

电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE



论文题目 云数据完整性验证的关键技术研究

专业学位类别 工程硕士

学号 201422060632

作者姓名 邱佳惠

指导教师 禹勇 副教授

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名： 仰伟杰 日期： 2017 年 6 月 18 日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名： 仰伟杰 导师签名： 高勇

日期： 2017 年 6 月 18 日

分类号 _____ 密级 _____

UDC ^{注1} _____

学 位 论 文

云数据完整性验证的关键技术研究

(题名和副题名)

邱佳惠

(作者姓名)

指导教师

禹勇

副教授

电子科技大学

成 都

(姓名、职称、单位名称)

申请学位级别 硕士 专业学位类别 工程硕士

工程领域名称 计算机技术

提交论文日期 2017.03 论文答辩日期 2017.05

学位授予单位和日期 电子科技大学 2017 年 6 月

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Research on the Key Technology of Cloud Data Integrity Verification

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Master of Engineering**

Author: **Jiahui Qiu**

Supervisor: **A. Prof. Yong Yu**

School : **School of Computer Science & Engineering**

摘要

云存储作为云计算提供的重要服务，允许用户将数据从本地存储系统迁移到云存储系统中。这种新型存储服务向用户提供价格便宜、位置无关、应用透明、可伸缩的存储服务，减轻用户数据管理和维护的负担。云存储已然成为云计算中一个较快的利润增长点，越来越多的个人和企业采用云存储服务存储和管理其数据。但是这种数据外包的存储模式存在安全问题，云服务器容易受到恶意攻击且云服务提供商自身并不是完全可信的，用户的数据安全可能得不到保障，因此，为用户设计一种能验证其外包数据完整性的机制是十分有必要的。数据去重在真实的云存储环境中具有很强的实用性，用户外包到云上的数据存在很多重复，所以服务器希望对不同用户的相同文件只保存一个副本。另外在云存储中云用户的密钥可能会经常更新，通过下载所有文件再计算文件的块标签，然后再将文件和更新的标签上传是不实用的。如何同时实现完整性验证和数据去重，以及同时实现安全的数据拥有性验证和密钥更新操作是一项挑战。本文具体工作包括：

1. 分析现有的具备去重的云数据完整性检测方案的安全性, Zheng提出的方案，能实现数据的去重操作，同时也能实现云数据的完整性检测，缺陷是方案中使用了大量的随机数，计算过程比较复杂，另一缺陷是在审计过程中会泄露审计数据的信息，不是零知识隐私保护的。
2. 通过分析数据可拥有性证明(PDP)和数据所有权证明(POW)之间的联系，本文提出了一个支持去重操作的云数据完整性检测方案。通过使用广播的可聚合签名，在同一框架中同时实现安全的数据去重操作，和数据可拥有性证明检测。此外，该方案还支持零知识隐私保护，在审计的过程中不会将文件信息泄露给第三方验证者。
3. 本文提出了一个支持密钥更新的数据审计方案，该方案利用单向代理重新签名，当用户私钥丢失或是过期时，需要更换其公钥，用户只需生成新的私钥以及重签名私钥，然后将重签名私钥发给服务器。该方案支持公开可验证和隐私保护的性质，并且证明了该方案的安全性以及分析实验结果。

关键词：云存储，数据去重，完整性验证，密钥更新，隐私保护

ABSTRACT

Cloud storage, which enables cloud users to move their data from local storage systems to the cloud, is an important service offered by cloud computing. This new storage service provides users with affordable, location-independent, application transparent, scalable storage services. It can greatly reduce the burden of user data management and maintenance. Cloud storage has become a faster profit growth point in cloud computing, more and more individuals and enterprises to adopt cloud storage service. However, there is a security problem with the data storage model, the cloud server is vulnerable to malicious attacks and the cloud service providers are not fully trusted, the user's data security may not be guaranteed. Therefore, providing a mechanism that can detect the integrity of outsourced data for user is necessary. In the real cloud storage environment, data deduplication has a strong practicality, there are many duplicates of data that users outsource to the cloud, so the cloud vendor wants to save storage space by storing a single copy of each data—no matter how many clients outsourced it. In addition, cloud users will often update the key in the cloud storage, if users by downloading all the files to calculate the file block tags, and then upload the file and update the label is not practical. How to achieve both integrity verification and data deduplication, and how to implement secure data integrity verification and key update operations simultaneously are challenges. This thesis researches this regard, including:

1. Analyzed the security of existing cloud data integrity checking with deduplication scheme, Zheng's scheme is not only to achieve secure data deduplication, but also support for integrity verification. The defect is that use of a large number of random numbers in the program, the calculation process is more complex. Another flaw is that users' data information will be leaked during the audit process, can not achieve zero knowledge privacy protection.
2. Analyzed the relationship between Provable Data Possession (PDP) and Proof of Ownership (POW), This thesis proposed a cloud data integrity checking with deduplication scheme. Based on aggregatable signature based broadcast encryption scheme, to achieve data integrity verification and deduplication actually co-exist within the same framework. Another bonus of our construction is privacy preserving, meaning that the third party auditor learns nothing about the stored data during the auditing process.

ABSTRACT

3. This thesis proposed a privacy preserving cloud data auditing with key update, the scheme utilize a unidirectional proxy re-signature, when a user needs to change his/her public key due to private key is lost or expired, the user only to generate a new secret key as well as the re-signing key, then forwards the resigning key to the cloud server. This construction also has the desirable property of publicly verification and privacy-preserving, This thesis gave the security proof of this protocol and analyzed the performance of this protocol by experiment.

Keywords: cloud storage, deduplication, integrity verification, key update, privacy-preserving

目 录

第一章 绪论	1
1.1 研究背景	1
1.1.1 云存储的背景	1
1.2 云数据完整性检测方案研究现状	4
1.3 本文贡献	5
1.4 章节安排	6
第二章 密码学基础知识	7
2.1 困难问题	7
2.1.1 离散对数困难问题	7
2.1.2 双线性对与Diffie-Hellman问题	8
2.2 Hash函数	9
2.3 离散对数等式的知识证明	10
2.4 基于广播的可聚合签名(ASBB)的密码体制	10
2.5 同态验证	11
2.6 随机预言机和可证明安全	11
2.6.1 随机预言机模型	12
2.6.2 可证明安全	12
2.7 本章小结	14
第三章 云存储中的数据去重	15
3.1 数据去重的概述	15
3.1.1 数据去重的特点优势	15
3.1.2 数据去重方法的分类	16
3.1.3 云存储中的数据去重过程	17
3.1.4 云存储中数据去重的安全问题	18
3.2 对Zheng等的POSD方案的分析	20
3.2.1 方案回顾	20
3.2.2 方案分析	24
3.3 本章小结	24
第四章 支持密钥更新的隐私保护的云数据审计方案	25
4.1 问题提出	25

4.2 问题描述	25
4.2.1 系统模型	25
4.2.2 系统组成	26
4.2.3 安全模型	27
4.2.4 零知识证明	29
4.2.5 代理重签名	29
4.3 具备密钥更新的隐私保护的云数据审计方案描述	30
4.3.1 方案描述	30
4.4 安全性证明	32
4.4.1 正确性	32
4.4.2 完备性	33
4.4.3 零知识隐私	34
4.5 性能分析和实现	34
4.5.1 参数选择	35
4.5.2 复杂度分析	35
4.5.3 实验结果	36
4.6 本章小结	41
第五章 具备去重功能的云数据完整性检测协议	42
5.1 问题提出	42
5.2 问题描述	43
5.2.1 系统模型	43
5.2.2 系统组成	44
5.2.3 安全模型	45
5.3 具备去重功能的云数据完整性检测方案描述	47
5.3.1 方案描述	47
5.4 性能分析	50
5.4.1 数值分析	50
5.4.2 实验结果	51
5.5 本章小结	53
第六章 全文总结与展望	54
6.1 全文总结	54
6.2 后续展望	55
致 谢	56
参考文献	57
攻硕期间取得的研究成果	60

第一章 绪论

1.1 研究背景

本文主要研究云存储中数据完整性检测问题，具体包括支持去重的云数据完整性检测协议的研究和支持密钥更新操作的可证明数据拥有性研究。主要利用广播的同态签名技术来构造，同时实现数据拥有性证明(Proof of Data Possession，简称PDP)和数据所有权证明(Proof of Ownership，简称 POW)，利用代理重签名构造支持密钥更新操作的数据完整性检测方案。本章首先描述了云存储的背景，然后云数据完整性验证方案的研究现状，最后描述本文的主要贡献和具体的章节安排。

1.1.1 云存储的背景

云计算是并行计算、分布式计算、虚拟化等多种计算机技术和网络技术发展相融合的产物，实现了人们长期以来的“把计算作为一种设施”的愿望，云计算被设想为企业的下一代信息技术，是我国信息技术史上的一次重要创新^[1]。云计算具有按需提供服务、无处不在的网络接入、位置独立的资源池、规模高扩张性、价格低廉、可靠性高的特点，倍受各个行业的高度关注^[2]。作为下一代计算模式，云计算秉承资源租赁、应用托管和服务外包的核心理念，将企业和用户从IT基础设施管理和维护的繁琐工作中解放出来，使之能更关注自身核心业务发展。只要云用户愿意，就可以在云计算环境中得到相应的资源，这种新型的计算服务供给就如同水和电的供给一样。目前，许多国际大型 IT公司如 Amazon, Google, Microsoft, IBM等已经纷纷建立并对外提供各种云计算服务，已经成功吸引了许多用户。

众所周知，云计算的本质就是服务。云存储作为云计算提供的重要服务，是应用于分布式网络环境中的存储技术，它将同一网络环境中的不同存储设备，如计算机、服务器和 DAS存储设备整合到一起协同工作^[3]，对外可以提供有效的数据存储和访问服务。在云存储系统中，大量低成本的存储设备构成了一个大的资源池，它是云存储系统的核心，从而便于对外提供数据存储和数据管理服务，云服务器根据用户提出的存储需求来进行合理地动态的分配存储空间。然而，在它提供服务的同时，也带来了安全隐患。云用户最关注的问题就是外包数据的完整性问题，一旦数据被外包到了云服务器上，云用户也就失去了对其数据所处物理环境的直接控制。更糟糕的是，云服务提供商不是完全可信的^[4]，由于利益驱使，

云服务提供商可能删除不被访问或者访问较少的数据，将节约的空间租赁给其他用户从而获得更多的利益；或者，由于内部原因包括服务器故障、管理失误或外部原因如受到恶意攻击，导致用户部分数据丢失或被篡改，但云服务提供商为了维护自己的声誉，会刻意隐瞒或掩盖数据丢失和数据被篡改事件。Google 的大量邮件删除事件^①，T-Mobile Sidekick用户的个人数据丢失事件^[5]等都对用户造成了巨大的损失。所以在云存储环境中，云用户对自己外包到云服务器的数据能时刻进行监控和审计是十分有必要的。

数据审计协议是用来检验云服务器是否完全正确地保存用户外包数据的新型技术，以确保云数据的完整性，主要包括可证明数据拥有 PDP(Proof of Data Possession，简称PDP)和数据可恢复证明PoR (Proof of Retrievability，简称PoR)。2007年，Ateniese等首次提出了可证明数据拥有 PDP的概念和安全模型^[6, 7]，使得用户在不下载自己存储在服务器上的文件的情况下，能够验证存储在云服务器上的文件的完整性。Ateniese等采用同态可验证标签(Homomorphic Verifiable Tag)技术^[6]解决这一问题，即用户预先为文件中的每个数据块计算一个标签，然后将文件数据块与标签一起上传到云端存储起来。需要验证时，用户随机选择一些挑战块发给服务器，然后，服务器利用这些挑战块及其标签计算数据拥有证明，而且由于同态性，所有挑战数据块的标签可以聚合成一个标签，极大地节省了通信带宽。

在 Ateniese等提出可证明数据拥有 PDP概念的同一年，Juels等提出了可回取证明 PoR的概念^[8]，该方案使用纠错码和随机抽样技术来确保服务器正确无误存储了数据。相比于 PDP，POR除了能验证数据的完整性，还提了用户可以“恢复”外包给云的数据的功能。这一概念也在多个方面得到了增强和扩展，2008 年，Shacham和 Waters^[9, 10]利用纠删码提出了两个高效且紧致的 PoR方案，并在 Juels等的安全模型下^[8]进行了严格的安全性证明。

考虑到用户存储到云服务器的数据往往是海量的，有研究表明^[11]，云存储中75%的数据都是重复数据，如果任由重复数据肆意存储，会大大增加云服务提供商的存储空间负担，白白浪费云存储系统的存储空间。云服务器希望对不同用户请求存储的相同文件只存储一份，来节约存储空间，降低存储成本。而去重技术^[12]可以消除数据的重复副本，并且在存储中只保留一个副本，高度符合云存储的存储效率需求。将去重技术加入到云存储中，可以显著地减少网络带宽消耗以及节省服务器的存储空间。最开始，去重的实现都是基于计算数据的 MD5值或是散列值并进行对比的方法来实行，通过这种方式来实现简单直接的文件去

^① <http://www.techcrunch.com/2006/12/28/gmail-disaster-reports-of-massemail-deletions/index.html>

重。然而，通过这种简单直接的方法实现数据去重的缺陷是，恶意用户可以通过仅获得文件的哈希值，而声称拥有对文件的所有权，欺骗云服务器获得到文件的下载链接。因此，在为用户发送文件的链接之前，服务器要对用户是否真的拥有该文件进行验证。Halevi^[13]等人率先提出了所有权证明(Proof of Ownership，简称POW)的概念，该方案基于 Merkle哈希树设计，使得用户可以向服务器证明其真正拥有该文件。Pietro^[14]等人提出一种更安全有效的 POW方案来增强[13]中的构造，该方案将计算成本降低到仅有恒定数量的伪随机函数运算。然而，这些方案都只处理了云存储上所有权证明的(POW)的问题，然而并不适用于处理云数据审计的种种问题。所以设计一种具备去重功能的云数据完整性审计方案是十分有必要的。

迄今为止，绝大多数云数据审计结构都是基于公钥基础设施(PKI)。在 PKI系统中，需要证书颁发机构(CA)，为认证用户进行数字签名和颁发证书。公钥加密体制利用证书来解决假冒问题，证书通过指定有效期，将个人、公司或任何其他实体的身份与公钥相关联，密钥过期表示用户的证书不再有效。因此，证书的撤销和重新签发是维护PKI安全性的关键因素，这是PKI在互联网上进行安全通信的基础。当用户的私钥泄露时，用户证书很可能变得无效，或证书被误签，该证书就不能再代表用户身份。对于没有撤销证书能力的CA，那么在该证书到期之前CA没有直接的方法将该证书标记为不受信任证书。在云存储中，由于各种原因，当用户需要更改他的公钥时，就会面临如下难题，用户如何用更新后的公钥验证外包到云上的数据的完整性，因为存储在云中的旧的认证子是使用之前的私钥产生的，用更新之后的公钥进行数验证是无效的。这个问题的一个简单的解决方案是从云端下载所有的文件块和块标签，重新计算每个块的标签，并将数据块和块标签重新上传到云中。然而这种方法将占用大量带宽，受到网络带宽的限制，在实际应用中是不可行的。所以解决数据审计协议中有效的密钥更新和数据块标签更新的问题对于云存储中的实用性也是至关重要的。

云数据审计的另一个重要问题是隐私问题，包括身份隐私和数据隐私，数据隐私要求在审计过程第三方验证者得不到被挑战文件的丝毫信息。注意，仅是加密数据是不可行的，因为云上的数据通常是动态变化的。同态加密方案在理论上可能解决这个问题，但是却是不实际的。早前的审计方案都不是隐私保护的，为了回应一个挑战，云服务器可能会发送 $\mu_i = \sum_{(i, v_i)} v_i m_i$, m_i 是挑战块, v_i 是由审计者产生的挑战。因此，每个 μ_i 会泄露数据块的部分信息，通过重复挑战这些块，验证者将知晓到数据信息。所以在数据审计过程中同时实现隐私保护是很重要的。

1.2 云数据完整性检测方案研究现状

目前, Deswarthe等人^[15]最先提出利用 HMAC哈希函数来实现远程节点数据的完整性验证是完整性验证最基本的方法, 用户的文件被传到云服务器之前, 用户先计算这些文件的 MAC值并自己保存下来, 要验证时, 用户需要从云端将文件下载到自己本地并计算下载文件的 MAC值, 将本次下载文件计算的 MAC值和之前存的 MAC值相对比来确定云服务器确实存储了该文件。很明显这种方案要将整个文件下载到本地进行验证, 有很大的带宽开销, 在实际应用中是不允许的。目前, 检验云服务器是否正确地保存了用户数据的常用技术有, PDP和 PoR。2007年, Ateniese^[6]等人提出采用概率性的策略方式来完成数据完整性验证, 使得用户可以在不取回文件的情况下, 验证存储在服务器上的文件的完整性, 该方案提出利用同态可验证标签的技术, 开始用户预先为文件中的每个数据块计算一个标签, 然后将文件数据块与标签一起上传到云端存储起来。需要验证时, 用户随机选择一些挑战块发给服务器, 然后, 服务器利用这些挑战块及其标签计算数据拥有证明, 由于同态特性, 证据聚集成一个较小的值, 大大降低了协议的通信开销。

2009年, Wang等人^[16]实现了一种支持全动态操作的 PDP方案, 该方案采用 Merkle哈希树来确保数据块在正确的位置上, 而数据块的值的正确性则通过 BLS签名机制来确保。为了减轻云用户的计算开销负担, 该方案还引入独立的第三方(TPA)来代替用户验证外包到云服务器上的数据的完整性。但采用这种方式在验证过程中会向 TPA泄露文件数据, 未能实现公开验证的隐私保护。

Juels等提出了可回取证明PoR的概念^[8], 在其方案中, 用户首先利用纠错码对每个数据块进行编码, 同时, 选择一些作为哨兵的随机数据块, 然后将哨兵安插到加密的编码文件中去。验证者(用户或是第三方TPA)要求服务器返回某些哨兵位置上的数据块, 然后通过将返回的哨兵与自己选择的哨兵进行比对, 来检查数据的完整性。对比于 PDP方案, POR方案不仅能识别远程节点上的数据是否已损坏, 且还能恢复已损坏的数据。纠错码和随机抽样技术确保服务器正确存储了数据, 并且用户在需要取回数据的时候能够取回其数据。但由于每次挑战都得牺牲一些哨兵, 所以审计次数会受到限制。2008年, Shacham和 Waters^[9, 17]利用纠删码设计了两个高效且精致的 PoR方案, 并在Juels等的安全模型下^[8]进行了严格的安全性论证。

2012年, Zheng等人^[18]提出了支持数据去重的云数据完整性验证方案, 他将 PDP/PoR和POW方案进行了结合, 提出了一种称为支持去重的存储证明的概念(Proof of Storage with Deduplication, 简称: POSD)的概念, 该方案支持公开可验证, 并且有效的减少了重复文件数据标签值的数量, 从而为服务器节省了存储空间,

但是该方案没有实现零知识隐私保护，如果将它应用到云存储的环境中，在审计过程中会将用户的文件信息泄露给第三方审计者。

2016年，Yu等^[19]提出了一种支持密钥更新的零知识隐私保护的公开云数据审计方案，当云用户进行了密钥更新后，用户只需下载并更新认证标签，不需要下载整个文件以及重新生成所有的数据块标签，相比于下载整个文件来重新计算块标签再将文件和块标签上传的方法，其方法大大节省了网络带宽的开销。

1.3 本文贡献

目前对云数据完整性检测的关键技术研究主要集中在数据完整性验证方案的功能性以及安全性，然而本论文在保证数据审计方案的功能性和安全性的同时，考虑了数据去重计算在云存储中的应用，因为数据去重在云存储中有具有很强的实用性，云服务器对不同用户的相同文件只存储一份，从而节省服务器的存储空间以及减少网络带宽消耗。本文还考虑到用户的密钥会丢失或过期，在更新密钥之后要更新所有外包到云上文件的块标签，采用下载所有文件重新计算标签然后再上传的方法是很低效的。综合上述情况，本文提出了具有特色和实用价值的云数据完整性检测的方案，本文的主要贡献如下：

1. 分析了 Zheng 等提出的支持去重的安全高效的存储证明方案，发现该方案能安全且有效的实现去重和数据完整性验证的功能，但是在审计过程中会泄露审计数据的信息，不是零知识隐私保护的。
2. 基于代理重签名设计了一种支持密钥更新的数据审计方案，当用户密钥改变或是过期时，只需下载文件的标签而不是下载整个文件，该方案利用离散对数的简单的零知识证明，在审计和密钥更新的过程中实现隐私保护，并且证明了该方案的安全性。
3. 基于广播的可聚合签名提出了一种具备去重的数据完整性检测方案，同时实现数完整性检测和数据去重，该方案还支持零知识隐私保护，在审计的过程中不会将文件信息泄露给第三方验证者。

本论文的研究使得云数据完整性验证协议不仅仅适用于云存储环境下数据审计，同时也支持云服务器端的文件去重，以及支持用户密钥更新操作，同时也为云存储健康和快速地实施提供安全保障，并促进了应用密码学的进一步发展。

1.4 章节安排

本文的结构安排如下：

第一章阐述云存储服务的优点和应用以及外包数据审计协议目前面临的一些难题，重点介绍了云存储中数据去重操作、用户密钥更新操作、数据零知识隐私保护的问题。然后简单的介绍了云数据完整性验证协议的研究现状，例如，仅支持数据完整性验证的，同时支持数据去重和数据完整性验证的，以及支持密钥更新的数据审计方案。最后，简要介绍了本文的主要贡献。

第二章主要讲述了一些密码方案中用到的密码学基础知识。首先介绍了本文中用到的困难问题，如，离散对数困难问题、双线性对的Diffie-Hellman问题，接着介绍了Hash函数、基于广播的可聚合签名(ASBB)的密码体制、同态验证技术。然后具体描述了离散对数等式的知识证明问题，并详细介绍了随机预言机引入的原因及其概念。最后，着重分析了可证明安全性的定义，并对其安全证明框架做了详细介绍。

第三章主要开始介绍了数据去重概述，如去重的特点、去重的分类方法、云存储中的数据去重过程以及云存储中数据去重的安全问题，最后回顾了Zheng等的POSD方案，并对该方案做了分析。

第四章提出了一个支持密钥更新的数据完整性检测方案。发现当用户密钥丢失或过期时，用户在更新密钥后，需要通过下载整个文件重新计算文件的块标签，再将文件和块标签上传到云服务器，于是利用代理重签名解决了密钥更新问题，通过此技术，用户只需要将重签名的密钥上传给服务器并且更新自己的公钥和重签名的密钥，其他的工作都交由服务器处理，并且该方案还满足零知识隐私性质。最后，本章证明了方案的安全性。

第五章提出了一个具备去重功能的云数据完整性检测方案。该方案使用基于广播的同态签名技术，在同一框架中同时实现数据去重和数据拥有性证明检测，并且该方案还具备公开可验证和零知识隐私保护性质。最后通过性能分析和实际实验数据结果表明了方案的高效性。

第六章总结全文，并对后面的工作进行展望。

第二章 密码学基础知识

本章将详细地对本文中用到的一些密码学基础知识的概念做出介绍。具体包括：密码学困难问题、Hash函数、离散对数等式的知识证明、基于广播的可聚合签名(ASBB)的密码学构造、随机预言机模型、和可证明安全。

2.1 困难问题

困难问题是密码学里的一个重要的基础理论知识，因为所有的密码学算法(加密算法和签名算法)都是在困难问题的基础上构建而成的，即，如果没有困难问题，那就构造不出一个安全的密码算法。如果想要构造一个安全的密码算法，那么首先必须要找到一个相关的困难问题，下面，本章就简单介绍一下本论文用到的一些困难问题。

2.1.1 离散对数困难问题

定义 2.1.1 离散对数问题：已知大素数 p ，其中 $p - 1$ 包含另外一个大素数 q ，那么，可以构造一个阶为 $p - 1$ 的乘法循环群 Z_p^* 。令 g 做为乘法循环群 Z_p^* 的一个生成元，如果已知 x ，那么很容易求解 $y = g^x \bmod p$ 的值，反过来，如果已知 g, p, y 的值，要求解满足等式 $y = g^x \bmod p$ 的 x 是非常困难的。

有限域上的离散对数问题仍是一个困难问题，至今还未能找到一个有效的方法能够解决该问题。由于，在离散对数的运算中，无论一个离散对数算法在生成的时候选择一个生成元为 γ 来生成，还是一个离散对数算法在生成的时候选择另外一个生成元 v 来生成，最后，这两个以不同的生成元为参数构造的离散对数算法，它们之间都是可以相互转变的，因此，我们可以知道离散对数问题的困难度与生成元的取值是无关，而是与群中元素的数量有关系。比如，在乘法群 Z_p^* 中，我们假设 p 是一个长度为 ρ 位的数，那么求解一个离散对数的复杂度就是 ρ 的多项式时间。再一次验证了，离散对数问题的困难程度是完全取决于阶 p 的长度 ρ 。如果将参数 ρ 的值设置的很小时，那么，我们就可以找到一个有效的方法解决乘法群上的离散对数算法，因为在参数 p 长度很小的情况下，该算法是容易求解的，那么它也就构不成一个困难问题。反过来，如果我们将选择的参数 ρ 的值设置成较大的数时，那也就没有办法找到一个有效的方法可以解决乘法群上的离散对数问题。在实际实践中以及通过分析现代的计算机的运行速度，我们可以得出下

述结论，当 ρ 的数值取大于等于 1024 位时，就找不到一个有效的方法可以解决群 Z_p^* 上的离散对数问题。

2.1.2 双线性对与Diffie-Hellman问题

定义 2.1.2 双线性对. 假设 λ 是安全参数。群 G 和 G_T 是阶为 P 的乘法循环群， g 是群 G 的其中一个生成元。假设群 G 和群 G_T 中的离散对数问题都是困难的，整数 a, b 是 Z_p 中的元素，整数 u, v 是 G 中的元素。 $\hat{e} : G \times G \rightarrow G_T$ 是一个双线性对，符合以下特性。

- (1) 对所有的 $u, v \in G$ 和所有的 $a, b \in Z_p$, $\hat{e}(u^a, v^b) = \hat{e}(u, v)^{ab}$ 。对所有的 $u_1, u_2 \in G$, $\hat{e}(u_1 \cdot u_2, v) = \hat{e}(u_1, v)\hat{e}(u_2, v)$ 。
- (2) 这个 \hat{e} 应该是非退化，满足 $e(g, g) \neq 1$ 。
- (3) 对于任意的 $u, v \in G$, 存在一个有效的算法可以计算 $e(u, v)$ 。

在现代密码学中，还有一些公钥密码方案是基于 Diffie-Hellman 问题的困难性构建的。Diffie-Hellman 问题相对于离散对数困难问题来讲是更容易求解的。最常用的 Diffie-Hellman 问题包括四种：计算性 Diffie-Hellman 问题、判定性 Diffie-Hellman 问题、双线性 Diffie-Hellman 问题和判定性双线性 Diffie-Hellman 问题。在我们的方案中用到的一些困难问题如下：

定义 2.1.3 双线性对问题. 给定 G , 它的生成元为 g , 以及 $\hat{e}(X, g) \in G_T$ 的值, 计算出 $X \in G$ 是困难的。

定义 2.1.4 计算性Diffie-Hellman问题(CDHP). 给定 (g, g^a, g^b) 和两个未知的整数 $a, b \in Z_p^*$, 计算 g^{ab} 是困难的。

定义 2.1.5 判定性Diffie-Hellman问题(CDHP). 给定 (P, aP, bP, cP) 和两个任意的整数 $a, b \in Z_p^*$, 判断式子 $c = ab \bmod p$ 是否成立. 如果 (P, aP, bP, cP) 可使式子 $c = ab \bmod p$ 成立, 则 (P, aP, bP, cP) 称为一个Diffie-Hellman 元祖。

定义 2.1.6 双线性Diffie-Hellman问题(BDHP). 给定 (g, g^a, g^b) 和两个未知的整数 $a, b \in Z_p^*$, 对一个任意的 $h \neq g \in G$, 计算 $e(h, g)^{ab}$ 是困难的。

定义 2.1.7 判定性双线性Diffie-Hellman问题(DBDHP). 该问题结合了上面分别介绍的判定性和双线性，具体来讲，该问题指的是给定 (g, g^a, g^b, g^c) 和三个未知的整数 $a, b, c \in Z_q^*$, 选择一个数 $Z \in G_T$, 很难判断 Z 与 $e(g, g)^{abc}$ 是否相等。

在特定的条件下，CDHP、双线性对问题、BDHP 和 DBDHP 都是困难的，找不到一个有效的方法去解决它们，所以一些密码体制经常是基于这四种困难问题构造的，本论文第五章中的方案构造也用到了上述四种困难问题。

2.2 Hash函数

Hash函数是一种具有单向性的函数^[20]，在现代的密码学的地位是十分重要的，几乎所有的密码方案中都使用到了Hash函数，不止在密码学中，在其他行业其应用很广泛。Hash函数的输入可以是任意长度的消息(数据) M ，经过Hash运算之后，变换成一个长度大小是固定值的消息(数据) $H(M)$ 。很明显，Hash函数可以将其消息(数据)压缩成更短固定长度的数据，通过这种技术，在云数据的完整性验证中，就可以通过Hash文件，将文件压缩，从而减少通信开销。通过使用不同的Hash函数，其输出的散列值也不同。在传统的云数据完整性验证的方案中，Hash函数通常用于为需要验证的消息产生一个短的数据摘要，开始时，用户先计算消息(数据)的Hash值，然后将消息(数据)的对应的Hash值发送到服务器上，在完整性验证过程中，验证者计算服务器发送给他的信息(数据)的Hash值，然后将用户发给他的一个之前保存在用户本地的信息(数据)的Hash值，通过对比来验证信息(数据)的完整性，如果两个值是相等的，说明信息(数据)是完整的，否则该信息(数据)的完整性就没被保证。Hash函数在一些场景中具有重要的作用，如数据完整性验证、数字签名、密钥交换、生成伪随机数和身份认证等场景中。

其实，Hash函数是一对多的映射关系，所有在实际使用Hash函数的过程中可能出现碰撞的情况，碰撞指的就是不同的原始消息(数据) m_1 和 m_2 利用Hash映射后，发现 $h(m_1) = h(m_2)$ ，所以一个安全可靠的Hash函数应满足如下定义：

定义 2.2.1 Hash函数. $H : \{0,1\}^* \rightarrow \{0,1\}^n$ 是满足下列性质的函数。

- (1) 函数的输入可以是任意长度的。
- (2) 函数的输出是固定长度的，但至少取 128 比特长，以便于抵抗生日攻击。
- (3) 对任意给定的 s ，能够很容易地计算其输出值 $h(x)$ ，这称为有效性。
- (4) 对任意给定的Hash值 s ，想要找到一个使等式 $h(x) = s$ 成立的参数 x ，在计算上是很难的或是行不通的，这称为Hash函数的单向性。
- (5) 对任意给定的输入串 s ，想要找到另一个输入串 x (s 与 x 不相等)，使得 $h(s) = h(x)$ 在计算上是行不通的，这称为抗弱碰撞性。
- (6) 任意找到两个输入串 s 和 x ，其中 s 与 x 不相等，使得 $h(s) = h(x)$ 在计算上是不可行的，这称为抗强碰撞性。

Hash函数可以用于数据完整性的检验，Hash在云存储中外包数据完整性的验证中所起的重要作用。目前，被广泛使用的Hash函数主要有 SHA0, SHA1, SHA224, MD4^[21] 和 MD5。

2.3 离散对数等式的知识证明

在密码学中，知识证明是一个交互式的证明，在证明过程中，证明者(prover)要让验证者(verifier)信息他真的知道一些事。在我们的方案中使用这个知识证明是为了证明，两个公开数据有相同的离散对数，在不泄露关于两个公开数数值的相关信息下前提下。我们假设 G 和 G_T 是阶为 p 的乘法循环群。 g 为群 G 的生成元， $A \in G_T$ 。 $H : \{0,1\}^* \rightarrow G$ 是一个Hash函数。给定两个公开的元素 $h \in G$ 和 $R \in G_T$ ，证明者(prover)需要证明 $\log_g h = \log_A R = x$ ，但是他不能泄露任何一点 x 的信息。我们使用在文献[22]展示的这个知识证明，具体阐释见下图2-1：

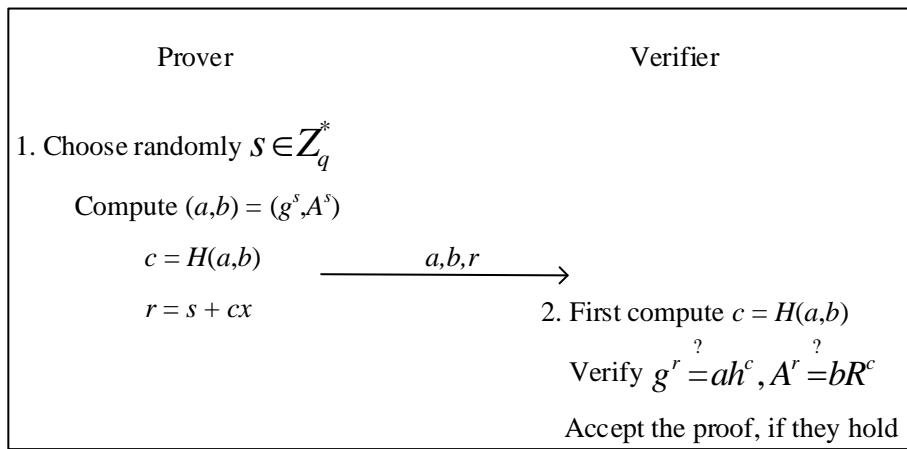


图 2-1 离散对数等式的知识证明

2.4 基于广播的可聚合签名(ASBB)的密码体制

文献[23]中为我们介绍了一种新的密码学加密方法，叫做基于广播的可聚合签名的密码体制，它具有可聚合的性质，从该方法中得到启发，本文第五章提到的方案运用了这一性质，该密码学体制的详细描述如下：

1. 公共参数：输入一个安全参数 λ ，生成一个双线性对，即 $\text{PairGen}(1^\lambda) \rightarrow (p, G, G_T, \hat{e})$ ，系统参数为 g, H, p, G, G_T, e 其中 g 为群 G 的一个生成元， $H : \{0,1\}^* \rightarrow G$ 是一个映射到群 G 的 Hash 函数。
2. 公钥私钥对：选择随机值 $r \in \mathbb{Z}_p^*$ ， $X \in G \setminus 1$ 。计算 $R = g^{-r}$ ， $A = \hat{e}(X, g)$ 。公钥为 $pk = (R, A)$ ，以及私钥为 $sk = (r, X)$ 。
3. 签名：为了给 $m \in \mathbb{Z}_p^*$ 生成一个签名，最开始选择一个随机数 $s \in \{0,1\}^*$ ，并且使用公钥俩计算 $\sigma = X^m H(s)^r$ ， m 的签名就为 (s, σ) 。

4. 验证：给定一个消息签名对 (m, s, σ) ，验证的等式为 $\hat{e}(\sigma, g)\hat{e}(H(s), R) = A^m$ 。如果这个等式是成立的，就输出1，表示这个签名是有效的，或者输出0，表示该签名是无效的。
5. 加密：对任何一个明文 $\omega \in G_T$ ，选择一个任意随机数 $t \in Z_p^*$ ，并且计算 $c_1 = g^t$, $c_2 = R^t$, $c_3 = \omega \cdot A^t$ ，得到的密文就为 (c_1, c_2, c_3) 。
6. 解密：给定密文 (c_1, c_2, c_3) ，以及任何一个有效的消息签名对 (m, s, σ) ，就可以计算出明文： $\omega = c_3 / (\hat{e}(\sigma, c_1)\hat{e}(H(s), c_2))^{m^{-1}}$ 。

下面我们来验证该方案的正确性：

$$\begin{aligned}
\omega &= c_3 / ((\hat{e}(\sigma, c_1)\hat{e}(H(s), c_2)))^{m^{-1}} \\
&= \omega \cdot A^t / ((\hat{e}(X^m H(s)^r, g^t)\hat{e}(H(s), R^t)))^{m^{-1}} \\
&= \omega \cdot \hat{e}(X, g)^t / ((\hat{e}(X^m, g)^t \hat{e}(H(s)^r, g^t) \hat{e}(H(s), R^t)))^{m^{-1}} \\
&= \omega \cdot \hat{e}(X, g)^t / \hat{e}(X, weg)^t \hat{e}(H(s)^r, g^t) \hat{e}(H(s), g^{-rt}) \\
&= \omega / \hat{e}(H(s), g)^{rt} \hat{e}(H(s), g)^{-rt} \\
&= \omega
\end{aligned}$$

2.5 同态验证

在云数据完整性检测方案中使用同态验证技术可以大大地减少网络带宽的消耗。对于给定消息(数据) M 的验证，一般是消息的验证标签 Tag 最先被计算出，然后使用该验证标签来验证消息(数据)的完整性。但是消息的验证标签的计算复杂度随着消息(数据)长度的增长呈线性增长，即消息(数据)越大，计算的时间就越长。如果消息被分块处理，再对消息的每个数据块进行标签验证计算，然后使用同态验证的技术，根据同态认证标签的可聚合的特性，从而对整个消息进行完整性验证，运用这种方法可以极大地减少计算验证标签的复杂度。

假设有一个文件 F ，将文件进行分块处理，分成 m_1, \dots, m_n 共 n 个数据块，然后计算出各数据块的验证标签 Tag_1, \dots, Tag_n ，那么文件该 F 的验证标签可用 Tag_1, \dots, Tag_n 计算得到。比如，在基于 RSA 的同态验证标签计算中，找一个与 N 互素并且具有最大阶数的整数 k ，那么数据块 m_1, \dots, m_n 的同态标签为 $Tag_1 = k^{m_1} mod N, Tag_2 = k^{m_2} mod N, \dots, Tag_n = k^{m_n} mod N$ ，由此可以得到的是，文件 F 的同态标签就为: $Tag = k^{m_1 + \dots + m_n} mod N = Tag_1 \dots Tag_n mod N$

2.6 随机预言机和可证明安全

在现代密码学中，验证一个密码体制是否是安全的最有效方法是可证明安全技术，该技术的出现打破了密码学中一直面临的“先使用，后证明”的尴尬局面。

一个安全的密码体制都要经过详细的安全性证明分析论证得来，使用的安全性证明方法就是可证明安全技术。本文在证明提出方案的安全性时，使用的也是可证明安全技术。即把一个方案的安全性归约到一个公认的难解的困难问题上，因为困难问题是难解的，从而推断出提出的方案是安全的。下面给出了可证明安全的详细分析过程。如果要设计一个安全的密码方案，那么掌握可证明安全技术是很必要的。

2.6.1 随机预言机模型

密码方案的安全性可以通过使用可证明安全性方法来证明。而可证明安全性的最终目标是：敌手 \mathcal{A} 能够攻破一个密码协议，然后，算法 \mathcal{C} 就可以利用敌手 \mathcal{A} 解决一个困难问题，前提是算法 \mathcal{C} 需要对敌手 \mathcal{A} 的询问做出回答。算法 \mathcal{C} 的回答需要符合下面几个标准：

- (1) 算法 \mathcal{C} 的给敌手的回答看起是合理的。
- (2) 预言机的回答始终都是一致的。
- (3) 算法 \mathcal{C} 的回答应该与敌手 \mathcal{A} 期望从真正的签名预言机那里获得的回答具有一样的概率分布。
- (4) 算法 \mathcal{C} 是在不知道私钥的前提下做出问答的。

对于一个安全的密码体制来讲，其中第(4)个标准是不符合实际的。为了回避这个问题，使用随机预言机模型。

定义 2.6.1 随机预言. 是一个理想的Hash函数，它满足以下几个性质。

- (1) 当敌手 \mathcal{A} 提出一个新的询问时，随机预言将一个随机值作为答案给敌手 \mathcal{A} 。
- (2) 当敌手 \mathcal{A} 询问之前已经提过的询问时，随机预言将和上次询问结果相同的值返还给敌手 \mathcal{A} 。
- (3) 敌手 \mathcal{A} 使用的是随机预言机返回的Hash值，而不是密码算法中定义的Hash值。
- (4) 对于敌手 \mathcal{A} 的签名询问，算法 \mathcal{C} 总是通过随机预言给出一个自己想要给出的Hash值。

如果一个密码体制在随机预言机模型下证明是安全的，只能说它在真实的世界里可能是安全的，而不能保证它在真实的世界里一定是安全的。

2.6.2 可证明安全

定义 2.6.2 可证明安全. 实际上是一种归纳方法，该方法能够证明密码协议在特定的安全模型下可以达到具体的安全目标。具体包括：

- (1) 确定密码协议的安全目标。
- (2) 根据敌手 \mathcal{A} 的攻击能力构造一个形式化的安全模型。

- (3) 若敌手 \mathcal{A} 能够在多项式时间内攻破一个密码协议，则存在一个算法 \mathcal{C} 可以借助敌手 \mathcal{A} 解决一个困难问题。
- (4) 证明通过困难问题被公认为是困难的，是无解的，所以推出敌手 \mathcal{A} 是不可能攻破密码协议的。

显然，可证明安全的前提条件是找到合适的安全目标和构建合适的安全模型。以数字签名方案为例。

- 数字签名方案的四种安全目标：

1. 完全攻破，即敌手 \mathcal{A} 可以生成一个与密钥拥有者相同的签名，此安全目标是最难攻破的。
2. 通用性攻破，即敌手 \mathcal{A} 可以伪造任意一个消息的签名。
3. 选择性伪造，即敌手 \mathcal{A} 可以伪造它自己选择的消息的签名。
4. 存在性伪造，即敌手 \mathcal{A} 可以伪造一个消息的签名，此安全目标是最容易攻破的。

- 数字签名方案的三种攻击模型：

1. 唯密钥攻击，敌手 \mathcal{A} 仅仅被告知签名方案的公钥，此模型下敌手 \mathcal{A} 的攻击能力较弱。
2. 已知消息攻击，敌手 \mathcal{A} 除了被告知一个公钥以外，还知道一些消息/签名对。
3. 适应性选择消息攻击，敌手 \mathcal{A} 能够通过询问签名预言机获得消息的签名，此模型下敌手 \mathcal{A} 的攻击能力最强。

定义了数字签名方案的安全目标和攻击模型之后，就能够定义数字签名方案的安全性。在适应性选择消息攻击下，如果数字签名方案可以抵抗存在性伪造，则该签名方案是安全的，记为EUF-CMA。

可证明安全的证明过程可以借助一个由敌手 \mathcal{A} 和算法 \mathcal{B} 进行交互的游戏来证明。在游戏中，通过分析敌手成功赢得游戏的概率来判断密码体制的安全性。如果敌手 \mathcal{A} 赢得安全游戏的概率 ϵ 是一个不可忽略的值，则该密码体制是不安全的。否则，如果敌手 \mathcal{A} 赢得安全游戏的概率 ϵ 是一个可以忽略值，则该密码体制是安全的。所以密码体制的安全性完全依照于安全游戏中敌手 \mathcal{A} 成功的概率。敌手 \mathcal{A} 和算法 \mathcal{B} 进行交互的安全性游戏详细的分析见图2-1。在该游戏中， \mathcal{A} 表示敌手， \mathcal{B} 表示和敌手交互的算法， X 表示一个公认的困难问题， ϵ/p 表示敌手在游戏中获胜的概率， Π 表示需要分析其安全性的密码体制。

- 假设在多项式时间内敌手 \mathcal{A} 攻击密码方案 Π 。

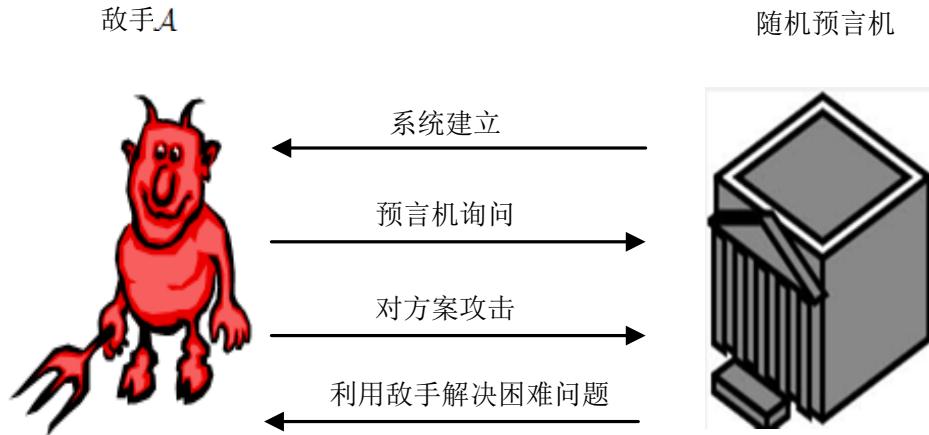


图 2-2 可证明安全有效

- 构造一个多项式时间的算法 \mathcal{B} , 该算法试图利用敌手 \mathcal{A} 来解决一个公认的困难问题 X 。敌手 \mathcal{A} 和算法 \mathcal{B} 模拟密码方案 Π :
 - 初始化阶段. 算法 \mathcal{B} 执行密钥生成算法生成一对公私钥对 (pk, sk) , 算法 \mathcal{B} 发送公钥 pk 给敌手 \mathcal{A} , 自己保存私钥 sk 。
 - 询问阶段. 敌手 \mathcal{A} 询问预言机, 与密码方案 Π 交互。
 - 攻击阶段. 敌手 \mathcal{A} 攻击密码方案 Π 。

假设在安全性游戏中, 敌手 \mathcal{A} 攻破了密码体制 Π , 那么根据可证明安全技术的定义可知, 算法 \mathcal{B} 能够解决困难问题 X 且概率至少为 $1/p$ 。如果概率 ϵ 是不可忽略的, 那么算法 \mathcal{B} 就是以不可忽略的概率 ϵ/p 解决了困难问题 X 。然而, 困难问题是没有办法解决的, 所以能够以不可忽略的概率攻破方密码案 Π 的敌手 \mathcal{A} 是不存在的, 进而证明密码方案 Π 是安全的。

2.7 本章小结

本文的研究方案在方案构造阶段使用Hash函数生成数据块的标签, 在标签生成的过程中用到了同态验证技术, 重点介绍了第五章方案中使用到的基于广播的可聚合签名(ASBB)的加密构造, 以及离散对数等式的知识证明。在安全性证明阶段使用了Diffie-Hellman问题、随机预言机模型和可证明安全性, 最后通过使用可证明安全技术分析了方案的安全性, 着重分析了可证明安全性的定义, 并对其安全证明框架做了详细介绍。

第三章 云存储中的数据去重

本章先对数据去重做一个概述，具体包括去重的概念、特点优势、云存储中数据去重的过程和去重分类，进而介绍云存储中数据去重的安全问题，回顾Zheng等^[18]提出的POSD方案，并对方案进行了分析。

3.1 数据去重的概述

随着互联网技术快速的普及和更新发展，数据信息的传递和分享在很大的程度上提高了互联网的速度和广度。据保守统计，现在信息产生的速度大概是，6亿的Email，18万小时的音乐下载，6000万次照片查看，14万次的应用下载以及390万次的视频观看可以在三分钟不到的时间内产生。现在社交网络以及移动设备工具的高度普及使得用户随时随地都能进行数据的转发和分享操作，如今很多移动设备的应用，还会随时收集关于用户的行为数据。移动数据量近年来急剧增加，智能电话作为移动设备的典型代表，其不仅提供传统蜂窝电话的基本电话功能，而且还结合若干其它数字设备的功能，特别是个人数字助理，全球定位系统，录音机和数码相机功能，它们通过会产生很多数据文件，包括电子邮件，电子表格，银行对账单和多媒体文件(视频、音频，图片)，存储这些数据的需要占用很大的存储空间，如何存储并处理这些日益庞大的数据，对于云存储服务提供商来说是一个不容回避的问题。本文提出一种在云存储中实现数据去重的方法，通过该数据去重方法在很大程度上可以减少云存储服务提供商在存储空间方面的开销，同时还可以减少网络带宽的消耗。

3.1.1 数据去重的特点优势

数据去重是一种让数据无损的冗余数据缩减技术，用来对重复数据进行处理。数据去重技术^[12]通过避免对相同的数据文件在存储系统中存储多次来实现，即多个相同的数据文件在存储系统中只存在一个副本，这样便可以极大的节省存储空间。例如，用户Alice已经将文件M存到云服务器上了，用户Bob请求存储相同的文件M，检测到文件M已经存到服务器上了，服务器不会再一次存储文件M，而是仅仅跟文件M相关联的元数据来表明Bob和Alice都存储了文件M。使用这种方法，相同的文件不是再次存储，这样的话u个用户存储一份文件的存储开销就从 $\mathcal{O}(uM)$ 降低到 $\mathcal{O}(u + M)$ 。重复文件去重的功能通过避免对相同的文件存储多次

来实现，节省的存储开销是非常明显的，尤其是对那些很占用存储空间的流行多媒体文件的去重，例如音乐和视频文件。

数据去重往往利用数据指纹技术，这里的指纹技术指的是使用某一种技术为数据生成的唯一标识，从而达到多个文件中重复的数据块可以被消去的目的，这与我们常用的数据压缩并不同，数据压缩通过某种特殊编码方法来处理文件，从而达到消去文件内冗余数据目的，是使用少的数据量表示初始数据的方式。去重中的数据块级别去重方法可以消去多个文件中重复的数据块，从而来实现跨文件的数据去重，而数据压缩不能跨文件压缩相同的数据，数据去重的优势是显而易见的。再者，在云存储系统中，数据量很多。数据的存在都是密集，相似度高的数据出现的概率性很大，即不同文件出现相同的数据块的概率性是很大的，因而数据去重在云存储中的应用价值更高，可以减少云服务器存储设备开销、网络带宽消耗已经提高数据处理的效率。

3.1.2 数据去重方法的分类

根据角度的不同，数据去重的方法可以分为不同的种类^[24]。

(1)根据数据去重的粒度大小来分，数据去重主要可分为数据块级和文件级数据去重。数据去重的效果随去重的粒度的减小而增大，但是粒度越小会导致计算的开销也就越大^[25]。数据级去重采用的方法是，文件先被分块处理，然后根据分好的数据块进行重复性检测，发现重复数据块就删除冗余数据块，对相同的块只保留一份，基于数据块级的去重获得的数据去重效果是更高的。文件级去重就是在文件的基础上进行直接去重，对比数据块级去重，文件级去重的计算速度是非常快的，但是去重效果却没数据块级去重佳，而且没有办法做到跨文件去重，即不能对不同文件中的相同数据块部分进行去重。本文提出的具备去重功能的云数据完整性验证方案是基于文件级数据去重的。

(2)根据从去重发生位置的来分，数据去重可以分为数据源端数据去重和数据宿端的数据去重^[26]。比如，基于文件级数据去重的两种类型，服务器端文件级数据去重和客户端去重，服务器端的去重非常简单，服务器收到文件之后，检查是否已经存储了该文件，如果检测到已经存储了，就丢弃该文件；如果没有存储的话，就为该文件创建存储空间进行存储。用户端去重采用的方法是，用户在上传文件之前，计算文件的哈希值并将文件哈希值发给服务器，一旦收到文件的哈希值，服务器就可以检测是否已经存储了该文件，如果已经存储了，用户不会被要求上传该文件，服务器会将该文件和用户进行关联；如果没有存储该文件，用户就会被要求上传文件。图一给出了详细的展示，其中用户1首先上传图1中的文件

在 $F1$ 和 $F2$, 图3-1(a)所示, 然后云用户2发送文件 $F1$ 和 $F3$ 的哈希值 $h(F1)$ 和 $h(F3)$ 到服务器, 服务器查到存在 $F1$ 的副本, 向用户2发送肯定确认和否定确认, 用户2根据确认值, 只将 $F3$ 上传到服务器, 图3-1(b)所示。通过这种方式, 如果服务器已经具有文件的副本, 则客户端不需要再次将其上传到服务器。这里的服务器端去重就对应数据源端数据去重, 客户端数据去重对应数据宿端的数据去重, 可以观察出服务器端的去重不能减少带宽, 因为去重在收到文件之后才进行, 而客户端去重避免了重复数据的再一次上传, 从而可以节省网络带宽量。所有在数据源端的数据去重可以减少数据的传输量, 从而减少网络带宽的消耗。在数据宿端进行数据去重, 数据需要被传送到宿主端再进行去重, 会造成不必要的网络带宽的消耗。

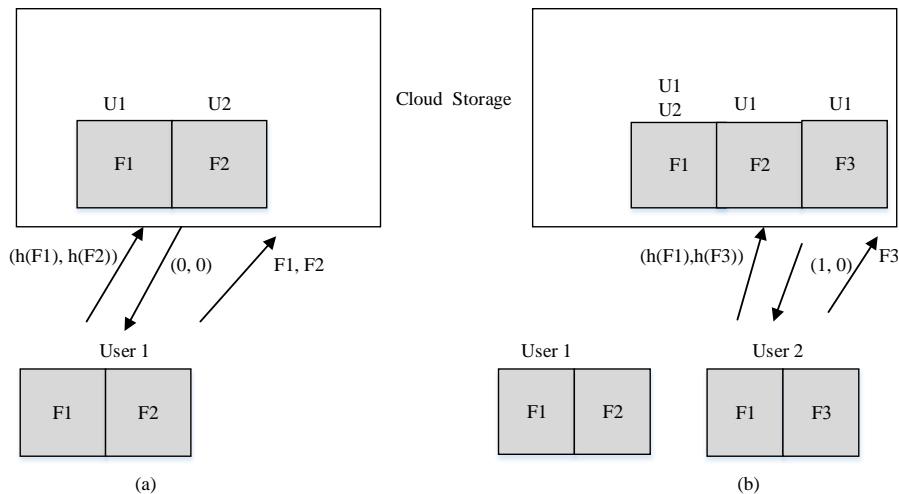


图 3-1 客户端数据去重:(a)用户1将文件 $F1$ 和 $F2$ 上传;(b)用户2将文件 $F1$ 和 $F3$ 上传

(3)根据数据去重发生时机的来分, 具体可分为离线数据去重和在线数据去重。离线数据去重指的是对已存入云存储系统中的数据进行数据去重, 这需要的是先将要处理的数据传到存储空间, 然后进行去重。而在线数据去重在数据存入磁盘之前进行去重。离线数据去重会使得网络开销很大, 而且开始会需要大的存储空间, 在线数据去重不会。

(4)根据数据去重范围来分, 可分为局部数据去重和全局数据去重^[26]两种。局部数据去重只在一个用户或者一个数据节点上进行数据去重, 而全局数据去重在多个用户或多个节点进行。

3.1.3 云存储中的数据去重过程

云存储系统中, 具体分析使用数据块级去重方法, 由于基于文件级去重过程比较简单, 对比于数据块级去重没有数据分块过程而已。具体过程如下:

1. 最开始，数据文件需要先被分块处理。这个过程中通常基于分块算法来对文件进行分块，比较常用的分块算法有固定长度的分块算法和变长分块法。本文提出的方案采用的是固定长度分块算法，根据设定的长度对数据文件进行分块。变长分块算法的计算消耗很大，数据分块速度也是很慢。
2. 使用哈希函数，每个数据块将被处理成一个唯一的散列值，用来区别出不同的数据块。若两个数据块的哈希值相同，就表示这两个数据块的内容是相同的，则可以删掉重复的数据块。常用来计算数据块散列值的哈希函数有 MD5 和 SHA-1，相比于 MD5，SHA-1 安全性更好，因为 SHA-1 的防碰撞性更高。
3. 由于云系统中存了文件及其对应的散列值，对每个将要被存储的数据块，将他们的散列值和已存储数据块的散列值进行比较，若未找到与已存储的某数据块相同的散列值，则说明该数据块是新数据块，为该数据块创建存储空间，同时更新存放块散列值的库；若找到了与已存在的某数据块相同的散列值，则说明该数据块是重复的，进行去重。

3.1.4 云存储中数据去重的安全问题

云存储中数据去重的实现都是基于上述提到的采用计算数据的 MD5 值或是散列值进行对比的方法来实行，通过这种方式来实现简单直接的文件去重，然而这也会带来许多安全隐患，下面介绍一些可能遭受的攻击表明使用短信息(哈希值)作为文件所有权的声明方式不是明智的。

公共哈希函数的使用，通常云存储中数据去重使用标准哈希函数。如 Dropbox 使用的是 SHA256^[27]。在这种情况下，攻击者是可以获取其他用户拥有的文件的哈希值，例如，如果这些哈希值在别处使用过或用户发布文件的哈希值。拥有文件哈希值的任何人都可以通过上传文件散列来获得文件的所有权，这是因为，当云服务器接收对已经被存储文件的存储请求时，避免冗余文件传输，就会为用户和该文件建立联系，表示用户拥有该文件的所有权，图3-2(a)中给出了详细示例。

使用存储服务作为内容分发网络(Content Distribution Network，简称CDN)，在这种情形下，敌手或是恶意用户将一个文件 M 上传到存储服务器，该文件可能是侵犯版权的，并发布该文件的哈希在她的网页上，那么任何希望获取该文件 M 的人都可以通过发送该哈希值给服务器，获得文件的所有权，然后云服务器上的文件 M 。因此，敌手或是恶意用户基本上将存储服务器用作CDN，图3-2(b)给出了详细示例。

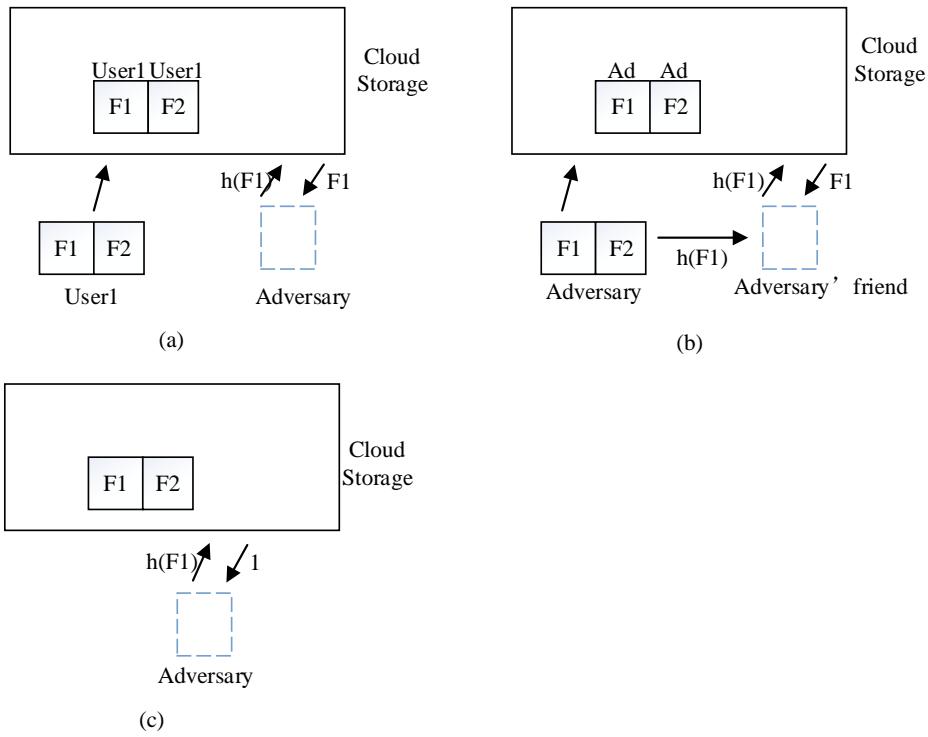


图 3-2 数据去重的安全问题:(a)仅通过hash值获取文件;(b)将服务器用作CDN;(c)推断文件的存在性

云中文件存在的隐私可能受到损害，因为恶意用户或是攻击者可能试图上传候选文件以查看候选文件的副本是否已经存在云服务器上了。如果发生客户端去重，这将是文件存在的指示符。否则，对手可以推断文件不存在云上。图3-2(c)中给出了说明性示例。情况更糟的是，当我们考虑低熵文件时，因为攻击者可以详尽地创建不同的文件并上传哈希以检查文件的存在，例如，好奇的同事可以通过上传不同的工资表来查询他/她的经理的工资，因为这些表的形式是很相似的。

攻击者入侵服务器，访问其内部缓存，其中包括所有最近访问的文件的哈希值。获得这些个所有的文件哈希值后，攻击者可以通过上传哈希值下载所有这些文件，这些文件可能包括机密文件。甚至攻击者可以发布这些哈希值，从而使世界上任何人都能获得这些文件。

这种上述出现的种种安全问题，主要是因为使用短信息(哈希值)作为文件所有者的声明方式，即通过学习关于文件的一小块信息，即其哈希值，攻击者能够从服务器获得整个文件，实际上文件哈希值并不是什么秘密，它可以通过由所有客户端共享的确定性公开算法从文件计算出来。所以仅仅靠检验文件哈希值来实现去重是不安全的，因为总有一些恶意或是不诚实的用户，他们声称拥有某文件，其实他们只有某文件的哈希值，以此来欺骗服务器获得到文件的链接。针对这些

可能出现的安全隐患，本文提出了一种安全有效的POW方案，通过该方案服务器能验证用户是否真的拥有文件而不仅仅是拥有文件的哈希值。

3.2 对Zheng等的POSD方案的分析

为了解决云存储中数据去重的安全问题，zheng等^[18]提出了一个能支持数据所有权证明的方案，该方案不仅能实现安全的数据去重，还支持云数据的完整性验证。

3.2.1 方案回顾

Zheng等的 POSD方案有四个算法，分别为密钥生成 (KEYGEN)，文件上传 (UPLOAD)，审计 (AUDITINT)以及去重 (DEDUP)四个算法，在该方案中将文件 F 分成 m 个符号的 n 个数据块，表示为 $F_i = (F_{i1}, \dots, F_{im})$, $F_i \in Z_p^m$, $i \in [1, n]$ 。

1. *KEYGEN*: 这个算法是用来生成密钥的。

- 随机均匀选择两个数 ν_1 和 ν_2 , $\nu_1, \nu_2 \in Z_p^*$, ν_1, ν_2 的阶为 q 。然后又均匀选择两个随机数 $s_{j1}, s_{j2}, s_{j1}, s_{j2} \in Z_p^*$, 其中 $1 \leq j \leq m$, 让 $z_j = \nu_1^{-s_{j1}} \nu_2^{-s_{j2}} \bmod p$, $1 \leq j \leq m$ 。
- 让参数 g 作为群 G 的生成元，随机均匀选择参数 $u, u \in G$ ，均匀选择随机数 $\omega, \omega \in Z_p^*$ ，并且设置 $z_g = g^\omega$ 。
- 设置 $PK_{int} = (q, p, g, u, \nu_1, \nu_2, z_1, \dots, z_m, z_g)$ 以及用户的私钥为 $SK_{int} = (s_{11}, s_{12}), \dots, (s_{m1}, s_{m2}), \omega$ 。这里使用了一个适当的伪随机函数(Pseudorandom Function, 简称 PRF), 可以大大地减少在用户端的存储，因为输入一个简单的密钥到 PRF就可以生成 $s_{11}, s_{12}, \dots, s_{m1}, s_{m2}, \omega$ 。
- 设置 $PK_{dup} = PK_{int}$ 以及 $SK_{dup} = null$, PK_{dup} 也是公开的。

2. *UPLOAD*: 这个算法一个用户和服务器之前执行，用户将文件 F 外包到云服务器上去，具体操作如下：

- 对每一个数据块 F_i , 其中 $1 \leq i \leq n$, 用户均匀地选择两个随机数 r_{i1}, r_{i2} , $r_{i1}, r_{i2} \in Z_p^*$ 并且进行如下计算:

$$\begin{aligned} x_i &= \nu_1^{r_{i1}} \nu_2^{r_{i2}} \bmod p \\ y_{i1} &= r_{i1} + \sum_{j=1}^m F_{ij} s_{j1} \bmod q \\ y_{i2} &= r_{i2} + \sum_{j=1}^m F_{ij} s_{j2} \bmod q \\ t_i &= ((H_1)(fid\|i) \cdot u^{H_2(x_i)})^\omega (\in G) \end{aligned}$$

用户发送 (fid, F, Tag_{int}) 给服务器， $Tag_{int} = \{(x_i, y_{i1}, y_{i2}, t_i)_{1 \leq i \leq n}\}$ 。

- 服务器一旦收到 (fid, F, Tag_{int}) ，服务器就将设置 $Tag_{dup} = Tag_{int}$ 。

3. AUDITINT: 这个算法在验证者和云服务器之间执行，验证者可以是用户自己，也可以是第三方验证者，云服务器需要向验证者证明用户的数据文件 F 完好无损的存储在云上。在该过程中用户不需要将任何的信息给验证者，除了用户的公钥 PK_{int} 和数据文件的标识符 fid 。

- 审计者即验证者选择一个 c 元素的集合 $I = \{\alpha_1, \alpha_2, \dots, \alpha_c\}$ ，其中 α_i 是从 $\{1, \dots, n\}$ 中随机选择的，并且选择一个同样有效的 β 集合 $\beta = \{\beta_1, \dots, \beta_c\}$ ，其中 β_i 是从 Z_p^* 中随机选择的。然后审计者就将 $chal = (I, \beta)$ 发送给服务器，为了更直观参见图3-3。

- 收到 $chal$ 后，服务器做如下计算：

$$\mu_j = \sum_{i \in I} \beta_i F_{ij} \bmod q$$

其中 $1 \leq j \leq m$ ，以及计算：

$$\begin{aligned} Y_1 &= \sum_{i \in I} \beta_i y_{i1} \bmod q \\ Y_2 &= \sum_{i \in I} \beta_i y_{i2} \bmod q \\ T &= \prod_{i \in I} t_i^{\beta_i} (\in G) \end{aligned}$$

服务器发送挑战回应给验证者 $resp = (\{\mu_j\}_{1 \leq j \leq m}, \{x_i\}_{i \in I}, Y_1, Y_2, T)$ ，其中 $x_i = \nu_1^{r_{i1}} \nu_2^{r_{i2}} \bmod p$ 是在 UPLOAD 算法中，用户生成的， $1 \leq i \leq n$ 。

- 一旦验证者收到挑战回应 $\{\{\mu_j\}_{1 \leq j \leq m}, \{x_i\}_{i \in I}, Y_1, Y_2, T\}$ ，计算：

$$\begin{aligned} X &= \prod_{i \in I} \beta_i \bmod p \\ W &= \prod_{i \in I} H_1(fid \| i)^{\beta_i} \end{aligned}$$

以及最后验证

$$\begin{aligned} X &\stackrel{?}{=} \nu_1^{Y_1} \nu_2^{Y_2} \prod_{j=1}^m z_j^{\mu_j} \bmod p \\ \hat{e}(T, g) &\stackrel{?}{=} \hat{e}(W u^{\sum_{i \in I} \beta_i H_2(x_i)}, z_g) (\in G_T) \end{aligned}$$

如果上面的两个等式都成立，就返回1，或者返回0。

4. DEDUP: 这一算法在用户和服务器之间执行，该用户声称他拥有文件 F 和文件的标识符 fid ，而且该文件已经被其他用户存储到服务其上了。该算法

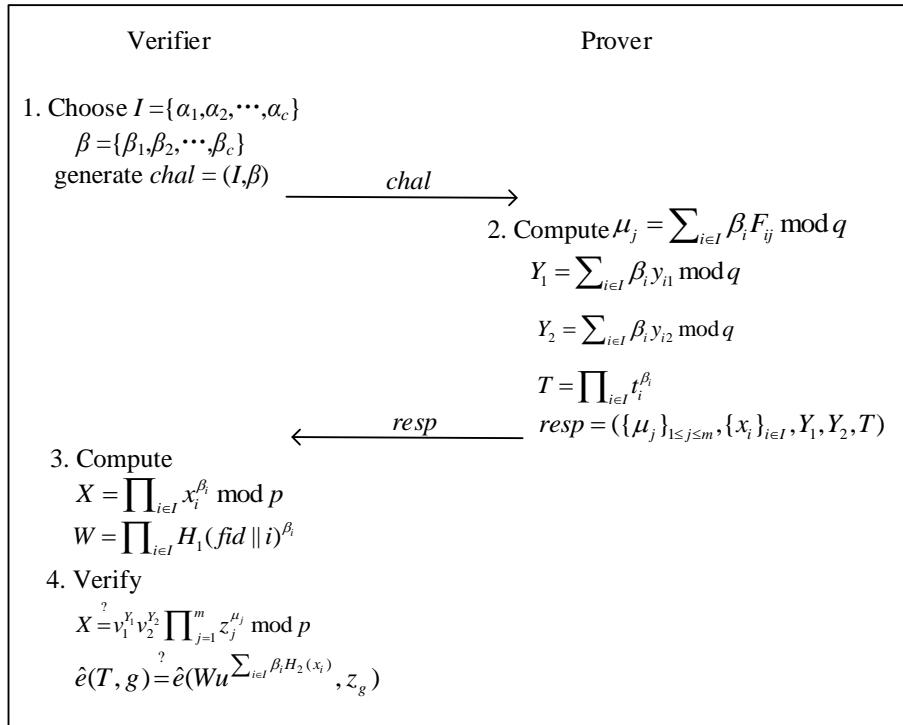


图 3-3 审计者(verifier)和服务器(prover)之间进行执行的AUDITINT协议过程

是上面 AUDITINT 算法的一个变形版本，审计者仅仅只要知道用户的公钥 PK_{int} 和文件的标识符 fid 。在这种情况下，让云服务器扮演审计者即验证者的角色，云服务器自然是知晓 PK_{int} 和 fid 。详细过程见图3-4

- 服务器选择一个 c 元素的集合 $I = \{\alpha_1, \alpha_2, \dots, \alpha_c\}$ ，其中 α_i 是从 $\{1, \dots, n\}$ 中随机选择的，并且选择一个同样有效的 β 集合 $\beta = \beta_1, \dots, \beta_c$ ，其中 β_i 是从 Z_p^* 中随机选择的。然后服务器就将 $chal = (I, \beta)$ 发送给声称有拥有文件 F 的用户。

- 收到 $chal$ 后，用户做如下计算：

$$\mu_j = \sum_{i \in I} \beta_i F_{ij} \bmod q$$

其中 $1 \leq j \leq m$ ，并将挑战相应 $resp = (\{\mu_i\}_{1 \leq i \leq m})$ 发送给服务器。

- 服务器收到挑战回应之后，从自己已经存储了的文件 F 得到标签 $Tag_{dup} =$

$\{(x_i, y_{i1}, y_{i2}, t_i)_{1 \leq i \leq n}\}$ 计算：

$$Y_1 = \sum_{i \in I} \beta_i y_{i1} \bmod q$$

$$Y_2 = \sum_{i \in I} \beta_i y_{i2} \bmod q$$

$$X = \prod_{i \in I} x_i^{\beta_i} \bmod p$$

$$W = \prod_{i \in I} H_1(fid \parallel i)^{\beta_i}$$

$$T = \prod_{i \in I} t_i^{\beta_i} (\in G)$$

以及服务器最后验证

$$X \stackrel{?}{=} \nu_1^{Y_1} \nu_2^{Y_2} \prod_{j=1}^m z_j^{\mu_j} \bmod p$$

$$\hat{e}(T, g) \stackrel{?}{=} \hat{e}(W u^{\sum_{i \in I} \beta_i H_2(x_i)}, z_g) (\in G_T)$$

如果上面的两个等式都成立，就返回1，或者返回0。

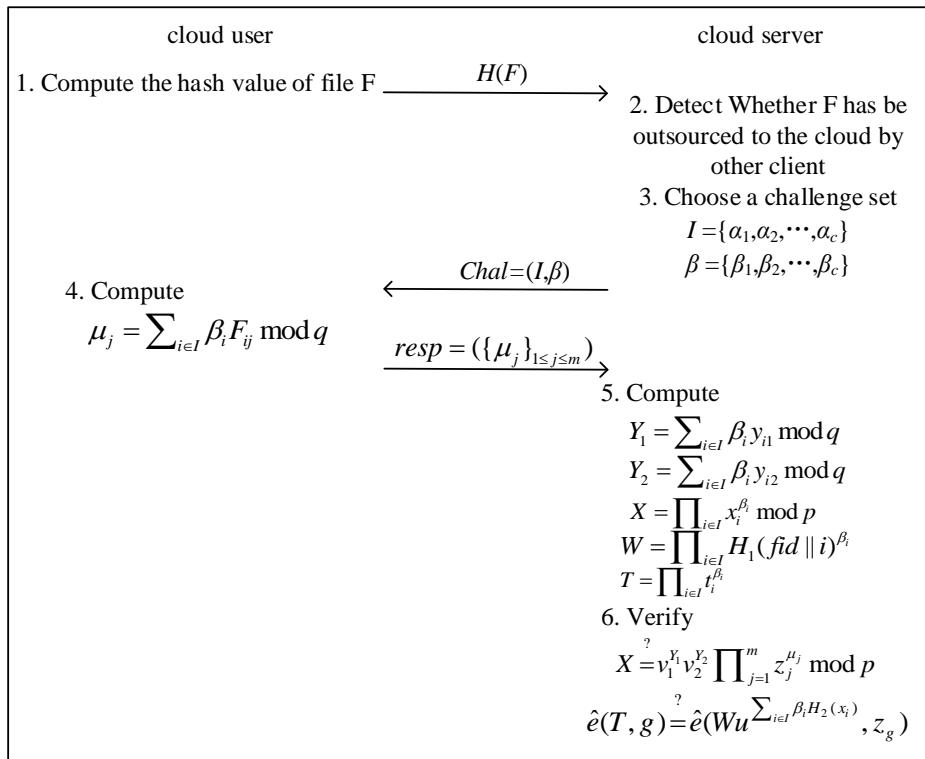


图 3-4 用户(prover)和服务器(verifier)之间进行执行的DEDUP协议过程

3.2.2 方案分析

下面从两个方面来分析Zheng等的POSD方案是否能实现支持去重的云数据完整性检测验证。

(1)数据完整性验证：在AUDITINT阶段，可以看到，验证者(第三方或是用户)通过判断 $X \stackrel{?}{=} \nu_1^{Y_1} \nu_2^{Y_2} \prod_{j=1}^m z_j^{\mu_j} mod p$ 和 $\hat{e}(T, g) \stackrel{?}{=} \hat{e}(W u^{\sum_{i \in I} \beta_i H_2(x_i)}, z_g)$ 是否成立来保证数据的正确无误性。

(2)数据所有权的验证：在DEDUP的阶段中，服务器通过验证下面等式 $X \stackrel{?}{=} \nu_1^{Y_1} \nu_2^{Y_2} \prod_{j=1}^m z_j^{\mu_j} mod p$ 和 $\hat{e}(T, g) \stackrel{?}{=} \hat{e}(W u^{\sum_{i \in I} \beta_i H_2(x_i)}, z_g)$ 是否成立来验证用户对文件的所有权证明。

该方案本质上使用两次PDP来同时实现数据完整性验证和数据拥有性证明，通过分析发现Zheng等的POSD方案可以实现支持数据去重的云数据完整性检测验证。但是，该方案存在一个缺陷就是在四个阶段都选择了大量的随机数来进行计算，导致计算比较繁琐和复杂。另一个缺陷就是不能实现零知识隐私保护，因为在数据完整性验证的过程中，服务器会发送 $\mu_j = \sum_{i \in I} \beta_i F_{ij} mod q$ ，只是将文件块的信息和对应的随机数进行相乘，然后将所有挑战的块做一个累加，当所有随机选择的数值是相同时，存在文件数据泄露的隐患，并且当挑战块的随机值*i*不一样时，不完全可信的第三方通过重复挑战相同的数据块依然可以获知用户文件的信息。

3.3 本章小结

本章对数据去重的相关概念、特点、过程和分类进行详细介绍，进而介绍了云存储中数据去重的安全问题，最后回顾Zheng等^[18]提出的POSD方案，并对方案进行了分析，在云服务器为了回应验证者的一个挑战时，云服务器会发送 $\mu_j = \sum_{i \in I} \beta_i F_{ij} mod q$ 。因为，每个 μ_j 会泄露数据块的部分信息，通过重复挑战这些块，验证者将知晓数据信息，发现该方案虽能同时实现数据去重和数据完整性验证，但是该方案不具备零知识隐私保护的性质，在审计的过程中会将审计的数据信息泄露给第三方验证者。

第四章 支持密钥更新的隐私保护的云数据审计方案

4.1 问题提出

在云存储中，由于云用户私钥丢失或是过期是常常发生的事情，当事故发生后，云用户需要更换当前的公钥。此时，云用户会面临一个问题，如何用更新后的公钥来验证外包到云服务器上的数据的完整性，因为服务器上存的旧的文件块的认证子是基于旧的私钥生成的，用更新后的公钥去验证是无效的。解决这个问题最直接的方法就是，用户从云服务器上下载所有的文件和文件块的认知子，用更新后的私钥重新计算文件中每个块的认证标签，再将更新后的认证标签和文件上传到服务器上。然而这种方法需要消耗大量的网络带宽，是不可取的。2016年，Yu等人^[28]提出了一个支持密钥更新和零知识隐私保护的公开云数据审计方案，该方案提出当用户更新自己的密钥时，并不需要下载所有的文件以及重新计算所有文件的数据块认证标签，而只需下载和更新认证标签。相比于下载整个文件再计算认证标签，该方案确实可以大大减少网络带宽的消耗。

不同于Yu的方案，本章描述的方案中，在认证标签更新的过程中使用一个单向的代理重签名，即使用户发布了重签名密钥，也没有必要担心泄露新的密钥，在进行密钥更换时，用户只需将重签名的密钥给云服务器，然后更新自己的公钥和重签名密钥，其他的事都交由云服务器完成，不仅大大节约网络带宽，还减少了云用户的计算开销。

4.2 问题描述

通过分析云存储环境中数据完整性检测和用户密钥更新所牵涉到的关键问题，提出解决该问题所需要涉及到的系统模型、系统组成和安全模型。

4.2.1 系统模型

一个云存储服务系统主要包括以下三个实体：即云用户，云服务器和第三方审计者(TPA)。云用户需要大的存储空间存储自己大量的数据文件，为了减轻本地存储的负担，将产生的数据备份存储到服务器上。云服务器具有大量的存储空间和计算资源，为云用户提供数据存储等各种服务。TPA可以是由政府管理的组织，他是一个可信的组织，替云用户检查外包到云上的数据的完整性。他具有用户不具有的专业技术和能力，能很好地提供数据审计服务。具体描述见图4-1。

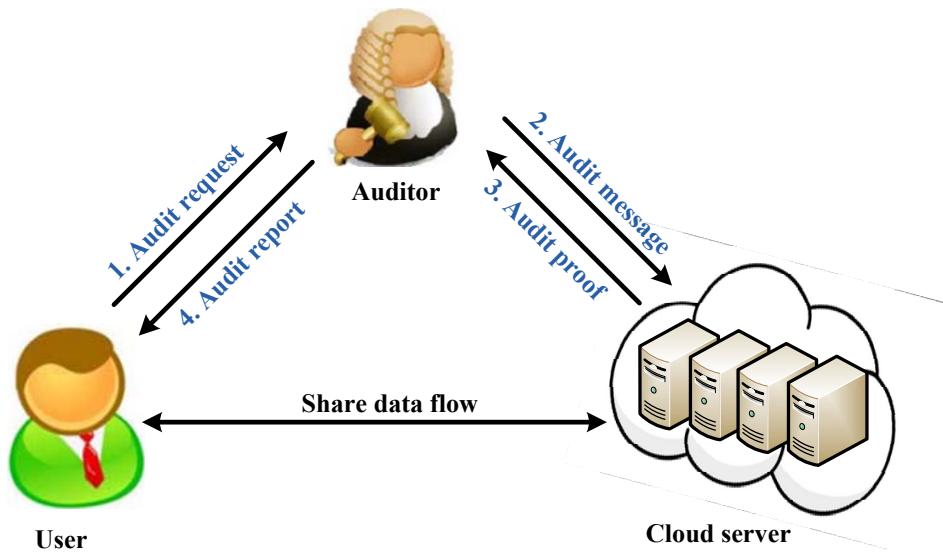


图 4-1 公开可验证的数据完整性检测协议的系统模型

4.2.2 系统组成

本章中具有密钥更新和标签更新的公开审计方案由 CrsGen, KeyGen, AuthGen, KeyUpdate, AuthUpdate 五个算法和证明者与验证者之间的交互式证明系统 Proof 组成。

CrsGen(1^{κ}): 这个算法输入一个安全参数 κ 输出一个包含所有公共参考的字符串 crs , 它里头的参数值是下面描述的所有算法的输入。

KeyGen(crs): 该算法输入 crs , 为云用户生成一个公钥 pk 和一个私钥 sk , 用户公布公钥 pk 私钥 sk 保密。该算法也用作密钥更新的算法

AuthGen(sk, F): 算法输入私钥 sk 和一个文件 $F = (m_1, \dots, m_n)$, 输出文件的标签 $\{D_i\}$ 和公共验证参数 ϕ , 在证明阶段用来检验数据的完整性。

KeyUpdate(sk, pk): 输入 $(l - 1)$ 和旧的密钥对 (sk_{l-1}, pk_{l-1}) , 算法输出新的密钥对 (sk_l, pk_l) 。

AuthUpdate($sk_l, pk_l, ft_{l-1}, \phi$): 输入新的密钥对 (pk_l, sk_l) , 和原始的文件标签 ft_{l-1} 和公开的验证参数 ϕ , 这个算法输出新的文件标签 ft_l 和更新的密钥 β_l , 在新密钥对下 β_l 是有效的。

Proof(\mathcal{P}, \mathcal{V}): 这是一个在验证者 (\mathcal{V}) 和证明者 (\mathcal{P}) 之间的交互式协议。输入到 $(\mathcal{P}, \mathcal{V})$ 是公钥 pk 和公开验证参数 ϕ 。 \mathcal{P} 另外还输入文件 $F = (m_1, \dots, m_n)$ 和这个文件标签的集合 $\{D_i\}$ 。最后, \mathcal{V} 输出 0 或 1 来表明外包到云上的文件是否损坏。为了更直观, 使用 $\mathcal{P} \Leftrightarrow \mathcal{V}(pk, \phi) = 1$ 来表示和 \mathcal{P} 交互后, \mathcal{V} 输出 1。当整个方案清晰时, 省略公开的参数 (pk, ϕ) 。

4.2.3 安全模型

完整性，完备性和数据隐私性是提出的隐私保护的公开云审计方案的三个安全要求。完整性表示当和完整保存数据的云服务器交互时，如果服务器和TPA都诚实地执行协议，交互协议的证明过程将总是返回1。

完备性指的是任何可以使验证者确信它存了数据文件的证明者，那么它真的如实保存了文件。基于Shacham 和Waters提出的安全模型^[17]，我们定义了带密钥更新和认证子更新的可以抵抗恶意敌手的公开审计方案的安全模型，其中包括了两个实体：敌手，可以看作不可信的服务器；挑战者，代表着一个用户或审计者(验证者)。敌手的目的是成功欺骗验证者，即，在没有完整存文件的前提下产生一个有效的回应。方案中的模型和之前的PDP或PoR模型的主要错别是我们包含了密钥更新和认证子更新。

完备性 为了实现完备性，我们首先扼要重述在文献[17]中定义的 ϵ -可容许的证明者的定义。一个算法 $P\beta$ 关于 (pk, ϕ) 是 ϵ -可容许的，如果 $\Pr[\mathcal{P}' \Leftrightarrow \mathcal{V}(pk, \phi) = 1] \geq \epsilon$ ，即它可以令人信服地回答一个诚实的 TPA 的占比为 ϵ 部分的验证挑战。

考虑下面的敌手 \mathcal{A} 和挑战者 \mathcal{C} 之间的游戏。

1.*Init*: 挑战者 \mathcal{C} 初始化一个计数器 $count = 0$ 和一个空的表 Tab 。它运行 $Crs-Gen$ 算法，以安全参数 k 为输入来获得 crs 。然后它运行 $KeyGen$ 算法生成一个密钥对 (pk_{count}, sk_{count}) 。将 crs, pk_{count} 发送给敌手 \mathcal{A} 。我们假设总有 \mathcal{A} 可以得到目前的 $count$ 和 Tab 的值。

2.*AuthQuery*: 我们可以安全地假设对于每次询问文件 F 都是不同的。如果一个相同的文件 F 提交了多次， \mathcal{C} 可以返回存在 Tab 表中的值给敌手 \mathcal{A} 。敌手 \mathcal{A} 提交一个文件 $F = (m_1, \dots, m_n)$ 。 \mathcal{C} 运行 $AuthGen$ 算法，输入 (sk_{count}, F) 来计算相应的认证子 $\{D_i\}$ 和公开的参数 ϕ_F 。然后 \mathcal{C} 在表 Tab 中增加一行 $(F, \{D_i\}, \phi_F)$ 。因为 \mathcal{A} 可以访问表 Tab ，因此这里省略返回值。

3.*ProofQuery*: 敌手 \mathcal{A} 可以和 \mathcal{C} 运行证明协议。这里， \mathcal{C} 扮演 TPA (\mathcal{V}) 的角色，输入 (pk_{count}, ϕ_F) 。协议结束后 \mathcal{V} 的输出发送给 \mathcal{A} 。

4.*Update*: 当 \mathcal{A} 调用这个询问， \mathcal{C} 首先设置 $count = count + 1$ ，然后， \mathcal{C} 调用 $KeyGen$ 算法获得新的密钥对 (pk_{count}, sk_{count}) 。对于表 Tab 中的每一行 $(F, \{D_i\}, \phi_F)$ ， \mathcal{C} 调用 $AuthUpdate(sk_{count-1}, pk_{count-1}, sk_{count}, pk_{count}, \{D_i\}, \phi_F)$ 来获得 $\{D'_i\}, \phi'_F$ 。将这行更新为 $(F, \{D'_i\}, \phi'_F)$ 。之后， \mathcal{C} 返回 $sk_{count-1}$ 给 \mathcal{A} 。

5.*Challenge*: 在某一时刻， \mathcal{A} 输出证明算法的描述 \mathcal{P}^* 。

我们首先扼要重述提取器的概念，一个提取器 ext 是一个算法，其输入为 ϵ -可容许的证明者 \mathcal{P}' ，公钥 pk 和验证参数 ϕ ，输入一个文件 F 。我们表示 $\text{ext}(\mathcal{P}', pk, \phi) = F$ 。

现在我们定义完备性。特别地，我们要求对于所有的 $(pk_{\text{count}}, \phi)$ 输出为一个 ϵ -可容许的证明者 \mathcal{P}^* 的多项式算法 \mathcal{A} ，存在提取器 ext ， $(\text{ext}(\mathcal{P}^*, pk_{\text{count}}, \phi), pk_{\text{count}}, \phi)$ 是表 Tab 中的一行。

换句话说，当一个算法可以以大于 ϵ 的概率通过证明协议，它一定以某种格式存储了原文件。而不论以哪种形式存储，通过提取器算法，文件最终可以恢复成原来的文件。

零知识数据隐私 这个性质保证 TPA 不能知道文件内容的任何信息除了基于公开可获得的信息得到存储文件的一个随机的文件名，为了增强这个性质，我们允许 TPA 选择两个等长的文件 $F_0 = (m_{0,1}, \dots, m_{0,n})$ 和 $F_1 = (m_{1,1}, \dots, m_{1,n})$ ，并且要求给定元数据和与服务器的交互，TPA 不能获得文件 F_0 和 F_1 的任何信息。一个正式的安全模型描述如下所示：

考虑敌手 \mathcal{A} 和挑战者 \mathcal{C} 之间的游戏：

1. *Init*: \mathcal{C} 先初始化一个计数器 $\text{count} = 0$ ，计数器 $j = 1$ 和一个空表 Tab 。它运行 CrsGen ，输入安全参数 k ，输出为 crs 。然后运行 KeyGen 产生一个密钥对 $(pk_{\text{count}}, sk_{\text{count}})$ 。 $\text{crs}, pk_{\text{count}}$ 发送给敌手 \mathcal{A} 。假设 \mathcal{A} 总是可以获得现在的 count 和 Tab 的值。 \mathcal{C} 初始化一个空行 $R^* = (0, \perp, \perp, \perp)$ ，它被称作挑战行。
2. *AuthQuery*(F): 我们可以安全地假定对于每一个 *AuthQuery* 询问，文件 F 都是不同的。如果一个相同的文件 F 被提交， \mathcal{C} 可以返回存储在表 Tab 中的值给敌手。 \mathcal{A} 提交一个文件 $F = (m_1, \dots, m_n)$ ， \mathcal{C} 运行 *AuthGen* 算法，输入 (sk_{count}, F) 来计算相应的认证子 $\{D_i\}$ 和公开的验证参数 ϕ_F 。然后 \mathcal{C} 在表 Tab 中增加一行 $(j, F, \{D_i\}, \phi_F)$ ，同时 j 增加 1。
3. *GenChallenge*: 这个询问只可以被询问一次。在某一时刻， \mathcal{A} 提交两个等长的文件 F_0 和 F_1 (它们和 *AuthQuery* 的所有输入都是不同的)。 \mathcal{C} 投掷硬币获得 $b \in_R \{0, 1\}$ ，运行算法 *AuthGen*，输入 (sk_{count}, F_b) 来计算相应的认证子 $\{D_i\}$ 和公开参数 ϕ_{F_b} 。 \mathcal{C} 设置挑战行 R^* 为 $(0, F_b, \{D_i\}, \phi_{F_b})$ 。 ϕ_{F_b} 发送给敌手 \mathcal{A} 。
4. *ProofQuery*(\hat{j}): \mathcal{A} 和 \mathcal{C} 进行证明协议。 j 是被 \mathcal{A} 指定的，可以参考表 Tab 的第 j 行或者如果 $\hat{j} = 0$ ，则是挑战行(假定已经进行了 *GenChallenge* 询问)。 \mathcal{A} 扮演 TPA 的角色， \mathcal{C} 扮演证明者的角色，输入为 $(pk_{\text{count}}, F, \{D_i\}, \phi_F)$ ，若 $j > 0$ ，则 $(j, F, \{D_i\}, \phi_F)$ 是表 Tab 的一行，若 $j = 0$ ，则为 $(pk_{\text{count}}, F_b, \{D_i\}, \phi_{F_b})$ 。

5. *Update*: 当 \mathcal{A} 调用这个询问, \mathcal{C} 首先设置 $\text{count} = \text{count} + 1$ 。然后, \mathcal{C} 再一次运行 KeyGen 算法获得新的密钥对 $(pk_{\text{count}}, sk_{\text{count}})$ 。对于表 $\text{Tab} \cup R^*$ 的每一行 $(F, \{D_i\}, \phi_F)$, \mathcal{C} 调用 $\text{AuthUpdate}(sk_{\text{count}-1}, pk_{\text{count}-1}, sk_{\text{count}}, pk_{\text{count}}, \{D_i\}, \phi_F)$ 来获得 $\{D'_i\}, \phi'_F$, 将这行更新为 $(F, \{D'_i\}, \phi'_F)$ 。之后, \mathcal{C} 将 $sk_{\text{count}-1}$ 返回给 \mathcal{A} 。

6. *Guess*: 在某一时刻, \mathcal{A} 输出 b' 的猜测。

如果 $b = b'$, 则 \mathcal{A} 赢得了游戏, \mathcal{A} 的优势为它赢得上述游戏的概率 0.5^①, 一个公开的云数据审计方案被称为零知识数据隐私如果任何多项式时间对手 A 的优势是可忽略的。

4.2.4 零知识证明

一个知识协议的零知识证明^[29]使证明者 (\mathcal{P}) 可以使验证者 (\mathcal{V}) 信服一些陈述是正确的, 但是验证者除了知道陈述的有效性, 其他的什么也不知道。下面, 我们回顾一个简单的但强有力的离散对数实例的零知识证明, 它是被 Schnorr 在他的身份识别方案^[30]中提出, 它在我们的方案中起了很重要的作用。这个知识的零知识证明允许 \mathcal{P} 向 \mathcal{V} 证明知识 $x \in Z_p$, 使得对于某个 $y \in G$, $y = g^x$ 成立。

Commitment. \mathcal{P} 挑选随机数 $\rho \in Z_p$, 计算 $T = g^\rho$ 并且发送 T 给 \mathcal{V} 。

Challenge. \mathcal{V} 挑选随机数 $c \in \{0, 1\}^\lambda$ 并且发送 c 给 \mathcal{P} 。

Response. \mathcal{P} 计算 $z = \rho - cx \pmod p$ 并且返回 z 给 \mathcal{V} 。

Verify. 当且仅当 $T = y^c g^z$, \mathcal{V} 接受该证据。

4.2.5 代理重签名

代理重签名的概念首先被 Blaze 等人^[31]于 1988 年提出, 一个半可信的代理可以将由 Alice 私钥签的一个消息签名转化成由 Bob 私钥签名的相同消息的一个签名。Ateniese 等人^[32] 通过下面的算法形式化了这个原语。

1. *Setup*(1^λ): 这个概率算法的输入为安全参数 1^λ , 它产生公共参数 pp 。

2. *Keygen*(pp): 该概率算法的输入为公共参数 pp , 输出一个签名密钥对 (pk, sk) 。

3. *ReKeygen*(pp, pk_i, sk_j): 输入公开参数 pp , 原来签名者 i 的公钥 pk_i 和一个签名者 j 的私钥 sk_j , 这个算法产生一个重签名密钥 R_{ij} 用来将用户 i 的签名转换为用户 j 的签名。

^① 这是为了抵消随机猜测的获胜概率

4. $\text{Sign}(pp, sk_i, m)$: 输入公开参数 pp , 签名者的私钥 sk_i 和一个消息 m , 这个算法产生一个用户 i 的签名 σ 。
5. $\text{ReSign}(pp, R_{ij}, m, \sigma, pk_i)$: 输入公共参数 pp , 一个重签名密钥 R_{ij} , 消息 m 和签名 σ , 这个算法首先检查是否 $\text{Verify}(pp, pk_i, m, \sigma) = 1$, 即验证是否成功, 如果是, 这个算法输出一个对应于用户 j 的签名 σ' , 或者输出 \perp .
6. $\text{Verify}(pp, pk_i, m, \sigma)$: 这是一个确定性算法, 输入公开参数 pp , 消息 m , 签名者的公钥 pk_i 和签名 σ , 它返回 1 或 0 来证明签名是否有效。

4.3 具备密钥更新的隐私保护的云数据审计方案描述

4.3.1 方案描述

本方案基于 Shacham 和 Waters 提出的紧致的 PoR 方案^[9], 他们的方案使用了一个基于短签名的同态认证子^[33], 使得所有公开可验证的可恢复性证明 (PoR) 的询问和回应更简短。

对于密钥更新和认证子的更换问题, 本方案在认证子更换的过程中使用一个单向代理重签名^[34], 因此即使用户丢失了重签名密钥, 用户的新密钥也不会被泄露。具体来说, 用户只要将重签名密钥传到云服务器并且更新自己的公钥的重签名的密钥, 这可以让云用户少做许多计算工作, 即只要生成新的私钥和重签名密钥, 这种新方法可以极大的减少通信开销。

关于数据隐私, 我们使用离散对数的零知识证明^[22], 在响应过程中被挑战块的认证子 σ 和组合 μ_i 都被随机化。因此, TPA 除了能知道用户随机选择的文件名字外, 得不到存储文件的丝毫信息。

方案的详细构造如下:

CrsGen(1^λ): 输入安全参数 λ , 这个算法输出有一个大素数 p 和阶为 p 群 G , G_T 。 $\hat{e}: G \times G \rightarrow G_T$ 表示一个双线性对, $H_0, H_1: \{0, 1\}^* \rightarrow G$ 表示两个抗碰撞的哈希函数。另外算法会选择随机数 $h, u_1, u_2 \dots, u_s \in G$ 并且计算 $\eta = e(g, h)$ 。公开的参数 **crs** 是 $(k, p, G, G_T, g, e, H_0, H_1, h, u_1, \dots, u_s, \eta)$ 。

KeyGen(**crs**): 输入公开的参数 **crs**, 用户生成一个签名密钥对 (spk_1, ssk_1) , $spk_1 = g^{ssk_1}$ 和另外一个密钥对 (α_1, v_1) , 这个密钥对用来为文件数据块生成认证标签, $\alpha_1 \in Z_p$, $v_1 = g^{\alpha_1}$ 。用户的私钥是 $sk = (\alpha_1, ssk_1)$, 公钥为 $pk = (spk_1, v_1)$ 。为了方便识别, 我们使用 η_i 来代表 $e(u_i, v)$, i 从 1 到 s 。值得注意的是, 给定 v 和 **crs**, $\eta_1 = e(u_1, v), \dots, \eta_s = e(u_s, v)$ 可以由相关方预先计算得到。

AuthGen(sk, F): 给定文件 F , 用户首先使用诸如 RS 码之类的擦除码来获得一个被处理的文件 F' , 并且将文件 F' 分成 n 块, 每一块又分成 s 个扇区 $\{m_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq s}$,

$m_{ij} \in Z_p$ 。用户从足够大的域中选择一个文件名为 Fn , t_0 为 $Fn||n$ 。用户计算 $t = (H_0(t_0))^{ssk}$ 以及表示文件标签 $ft = t_0||t$ 。然后用户对每个块计算标签

$$\sigma_i = (H_1(Fn||i) \cdot \prod_{j=1}^s u_j^{m_{ij}})^{\alpha_1}.$$

$i, 1 \leq i \leq n$ 。最后，用户存储

$$ft||\{m_{ij}\}_{(1 \leq i \leq n, 1 \leq j \leq s)}||Metadata$$

到服务器, $Metadata = \{\sigma_i\}_{1 \leq i \leq n}$ 。注意, 此算法中存在严格的访问控制, 即谁可以访问存储的文件和认证子。

KeyUpdate(pk_{l-1}, sk_{l-1}): 如果第 $(l-1)$ 个密钥过期或是被泄露, 那么用户通过生成一个新的密钥对 (ssk, spk) 将 (spk_{l-1}, ssk_{l-1}) 更换成 $pk_l = (spk_l, v_l), sk_l = (\alpha_l, ssk_l)$, $spk = g^{ssk}, v_l = g^{\alpha_l}$ 。

AuthEvolve(pk_{l-1}, sk_{l-1}): 用户从云上下载 ft , 并且如下计算文件标签和元数据

1. 计算 $t_l = t^{ssk_l/ssk_{l-1}}$ 并且使 $ft_l = t_0||t_l$.
2. 计算 $\beta_{l-1} = g^{\alpha_l/\alpha_{l-1}}$.
3. 随后将 t_l 和 β_{l-1} 发送到云服务器。

所以, 新的 $Metadata = (\sigma_1, \dots, \sigma_n, \alpha_1, \dots, \alpha_{l-1}, \beta_1, \dots, \beta_{l-1})$, 认证子的更新由服务器完成, 用作之后的验证。

Proof($P(F, \{\sigma_i\}, ft), V(pk)$): 这是在证明者(云服务器)和验证者(TPA)之间执行的交互式证明协议, 具体过程如下。

- 1.TPA 选择随机整数 c 和 $k, \psi \in Z_p$, 计算 $\Psi = g^k h^\psi$ 。对每个 $1 \leq i \leq c$, TPA 选择随机数 $v_i \in Z_p$ 。将承诺 Ψ 和挑战信息 $chal = \{i, v_i\}_{1 \leq i \leq c}$ 发送给云服务器。
- 2.一旦收到 $(chal, \Psi)$, 云服务器最开始随机选择参数 $r, \rho_r, \rho_1, \dots, \rho_s \in Z_p$, 然后计算

$$\Pi = \prod_{(i, v_i) \in chal} \sigma_i^{v_i} \cdot h^r, T = \eta^{\rho_r} \eta_1^{\rho_1} \cdots \eta_s^{\rho_s}$$

之后将 (T, Π) 发送给TPA。

- 3.TPA 发送 (k, ψ) 给服务器。
- 4.服务器验证等式 $\Psi \stackrel{?}{=} g^k h^\psi$ 是否成立。如果等式不成立, 服务器就不再做后续处理, 或者, 服务器计算 $z_r = \rho_r - kr, \mu_j = \sum_{(i, v_i) \in chal} v_i m_{ij}, z_j = \rho_i - k \mu_i (1 \leq i \leq s)$ 并且发送 $(z_r, z_1, \dots, z_s, \alpha_1, \dots, \alpha_{l-1}, \beta_1, \dots, \beta_{l-1})$ 给TPA。

5.TPA通过验证以下等式是否成立，来验证文件标签 ft

$$e(g, t) = e(spk, H_0(t_0)).$$

如果验证失败，发送0，或者，TPA再验证以下等式是否成立。

$$\begin{cases} \left(\frac{e(\Pi, g)}{e\left(\prod_{(i, v_i) \in chal} H_1(Fn||i)^{v_i}, v\right)} \right)^k \stackrel{?}{=} \frac{T}{\eta^{z_r} \eta_1^{z_1} \cdots \eta_s^{z_s}}. \\ e(v_{l-1}, g) \stackrel{?}{=} e(g^{\alpha_l}, \beta_{l-1}). \\ e(v_j, g) \stackrel{?}{=} e(\alpha_{j+1}, \beta_j), j \in 1, 2, \dots, l-2. \end{cases}$$

4.4 安全性证明

在这一小节，我们将证明提出的方案满足正确性，完备性和零知识隐私，且密钥的更新和认证子的进化都是十分有效的。正确性证明了协议的顺利执行能够得到有效的结果，完备性保证了对不完全可信服务器的安全性。

4.4.1 正确性

在密钥被更换前，可以通过验证双线性对的性质直接得出协议是正确的。如下在密钥对更换之后，我们证明验证式依旧是正确的。如果用户和服务器都是诚实的，那么我们有，

$$\begin{aligned} t_l &= t^{\frac{ssk_l}{ssk_{l-1}}} \\ &= ((H_0(t_0))^{ssk_{l-1}})^{\frac{ssk_l}{ssk_{l-1}}} \\ &= H_0(t_0)^{ssk_l} \end{aligned}$$

因此， $e(g, t_l) = e(spk_l, H_0(t_0))$ 成立关于更新的认证子，我们有

$$\begin{aligned} e(v_{l-1}, g) &= e(g^{\alpha_{l-1}}, g) \\ &= e(g^{\alpha_l}, g^{\frac{\alpha_{l-1}}{\alpha_l}}) \\ &= e(pk_l, \beta_{l-1}) \end{aligned}$$

并且对所有的 $j \in 1, \dots, l - 1$

$$\begin{aligned} e(v_j, g) &= e(g^{\alpha_j}, g) \\ &= e(g^{\alpha_{j+1}}, g^{\frac{\alpha_j}{\alpha_{j+1}}}) \\ &= e(\alpha_{j+1}, \beta_j) \end{aligned}$$

因此，带有密钥更新和认证子进化的审计协议是正确的。

4.4.2 完备性

一个审计协议具有完备性如果任何不诚实的证明者能够说服一个验证者他存储着一个文件 F ，那么他真的存着那个文件。换句话说，存在一个提取算法能够从这个不诚实的验证者处提取一些文件块。协议的完整性证明是基于底层的 Shacham 和 Waters 协议^[9]的完备性和单向代理重签名的不可伪造性。确切地说，如果存在一个敌手能够攻破我们方案的完备性，那么我们就能够找到另外一个算法来攻破 Shacham 和 Waters 方案(SW 方案)的完备性和 Libert 和 Vergnaud 方案^[34](LV 方案)的不可伪造性。

证明：假定存在一个敌手 \mathcal{A} 能够攻破我们方案的完备性，那么我们可以构造一个模拟器 \mathcal{B} 来攻破 SW 方案的完备性和 LV 方案的不可伪造性。

1. 给 \mathcal{B} 关于 SW 方案的公钥。则 \mathcal{B} 能够通过这样的方式得到 h 的值：直接计算 g 的某一次方，而 g 是 SW 方案中公钥的一部分，是已知的， h 和 pk^* 中对应的元素作为 crs。只有公钥中的元素 (v, ssk) 被视为是公钥 pk^* 。 n 是 \mathcal{A} 进行更新询问的次数。 \mathcal{B} 选择一个随机的索引值 $\hat{i} \in \{1, \dots, n\}$ 并且设置 $pk_{\hat{i}} = g^a$ 。对于 $i \in \{1, \dots, n\} \setminus \{\hat{i}\}$ ， \mathcal{B} 选择一个随机的 sk_i 并且计算 $pk_i = g^{sk_i}$ 。 pk_1 作为系统第一个公钥，是 \mathcal{A} 已知的。

2.(除了时间片 \hat{i} 的询问)。对于不等于 \hat{i} 的时间片， \mathcal{B} 有私钥，因此能够回答敌手的所有询问，因此在此不做过多赘述。

3.($count = \hat{i} - 1$ 的更新询问)。当 $count = \hat{i} - 1$ 时，若 \mathcal{A} 调用更新请求， \mathcal{B} 没有相应的私钥 $sk_{\hat{i}}$ ，并且将对 SW 协议使用 AuthQuery 算法通过重计算所有的认证子来回答这个询问，并且重新产生 LV 方案签名询问的验证信息。值得注意的是，计算出的认证子的分布与更新询问中计算出来的一样。

4.($count = \hat{i}$ 的更新询问)。当 $count = \hat{i}$ 时，若 \mathcal{A} 调用更新请求， \mathcal{B} 将通过重新计算所有认证子的方式重新计算所有的私钥 $sk_{\hat{i}+1}$ ，并且通过 $pk_{\hat{i}}$ 和 $sk_{\hat{i}+1}$ 产生验证信息。计算出的认证子的分布与更新询问中计算出来的一样。

5.(输出). 最终, \mathcal{A} 输出了一个 P_* . 以 $1/n$ 的概率, P_* 是在时间片 \hat{i} 输出的. 在其他时间 \mathcal{B} 终止。现在我们将证明 \mathcal{B} 是如何输出一个 P' 的。

6.(P' 的构造). 在 P_* 的一次执行中, \mathcal{B} 获得 $(T, \Pi, k, z_r, z_1, \dots, z_s)$. \mathcal{B} 将返回步骤三并重新回复一个不同的值 k' . 这种可能性是存在的, 因为 \mathcal{B} 知道底为 g 的离散对数 h , 所以可以回复对应的值 ψ' . 算法以不同的副本 $(T, \Pi, k', z'_r, z'_1, \dots, z'_s)$ 结束。从这两个等式中, \mathcal{B} 能够计算 (r, μ_1, \dots, μ_s) . \mathcal{B} 计算 $\sigma = \Pi/h^r$ 并且 P' 代表输出 $(\sigma, \mu_1, \dots, \mu_s)$. 值得注意的是, 如果 P_* 是 ϵ -可接受的, 那么 P' 对于底层的 SW 方案是 ϵ^2 -可接受的, 对 LV 方案是 $\epsilon/(q_s + q_{rs} + 1)$ 可接受的, 这里 q_s 和 q_{rs} 分别代表签名和重签名的询问次数。换言之, 如果 SW 方案的完备的, LV 方案是不可伪造的, 那么我们的方案也是完备的。

4.4.3 零知识隐私

这个新的协议取得了零知识的隐私性, 也就是说, 在数据审计过程中不泄露外包数据的任何信息。为了证明这一性质, 我们在数据所有者和与服务器之间构建了一个模拟器 \mathcal{S} 。证明: 我们展示了敌手 \mathcal{A} 能够以非常接近 $1/2$ 的概率来成功的输出对比特的猜测。我们的思路是阐明如何对 \mathcal{A} 产生一个挑战而又与 F_0 和 F_2 无关的。回顾一下, 这两个文件有着相同的长度并且文件名均为 f_n , 因为 TPA 应该知道文件名和文件的块数。为了模拟一个对挑战文件的证明询问, \mathcal{S} 需要严格遵守这个协议, 直到第三步, 当收到值 (k, ψ) 时。 \mathcal{S} 将 \mathcal{A} 倒回第二步并选择一个随机的 $\Pi, z_r, z_1, \dots, z_s$ 计算

$$T = \left(\frac{e(\Pi, g)}{e\left(\prod_{(i, v_i) \in chal} H_1(Fn || i)^{v_i}, v\right)} \right)^k \eta^{z_r} \eta_1^{z_1} \cdots \eta_s^{z_s}.$$

\mathcal{S} 发送 T, Π 给 \mathcal{A} 。现在 \mathcal{A} 返回相同的 (k, ψ) , 因为这一对已经被它的第一个消息 Ψ 固定了。 \mathcal{S} 用 (z_r, z_1, \dots, z_s) 来回复。我们可以看到, 能够通过验证的值是被正确地分布着的。除此之外, \mathcal{A} 的视角跟 F_0 和 F_1 是无关的, 所以 \mathcal{A} 能够回答成功的概率一定是 $1/2$ 。

4.5 性能分析和实现

在本节中, 首先对本方案的通信开销、计算开销和存储开销进行分析, 然后描述实验结果。

4.5.1 参数选择

安全参数 λ 的选择是 80。由于本章方案是公开可验证的, p 是一个 $2k = 160$ 比特的素数, 并且选择椭圆曲线以使离散对数为 2^k -secure, 对于 λ 最高到 128, 由于 Barreto 和 Naehrig^[35] 对友好素数阶的椭圆曲线可以被使用。 $n \gg k$ 表示的是一个文件中数据块的数量, 关于关于擦除码的选择, 我们遵循 SW 方案。传统的 RS 类型的擦除码可以应用于实现的构造, 但编码解码过程需要 $O(n^2)$ 的时间。对于可信的服务器, 可以使用系统代码, 其中编码文件的第一个 m 块是文件本身, 可以用于更高效的公共检索。

4.5.2 复杂度分析

通信开销 在证明阶段, TPA 发送 Φ 和 $chal$ 到云服务器, 二进制长度为 $\log_2 c + (c+1)\log_2 p$ 。我们可以通过选择一个伪随机排列来计算挑战块的位置 i 和一个伪随机函数来计算随机挑战值 v_i 从而大大缩短挑战长度。在这种情况下, TPA 仅仅发送伪随机排列和伪随机函数的密钥, 长度为 $\log_2 p$ 比特每个。在第二阶段, 云服务器返回两个椭圆曲线上的点, T 和 Π 给 TPA, 长度为 320 比特。在第三阶段, TPA 发送 (k, ϕ) 给服务器, 二进制长度为 $2\log_2 p$ 。在下一阶段, 云服务器发送 (z_r, z_1, \dots, z_s) 给 TPA, 它的二进制长度为 $(s+1)\log_2 p$ 。

存储开销 在方案的存储开销方面, 由于公开可验证, 所有的文件和相对应的元数据包括文件的标签和块认证器需要存储到服务器。我们的方案具有存储和通信之间的灵活权衡的优势在 [9] 中提到, 也就是说, 使用参数 s 来给出响应长度和存储开销之间的折衷。每个块由 Z_p 中元素的 s 组成, s 称为扇区。每个块都有一个认证器, 将存储开销降低到 $(1 + \frac{1}{s}) \times$ 。用户仅仅需要存储公钥 $pk = (spk, v)$, 和私钥 $sk = (\alpha, ssk)$, 所以用户的存储开销大致为 $2\log_2 p + 320$ 比特, TPA 需要存储用户的公钥, 长度为 320 比特。在审计的过程, TPA 存储 $k, \phi, chal, \Phi, T, z_r, z_1, \dots, z_s$ 用来验证服务器的响应, 它的二进制长度总共为 $(c+s+3)\log_2 p + 320$ 比特。

计算开销 下面将从系统模型中的三个实体分析方案的计算开销。仅仅考虑计算耗时的操作包括双线性对映射, 指数运算, 以及群 G 和 G_T 乘法运算。 T_{pair} 表示的是计算椭圆曲线两点的双线性映射的时间开销, 忽略一些高效的计算操作, 例如, 计算一个哈希函数。 T_{expg} 和 T_{expgt} 表示在 G 和 G_T 中指数操作的计算开销, T_{mulg} 和 T_{mulgt} 表示在 G 和 G_T 上乘法操作的计算开销。

用户的主要计算就是为文件块生成认证器 $\sigma_i = (H_1(Fn||i) \cdot \prod_{j=1}^s u_j^{m_{ij}})^\alpha$, 对于每一个认证器计算开销是 $(s+1)T_{expg} + sT_{mulp}$ 。考虑到认证器的更新, 用户需要在 G 中完成 $n+1$ 个指数运算, 时间开销为 $(n+1)T_{expg}$ 。因此, 在方案中用户

主要的计算开销为 $(sn + 2n + 1)T_{expg} + nsT_{mulp}$ 。为了生成和验证一个证明, TPA需要计算 Φ 以及验证文件标签和响应, 所有 TPA总共的计算开销为 $(4T_{pair} + (c + 2)T_{expg} + (s + 2)T_{expgt})$ 。为了生成证明服务器需要计算 Π, T 和 z_r, z_1, \dots, z_s , 服务器的主要计算开销为 $(c + 2)T_{expg} + (c + 1)T_{mulg} + (s + 1)T_{expgt} + sT_{mulgt}$ 。

4.5.3 实验结果

方案的实现使用JPBC 2.0.0^[36]和PBC包装包^[37], 在处理器为 Intel(R) Core(TM) i5-6200U CPU @ 2.3GHz的电脑运行得到测试结果的, 内存总是足够的, 该方案实验的源代码在[38]可以看到。我们使用参数 a.param, pbc库的标准参数设置之一, 参数a.param在所有默认参数之间提供最快速度的对称配对, 协议的实现由以下四个部分组成。 第一部分包括协议的离线阶段, 包括公共参考字符串生成、密钥生成和认证器生成。在实验中, 我们假设一个大小不变的1MB的文件, 该方案要求每个扇区是 \mathbb{Z}_p 中的元素, 大小是160比特, 或20字节, 扇区总数 $(n \times s)$ 约为500,000。在这种情况下, 唯一的剩余变量是块数(n), 它也控制每个块中扇区数(s)。我们将块的数量从10增加到1000, 每一步的增量为10。注意, 在密钥生成阶段, 不包括计算 η_i 的时间, 因为用户不需要自己生成这些值。

图4-2, 4-3, 4-4展示了实验结果。如图所示, crs的生成时间和 s 是线性关系, s 是每个块的扇区数。其余的计算几乎是稳定的, 不管块的数量有多大, 这是根据经验分析来的, 因为在生成crs时, 计算开销操作就是生成 $u_1, u_2 \dots, u_s \in G$, s 是一个块中扇区的数量。对一个固定大小的文件来说, 随着数据块数量的增加, 每个块的扇区数是减少的, 这导致生成的 u_i , 因此生成crs的时间是减少的。在 Key-Gen中, 很明显, 只涉及到两个指数, 这会时间开销是常数级的。最昂贵的部分是为文件生成认证子。我们可以看到1MB的文件, 计算文件的认证子大约需要70秒。这是合理的, 因为对于具有固定大小的文件, 扇区数也是固定的。因此, 整个文件的主要计算 $\prod_{j=1}^s u_j^{m_{ij}}$ 几乎是不变的。但是随着文件大小的增加, 时间的增加几乎是线性的, 因为扇区数与文件大小之间存在线性关系。

表 4-1 每个算法的时间开销

参数			离线计算时间			在线计算时间	
s	n	c	crsGen	KeyGen	AuthenGen	TPA load	Server load
83	600	1% n	144.0ms	3.3ms	68.8s	47ms	47.1ms
166	300	5% n	284.0ms	3.4ms	67.6s	99.1ms	97.8ms

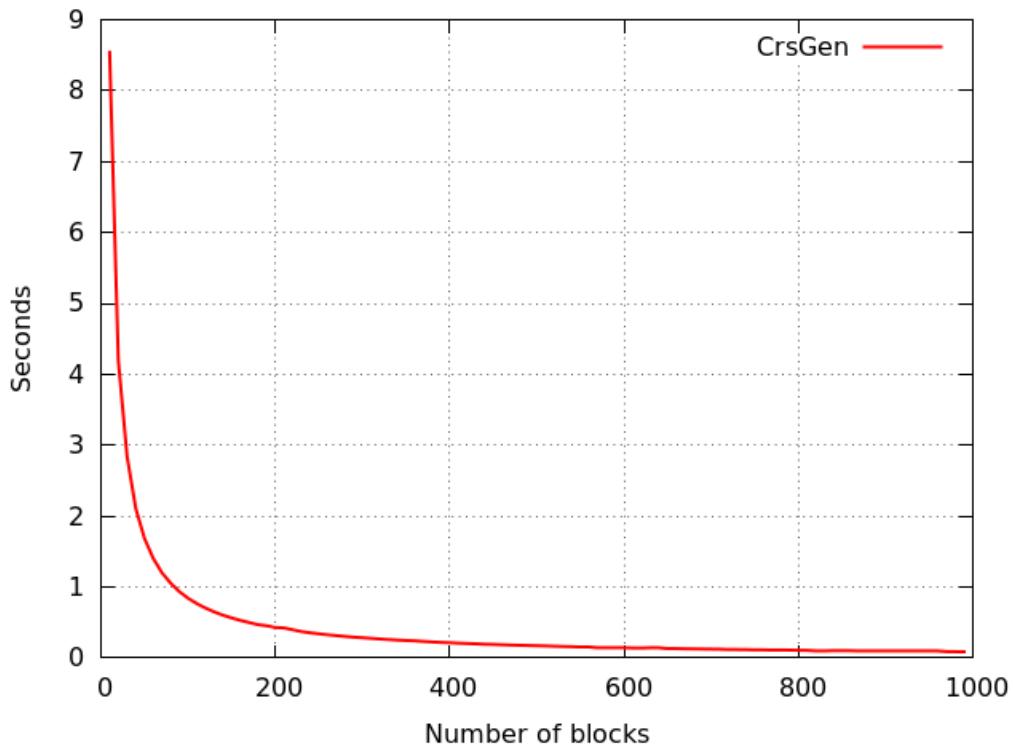


图 4-2 随着块数量的增加crs的生成时间

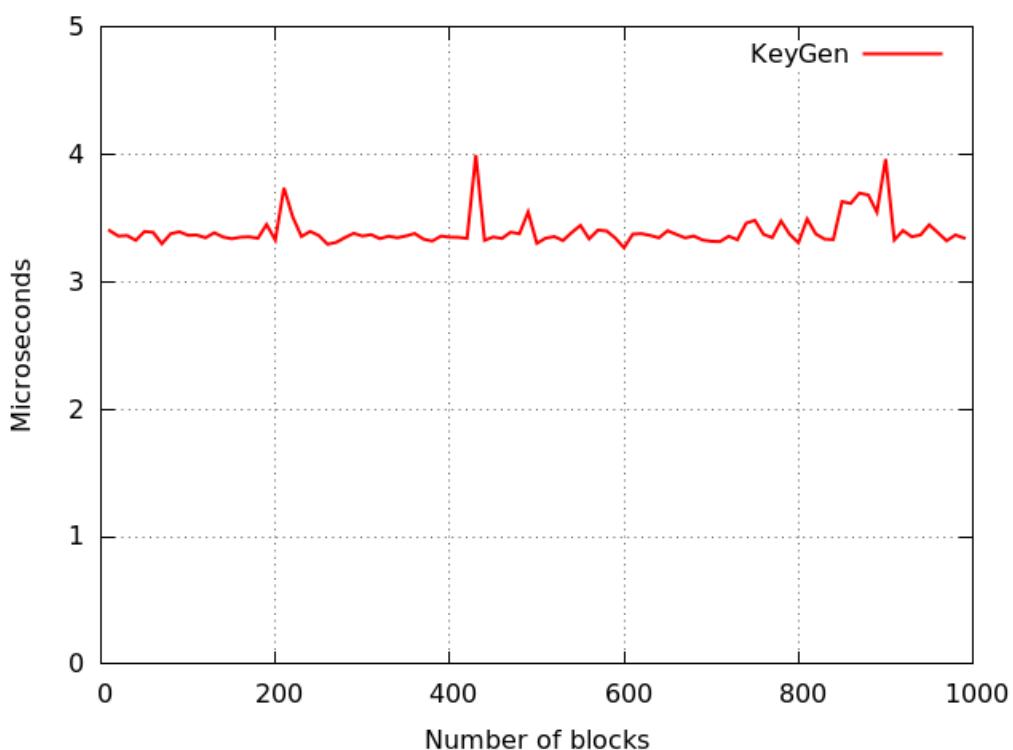


图 4-3 随着块数量的增加密钥生成的时间

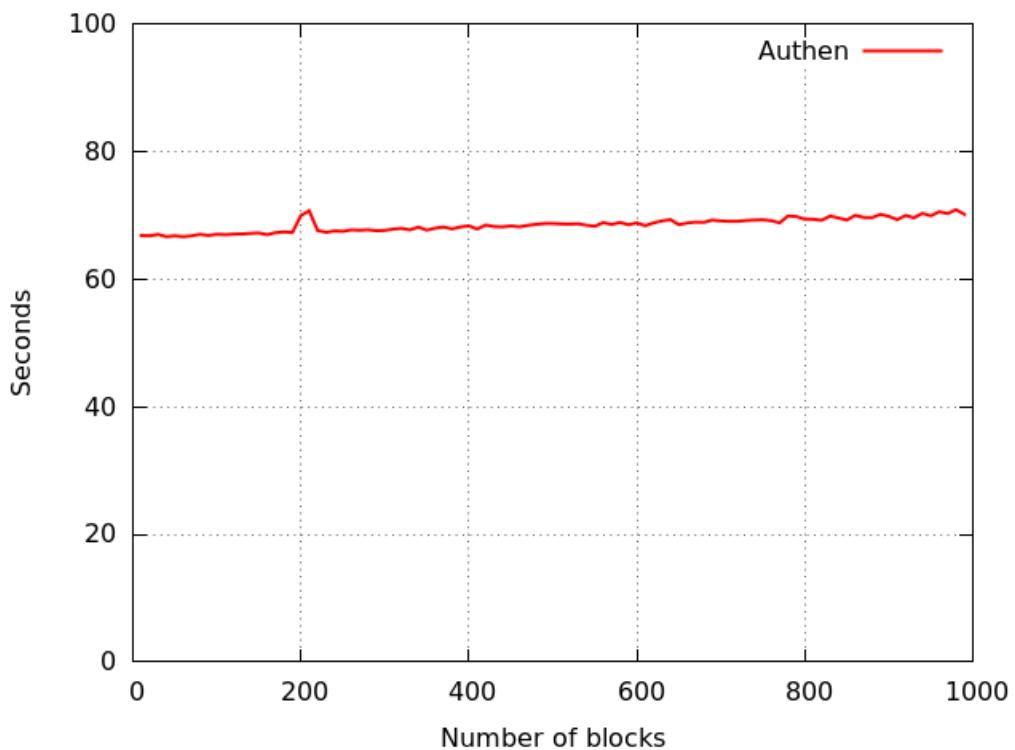


图 4-4 随着块数量的增加认证标签的生成时间

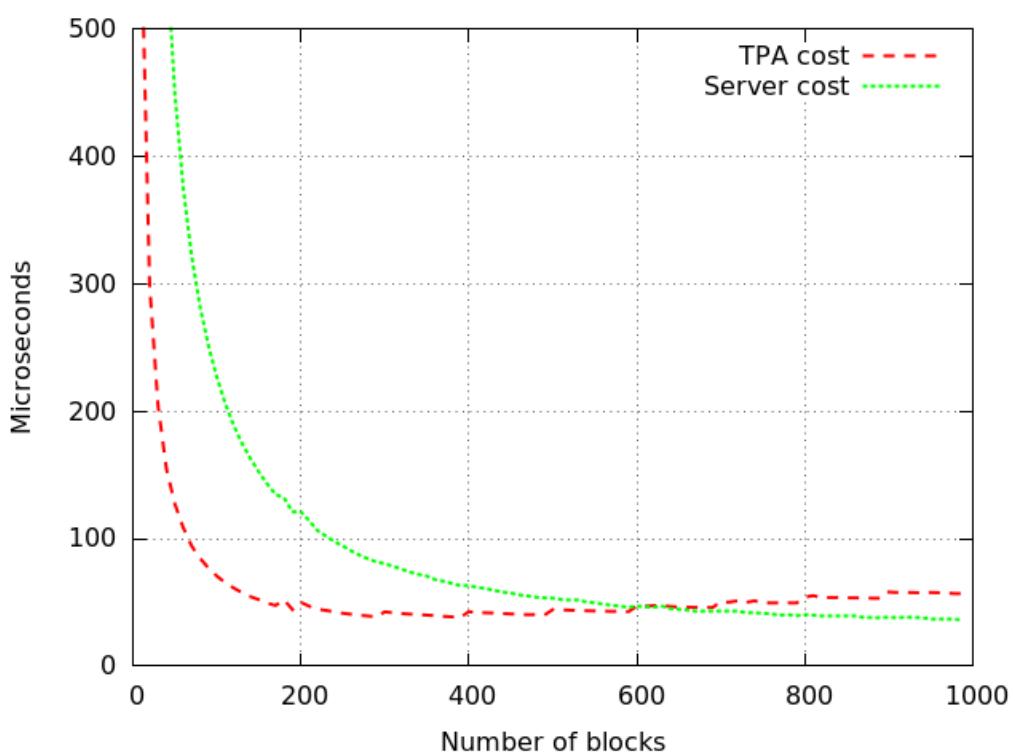


图 4-5 判断最佳的块大小

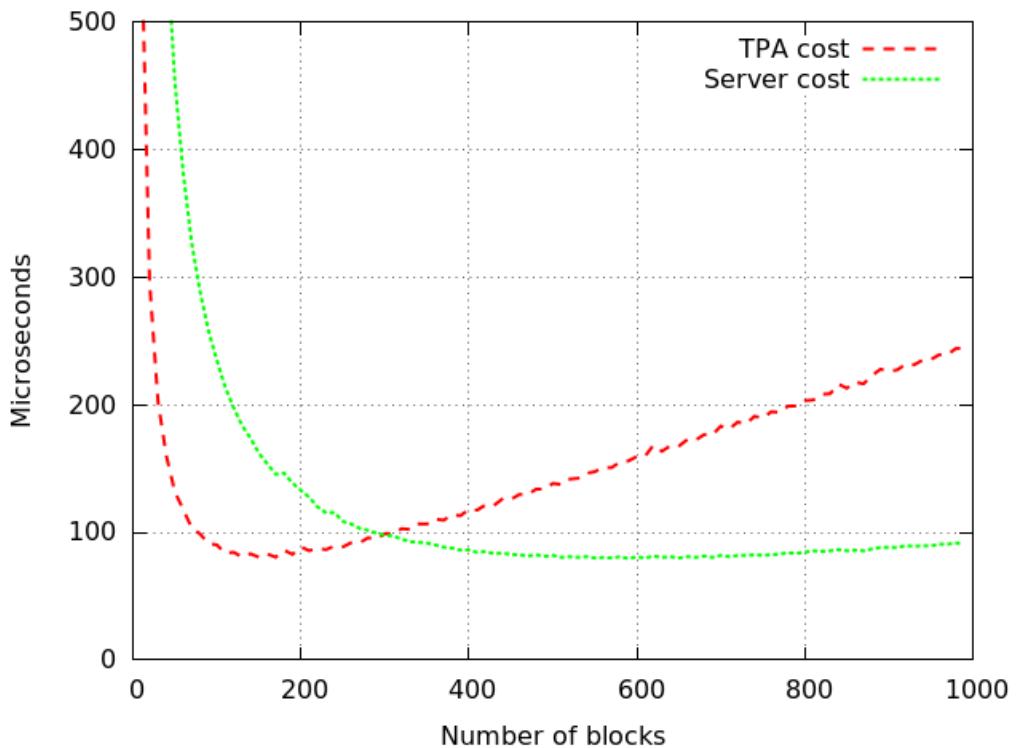


图 4-6 判断最佳的块大小

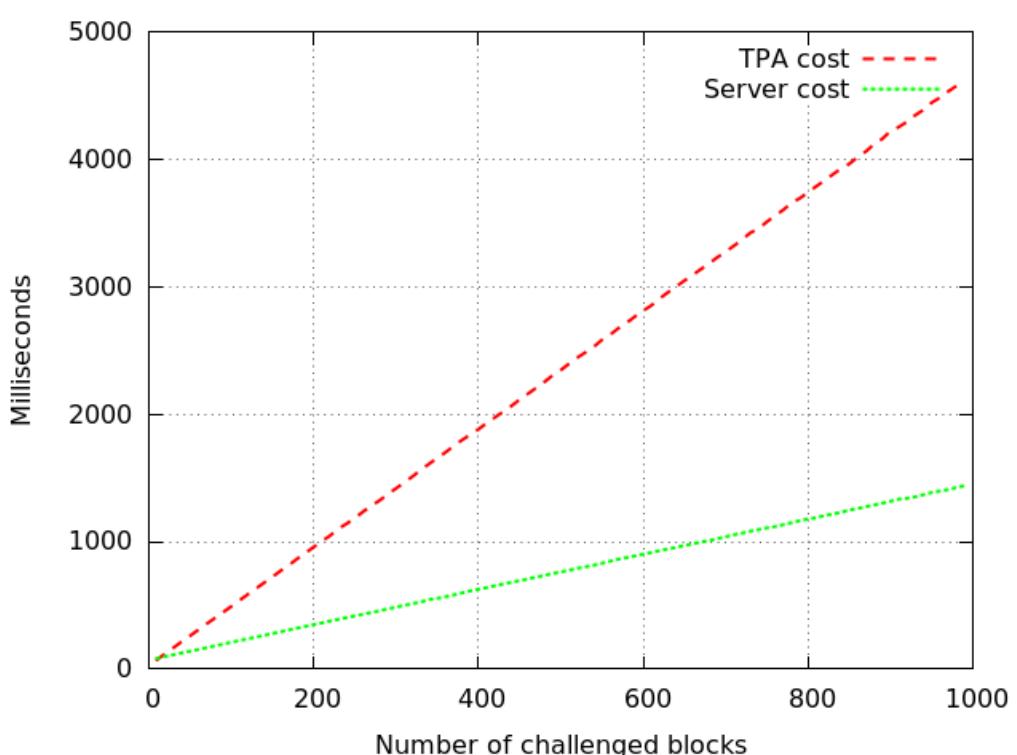


图 4-7 随着挑战块的增加TPA和Server的计算时间的开销

表 4-2 本文方案和其他方案的对比

	简单方法	Yu.[28]	本文方案
通信开销	1 MB	< 20 KB	160 bits
认证子更新时间	~ 70 s	< 1 s	5l ms
*l 是密钥更新的次数.			

实验的第二部分，对给定挑战我们尝试找到最佳块数(n)。我们将文件大小固定为1MB，从块数量的1%和5%设置挑战块数。挑战块的数量足以通过随机抽查来检测是否损坏。Ateniese等在^[7]中列出了一个例子，10,000块的文件，如果该文件存在1%的块已损坏，那么TPA可以通过仅挑战460个块，99%的概率能检测到文件被损坏。在这种情况下，被挑战的块只占整个文件的4.6%。

服务器需要执行大约 $(s + c)$ 个指针操作在证明阶段的步骤2中。在图4-7，可以看到随着块数的增加，服务器的负载由于扇区(s)数减少而急剧下降，然后随着挑战块的增加再次增加。因此，我们选择服务器和TPA的开销优化的最佳点，很明显，当对文件的所有块的1%进行挑战时，文件大小为1 MB的块的最佳数量达到近620，这意味着每个块应该接近1.6，然而当所有块的5%作为挑战块，文件块的最佳数量达到3.3KB。与此同时，当所有块的5%受到挑战时，服务器的不当行为可能被捕获的可能性非常高。因此，我们建议在实践中块大小可以是3KB，以尽量减少服务器的时间开销和TPA的工作量。在这种情况下，每个算法的时间开销总结在表4-1有显示。

确定最佳块大小后，第三部分的实验是测试验证协议的效率，用于频繁检查云数据的完整性。我们使用10,000个块的文件进行测试，每个块的大小为3KB，150个扇区。我们记录TPA和云服务器的时间开销，通过增加随机选择的挑战块的数量，从1到1000，结果如图4-4所示。可以发现Proof协议对于验证者和证明者都是非常高效的，例如，让我们观察曲线上的两点，Ateniese等^[6]表明，如果服务器已经损坏了1%的块，那么TPA可以挑战300个block和460个block，以达到检测概率至少为95%和99%。我们可以看到，TPA花费1.42秒，服务器0.48秒，当挑战为300块时，当挑战为460块时，TPA花费2.16秒，服务器0.7秒。随着挑战块的增加，双方的时间间接增加。这与经验分析一致，因为当挑战块的数量上升时，云服务器的计算量增加为 $\sigma_i^{v_i}$ ，而TPA在 $\prod_{(i,v_i) \in chal} H_1(Fn||i)^{v_i}$ 中具有更多的乘幂运算和乘法运算。

在实验的最后一部分，对比了本章提出的密钥更新方法与其他直接简单的密钥更新解决方案的效率，其他简单直接的密钥更新解决方案中，用户从云上下载

整个文件以及重新计算文件的认证子，这意味着用户必须再次执行 AuthenGen 算法。在文献[28]中，用户需要上传 n 个认证子，认证子是群 G 中的元素。在本文的实验中发现，选择 $n = 300$ ，而 $a.\text{param}$ 每个元素存储在 64 字节中，因此，[28] 的通信开销低于 20KB。然而，在我们的构造中，用户需要上传到服务器的是文件标签，它是群 G 中的一个元素。因此，我们的解决方案的通信成本只有 160 比特。在更新认证子时间开销方面，使用简单直接的方法，我们需要重新计算所有的认证子，这大约花费在 70 秒，如我们在实验的第一部分中所示。在[28]中，需要进行大约 n 次的指数运算，对于 $n = 300$ ，它需要不到 1 s 的时间。在本章的方案中，如果我们更新密钥对 l 次，需要执行 l 次对运算，每次花费约为 5ms，因此，本章解决方案中密钥更新的时间与更新次数呈线性关系。时间结果在表 4-2 中可以看到，本章提案的方案在计算和通信开销方面都比其他两个方案的要小的多。

4.6 本章小结

在 Yu 等^[28] 提出的支持密钥更新和零知识隐私保护的公开云数据审计方案基础上，本章在认证标签更新的过程中使用一个单向的代理重签名，即使用户发布了重签名密钥，也没有必要担心泄露新的密钥，在进行密钥更换时，用户只需将重签名的密钥给云服务器，然后更新自己的公钥和重签名密钥，其他的事都交由云服务器完成即可。此外，该协议还支持零知识隐私保护和公开可验证，本章给出了该协议的安全性证明，并通过实验数据展示改进协议的性能。

第五章 具备去重功能的云数据完整性检测协议

5.1 问题提出

云存储作为云计算提供的主要服务，允许用户远程备份本地数据到云端，之后通过网络随时随地访问和管理存储在服务器上的数据^[39]。现今，许许多多的团体和个体将数据存到云服务器上，大大缓解他们存储空间受限的状况，以便于他们不用花太多的时间在繁琐的数据存储和管理方面，从而专心于自身的核心要务。然而，一旦数据被上传到云中，用户也就对自己数据的所处物理环境失去了直接的控制。为了防止移动云存储的各种安全威胁，合乎标准规范的云服务提供商必须采取及时的措施来保证用户云端数据的安全，然而云服务提供商并不是完全可信的^[4]。例如，由于黑客或是攻击者对云服务器的攻击，从而导致数据的丢失；或在利益的驱使下，云服务器删除不被访问或者访问较少的数据，导致用户数据的丢失；服务器为了完成业务操作进行频繁地写数据和读数据的操作，导致磁盘被损坏的机率增加，进而发生接连的数据丢失事件。让情况更糟糕的是，云服务器为了维护自己所谓的名誉极可能对用户隐藏数据丢失事件。数据被泄露和丢失一直是云存储中最常发生的事故，因此，用户必须能时刻对外包到云的数据进行完整性检测，用户也就可以得到强有力的可以证明数据有无被丢失的证据，一旦发生数据丢失事件，用户能以此证据向云服务商进行索赔。

为了检验数据的完整性，在2007年，Ateniese等人^[6, 7]提出了 PDP机制，该机制使得用户在不下载整个文件的前提下，通过概率性抽查来验证外包数据的完整性，Ateniese等使用同态可验证标签技术提出了两个高效验证数据完整性并可证明安全的 PDP方案。随后，Juels等人^[8]提出了 PoR的机制，相比PDP该机制不仅具备PDP机制的特性，即完成数据的完整性检测，并且还提供数据恢复功能，通过生成用户可以恢复的简单证明来实现，随后他们使用纠错码提出了基于哨兵的PoR方案，在2008年，Shacham和 Waters^[9, 10] 在这基础上描述了两个高效且紧致的PoR方案。在2009年，Ateniese等^[40]提出了通过构建公钥同态线性认证子模型，将任意其认证子转变成一个公开可验证的 PDP方案，验证次数可以是无限的。接下来。在一些经典的 PDP和 PoR方案^[8-10, 41-43]的基础上，大量数据完整性检测协议^[44-47]被提出用来保证云数据的完整性。

存储到云服务器的数据是海量，且这些数据的重复率是非常高的，如果对这么多的重复数据都进行存储的话，在很大程度上加大了服务器的存储负担。所以云服务器希望对不同用户请求存储的相同文件只存储一份，来节约存储空间，降

低存储成本，也提高存储的效率。将去重技术应用到移动云存储中可以消除数据的重复副本，从而在存储中只保留一个数据副本，不仅可以减小服务器存储开销，还能显著地减少网络带宽消耗。最开始，云存储中的去重的实现都是基于采用计算数据的散列值进行对比的方法来实行，然而这种简单直接的数据去重回带来很大安全隐患。为了解决使用一个短小的Hash值来做为整个文件的”代理”而出现的安全问题，Halevi^[13]提出了所有权证明 (POW)的概念。

通过仔细观察，我们会发现，这有点类似于可恢复性证明 (PoR)和数据拥有证明 (PDP)，只是角色发生了颠倒而已，在 PDP方案中用户是证明者，而服务器是验证者。但是值得注意的是，之前提出的那些经典的数据完整性验证方案并不适用于POW机制中。如上所述，所有权证明与可恢复性证明 (PoR)和数据拥有证明 (PDP)是密切相关。具体来讲，它们的本质都是来验证数据的持有性证明。它们之间的主要差别是，可恢复性证明 (PoR)和数据拥有证明 (PDP)使用的预处理步骤不能用于所有权证明 (POW)，在可恢复性证明 (PoR)和数据拥有证明 (PDP)中，用户通过在编码后的数据文件中嵌入一些私密信息来做预处理，借助这些嵌入的秘密信息来验证外包到服务器上数据文件的完整性。在POW方案中，一个用户需要证明的是他就只是拥有原始文件本身，从而不能在文件中嵌入秘密信息来达到对文件所有权的证明。所以基于MAC或签名的解决方法并不能应用到所有权证明的方案中来，例如[6, 8, 9, 48]中提出的方案。目前，许多数据完整性验证方案被提出，但是却不适合应用到 POW方案中来验证用户对文件的所有权，那些被提出的POW方案却也不是适合应用到 PDP方案中用来实现数据完整检测。出于数据安全性和云服务器存储效率性的考虑，设计一种具备去重功能的云数据完整检测方案是十分有实用价值的。

5.2 问题描述

通过分析云存储环境中数据完整性检测和去重的关键性问题，提出解决该问题所需要的系统模型、系统组成和安全模型。

5.2.1 系统模型

一个云存储服务系统主要包括以下三个实体：云用户、云服务器、审计者，和第四章提出方案的系统的组成者是极其类似的。

1. 用户：用 C表示，使用云服务的用户，需要大的存储空间存储自己大量的数据文件，但是受自身设备固有缺陷性质，存储空间有限、计算能力有限、电池电源使用时间有限的限制，更是为了减轻本地存储的负担，将产生的数据备份存储到服务器上。

2. 云服务器：用 S 表示，拥有大量存储空间和计算资源，为云用户提供存储服务和计算资源，存储用户的数据并提供数据访问服务。
3. 审计者：用 $Auditor$ 表示，也称第三方审计者。他是一个可信的组织，替云用户检查外包到云上的数据的完整性。他具有移动用户不具有的专业技术能力和能力，能很好地提供数据审计服务。

5.2.2 系统组成

$F = (m_1, \dots, m_n)$ 表示一个文件由 n 块组成, $m_i \in Z_q^*$ 。 fid 是文件 F 的标识符, 且是唯一标识符。我们考虑下面两个标签, Tag_{int} 表示审计数据完整性的认证标签, Tag_{dup} 表示重复副本检查的认证标签。[] 表示一个函数或一个算法中可选择的参数, 例如, 算法 $\text{Algorithm}(a, b, [c])$ 意味着算法 Algorithm 有两个参数 a 和 b , 和第三个可选择的参数 c 。 ι 表示一个安全参数, 如果对任意的常数 $const$ 和足够大的 ι , 函数 $\epsilon(\iota)$ 都比 ι^{-const} 要小, 那么就认为 $\epsilon(\iota)$ 是可以忽略的。具备去重功能的隐私保护公开云审计协议由以下四个算法组成, 具体包括: 密钥生成 (KeyGen), 上传 (UploadStore), 审计 (AuditINT), 副本检查 (ReplicaVer)。

1. **KeyGen:** 这是密钥生成算法, 它将一个安全参数 λ 作为输入, 输出两对公钥私钥对 (PK_{int}, SK_{int}) 和 (PK_{dup}, SK_{dup}) 。 PK_{int} 是用户的公钥, SK_{int} 是相对应的私钥, 这对密钥用于完整性的验证。同样的, PK_{dup} 和 SK_{dup} 是服务器的密钥, 这对密钥用于实现安全的去重。
2. **UploadStore:** 这个协议在云用户 C 和云服务器 S 之间执行。假设 C 想上传一个新的文件 F 到云服务器 S 上, 如果服务器 S 上没有存储该文件, 用户 C 就用文件 F , F 的唯一标识符 fid 和自己的私钥 SK_{int} 生成一个可以用于文件 F 完整性审计的认证标签 Tag_{int} , 并将 (fid, F, Tag_{int}) 上传。服务器存储 (fid, F, Tag_{int}) 和可能用于重复副本检查的 Tag_{dup} 以及相对应的私钥 SK_{dup} 。服务器可能还会保存文件 F 的 $Hash$ 值以方便用来数据文件的重复检查。
3. **AuditorINT:** 这是数据完整性验证的算法, 它在服务器 S 和审计者 $Auditor$ 之间运行。审计者 $Auditor$ 用文件的标识符 fid 和对应用户的公钥 PK_{int} 产生一个挑战 $chal$, 随后将挑战发给服务器。服务器 S 用存储在本地的文件 F 、文件标识符 fid 、文件 F 的认证标签 Tag_{int} 和收到的挑战值 $chal$ 生成一个响应 $resp$ 。如果 $resp$ 是根据挑战 $chal$ 和其他相关信息产生的有效值, 审计者 $Auditor$ 输出 1, 表示文件 F 的完整性是被确保的, 0 表示完整性是不被

保证的。可以用如下的形式表示:

$$b \leftarrow (Auditor(fid, PK_{int}) \iff S(fid, F, Tag_{int})), b \in \{0, 1\}$$

4.ReplicaVer: 这是重复文件验证的算法，在云服务器 S和云用户 C之间执行。一个云用户声明他拥有文件 F，由于副本的检测可以通过 C发送该数据文件的 Hash值给云服务器 S来实现，S可以判断这份文件是否已经被自己存储了。这个算法和 AuditorINT很是相似，具体实现是，云服务器发送一个挑战 *chal*给云用户 C，云用户利用自己拥有的文件 F和可能用到的其他的信息生成一个响应 *resp*给服务器 S。S使用自己的 *Tag_{dup}*和 *PK_{dup}*来验证 *resp*是否是有效的，如果是有效的，就输出1，意味着云用户 C的确拥有文件 F,或者输出0。可以用如下的形式来表示:

$$b \leftarrow (S(fid, Tag_{dup}, [SK_{dup},]PK_{dup}) \iff C(fid, F)), b \in \{0, 1\}$$

5.2.3 安全模型

我们使用游戏的形式来定义我们方案的安全性，明确规定了敌手的行为(如，允许敌手做什么)和获胜的条件(如，在什么情况下我们就说攻击是成功的)。因此，我们的方案必须具备正确性的和完备性。正确性就意味着，如果证据 P的产生是源自合法的证明者的有效的数据块和标签，那么 AuditorINT算法的输出值为0的概率就是可以忽略的了。完备性^[9]表示的是，如果 AuditorINT算法输出的是1，那么证明者就确确实实一定是完整的存储了文件的。这被形式化地定义为，存在一个提取器能够通过与服务器的交互使用auditint协议提取文件。具体的是，提取器算法 *Extr*(*fid*, *PK_{int}*, *P*)将 *fid*, *PK_{int}*以及在 AuditorINT协议中实现证明者 prover角色的交互式图灵机的描述作为输入，这个提取算法的输出就是文件 *F*。在这模型中，我们提出了一个称为完备性的安全概念，它和文献[6]中的数据拥有性游戏定义的安全性概念相似。我们的方案是应该 (κ, θ) -uncheatable^[18]。下面是敌手 (*A*)和模拟器之间的交互游戏，我们称这个游戏为 setup 游戏。

1. 模拟器执行 KeyGen算法生成密钥对 (PK_{int}, SK_{int}) 和 (PK_{dup}, SK_{dup}) ，其中 PK_{int} 和 PK_{dup} 是公开的，将 SK_{int} 给相应的用户以及将 SK_{int} 给敌手 (*A*)。
2. 敌手同模拟器进行多次适应性的 KeyGen和 UploadStore询问。敌手 (*A*)可以询问其他任何用户的私钥(这些用户可能和 (*A*)串通或是被 (*A*)攻击了)，除了当前用户的。模拟器运行算法 KeyGen计算出相应的私钥 SK_{int} 给敌手，敌手可以对一个存储的预言机进行询问，例如，对每次询问的一些文件，模拟器运行 UploadStore算法，并把算法的输出 (F, Tag_{int}, fid) 给敌手。

3. 对于已经进行了 UploadStore 询问的文件 F , 敌手可以通过指定文件的 fid 来得到 AuditINT 协议的执行。模拟器扮演验证者 (Auditor/verifier) 的角色, 敌手扮演证明者 prover 的角色: $\text{Auditor}(\text{fid}, \text{PK}_{\text{int}}) \rightleftharpoons A$ 。当执行完这一协议, 敌手可以获得审计者的输出。
4. 最后, 敌手选择一个文件 F^\dagger , 存在着一个用 F^\dagger 作为输入的 UploadStore 询问, 它的输出是 $\text{Tag}_{\text{int}}^\dagger$ 和 fid^\dagger 。敌手要输出一个证明 P^\dagger 的描述, 下面给出了证明 P^\dagger 是 ϵ -admissible 的定义。

我们说这个假的证明 P^\dagger 是 ϵ -admissible, 如果它能令人信服的回答一个 ϵ , ϵ 是验证挑战的一部分, 即如果 $\Pr[(\text{Auditor}(\text{fid}^\dagger, \text{PK}_{\text{int}}) \rightleftharpoons P^\dagger) = 1] \geq \epsilon$ 。这个概率比验证者 verifier 和证明者 prover 的猜测概率要大。

如果存在一个提取算法 Extr , 对所有的敌手 (A), 每当 (A) 玩 setup 游戏, 输出一个对应文件 F^\dagger 的 ϵ -admissible 假证明 P^\dagger , 提取算法 Extr 从证明 P^\dagger 中恢复出文件 F^\dagger , 即 $\text{Extr}(\text{fid}^\dagger, \text{PK}_{\text{int}}, P^\dagger) = F$ 的概率是不可忽略的, 我们的方案就被称为是 ϵ -sound。直观的来说, 如果给定一个具有最小熵的文件 F , 不存在一个恶意的用户, 他可以找到包含文件 F 的 θ -bit 香农熵的另一个文件 F' , 然后他让服务器相信他真的拥有文件 F 的概率超过 $2^{-(\kappa-\theta)}$, 是不可忽略的, 我们就认为我们的方案是 (κ, θ) uncheatable。具体的, 我们下面来看敌手 (A) 和挑战者之间的游戏, 其中敌手 (A) 扮演不怀好意的用户角色, 挑战者扮演服务器的角色。

1. 运行算法 KeyGen 来生成密钥对 $(\text{PK}_{\text{int}}, \text{SK}_{\text{int}})$ 和 $(\text{PK}_{\text{dup}}, \text{SK}_{\text{dup}})$, 其中 PK_{int} 和 PK_{dup} 是公开的, 将相应的 SK_{int} 给敌手 (A), 然而, 一个忠实用户相应的 SK_{int} 和 SK_{dup} 仅仅只给挑战者, 敌手是不知道的。
2. 挑战者选择一个具有 κ -bit 最小熵的文件 F , 文件的唯一标识符为 fid 。挑战者执行 ReplicaVer 算法, 并将公开可观的信息给敌手 (A)。
3. 敌手 (A) 和挑战者进行交互式的询问, 想从询问过程中获得文件 F 的内容: $S(\text{fid}, \text{Tag}_{\text{dup}}, [\text{SK}_{\text{dup}},] \text{PK}_{\text{dup}}) \rightleftharpoons A$ 。敌手 (A) 也许能通过其他的方法渗透到云服务器从而来获得文件 F 的部分信息, 但是不能推断出 κ 的大小。假设敌手 (A) 获得了文件 F 的 θ -bit 香农熵。
4. 最后, 敌手 (A) 输出一些文件 F' , 只要其中一个文件能满足如下式子, 就认为敌手 (A) 胜利。

$$1 \leftarrow (S(\text{fid}, \text{Tag}_{\text{dup}}, [\text{SK}_{\text{dup}},] \text{PK}_{\text{dup}}) \iff C(\text{fid}, F'))$$

如果对任意一个多项式时间算法的敌手 (A), 他获胜的概率大于 $2^{-(\kappa-\theta)}$, 是可忽略的, 我们就说我们的方案是 (κ, θ) uncheatable。要注意的是 $\kappa - \theta \geq \iota$ 将是常见的

情况，因为我们主要处理大的数据文件，因此 A 的获胜概率在 ν 中要求是可以忽略的。

5.3 具备去重功能的云数据完整性检测方案描述

5.3.1 方案描述

具备去重功能的云数据完整性检测方案的具体构造如下：

1. *KeyGen*: 这是密钥生成算法，详细过程如下：

- G 和 G_T 是阶为 p 的乘法循环群， $\hat{e}: G \times G \rightarrow G_T$ 是一个双线性对， g 为群 G 的生成元，Hash 函数 $H: \{0,1\}^* \rightarrow G$ ，系统参数分别是 $(g, H, p, G, G_T, \hat{e})$ 。云用户选择一个随机数 $X \in G \setminus \{1\}$, $r \in Z_p^*$ 。计算 $R = g^{-r}$ 和 $A = \hat{e}(X, g)$ 。
- 让 $PK_{int} = (R, A)$ 以及用户的私钥为 $SK_{int} = (r, X)$ 。
- 让 $PK_{dup} = PK_{int}$, $SK_{dup} = null$, 其中 PK_{dup} 是公开的。

2. *UploadStore*: 这一协议在云用户和云服务器之间执行，具体过程如下：

- 对每一块数据块 $m_i, i \in [1, n]$, 用户使用 SK_{int} 去计算：

$$T_i = X^{m_i} H(fid \parallel i)^r \in G$$

用户发送 (fid, F, Tag_{int}) 给服务器，其中 $Tag_{int} = \{T_i\}_{i \in [1, n]}$ 。

- 服务器根据收到的 (fid, F, Tag_{int}) , 设置 Tag_{dup} 的值, $Tag_{dup} = Tag_{int}$ 。

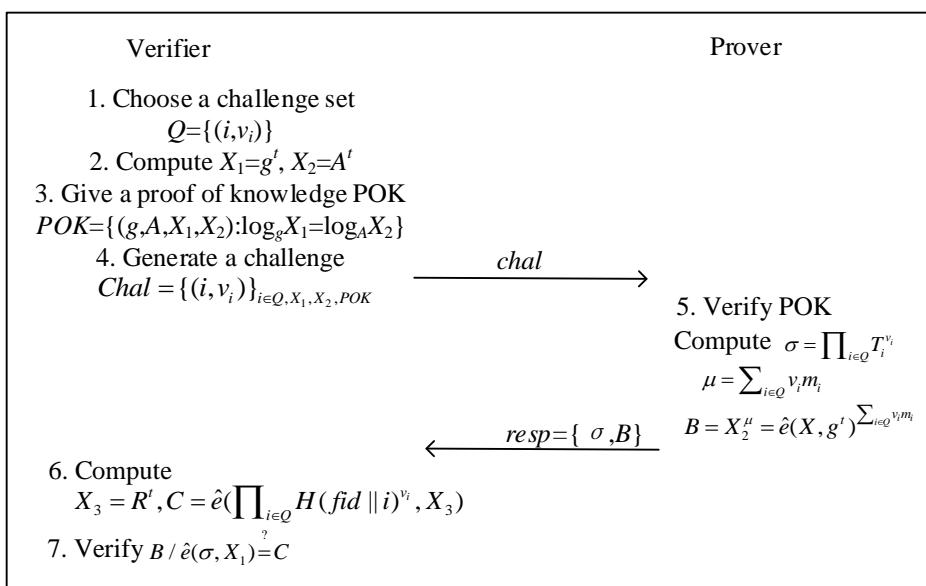


图 5-1 审计者(verifier)和服务器(prover)之间执行 AuditorINT 协议过程

3. *AuditorINT*: 这一协议在审计者和云服务器之间执行，当然也可以是在云用户和服务器之间执行，用来验证用户存储到云服务器的数据文件的完整

性，图4-3做了详细的阐述。如果用户是委托第三方审计者来验证数据的完整性，用户不需要给任何的信息给审计者，除了用户公开的公钥 PK_{int} 和文件的标识符 fid 。

- 为了生成审计时用的挑战 $Chal$ ，审计者首先挑选一个随机的 $t \in Z_p^*$ ，然后选择一些数据块来组成一个挑战，用 Q 来表示，以及为每个 $m_i, (i \in Q)$ 选择一个随机数 $v_i \in Z_p^*$ ，使用[]中的方法计算：

$$X_1 = g^t$$

$$X_2 = A^t$$

审计者必须给一个知识证明， X_1 和 X_2 具有相同离散对数，对应于 g 和一个 $POK\{(g, A, X_1, X_2) : \log_g X_1 = \log_A X_2\}$ 。给云服务器的挑战值是 $Chal = \{(i, v_i)_{i \in Q}, X_1, X_2, POK\}$ 。

- 收到审计者的挑战 $Chal$ 后，云服务器最开始验证 POK 是否有效，如果是无效的，审计失败；否则，服务器计算：

$$\begin{aligned} \sigma &= \prod_{i \in Q} T_i^{v_i} \\ \mu &= \sum_{i \in Q} v_i m_i \\ B &= X_2^\mu = \hat{e}(X, g^t)^{\sum_{i \in Q} v_i m_i} \end{aligned}$$

通过上面的计算之后，服务器发送回应 $resp = \{\sigma, B\}$ 给审计者。

- 审计者收到 $resp$ ，解析 $resp = \{\sigma, B\}$ 并计算

$$\begin{aligned} X_3 &= R^t \\ C &= \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, X_3\right) \end{aligned}$$

以及验证

$$B / \hat{e}(\sigma, X_1) \stackrel{?}{=} C$$

如果上面的等式成立，就将1返回给云用户；或者返回0。

4. *ReplicaVer*: 该协议在云用户和服务器之间执行，云用户声明他拥有标识符为 fid 的文件 F ，而该文件已经被其他的用户上传到云服务器上了，图4-4做了详细的阐述。该协议是上面的 *AuditorINT* 协议的一个简单的变形，在该协议实施的过程中，审计者只需要知道用户的公钥 PK_{int} 和文件的表示符 fid 。我们让服务器充当审计者的角色，用户就成了证明者，在该算法

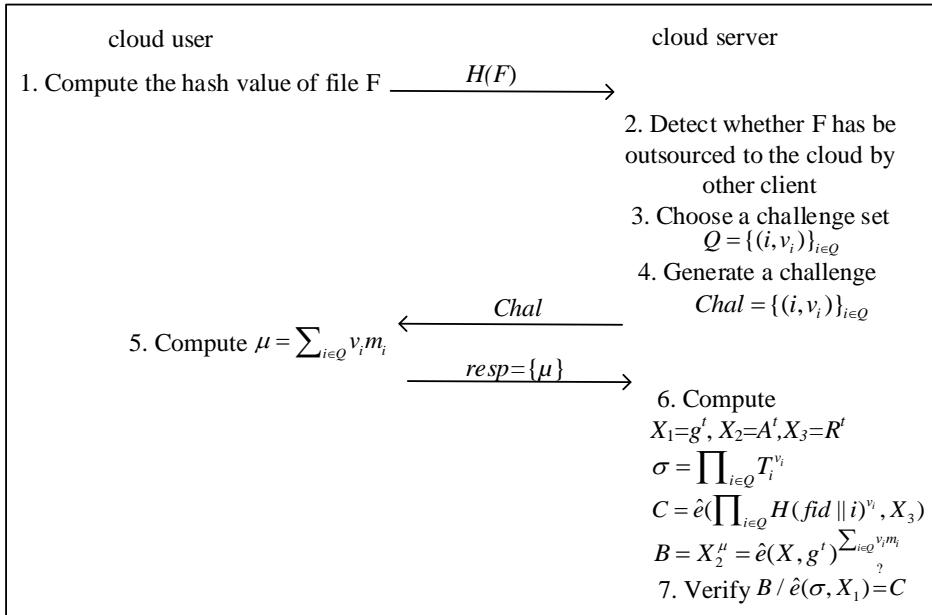


图 5-2 用户(prover)和服务器(verifier)之间执行的ReplicaVer协议过程

中做了一些调整，因为在产生 $resp$ 过程中，有些信息是用户不知道的。

- 云服务器首先选择随机数 $t \in Z_p^*$ ，然后选择一些数据块来组成一个挑战，用 Q 来表示，以及为每个 $m_i, (i \in Q)$ 选择一个随机数 $v_i \in Z_p^*$ ，服务器发送 $Chal = \{(i, v_i)_{i \in Q}\}$ 给用户。
- 用户计算

$$\mu = \sum_{i \in Q} v_i m_i$$

并发送 $resp = \{\mu\}$ 给服务器。

- 服务器选择随机数 $t \in Z_p^*$ ，从 $Tag_{dup} = \{T_i\}_{i \in [1, n]}$ 中计算下面的值：

$$\begin{aligned}
 X_1 &= g^t \\
 X_2 &= A^t \\
 X_3 &= R^t \\
 C &= \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, X_3\right) \\
 B &= X_2^\mu = \hat{e}(X, g^t)^{\sum_{i \in Q} v_i m_i}
 \end{aligned}$$

服务器验证

$$B / \hat{e}(\sigma, X_1) \stackrel{?}{=} C$$

如果上面的等式成立，就将1返回给云用户；或者返回0。

具备去重功能的云数据完整性检测方案的正确性验证如下：

$$\begin{aligned}
 & B/\hat{e}(\sigma, X_1) \\
 &= \hat{e}(X, g^t)^{\sum_{i \in Q} v_i m_i} / \hat{e}(\sigma, g^t) \\
 &= \hat{e}(X^{\sum_{i \in Q} v_i m_i}, g^t) / \hat{e}\left(\prod_{i \in Q} T_i^{v_i}, g^t\right) \\
 &= \hat{e}\left(\prod_{i \in Q} X^{v_i m_i}, g^t\right) / \hat{e}\left(\prod_{i \in Q} X^{m_i v_i} H(fid \parallel i)^{rv_i}, g^t\right) \\
 &= \hat{e}\left(\prod_{i \in Q} X^{v_i m_i}, g^t\right) / \hat{e}\left(\prod_{i \in Q} X^{m_i v_i}, g^t\right) \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{rv_i}, g^t\right) \\
 &= 1 / \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{rv_i}, g^t\right) \\
 &= 1 / \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, g^{tr}\right) \\
 &= \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, g^{-r}\right)^t \\
 &= \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, R\right)^t \\
 &= \hat{e}\left(\prod_{i \in Q} H(fid \parallel i)^{v_i}, X_3\right) \\
 &= C
 \end{aligned}$$

5.4 性能分析

在本节中，我们先数值分析我们方案的开销成本，然后展示实验的数据结果。

5.4.1 数值分析

在这一小节中，我们为方案中的计算开销和通信开销做了数值分析。为简单起见，在本文的余下部分，我们使用 M_{G_1} , M_{G_2} 表示一个乘法运算的复杂性以及用 E_{G_1} , E_{G_2} 表示一个指数运算的复杂性在群 G 和 G_T 中。我们还使用 E 和 M 来表示乘幂运算和乘法运算的复杂性。 P 是双线性对运算， H 是映射到点放的 Hash 函数运算。

计算开销: 如上介绍的，我们的方案由四个算法组成：KeyGen, UploadStore, AuditInt 和 ReplicaVer。算法 KeyGen 是生成系统参数和用户的密钥，所有它主要的计算开销就是 $E_G + 1BP$ ，但是这个算法运行极其快，它的时间复杂度通常都

是常数级的。为编码文件块生成验证标签 $T_i = X^{m_i} H(fid \parallel i)^r$, 一个文件有 n 个文件块, 计算验证标签需要 $2nE_G$ 和 nH 操作, 这一运算可以在线下做。在数据完整性审计的过程中, 审计者通过选择随机数量的常数来生成挑战信息的计算开销是可以忽略的, 仅需要 $1E_G$ 和 $1E_{G_T}$ 来生成 X_1 和 X_2 。一旦云服务器收到挑战信息 $Chal$, 需要 $cE_G, cE_{G_T}, 1M_G, 1M_{G_T}$ 和 cP 操作来计算验证所要的信息。最后, 审计者需要执行 $c + 1E_G, 1M_G, 1M_{G_T}, 2P$ 和 cH 操作来验证 $resp$ 的有效性。因此, 算法 AuditorINT 的总计算开销就是 $\mathcal{O}(1)E + \mathcal{O}(1)M + \mathcal{O}(1)P + \mathcal{O}(1)H$ 。方案中的 ReplicaVer 算法, 其中用户的计算开销是可以忽略的, 服务器的计算复杂度为 $\mathcal{O}(1)E + \mathcal{O}(1)M + \mathcal{O}(1)P + \mathcal{O}(1)H$ 。

通信开销: 审计过程的通信开销主要是由 $Chal = \{(i, v_i)_{i \in Q}, X_1, X_2, POK\}$ 和 $resp = \{\sigma, B\}$ 的原因。在实际的情况中, 我们可以使用伪随机函数 θ 和伪随机置换 ψ 作为公共参数。验证者不需要发送挑战集合 Q , 只需要发送伪随机函数的健值 k_1 和伪随机置换的健值 k_2 , 这可以大大减少通信开销。在这种情况下, 二进制长度的通信开销为 $\log_2 X_1 + \log_2 X_2 + \log_2 k_1 + \log_2 k_2 + \log_2 POK$ 。在回应挑战 $Chal$ 的过程中, 云服务器返回 $\{\sigma, B\}$ 给验证者, B 是椭圆曲线上的点, 长度为 320 bits, 所有发送 $resp$ 的通信开销为 $\log_2 \sigma + 320$ 。在 ReplicaVer 协议的过程中, 服务器发送 $Chal = \{(i, v_i)_{i \in Q}\}$ 给用户, 而用户只需要发送 $\mu = \sum_{i \in Q} v_i m_i$ 给云服务器证明他真的拥有该文件。因此, 这个过程的通信开销是 $\log_2 k_1 + \log_2 k_2 + l$ (假定 $m_i \in \{0, 1\}^l$)。

5.4.2 实验结果

为了表明我们的方案是高效的, 方案实现使用 JPBC 2.0.0^[49] 和 PBC 封装包^[37], 在处理器为 Intel(R) Core(TM) i5-6200U CPU @ 2.3GHz 的电脑运行得到测试结果的。内存总是足够的, 因为方案需要空间不大。实验中, 我们使用参数 `a.properties`, 它是 jpbc 库的标准参数的设置之一。参数 `a.properties` 提供了所有默认参数中速度最快的对称配对。为了验证我们的方案在用户端的计算成本, 我们将存储在云服务器上的文件大小控制 200KB 到 2MB 之间, 从而得到实验的测试数据。第一步, 我们采用一个大小为 2MB 的文件, 并观察挑战块的数量对时间开销的影响。在我们的设置中, 数据块的大小是一个限定值, 为 160bits。因此, 大小为 2MB 的文件

表 5-1 1MB 大小文件各个过程的计算时间

密钥生成	离线tag生成	在线	挑战生成	验证
113ms	448.0s	44s	336ns per challenge	1.05ms per challenge

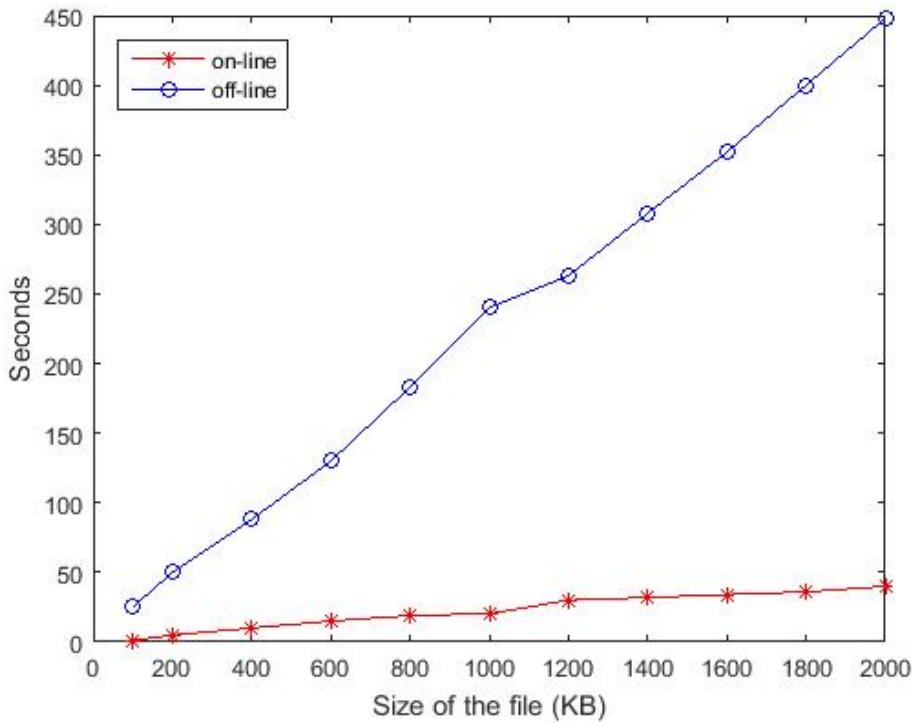


图 5-3 随着文件大小的增加标签生成的时间变化

总共有100000个块。这意味着 KenGen 和 UploadStore 过程的运算时间基本上是恒定的，因为主要的计算开销是验证标签的产生，详细的情况可以看表5-1。可以看到 KeyGen 算法是极其快的，它选择一些随机值并且计算群 G 中的指数的模运算和双线性对运算，花费113ms。认证标签生成的计算成本是最昂贵的，标签生成的时间消耗来源于两部分，当生成标签 $T_i = X^{m_i} H(fid \parallel i)^r, H(fid \parallel i)^r$ 部分的计算不需要知道文件信息，这部分计算可以在线下完成；文件的拥有者需要对文件的每个数据块计算 X^{m_i} ，这部分必须是线上才能完成的，我们只需关注线上的计算时间。为计算具有100000个数据块的文件生成标签，花费大约44秒。

第二步，我们主要通过变化挑战块的数量来观察 AuditorINT 和 ReplicaVer 过程的时间消耗，挑战块的数量变化范围是50到1000之间。正如图5-4所展现的一样，计算时间的花费显然是因为需要选择更多的随机值，执行更多的取幂，乘法运算，对计算和哈希运算。AuditorINT 和 ReplicaVer 的时间消耗主要是由于产生挑战信息 Chal 的计算，生成证据的计算以及验证证据的计算。但是公平地说，我们所提出的方案确实是有高效的。我们表明，验证者可以随机挑战 $Q = 460$ 个数据块，以确保至少99%的错误检测概率，云服务器可以挑战300或460个块，以实现95%或99%准确率用户真的拥有整个数据文件。最后，我们测试最昂贵的计算成本，通过将文件大小从200 KB增加到2 MB来生成验证标签，即从10000块到100000块。如预期的一样，为测试文件生成认证标签的时间，标签计算在线上

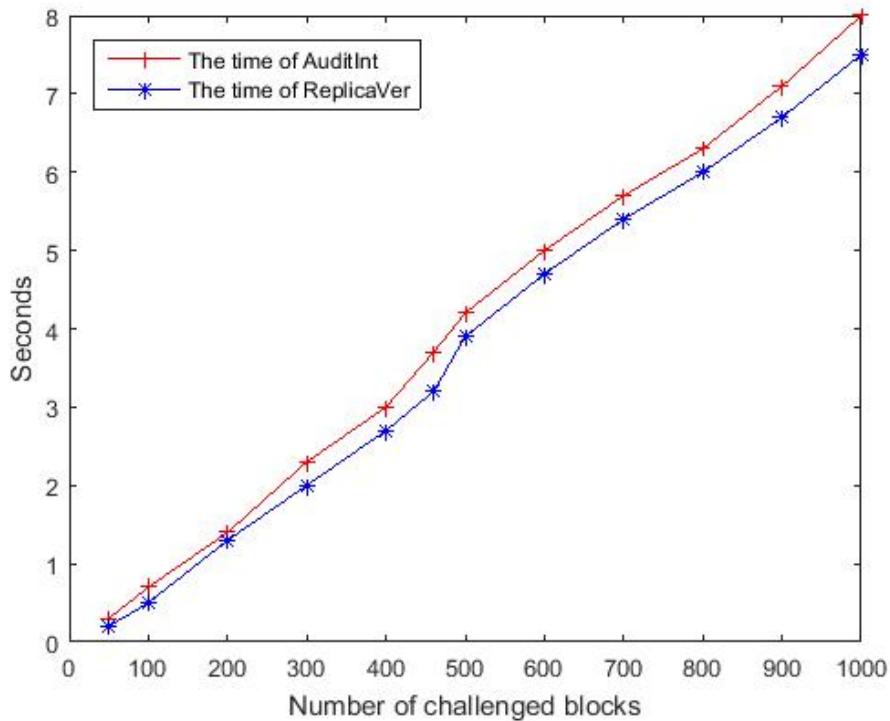


图 5-4 随着挑战块的增加AuditInt 和 ReplicaVer 算法的计算时间

和离线时间几乎都是随着文件大小的增加而线性增加。图5-3显示了文件标签生成时间随文件大小的增加而增加，实验表明生成标签的时间消耗比其他部分的都要高，如上所述，云用户可以完全事先并行地进行离线工作，所以我们注意在线成本。与现有的一些具备去重功能的云数据完整性验证方案相比，如[18]，我们可以看到我们方案的标签生成的高效性。

5.5 本章小结

本章开始分析可恢复性证明 (PoR) 和数据拥有证明 (PDP) 两者和所有权证明 (POW) 之间的密切联系，它们的本质都是来验证数据的可持有性证明，然而并不是所有的 PoR 和 PDP 方案都适用于 POW，也不是所有 POW 方案就能用于数据完整性验证，这是它们之间的区别。本章使用基于广播的可聚合签名提出一种具备去重的数据完整性检测方案，同时实现数据拥有性证明 (PDP) 和数据的所有权证明 (POW)，该方案还支持零知识隐私保护，在审计过程中证明者 (云服务器) 在计算有关于文件数据信息 μ 时，不是直接将 μ 发送给第三方验证者，而是通过计算 $B = X_2^\mu$ ，将 B 发送给验证者，在审计的过程中不会将文件信息泄露给第三方验证者。最后展示了本协议的实验结果。

第六章 全文总结与展望

6.1 全文总结

云存储作为云计算提供的重要服务，允许用户将数据从本地存储系统迁移到云存储系统中，减轻用户管理和维护数据的负担，以及提供地理位置无关的数据访问服务。目前，越来越多的用户喜欢将他们的数据存储在云中。而用户外包到云端的大量数据存在重复，数据去重技术可以消除数据的重复副本，并且在存储中只保留一个副本，极大节省服务器的存储空间，本文利用新型的密码学签名技术和数据拥有性证明技术同时解决数据完整性问题和数据去重的文体。另外在云存储中云用户的密钥可能会经常更新，通过下载所有文件再计算文件的块标签，然后再将文件和更新的标签上传是不实用的，本人提出一个在云数据审计中实用的密钥更新方法，使得用户只需将更新之后的私钥通过处理传到服务器，用户存储在云上所有文件的块标签的更新由服务器计算完成，极大减少了云用户的计算开销和网络带宽的消耗，该方案还具有公开验证和隐私保护的性质，本文具体研究成果如下：

1. 分析现有的具备去重的云数据完整性检测方案的安全性，Zheng提出的支
持去重的安全高效的存储证明方案，能实现数据的去重操作，同时也能实现云数
据的完整性检测，缺陷是方案中使用了大量的随机数，计算过程比较复杂，另一
缺陷是在审计过程中会泄露审计数据的信息，不是零知识隐私保护的。
2. 通过分析数据可拥有性证明(PDP)和数据所有权证明(POW)之间的联系，本
文提出了一个支持去重操作的云数据完整性检测方案。通过使用广播的可聚合签
名，在同一框架中同时实现安全的数据去重操作，和数据可拥有性证明检测。此
外，该方案还支持零知识隐私保护，在审计的过程中不会将文件信息泄露给第三
方验证者。
3. 本文提出了一个支持密钥更新的数据审计方案，当用户私钥丢失或是过期
时，需要更换其密钥，用户只需将更新之后的私钥通过处理传到服务器，用户存
储在云上所有文件的块标签的更新由服务器计算完成，该方案利用离散对数的简
单的零知识证明，在审计和密钥更新的过程中实现隐私保护，并且证明了该方案
的安全性

6.2 后续展望

目前，云数据完整性验证的关键技术研究得到了大量的关注和研究，本文提出了支持去重操作的数据完整性检测研究方案和支持密钥更新操作的云数据审计方案，但依然存在以下问题：

1. 在本文提出的支持去重操作的数据完整性检测方案中，对方案的安全模型做了详细描述，但是未给出安全性证明。
2. 本文提出的支持去重操作的云数据完整性检测方案针对的是文件级的数据去重，文件级的数据去重就是在文件的基础上直接进行去重，没有办法做到跨文件去重，去重效果不是很好。所以，设计一个支持数据块级去重的数据审计方案是一个值得考虑的问题。

致 谢

在电子科技大学，我度过了人生中最难忘的三年，在这三年的时光里，我收获颇丰，为此我要衷心感谢所有给予过我帮助和支持的老师和同学。

首先我要感谢的是我最亲爱最敬爱的导师***副教授，一直以来对我生活和学业上无微不至的关怀和帮助，对我来说他既是老师又是朋友。在学习上不管越到什么困难，他耐心地给予解答并悉心进行指导，在平时的生活中，他也和我们一块讨论周围发生的趣事，一起欢笑。***在科研上严谨的态度和敏锐的洞察力，以及在生活中积极乐观的人生态度深深感染着我。记得我第一次给大家做讨论分享，由于比较紧张和文章中有许多地方没有看透看懂，讲的结结巴巴，讨论会后心里很难受，***就赶紧安慰了鼓励我，让我不要有太多压力，下次再接再厉，他相信我一定会做的更好。十分感谢***在学业上给予的耐心指导，在生活上给予温暖。在本论文的选课、方案分析、实现以及最后的修改都给予了指导和支持。

感谢学院所有给予我帮助的老师，感谢你们在学生和生活上给予的大力帮助。还要感谢同教研室的同学，感谢你们三年的陪伴，特别感谢师姐张亚芳在学习和生活各方面对我的帮助，师姐为人热情，在生活中给予指导和帮助，在学习上，她勤奋踏实的学习态度和积极乐观的生活态度十分值得我学习，感谢***、***、***、***、***、***，在本论文的修改过程中给予了最大的帮助。感谢我的家人，感谢父母，在这三年里给予我最好的物质条件，以及在生活上给予的最大关怀，感谢你们默默无私的付出，感谢弟弟，总是给予我支持和鼓励。

最后，感谢各位评审老师对本论文的评阅，向你们致以最诚挚的谢意。

参考文献

- [1] 李国杰. 信息科学技术的长期发展趋势和我国的战略取向[J]. 中国科学: 信息科学, 2010, 40(1):128–138
- [2] P. Mell, T. Grance. Draft NIST Working Definition of Cloud Computing[J]. National Institute of Standards and Technology, 2009, 53:50
- [3] 刘思得. 基于网络的云存储模式的分析探讨[J]. 科技通报, 2012, 28(10):206–209
- [4] D. Zissis, D. Lekkas. Addressing cloud computing security issues[J]. Future Generation computer systems, 2012, 28(3):583–592
- [5] M. Shiels. Phone sales hit by sidekick loss[J]. URL: <http://news.bbc.co.uk/2/hi/technology/8303952.stm>, 2009
- [6] G. Ateniese, R. Burns, R. Curtmola, et al. Provable data possession at untrusted stores[M]. 2007, 598–609
- [7] G. Ateniese, R. Burns, R. Curtmola, et al. Remote data checking using provable data possession[J]. ACM Transactions on Information and System Security (TISSEC), 2011, 14(1):12
- [8] A. Juels, B. S. K. J. PORs: Proofs of retrievability for large files[C]. Proceedings of the 14th ACM conference on Computer and communications security, Alexandria, 2007, 584–597
- [9] H. Shacham, B. Waters. Compact Proofs of Retrievability[C]. in Proc. International Conference on the Theory and application of cryptology and Information security (ASIACRYPT), Sydney, 2008, 90–107
- [10] H. Shacham, B. Waters. Compact proofs of retrievability[J]. Journal of cryptology, 2013, 26(3):442–483
- [11] J. Gantz, D. Reinsel. The digital universe decade—are you ready[J]. IDC White Paper, 2010:1–16
- [12] G. H. Forman, F. Safai, B. Zhang. Data de-duplication[M]. US, 2007
- [13] S. Halevi, D. Harnik, B. Pinkas, et al. Proofs of ownership in remote storage systems[M]. 2011, 491–500
- [14] R. D. Pietro, A. Sorniotti. Boosting efficiency and security in proof of ownership for deduplication[M]. 2012, 81–82
- [15] Y. Deswarte, J. J. Quisquater, A. Saïdane. Remote integrity checking[M]. Springer, 2004, 1–11
- [16] Q. Wang, C. Wang, J. Li, et al. Enabling public verifiability and data dynamics for storage security in cloud computing[M]. 2009, 355–370
- [17] H. Shacham, B. Waters. Compact Proofs of Retrievability[J]. Journal of Cryptology, 2013, 26(3):442–483

- [18] Q. Zheng, S. Xu. Secure and efficient proof of storage with deduplication[M]. 2011, 1–12
- [19] Y. Yu, H. A. Man, Y. Mu, et al. Enhanced privacy of a remote data integrity-checking protocol for secure cloud storage[J]. International Journal of Information Security, 2015, 14(4):307–318
- [20] X. Lai, J. L. Massey. Hash functions based on block ciphers[M]. 1992, 55–70
- [21] R. Rivest. RSA Data Security, Inc.,” The MD4 Message Digest Algorithm[J]. Network Working Group, RFC-1321, 1992
- [22] D. Chaum, T. P. Pedersen. Wallet Databases with Observers[M]. 1992, 89–105
- [23] Q. Wu, Y. Mu, W. Susilo, et al. Asymmetric group key agreement[M]. 2009, 153–170
- [24] N. Mandagere, P. Zhou, M. A. Smith, et al. Demystifying data deduplication[M]. 2008, 12–17
- [25] D. T. Meyer, W. J. Bolosky. A study of practical deduplication[J]. Acm Transactions on Storage, 2012, 7(4):14
- [26] M. Vrable, S. Savage, G. M. Voelker. Cumulus: Filesystems backup to the cloud[J]. the magazine of USENIX and SAGE, 2009, 34(4):7–13
- [27] W. V. D. Laan. Dropship-dropbox api utilities[M]. 2011
- [28] Y. Yu, Y. Li, M. H. Au, et al. Public cloud data auditing with practical key update and zero knowledge privacy[M]. 2016, 389–405
- [29] S. Goldwasser, S. Micali, C. Rackoff. The knowledge complexity of interactive proof systems[J]. SIAM Journal on computing, 1989, 18(1):186–208
- [30] C. P. Schnorr. Efficient signature generation by smart cards[J]. Journal of Cryptology, 1991, 4(3):161–174
- [31] M. Blaze, G. Bleumer, M. Strauss. Divertible protocols and atomic proxy cryptography[J]. Lecture Notes in Computer Science, 1998, 1403:127–144
- [32] G. Ateniese, S. Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications[M]. 2005, 310–319
- [33] D. Boneh, B. Lynn, H. Shacham. Short Signatures from the Weil Pairing[M]. 2001, 514–532
- [34] B. Libert, D. Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption[M]. 2008, 360–379
- [35] P. S. L. M. Barreto, M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order[M]. Springer Berlin Heidelberg, 2006, 319–331
- [36] Y. Yu, Y. Li, M. H. Au, et al. Public cloud data auditing with practical key update and zero knowledge privacy[M]. 2016, 389–405
- [37] A. Kate. The Pairing-Based Cryptography (PBC) Library-C++ Wrapper Classes (0.8. 0). on line: <http://crysph.uwaterloo.ca/software>[M]. PBCWrapper

- [38] Y. Yu, L. Xue, H. A. Man, et al. Cloud data integrity checking with an identity-based auditing mechanism from RSA[J]. Future Generation Computer Systems, 2016, 62(C):85–91
- [39] R. Buyya, C. S. Yeo, S. Venugopal, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation computer systems, 2009, 25(6):599–616
- [40] G. Ateniese, S. Kamara, J. Katz. Advances in Cryptology—ASIACRYPT 2009[M]. Tokyo: Springer, 2009, 319–333
- [41] R. Curtmola, O. Khan, R. Burns. Robust remote data checking[C]. Proceedings of the 4th ACM international workshop on Storage security and survivability, Alexandria, 2008, 63–68
- [42] K. D. Bowers, A. Juels, A. Oprea. Proofs of retrievability: Theory and implementation[C]. Proceedings of the 2009 ACM workshop on Cloud computing security, Chicago, 2009, 43–54
- [43] Y. Dodis, S. Vadhan, D. Wichs. Theory of cryptography[M]. 北京: Berlin, 2009, 109–127
- [44] Q. Wang, C. Wang, K. Ren, et al. Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing[J]. IEEE Transactions on Parallel & Distributed Systems, 2010, 22(5):847–859
- [45] Y. Zhu, H. Hu, G. J. Ahn, et al. Efficient audit service outsourcing for data integrity in clouds[J]. Journal of Systems & Software, 2012, 85(5):1083–1095
- [46] K. Yang, X. Jia. An Efficient and Secure Dynamic Auditing Protocol for Data Storage in Cloud Computing[J]. IEEE Transactions on Parallel & Distributed Systems, 2013, 24(9):1717–1726
- [47] C. Wang, S. S. M. Chow, Q. Wang, et al. Privacy-Preserving Public Auditing for Secure Cloud Storage[J]. IEEE Transactions on Computers, 2013, 62(2):362–375
- [48] K. D. Bowers, A. Juels, A. Oprea. HAIL:a high-availability and integrity layer for cloud storage[M]. 2008, 187–198
- [49] H. Wang. Identity-based distributed provable data possession in multicloud storage[J]. IEEE Transactions on Services Computing, 2015, 8(2):328–340

攻硕期间取得的研究成果

- [1] Y. Yu, J. H. Qiu, Y. N. Li, et al. Cloud Data Integrity Checking with Deduplication for Confidential Data Storage[J]. Future Generation Computer Systems, 2016 under review
- [2] 禹勇, 薛靓, 李艳楠, 邱佳惠, 张亚芳. 一种具备去重功能的云数据公开审计方法[P]. 中国, 2016年
- [3] 禹勇, 李艳楠, 邱佳惠, 张亚芳. 云存储的密钥更新方法及云数据审计系统的实现方法[P]. 中国, 2015年
- [4] 禹勇, 张亚芳, 倪剑兵, 李艳楠, 邱佳惠. 云计算中一种基于身份的数据存储与完整性验证方法[P]. 中国, 2015年



专业学位硕士学位论文

MASTER THESIS FOR PROFESSIONAL DEGREE