authenticity.

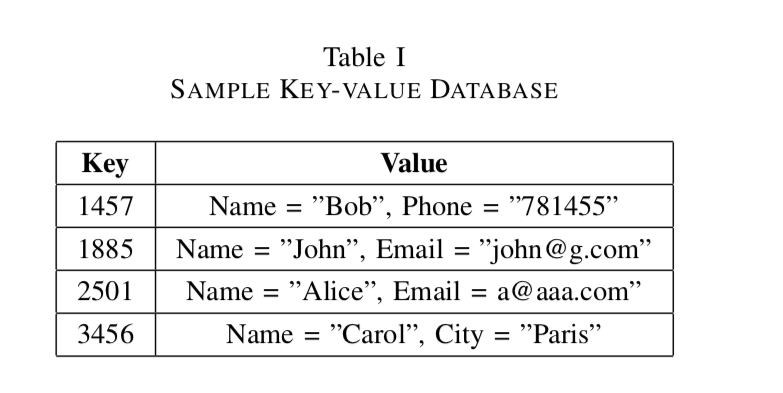
3) Data Authentication: To verify data authenticity we use message authentication codes (MAC’s). The DO signs its data according to the MAC scheme (e.g. HMAC [5]) and stores the MAC value in the cloud along with the signed data. Then, based on the MAC value and the re- ceived data, clients can verify data authenticity. Signing and verification operations are denoted by Sign(data, key) and V erif y(data, M AC, key).

4) Data Encryption: Sensitive data that is stored in the cloud is encrypted by the DO and then decrypted by the clients by using symmetric encryption (e.g. AES [6]). We denote encryption and decryption operations as Enc(plaintext, key) and Dec(ciphertext, key).

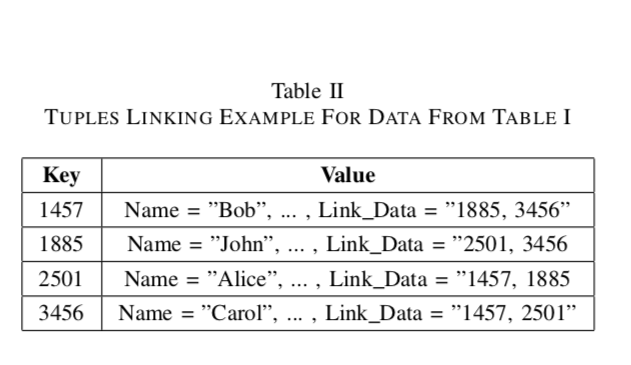
B. Completeness Verification Scheme

Our goal is to verify that no data was omitted from the results of the client queries. In our approach, we use two techniques - tuples linking and crowdsourced verification described below.

1) Tuples Linking: The Intuition behind the tuples linking is that every tuple is aware about the existence of some other tuples. For example, consider the following key-value database (Table I), where key is a user id and value is a user object.



If we would apply tuples linking to the sample data from Table I, the result might look as in Table II.



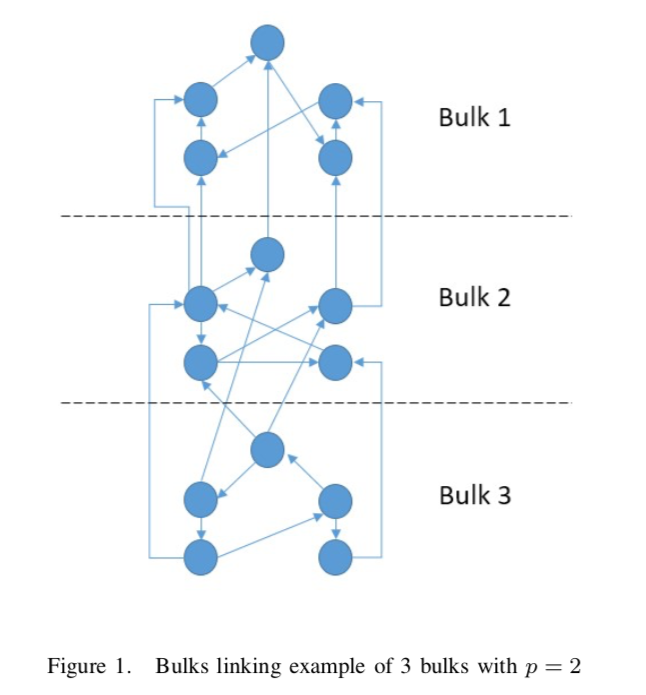
The formal definition of tuples linking is as follows: ∀(k1, v1), (k2, v2) ∈ D, (k1, v1) is linked to (k2, v2) ⇐⇒ k2 ⊂ v1 (1)

The DO is responsible for the linking between the tuples when uploading the data to the cloud. Afterwards, the clients can rely on this linking data to verify the result completeness. For example, consider a query ”get users with ids 1457, 1885” on data from Table II and the result that contains only tuple with id 1457. By checking linking data of the tuple 1457, the client knows that tuple 1885 should be a part of the result and thus detects the attack. The implementation of the tuples linking is based on the assumption that the DO uploads its data to the cloud by bulk loading, and therefore tuples inside a bulk may be linked one to another forming a bulks linking. However, it is not enough to link between the tuples of the same bulk, because if the CP removes all the bulk there is no way to detect that. We suggest that the DO keeps the copy of the previous bulk locally and when the new bulk is arrived, links its tuples to both the current bulk and the previous one. The number of linked tuples is defined by a parameter p. The bulks linking can be formally defined as follows:

∀Bi,Bi+1 ∈ D,x ∈ Bi,p > 0, there are tuples in and tuples in that are linked to x(2)

The illustration of the bulks linking of 3 bulks with 5 keys each is shown in Fig. 1:

* Graph nodes represent database tuples.
* An edge from the node A to the node B depicts that A is linked to B (A knows about the existing of B).2) Crowdsourced verification: In crowdsourcing (CS) systems [7] users collaborate to achieve a goal that is beneficial to the whole community. The collaboration may be explicit (e.g. Wikipedia and Linux projects) or implicit as in ESP game [8] where users label images as a side effect of playing the game. In our approach we build CS system where users implicitly collaborate to achieve a mu- tual goal database integrity assurance. Unlike the previous work which focused on the query integrity (clients verify only their own queries), our focus is on the database integrity (clients execute random verification queries to verify that the CP behaves honorably). The rationale behind this strategy is



based on the observation that the database integrity is more important than the query integrity; clients would not like to work with the CP that returns provably wrong results to some of the queries (even if these queries are fake and were issued by other users). We rely on this observation in our CS system, where each client contributes a little bit of his computational power in order to verify that the CP can be trusted. It works as follows:

* A client sends a query to the CP.
* The CP sends the query result along with the linking data back to the client.
* The client builds verification queries based on the received linking data and sends them to the CP.
* TheCPsendstheresultoftheverificationqueriesback to the client.
* The client verifies that the result of the verification queries matches the linking data.

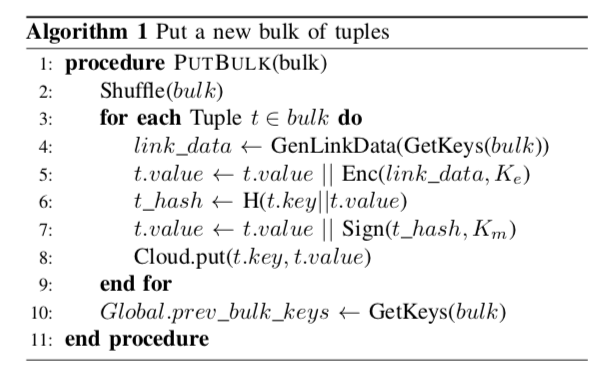
Verification queries (step 3) are built such that the CP can- not distinguish between them and the regular client queries (step 1). Thanks to that, CP’s malicious behavior with the client queries will inevitably cause malicious behavior with the verification queries as well and thus will be detected. For example, consider a query ”get user with id 1885” on data from Table II. The verification queries for this client query will be ”get user with id 2501” and ”get user with id 3456”. If the result of at least one of these queries is empty, the attack is detected.

Note that there is no dependency between client query (step 1) and verification queries (step 3) and hence steps 3- 5 can be executed asynchronously (i.e. without hurting reads latency).

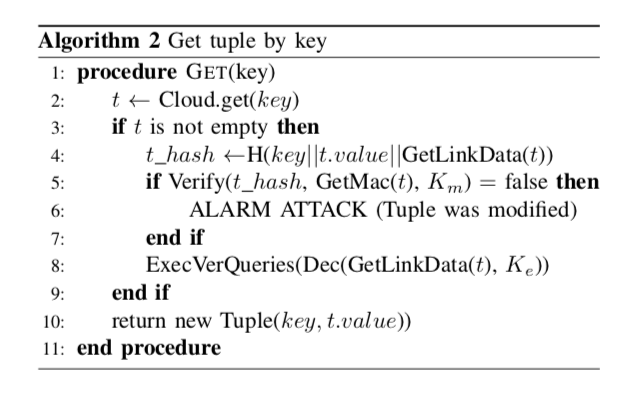
C. Pseudo-Code

The pseudo-code of the algorithms that provide both correctness and completeness verification in our model is presented below.

Algorithm 1 puts a bulk of tuples into the cloud database. It iterates over all the tuples in the bulk (lines 3-9) and calculates linking data for each tuple. Linking data then is encrypted and stored as a part of the tuple’s value. In order to be able to link the next bulk to the current, we store the current bulk keys in a global variable (line 10).



Algorithm 2 returns a tuple by key. First the correctness of the tuple is verified in lines 4-6. Completeness verification is based on verification queries which are built based on the decrypted linking data (line 8).



IV. SECURITY ANALYSIS

A. Correctness

Our correctness verification scheme is directly based on the well-known primitives (MAC scheme and collision resistant hash function) and is secure as long as they are.

B. Completeness

Our approach for completeness verification is based on two techniques: tuples linking and crowdsourced verifica- tion, described above.

Assuming a uniform distribution of both deleted tuples and range of queries, the probability that the clients will detect that d tuples were deleted from the DB (or omitted from the result) with |D| tuples after Q queries is:

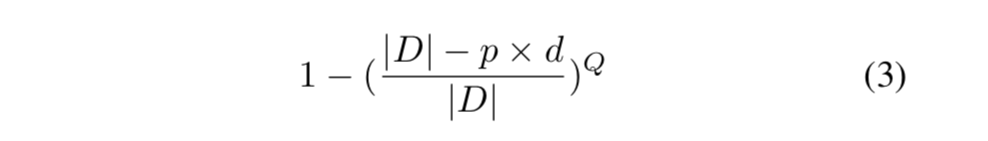
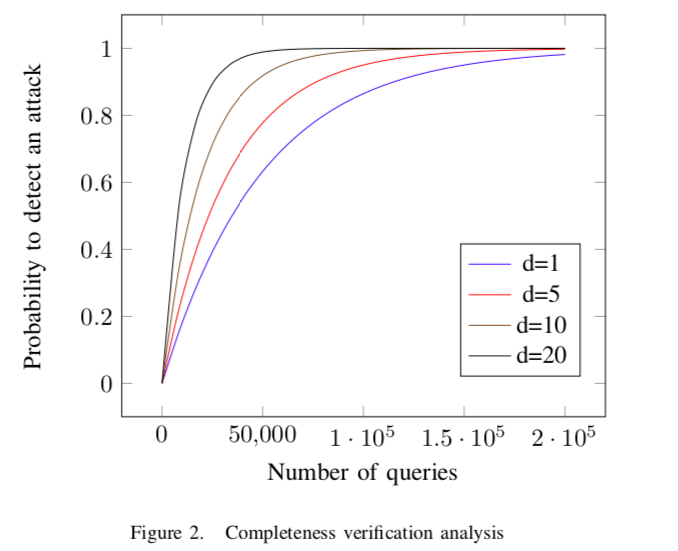


Fig. 2 shows the probability to detect an attack as a function of number of queries performed by the clients with |D| = 1.000.000, p = 4 and d ∈ {1,5,10,20}. It can be seen that even with p as small as 4, after a relatively small number of queries (production systems receive tens of thousands of queries per second [25]) and deleted tuples, the chance of the CP to escape from being caught is very low.



V. IMPLEMENTATION AND EXPERIMENTAL RESULTS

For experimental evaluation, we have implemented a prototype of our solution. As a cloud key-value store we use Redis [9]. In order to evaluate our solution, we use Yahoo! Cloud Serving Benchmark (YCSB) framework [10]. The only thing needs to be done in order to benchmark a particular database with YCSB is to implement a database interface layer. This will allow framework client to perform operations like ”read tuple” or ”insert tuple” without having to understand the specific API of the database.

We use YCSB in the following way:

• YCSB framework already has Redis client implemen-  tation called RedisClient.

• We implemented our version of Redis client (IRe-  disClient) based on algorithms from Section III. Our  implementation is available online [11].

• We configured a workload in which predefined num-  ber of interchangeable read and insert operations are executed against the database.

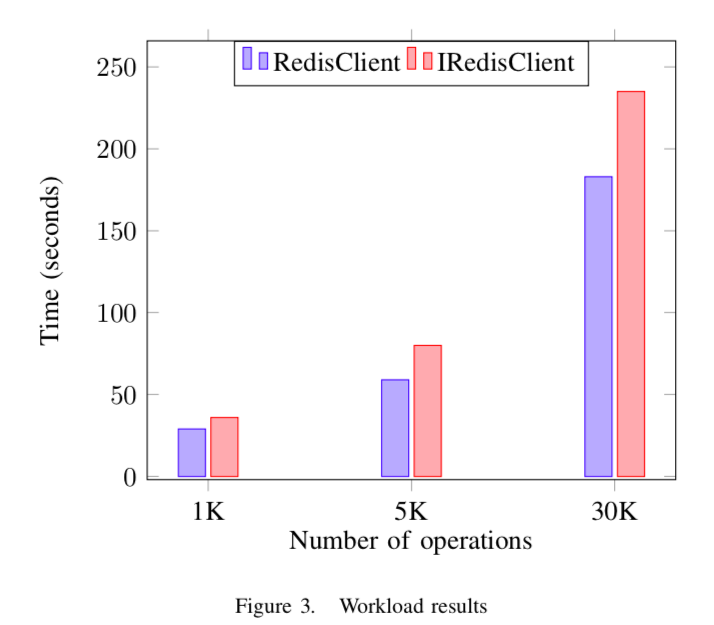
• We executed this workload on both clients and com- pared their execution time. The results are presented below.

A. Setup

We used tuples of 1KB size with random values. We defined a bulk size to be 100 and the parameter p (number of linked tuples) to be 4. The workload was performed on database that initially contained 30.000 tuples. We executed the workload three times for each client with 5.000, 10.000 and 30,000 operations. The results below represent the average value of these three executions.

B. Performance Analysis

The cost of IRedisClient insert operation is dominated by two encryption and one hash operations (MAC calculation and encryption of linking data). The cost of IRedisClient read operations is similar to the cost of inserts (MAC calculation and linking data decryption) with an additional cost of p verification queries. Experimental results (Fig. 3) show that this overhead increases execution time by 29 % in average.



VI. RELATED WORK

Existing solutions can be mainly categorized into three types. The first type is based on the Merkle Hash Tree (MHT), the second is based on digital signatures (DS), and the third uses a probabilistic approach.

A. MHT-based approach

MHT [12] is a binary tree, where each leaf is a hash of a data block, and each internal node is a hash of the concate- nation of its two children. Devanbu et al. in [13] introduce a method that uses MHT as Authenticated Data Structure (ADS) to provide data integrity assurance in DBaaS model. The general idea is to build MHT for every database table such that MHT’s leaves are hashes of table’s records ordered by a search key. To reduce I/O operations cost in both client and server sides, instead of using binary trees, trees of higher fanout (MB-Trees) can be used [14]. Different MHT- based techniques to provide efficient integrity assurance for join and aggregate queries are presented in [15] and [16] respectively. MHT-based approach does not suite our system model since it was designed for range queries and does not support efficient queries on arbitrary keys. It also requires significant server side changes and that is not allowed in our system model.

B. DS-based approach

A natural and intuitive approach to provide data integrity in RDBMS is to use digital signatures scheme (e.g. RSA [17]) in the following way:

* An additional column containing a hash of concate- nated record values signed by the DO’s private key is added to every table.

Clients verify records integrity by using the DO’s public key.  To reduce the communication cost between client and server and the computation cost on the client side, signature ag- gregation technique [18] can be used to combine multiple record signatures into a single one. In our approach, we use a similar technique for data correctness (the only difference is that we use MAC’s instead of DS and do not use aggregation). To guarantee completeness, rather than sign individual records, the DO signs consecutive pairs of records [19]. In order to do that, the DO must either have the copy of the database locally or be able to get the consecutive records from the cloud; both options are not possible in our model.  C. Probabilistic approach  Probabilistic approaches provide only probabilistic in- tegrity assurance, but do not require DBMS modifications and have better performance than MHT-based and DS-based approaches. In this approach, a number of additional records is uploaded to the cloud along with the original records. The more additional records are being uploaded, the higher is the probability to detect data integrity attack. All data is en- crypted on a client side so the CP cannot distinguish between the original and the additional records. These additional records may be completely fake as was proposed in [20] or original records encrypted with a different (secondary) secret key as was proposed in dual encryption scheme [21]. Fake-records approach is appropriate only for range queries and therefore is not relevant for our model, whereas dual-encryption scheme could be applied to our model, but requires that the whole database must be encrypted. In our scheme only a small number of keys (i.e. linking data) in each tuple are encrypted. Compared to our approach,   dual-encryption scheme also requires much more additional storage on the cloud side due to records duplication.

VII. CONCLUSION

In this paper, we present our novel method for data integrity assurance in cloud key-value stores. Our method relies on crowdsourcing paradigm - users collaborate to achieve a mutual goal - database integrity assurance. The main advantage of our method over existing approaches is its high applicability - it can be applied to existing systems without modifying server side. We implemented a proof-of-concept prototype of our protocol and conducted experimental evaluation thereof. The results show that our scheme imposes a reasonable overhead. For future work, we plan to extend our scheme on additional types of cloud databases (e.g. column stores).

REFERENCES

1. Leavitt, Neal. ”Will NoSQL databases live up to their promise?.” Computer 43.2 (2010): 12-14.
2. Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.
3. Hacigms, H., Iyer, B., Mehrotra, S. (2002). Providing database as a service. In Data Engineering, 2002. Proceed- ings. 18th International Conference on (pp. 29-38). IEEE.
4. Standard, S. H. National Institute of Standards and Technol- ogy (NIST), FIPS Publication 180-2, Aug 2002.
5. Krawczyk, H., Canetti, R., Bellare, M. (1997). HMAC: Keyed-hashing for message authentication.
6. Pub, N. F. (2001). 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication, 197, 441-0311.
7. Doan, A., Ramakrishnan, R., Halevy, A. Y. (2011). Crowd- sourcing systems on the world-wide web. Communications of the ACM, 54(4), 86-96.
8. Von Ahn, L., Dabbish, L. (2004, April). Labeling images with a computer game. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 319-326). ACM.
9. Redis. http://redis.io/
10. Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R. (2010, June). Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143-154). ACM.
11. IRedisClient. https://github.com/grishaw/ycsb-iredis-binding
12. Merkle, R. C. (1989, August). A certified digital signature. In Advances in CryptologyCRYPTO89 Proceedings (pp. 218- 238). Springer New York.
13. Devanbu, P., Gertz, M., Martel, C., Stubblebine, S. G. (2003). Authentic data publication over the internet. Journal of Computer Security, 11(3), 291-314.
14. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2006, June). Dynamic authenticated index structures for outsourced databases. In Proceedings of the 2006 ACM SIGMOD inter- national conference on Management of data (pp. 121-132). ACM.
15. Yang, Yin, et al. ”Authenticated join processing in outsourced databases.” Proceedings of the 2009 ACM SIGMOD Interna- tional Conference on Management of data. ACM, 2009.
16. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2010). Authenticated index structures for aggregation queries. ACM Transactions on Information and System Security (TISSEC), 13(4), 32.
17. Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.
18. Mykletun, E., Narasimha, M., Tsudik, G. (2006). Authentica- tion and integrity in outsourced databases. ACM Transactions on Storage (TOS), 2(2), 107-138.
19. Narasimha, M., Tsudik, G. (2005, October). DSAC: in- tegrity for outsourced databases with signature aggregation and chaining. In Proceedings of the 14th ACM international conference on Information and knowledge management (pp. 235-236). ACM.
20. Xie, M., Wang, H., Yin, J., Meng, X. (2007, September). Integrity auditing of outsourced data. In Proceedings of the 33rd international conference on Very large data bases (pp. 782-793). VLDB Endowment.
21. Wang, H., Yin, J., Perng, C. S., Yu, P. S. (2008, October). Dual encryption for query integrity assurance. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 863-872). ACM.
22. Hacigm, H., Iyer, B., Li, C., Mehrotra, S. (2002, June). Executing SQL over encrypted data in the database-service- provider model. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (pp. 216- 227). ACM.
23. Damiani, E., Vimercati, S. D. C. D., Jajodia, S., Paraboschi, S., Samarati, P. (2003, October). Balancing confidentiality and efficiency in untrusted relational DBMSs. In Proceedings of the 10th ACM conference on Computer and communica- tions security (pp. 93-102). ACM.
24. Ciriani, V., Vimercati, S. D. C. D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P. (2010). Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC), 13(3), 22.
25. Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M. (2012, June). Workload analysis of a large-scale key- value store. In ACM SIGMETRICS Performance Evaluation Review (Vol. 40, No. 1, pp. 53-64). ACM.
26. 翻译

**针对云中键值存储的众包数据完整性验证**

Grisha Weintraub以色列开放大学 数学与计算机科学系

Ehud Gudes 以色列内盖夫贝尔谢巴本古里安大学 计算机科学系

摘要

由于其高可用性，可扩展性和可用性，云数据库已成为主要的云服务之一。但是，由于云用户在物理上不拥有其数据，因此数据完整性可能存在风险。在本文中，我们提出了一种新的协议，利用众包在键值云数据库中提供实用的数据完整性保证。我们的协议相对于以前的工作的主要优点是其高适用性 ，与现有方法相反，我们的方案不需要在云端进行任何系统更改，因此可以直接应用于任何现有系统。我们通过原型实现及其评估证明了我们的方案的可行性。

**关键词**：云计算，NoSQL，键值存储，数据完整性，安全存储

一、介绍

关系数据库管理系统（RDBMS）不再是数据存储的唯一解决方案。近年来已经开发了大量非关系数据库，以便管理不断增长的数据量。这些系统的主要特征，通常称为NoSQL [1,2]，具有灵活的模式，水平扩展和宽松的一致性。它们通常跨数据中心在分布式系统中存储和复制数据，从而实现可扩展性和高可用性。 NoSQL数据库通常根据其数据模型分为三组：

•键值存储：数据存储为键值对，使得键是唯一标识符，值是任意条目。

•基于文档的存储：数据存储在类似文档的结构中，例如JSON或XML。

•面向列的存储：数据按表组织，包括行键和列键。列键被分组为称为列族的集合。

在本文中，我们将重点关注第一组——键值存储。

键值存储可用作内部数据库系统或云数据库-云服务，可为用户提供数据访问，而无需管理硬件或软件。但是，将数据存储在云中会引入一些安全问题。尤其是云用户没有物理上的数据，数据完整性可能处于危险之中。云提供商（或某些恶意实体）可以更改用户的数据，从查询结果中省略一些数据或返回不是最新版本的数据。 其他文字，数据正确性，完整性和新鲜度可能妥协。外包关系数据库中的数据完整性问题已经被研究了几年[13-16,18-21]。 然而，现有的解决方案不适合用于键值存储。

原因如下：

•键值存储中的数据量预计比RDBMS中的数据量高得多，因此数据（通常）分布在许多不同的节点上。

•键值存储的查询模型比RDBMS简单得多。

RDBMS和键值存储之间的这些差异带来了挑战和机遇。一方面，由于其分布式架构和高数据量，键值系统中的数据完整性保证需要更复杂的解决方案。另一方面，它们极其简单的查询模型可以让我们设计更简单有效的数据完整性验证协议。我们特别感兴趣的是开发一种不需要服务器端修改的方法，使它可以无缝地应用于现有的真实世界系统。

本文的目的是证明无需服务器端修改和合理的性能开销即可验证云中键值存储的数据完整性。我们的主要贡献如下：

•开发一种新的概率方法，允许用户验证云密钥值存储及其分析中数据的完整性。

•通过原型实施及其实验评估证明我们方法的可行性。

论文的其余部分结构如下：第二节概述了系统和威胁模型。第三节介绍了我们的数据完整性验证方法。第四节介绍了我们方法的安全性分析。第五节介绍了我们的概念验证实现，并对其进行了实验评估。第六节回顾相关工作。第七章总结全文。

二、系统和威胁模型

A.系统模型和假设

数据库即服务范式（dbaas）在十多年前首次引入[3]，已成为当今云提供商的一种流行服务。Micorosoft的Azure SQL数据库，亚马逊的DynamoDB和Google的Cloud BigTable就是一些例子。

我们假设DBAS模型中有三个实体:

•数据所有者（DO）：将数据上传到云。

•云提供商（CP）：存储并提供对该服务的访问 数据。

•客户端：从云中检索数据。

在我们的模型中只有一个DO和CP实例，而客户端数量不受限制。 我们假设上传到云的数据存储在键值数据库中。我们将键值数据库建模为一组元组 *D*=*{*（*k，v*）*}*，其中*k*是键，*v*是与 *k* 相关的值。DO通过发送将数据上传到云端到预定义*B* =*{*（*k，v*）*}*。通过向云发送密钥*k*，客户端检索元组（*k，v*）*∈d。*

我们的系统模型是写入和读取密集型的。我们假设DO只将新值附加到数据库中;现有元组未更新。虽然云中的隐私是一个备受关注的话题（例如参见[22-24]），但我们的模型侧重于数据完整性。如上所述，我们对高度适用的解决方案感兴趣，因此我们假设不能在云端执行服务器更改。

B.完整性和攻击模型

我们假设DO和客户端都不信任CP，并且它可能以任何可能的方式恶意破坏数据完整性。例如，CP可能会修改某些元组，添加或删除元组，或将部分（或空）结果返回给客户端的查询。

我们专注于以下两个方面的数据完整性保护：

•正确性：客户收到的数据最初是 由DO上传到云端，未在云端恶意或错误地修改。

•完整性：CP返回所有客户端 与查询匹配的数据。 换句话说，结果中没有数据被省略。

新鲜度是数据完整性的另一个重要方面，这意味着客户端可以获得上载到云端的最新数据版本。但是，由于在我们的系统模型中没有更新，因此新鲜度不是问题。

三、我们的方法

在我们的模型中，客户端通过“按键获取值”查询来查询云。CP返回的结果可能是空的，也可能不是空的。如果它不是空的，客户端只需要检查其正确性，并且可以通过数据验证轻松实现。我们工作的重点是确保空结果应该是空的。换句话说，我们希望确保客户查询的结果是完整的。

下面我们将介绍我们用于正确性和完整性验证的保护技术。

A.预赛

1）散列函数：我们使用抗冲突散列函数，它具有一个属性，在计算上很难找到散列到同一输出的两个输入。SHA-256和SHA-512 [4]就是这种功能的例子。值x的 哈希操作由H（x）表示。

2）秘密密钥：我们假设DO和客户端共享两个密钥{Ke，Km}，Ke用于数据加密，Km用于数据真实性。

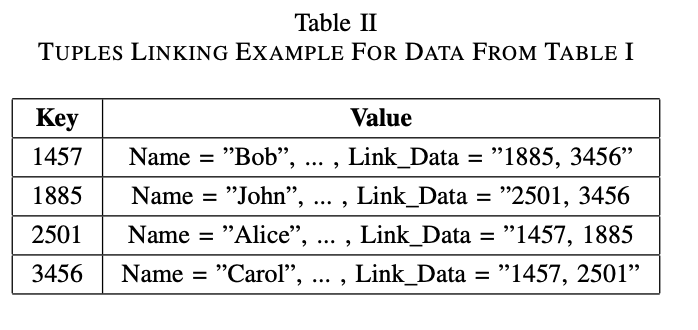
3）数据验证：验证我们的数据真实性 使用消息验证码（MAC）。DO根据MAC方案（例如HMAC [5]）签署其数据，并将MAC值与签名数据一起存储在云中。然后，基于MAC值和接收的数据，客户端可以验证数据的真实性。签名和验证操作由Sign(data, key) 和Verify(data, MAC, key) 表示 。

4）数据加密：存储的敏感数据在云由DO加密，然后由客户端使用对称加密解密（例如AES [6]）。我们将加密和解密操作表示为Enc(plaintext,key) 和Dec(ciphertext, key) 。

B.完整性验证方案

我们的目标是验证客户端查询的结果中是否没有遗漏任何数据。在我们的方法中，我们使用两种技术——元组链接和下面描述的众包验证 。

1）元组链接：元组联背后的原理是，每一个元组知道其他的一些元组的存在。例如，考虑以下键值数据库（表I），其中key是用户id，value是用户对象。



元组链接的正式定义如下：

∀(k1, v1), (k2, v2) ∈ D, (k1, v1) 被链接到 (k2, v2) ⇐⇒ k2 ⊂ v1 （1）

在将数据上传到云时，DO负责元组之间的链接。之后，客户端可以依赖此链接数据来验证结果的完整性。例如，考虑对表II中的数据进行查询“获取具有ID 1457,1885的用户”以及仅包含具有id 1457的元组的结果。通过检查元组1457的链接数据，客户端知道元组1885应该是其中的一部分结果，从而检测到攻击。

元组链接的实现基于这样的假设：DO通过批量加载将数据上传到云，因此，同一个批量内的元组可以彼此链接形成批量链接。但是，在同一批量的元组之间进行链接是不够的，因为如果CP删除了所有批量，则无法检测到它。我们建议DO在本地保留上一个批量的副本，当新批量到达时，将其元组链接到当前批量和前一个批量。链接元组的数量由参数p定义。批量链接可以正式定义如下：

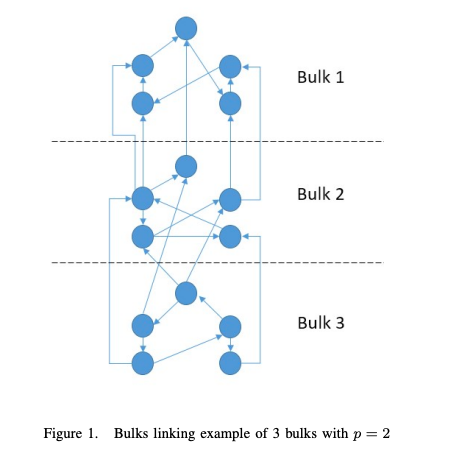


中有个元组，中有个元组，它们被链接到x。

图1显示了3个批量与5个密钥相关的批量连接图示：

•图形节点表示数据库元组。

•从节点A到节点B的边缘描绘了A链接到B（A知道B的存在）。



2）众包验证：在众包（CS）系统中，[7]用户通过合作实现一个有利于整个社区的目标。协作可以是显式的（例如维基百科和Linux项目），也可以是隐式的，例如，在ESP游戏[8]中，用户将图像标记为玩游戏的副作用。在我们的方法中，我们构建了CS系统，用户通过隐式地协作以实现多目标数据库完整性保证。与之前关注查询完整性的工作（客户端仅验证自己的查询）不同，我们关注的是数据库完整性（客户端执行随机验证查询以验证CP的行为是否令人满意）。这一战略背后的理由是：基于观察，数据库完整性比查询完整性更重要。客户不愿意使用CP，因为CP会向某些查询返回可证明错误的结果（即使这些查询是假的并且是由其他用户发出的）。我们在CS系统中依赖于这种观察，其中每个客户端贡献一点他的计算能力，以验证CP是否可信。它的工作原理如下：

（1）客户端向CP发送查询。

（2）CP将查询结果连同链接数据一起发送回客户端。

（3）客户端根据收到的链接数据构建验证查询，并将它们发送给CP。

（4）CP将验证查询的结果发送回客户端。

（5）客户端验证验证查询的结果是否与链接数据匹配。

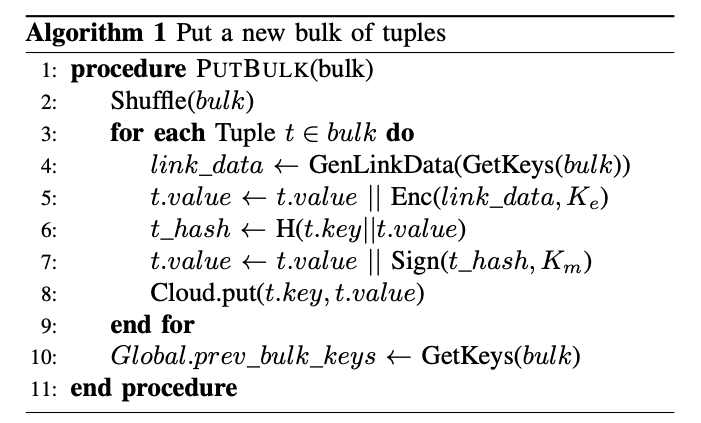
构建验证查询（步骤3），使得CP不能区分它们和常规客户端查询（步骤1）。 由于这一点，CP的客户端查询的恶意行为将不可避免地导致验证查询的恶意行为，从而被检测到。 例如，考虑对表II中的数据进行查询“获取id为1885的用户”。此客户端查询的验证查询将是“获取ID为2501的用户”和“获取ID为3456的用户”。如果至少其中一个查询的结果为空，则检测到攻击。

注意，客户端查询（步骤1）和验证查询（步骤3）之间没有依赖性，因此步骤3-5可以异步执行（即不会损害读取延迟）。

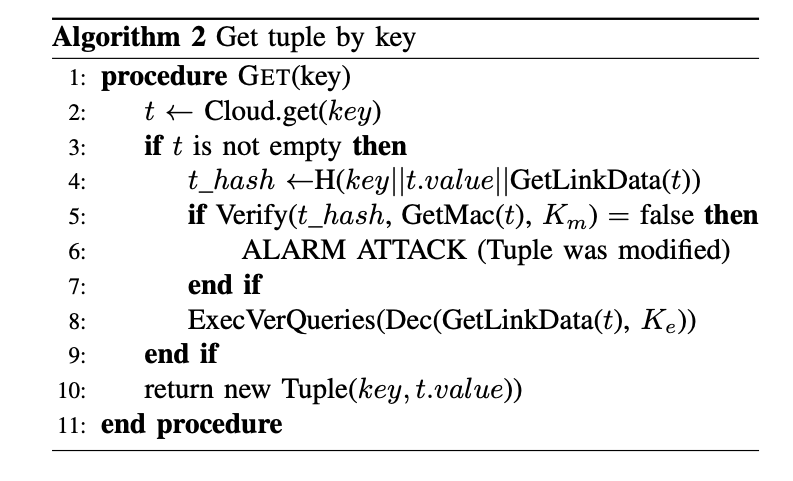
C.伪代码

下面给出了在我们的模型中提供正确性和完整性验证的算法的伪代码。

算法1将大量元组放入云数据库。它遍历批量中的所有元组（第3-9行）并计算每个元组的链接数据。 然后链接数据被加密并存储为元组值的一部分。为了能够将下一个批量链接到当前，我们将当前批量密钥存储在全局变量中（第10行）。



算法2按键返回元组。首先，在第4-6行验证元组的正确性。完整性验证基于验证查询，验证查询是基于解密的链接数据构建的（第8行）



四、安全分析

A.正确性

我们的正确性验证方案直接基于众所周知的原语（MAC方案和抗碰撞哈希函数），只要它们是安全的。

B.完整性

我们的完整性验证方法基于两种技术：元组链接和众包验证，如上所述。

假设删除的元组和查询范围的分布都是统一的，那么在Q查询之后，客户端检测到从数据库中删除了d个元组（或从结果中省略了d个元组）和个元组的概率是：

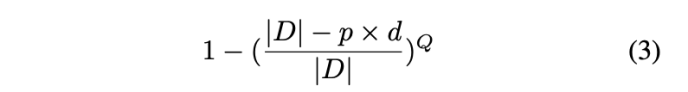
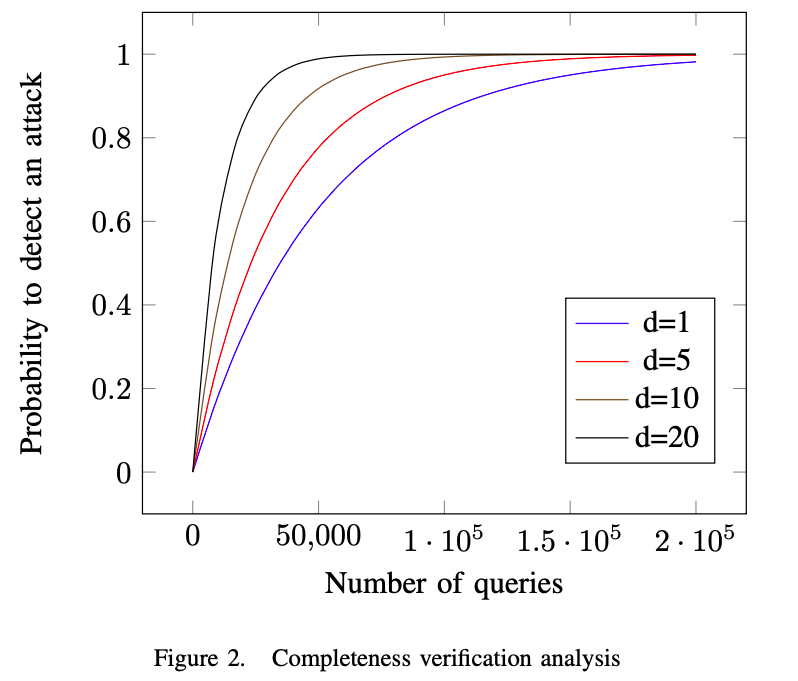


图2显示了检测攻击的概率，作为客户端使用执行的查询数量的函数， p = 4 和 d ∈ {1,5,10,20} 。可以看出，即使 P很小，在相对少量的查询（生产系统每秒接收数万个查询[25]）和删除元组之后，CP逃脱被捕的可能性非常低。



五、实施和实验结果

对于实验评估，我们已经实现了我们的解决方案的原型。 作为云键值存储，我们使用Redis [9]。为了评估我们的解决方案，我们使用Yahoo! 云服务基准（YCSB）框架[10]。为了使用YCSB对特定数据库进行基准测试，唯一需要做的就是实现数据库接口层。这将允许框架客户端执行诸如“读取元组”或“插入元组”之类的操作，而不必理解数据库的特定API。

我们以下列方式使用YCSB：

•YCSB框架已经有Redis客户端实现，名为RedisClient。

•我们基于第III节中的算法实现了我们的Redis客户端版本（IRe-disClient）。 我们的实施可在线获取[11]。

•我们配置了一个工作负载，其中对数据库执行预定义数量的可互换读取和插入操作。

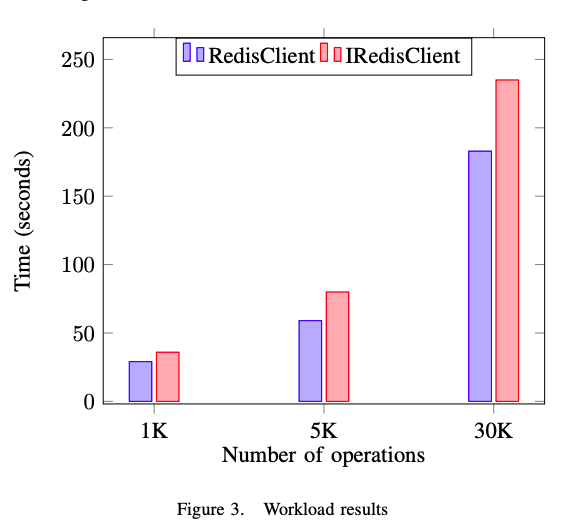
•我们在两个客户端上执行了这个工作负载，并比较了它们的执行时间。 结果如下。

A.建立

我们使用1KB大小的元组和随机值。 我们将批量大小定义为100，将参数 p （链接元组的数量）定义为4。工作负载在最初包含30000元组的数据库上执行。 我们为每个客户执行了三次工作量，分别为5000，10000和30000次操作。下面的结果代表了这三次执行的平均值。

B.业绩分析

IRedisClient插入操作的成本主要由两个加密和一个散列操作（MAC计算和链接数据的加密）决定。 IRedisClient读取操作的成本类似于插入成本（MAC计算和链接数据解密），另外还有 p 验证查询的成本。实验结果（图3）表明，这种开销平均增加了29％的执行时间。



六、相关工作

现有解决方案可以主要分为三种类型。 第一种类型基于Merkle哈希树（MHT），第二种类型基于数字签名（DS），第三种类型使用概率方法。

A.基于MHT的方法

MHT [12]是二叉树，其中每个叶子是数据块的散列，并且每个内部节点是其两个子节点的连接的散列。 Devanbu等人在[13]中介绍了一种使用MHT作为认证数据结构（ADS）的方法，以在DBaaS模型中提供数据完整性保证。一般的想法是为每个数据库表构建MHT，使得MHT的叶子是由搜索关键字排序的表记录的哈希。为了减少客户端和服务器端的I / O操作成本，可以使用更高扇出（MB-Trees）的树[14]，而不是使用二叉树。 在[15]和[16]中分别给出了为连接和聚合查询提供有效完整性保证的不同基于MHT的技术。基于MHT的方法不适用于我们的系统模型，因为它是为范围查询而设计的，不支持对任意键进行有效查询。 它还需要进行大量的服务器端更改，这在我们的系统模型中是不允许的。

B.基于DS的方法

在RDBMS中提供数据完整性的自然而直观的方法是使用数字签名方案（例如RSA [17]），方法如下：

•包含由DO私钥签名的连续记录值的散列将被添加到每个表中。

•客户端使用DO的公钥验证记录的完整性。

为了降低客户端和服务器之间的通信成本以及客户端的计算成本，可以使用签名ag-gregation技术[18]将多个记录签名组合成一个。 在我们的方法中，我们使用类似的技术来确保数据正确性（唯一的区别是我们使用MAC代替DS而不使用聚合）。 为了保证完整性，DO不签署单独的记录，而是签署连续的记录对[19]。为此，DO必须在本地拥有数据库副本，或者能够从云中获取连续记录。这两种选择在我们的模型中是不可能的。

C.概率方法

概率方法仅提供概率性完整性保证，但不需要DBMS修改，并且具有比基于MHT和DS方法更好的性能。 在此方法中，许多其他记录与原始记录一起上传到云中。传的附加记录越多，检测数据完整性攻击的可能性就越高。所有数据都在客户端加密，因此CP无法区分原始记录和附加记录。这些附加记录可能完全是伪造的，如[20]中提出的，或者是使用不同（次要的）密钥加密的原始记录，就像方案[21]中提出的那样。伪记录方法仅适用于范围查询，因此与我们的模型无关，而双加密方案可以应用于我们的模型，但要求整个数据库必须加密。在我们的方案中，每个元组中只有少量密钥（即链接数据）被加密。 与我们的方法相比，由于记录重复，双加密方案还需要在云端更多的额外存储。

七、结论

在本文中，我们提出了我们用于云密钥值存储中的数据完整性保证的新方法。 我们的方法依赖于众包范式——用户协作实现共同目标——数据库完整性保证。 我们的方法优于现有方法的主要优点是其高适用性——它可以应用于现有系统而无需修改服务器端。我们实施了协议的概念验证原型并进行了实验评估。结果表明我们的方案施加了合理的开销。为了将来的工作，我们计划在其他类型的云数据库（例如列存储）上扩展我们的方案。

参考文献

1. Leavitt, Neal. ”Will NoSQL databases live up to their promise?.” Computer 43.2 (2010): 12-14.
2. Cattell, R. (2011). Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 39(4), 12-27.
3. Hacigms, H., Iyer, B., Mehrotra, S. (2002). Providing database as a service. In Data Engineering, 2002. Proceed- ings. 18th International Conference on (pp. 29-38). IEEE.
4. Standard, S. H. National Institute of Standards and Technol- ogy (NIST), FIPS Publication 180-2, Aug 2002.
5. Krawczyk, H., Canetti, R., Bellare, M. (1997). HMAC: Keyed-hashing for message authentication.
6. Pub, N. F. (2001). 197: Advanced encryption standard (AES). Federal Information Processing Standards Publication, 197, 441-0311.
7. Doan, A., Ramakrishnan, R., Halevy, A. Y. (2011). Crowd- sourcing systems on the world-wide web. Communications of the ACM, 54(4), 86-96.
8. Von Ahn, L., Dabbish, L. (2004, April). Labeling images with a computer game. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 319-326). ACM.
9. Redis. http://redis.io/
10. Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R. (2010, June). Benchmarking cloud serving systems with YCSB. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 143-154). ACM.
11. IRedisClient. https://github.com/grishaw/ycsb-iredis-binding
12. Merkle, R. C. (1989, August). A certified digital signature. In Advances in CryptologyCRYPTO89 Proceedings (pp. 218- 238). Springer New York.
13. Devanbu, P., Gertz, M., Martel, C., Stubblebine, S. G. (2003). Authentic data publication over the internet. Journal of Computer Security, 11(3), 291-314.
14. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2006, June). Dynamic authenticated index structures for outsourced databases. In Proceedings of the 2006 ACM SIGMOD inter- national conference on Management of data (pp. 121-132). ACM.
15. Yang, Yin, et al. ”Authenticated join processing in outsourced databases.” Proceedings of the 2009 ACM SIGMOD Interna- tional Conference on Management of data. ACM, 2009.
16. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L. (2010). Authenticated index structures for aggregation queries. ACM Transactions on Information and System Security (TISSEC), 13(4), 32.
17. Rivest, R. L., Shamir, A., Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), 120-126.
18. Mykletun, E., Narasimha, M., Tsudik, G. (2006). Authentica- tion and integrity in outsourced databases. ACM Transactions on Storage (TOS), 2(2), 107-138.
19. Narasimha, M., Tsudik, G. (2005, October). DSAC: in- tegrity for outsourced databases with signature aggregation and chaining. In Proceedings of the 14th ACM international conference on Information and knowledge management (pp. 235-236). ACM.
20. Xie, M., Wang, H., Yin, J., Meng, X. (2007, September). Integrity auditing of outsourced data. In Proceedings of the 33rd international conference on Very large data bases (pp. 782-793). VLDB Endowment.
21. Wang, H., Yin, J., Perng, C. S., Yu, P. S. (2008, October). Dual encryption for query integrity assurance. In Proceedings of the 17th ACM conference on Information and knowledge management (pp. 863-872). ACM.
22. Hacigm, H., Iyer, B., Li, C., Mehrotra, S. (2002, June). Executing SQL over encrypted data in the database-service- provider model. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (pp. 216- 227). ACM.
23. Damiani, E., Vimercati, S. D. C. D., Jajodia, S., Paraboschi, S., Samarati, P. (2003, October). Balancing confidentiality and efficiency in untrusted relational DBMSs. In Proceedings of the 10th ACM conference on Computer and communica- tions security (pp. 93-102). ACM.
24. Ciriani, V., Vimercati, S. D. C. D., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P. (2010). Combining fragmentation and encryption to protect privacy in data storage. ACM Transactions on Information and System Security (TISSEC), 13(3), 22.
25. Atikoglu, B., Xu, Y., Frachtenberg, E., Jiang, S., Paleczny, M. (2012, June). Workload analysis of a large-scale key- value store. In ACM SIGMETRICS Performance Evaluation Review (Vol. 40, No. 1, pp. 53-64). ACM.