一种面向云的大数据完整性检测协议

张也弛[1,2],周文钦[1,2],石润华[1,2]
(1. 安徽大学 计算智能与信号处理教育部重点实验室,安徽 合肥 230039;
2. 安徽大学 计算机科学与技术学院,安徽 合肥 230601)

摘 要:云计算的出现为计算机的发展提出了一个新的发展方向,但是其中的很多安全问题一直限制着云计算的发展,其中云存储的完整性检测问题最为突出。由于大数据的特性,现有的关于数据完整性检测的协议并不适合大数据的检测。通过对 Juels 协议的改进,文中提出一种新的更加适合大数据存储的数据完整性检测协议,改进了原有协议只能进行有限次数检测的缺陷,引入了纠删码技术,使得数据的可恢复性得到大幅提升。通过理论分析表明该协议在数据的计算量和用户数据存储量上都有明显优势。

关键词:云计算;大数据;数据完整性检测
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2014)09-0068-05
doi:10.3969/j.issn.1673-629X.2014.09.015

A Big Data Integrity Checking Protocol for Cloud

ZHANG Ye-chi[1,2], ZHOU Wen-qin[1,2], SHI Run-hua[1,2]
(1. Key Laboratory of Intelligent Computing & Signal Processing of Ministry of Education, Anhui University, Hefei 230039, China;
2. School of Computer Science and Technology, Anhui University, Hefei 230601, China)

Abstract: With the emergence of cloud computing, a new developing direction of the computer has been pointed out. But there are still some security problems restricting the development of cloud computing, one of the most important which is data integrity checking. The current known protocols for data integrity checking are not suitable for big data which are so large and complex that it becomes difficult to check their integrity. In this paper, a new data integrity checking protocol for big data in the cloud storage is proposed, by improving Juel's protocol in which it only allows the data detection within limited times and bringing in the technology of erasure code to elevate the data's restorability significantly. By theoretical analysis, the protocol proposed is proved to have obvious advantages in data computing and memory space of the users.

Key words: cloud computing; big data; data integrity checking

0 引 言

随着云计算技术的发展,云计算的安全问题[1]也逐渐引起人们的重视。其中,云计算中数据的存储安全成为最大的问题[2]。由于数据在云端的存储时间最长,所以云计算中数据的存储安全一直是学术界研究的热点。如何保证存储在云端的数据没有被服务器篡改或者由于服务器故障而丢失正是云计算中面临的诸多安全问题之一。

以目前的发展趋势来看,互联网即将进入大数据时代,庞大数据的计算对于服务器来说也将成为巨大的负担。生活中充满了各种数据。人们的生活习惯和各种行为也被抽象成各种数据,这些数据意义重大,可以用于分析和挖掘重要信息。从某种意义上说,这些数据就是资源。对于大数据[3]和普通数据的安全保护,最大区别在于对大数据的加密和编码的代价非常高,若采用和普通数据一样的通过多次加、解密实现对数据的保护,其效率是非常低的。因此,大数据和普通数据的管理与保护还是有一定差异的。

收稿日期:2013-11-11 **修回日期**:2014-02-17 **网络出版时间**:2014-07-17
基金项目:国家自然科学基金资助项目(61173187,61173188);安徽省自然科学基金(11040606M141);安徽大学博士科研启动经费项目(33190187);安徽大学"信息安全"新专业项目(17110099)
作者简介:张也弛(1987-),男,安徽蚌埠人,硕士生,研究方向为保护隐私的多方协作计算;石润华,博士,教授,研究方向为现代密码学与安全多方计算。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20140717.1229.030.html

数据完整性检测最早是在 2004 年由 Lillibridge 等[4]提出,并且在同年由 Deswarte 等构造了一个具体的协议[5]。Naor 和 Rothblum[6]在 2005 年提出了基于 MAC 的检测协议。Filho[7]和 Ateniese 等人[8]分别在 2006 年、2007 年提出了基于数据同态性的完整性检测方案。2008 年 Shacham 等人提出了基于 BLS 的检测协议[9],同年,Sebe 等人[10]对 Filho 的方案进行了改进。随后,Zhu 等人提出了 IPDP 协议[11]。近几年,也有不少专家和学者提出新的协议,例如,基于访问控制的检测协议[12]、基于子树的验证协议[13]。但是这些协议,对大数据的处理都不是很有效。

随着云计算的发展,PC 机和 PDA(手持终端)都将成为云服务的终端设备,如清华大学的透明计算平台[14]。另外确保多任务的同时进行,也是未来用户需求之一。因此减少数据完整性检测时对用户终端有限资源(计算资源、存储资源、网络带宽等)的占用是未来数据检测协议的重要发展方向。

针对大数据的远端存储,2007 年 Juels 等人[15]提出了一个基于"哨兵"的数据完整性检测协议。然而,该协议只能进行有限次检测,且检测次数由"哨兵"的数量决定。另外,用户本地需要存储每个"哨兵"的具体数值和位置,以便检测时使用。2011 年 Sravan Kumar R 和 Ashutosh Saxena[16]对 Juels 的协议做了改进。在原有协议的基础上,通过两个公式计算出"哨兵"的存储位置和存储数值,即"哨兵"不再是对应块的纠错码,而是通过公式生成得到。同理,"哨兵"的位置也是通过类似的方法生成得到。这样就使本地存储的复杂度从 $O(n)$ 降低到 $O(1)$。然而,在改进的协议中,关于数据的有限次数的检测问题仍然没有得到解决。

文献[17]对近几年的数据完整性检测做过相关分析,与现有的其他协议相比(如文献[8-9,18]),Juels 的协议计算量和通信量都比较低,对网络带宽、终端配置(计算机或者 PDA)的要求也相对较低,更适合大数据的处理和检验。

基于此,文中主要对 Juels 协议进行了改进,通过随机插入和拟合检测的方式使得检测次数从有限次数提升为无限次数,即云服务器无法判断出"哨兵"在每块数据中的位置和具体内容。Juels 的协议是通过对分块后的数据进行纠错码编码,再使用随机数作为"哨兵",用户把"哨兵"嵌入数据块并记录每个相应的位置和"哨兵"数值。通过对"哨兵"数值的比对判断数据的完整性。借鉴 Juels 协议的分块思想,通过对数据分块,在每块数据中插入不同的"哨兵",而在检测时,要求云服务器传输指定的数据段。与此同时,引入文献[16]中对"哨兵"的存储位置和存储数值的处理方式,使得用户只需要存储固定数量的参数即可。通过对数据段中"哨兵"值的检验判断数据是否被修改,另外利用纠删码[19]的特性实现数据的恢复功能。

1　建议的协议

当用户直接把数据交由云服务器存储后,很难再确认当初存储的数据的完整性,即存储的数据是否遭到了服务器的恶意篡改或者由于其他硬件故障等原因所造成了数据的丢失。一般地,用户在上传数据前,需要对存储的数据进行一定额外的处理,以备后期实现对数据完整性的检测。此外在数据部分损毁或者被篡改后,也要有一定的数据恢复能力。对此,已有研究者提出了多种数据完整性检测协议。但多数协议在设计之初并没有考虑大数据。若用于大数据,这些协议需要的额外开销过大,很难适应资源受限的用户终端。

文中提出了一种更适合大数据的完整性检测方案。在下面的协议中,参与方只有数据拥有者和云存储服务器,没有任何可信的第三方。

协议描述如下:

假定每个用户有一对公私钥 PK,SK。协议主要包含两个阶段:预处理阶段和数据检测阶段。

1)预处理阶段。

(1)分块处理:用户使用公式 $\mathrm{Gen}(\,)\rightarrow(K_{\mathrm{prv}},k)$ 生成 K_{prv} 和 k。K_{prv},k 均为用户秘密。以 k 为密钥对数据 F 使用某种对称加密算法加密:$E_k(F)\rightarrow F'$。再把 F' 分成 n 块,进而利用纠删码技术,把每块数据分成 m 个子秘密:$\begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix}$。

(2)"哨兵"的生成:对于每块数据 a_{ij},利用自定义公式生成 l 个随机数,作为插入"哨兵"的位置,即 $f(K_{\mathrm{prv}},i,j,l)\rightarrow(K_{i,j,1},\cdots,K_{i,j,w},\cdots,K_{i,j,l})$。这里 $K_{i,j,w}$ 表示第 w 个"哨兵"的插入位置。接着,对于每个插入位置 $K_{i,j,w}$,再利用另外自定义的公式 $f(K_{\mathrm{prv}},i,j,K_{i,j,w},p)$ 生成一个长度为 p 的"哨兵" $S_{i,j,w}$。其中 $\dfrac{l*p}{\mathrm{len}(a_{ij})+l*p}$ 为"哨兵"总量占整个存储文件的比例,p 和 l 的数值由用户自己设置($l\geqslant 2$),$\mathrm{len}(a_{ij})$ 表示数据块 a_{ij} 的位长。

对于具体的自定义公式 $f(\,)$ 和 $f'(\,)$,这里建议如下:

首先计算 $K_{i,j,w}^{\sim}=H^w(\mathrm{ID}\,\|\,S(i,j)\,\|\,K_{\mathrm{prv}}\,\|\,i\,\|\,j\,\|\,l)\bmod(\mathrm{len}(a_{ij}'))$,其中 ID 为用户名,$S(i,j)$ 为伪随机数函数,$\mathrm{len}(a_{ij})+l*p=\mathrm{len}(a_{ij}')$,即每块数据嵌入"哨兵"后的大小。这里 $H^1(\cdot)=H(\cdot)$,$H^2(\cdot)=$

$H(H^1(\cdot))$,…, $H^l(\cdot)=H(H^{l-1}(\cdot))$,其中 $H(\cdot)$ 为哈希函数。然后按照单调递增顺序对 $K^{\sim}_{i,j,1}$,…, $K^{\sim}_{i,j,w}$,…, $K^{\sim}_{i,j,l}$ 排序,最终得到有序序列 $(K_{i,j,1},\cdots,K_{i,j,w},\cdots,K_{i,j,l})$。另外定义 $S_{i,j,w}=\text{front}_p[H^w(\text{ID}\|S(i,j)\|K_{\text{prv}}\|i\|j\|l)]$,其中函数 $\text{front}_p(x)$ 定义为取 0/1 串 x 的前 p 位子串。秘密保存公式 $f(\cdot)$ 和 $f'(\cdot)$ 也即秘密保存$\{K_{\text{prv}},l,\text{len}(a'_{ij})\}$。实际上,公式 $f(K_{\text{prv}},i,j,l)$ 和 $f'(K_{\text{prv}},i,j,K_{i,j,w},p)$ 可以公开,但其中的输入参数需要用户秘密保存。

(3)“哨兵”的插入:插入“哨兵”时,利用 $f(K_{\text{prv}},i,j,l)$ 和 $f'(K_{\text{prv}},i,j,K_{i,j,w},p)$ 生成的 $K_{i,j,w}$ 和 $S_{i,j,w}$ 对 a_{ij} 进行“哨兵”插入,由于 $K_{i,j,w}$ 是在“哨兵”插入前生成的,而每插入一个“哨兵”都会使得数据长度发生变化,为了使 $K_{i,j,w}$ 所指向的位置固定不变就需要对其进行处理,即第 w 个数据的具体插入位置 $K'_{i,j,w}=K_{i,j,w}+(w-1)*p$,$w\in[1,2,\cdots,l]$。此时的 $K'_{i,j,w}$ 就可以用于插入操作了。然后,就可以根据坐标 $K'_{i,j,w}$ 和“哨兵” $S_{i,j,w}$ 对每个 a_{ij} 进行更新,执行 insert $(a_{ij},K'_{i,j,w},S_{i,j,K_{i,j,w}})\rightarrow a'_{ij}$。生成最终数据 $\boldsymbol{M}=\begin{pmatrix}a'_{11}&\cdots&a'_{1m}\\ \vdots&\ddots&\vdots\\ a'_{n1}&\cdots&a'_{nm}\end{pmatrix}$。这里,insert (a,k,s) 表示在数据 a 的第 k 位后插入数据 s。

(4)计算 $\text{lpnm}=E_{\text{PK}}(K_{\text{prv}},l,p,n,m,\text{len}(a'_{ij}),H(K_{\text{prv}}\|l\|p\|n\|m\|\text{len}(a'_{ij})))$,并将 lpnm 附在 $\boldsymbol{M}$ 后一同存储在云服务器中。存储 lpnm 主要是为了确保各种秘密参数没有被篡改。这里 $E_{\text{PK}}(\cdot)$ 表示公钥加密算法,其中 PK 为公钥,解密时需要私钥 SK。

2)数据检测阶段。

(1)采用传统的挑战—响应方式,用户首先向服务器发出检验请求。

(2)服务器接收到检验请求后,立即验证用户身份,若是合法用户,则返回相应的小数据 lpnm,即 $\text{lpnm}=E_{\text{PK}}(K_{\text{prv}},l,p,n,m,\text{len}(a'_{ij}),H(K_{\text{prv}}\|l\|p\|n\|m\|\text{len}(a'_{ij})))$。

(3)用户使用私钥 SK 解密 lpnm,得到所需的各种秘密参数 $\{K_{\text{prv}},l,p,n,m,\text{len}(a'_{ij})\}$,进而计算出 $\{K_{\text{prv}}\|l\|p\|n\|m\|\text{len}(a'_{ij})\}$ 的哈希值 $H(K_{\text{prv}}\|l\|p\|n\|m\|\text{len}(a'_{ij}))$,并将其与从 lpnm 解密获得的哈希值进行比对,确保参数和公式没有被篡改。

(4)用户使用 random(n,m,r) 函数计算 r 对随机数组 $I=[(i_1,j_1),(i_2,j_2),\cdots,(i_r,j_r)]$,其中 n, m 具有 $n<m$ 的关系,而生成的 i, j 具有 $i<j$ 以及 $i<n, j<m$ 的关系,r 的数值由用户根据自身情况设置,当时间充裕,硬件设备良好,通信良好的情况下可以增大 r 的值(随机检测 r 块数据的完整性)。反之,如果硬件设备状态一般如 PDA,或者带宽、时间有限,则可以减小 r 的数值。接着使用公式 $f(K_{\text{prv}},i,j,l)$ 生成 $K_{i,j,w}$,继而得到 $K'_{i,j,w}=K_{i,j,w}+(w-1)*p$,最后计算出 r 个 a'_{ij} 中所有“哨兵”的位置。

(5)采用滑动窗口方法确定需要检测的数据区域。初始选取“窗口步长”为 $\text{len}(a'_{ij})/l$,通过对“窗口”位置的滑动,寻找出一个区域(起始位置 s 至终止位置 e,如图 1 所示),包含的完整或部分“哨兵”的总位长最大,且满足一个阈值。当“窗口”滑动到数据块尾端仍然无法找到达到指定的阈值时,“窗口步长”扩大一倍,再从数据开头重新滑动寻找。即统计下列有效长度的最大值。需要对窗口的起始和终止位置进行分析。由于“哨兵”位置的不确定性,窗口内囊括的“哨兵”可能并不是完整的,且囊括的“哨兵”有可能是一个或者数个,所以有效长度即窗口内“哨兵”的含量为:不完整的哨兵大小+完整哨兵的个数 * 单个哨兵的大小。由于不完整的哨兵在一个数据块中最多为两个,即由窗口选择所致,且可以根据窗口的位置判断出不完整哨兵的数量,所以接下来只要确定窗口内包含的“哨兵”总数量即可。确定一行内窗口囊括的哨兵个数:

①可以得知所有“哨兵”的起始坐标 $K_{i,j,w}$ 且“哨兵”的长度固定。当 s 在区间集合$[K_{i,j,w},K_{i,j,w}+p]$内时,即 s 在哨兵内时(如图 1 中 1,2 所示),记录所在区间首个哨兵起始位置 $K_{i,j,w}$ 中的下标 w(下标 w 表示的是 a_{ij} 中的第 w 个哨兵),记为 w_s。当 s 不在该集合内时,即 s 不在哨兵内部(如图 1 中 3,4 所示),寻找 s 后紧接着的 $K_{i,j,w}$,记为 w_s。

②对于终点位置 e,无论 e 是在哨兵内部或者外部(如图 1 所示),均记录紧接着 e 后的 $K_{i,j,w}$(即大于 e 且和 e 差值最小的 $K_{i,j,w}$),记其下标 w 为 w_e。

③ w_e-w_s 的数值即为窗口内哨兵总数(部分或全部哨兵个数)。

窗口位置的四种情况见图 1。

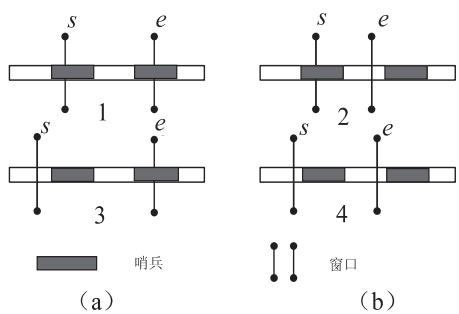

图 1 窗口内哨兵情况

(6)用户把最终得出的起始位置 s,终止位置 e,以及需要检测的数据块编号集合 $I=[(i_1,j_1),(i_2,j_2),\cdots,(i_n,j_n)]$ 发送给服务器。服务器按照要求,返回相

应的数据,即集合 I 中每个数据 $a_{ij}^{'}$ 的第 s 位到第 e 位的数据及其对应的 (i,j) 发送给用户。

(7)用户根据"窗口"区域 (s,e) 、公式 $f(K_{prv},i,j,l)$ 和 $f(K_{prv},i,j,K_{i,j,w},p)$ 计算出 s 位到 e 位之间囊括的"哨兵"编号,和其中的"哨兵"值,再对服务器传回的数据进行匹配比较。如果所有检测数值均相同,则认为数据是完整的。

2　协议分析

由于云服务器对数据的完整性保证是通过数据冗余实现的,因此存储空间对于云服务器来说是廉价的。以云服务器存储空间换取计算量的大量减少是值得的。

1)计算量。

和其他协议相似,在预处理阶段中,最大开销是对文件加密及其他对整个大数据的编码。其次,主要运算是计算出每块数据中的 l 个"哨兵"值及"哨兵"的插入位置。对每个"哨兵"值及相应的插入位置的计算,需要执行一次哈希函数的计算,总的需要 nml 次哈希计算。与 Juels 协议中随机生成"哨兵"相比,文中提出的协议需要更多的计算开销来生成"哨兵"。一方面免去了用户需要存储所有的哨兵值及具体的插入位置,另一方面又换来了后面检测阶段的高效运行。其他对一些小参数的加密等计算相对前面的主要计算来说都是轻量级的开销。

下面分析检测阶段用户和云服务器的计算开销。

用户:

(1)检测数据 lpnm 的哈希值。需要进行使用私钥解密和哈希计算各一次。

(2) random(n,m,r) 是一个单参数的随机算法。其中 n,m 是限制生成的 $i<n,m<j$,例如对生成的随机数进行 mod n 和 mod m 计算。公式生成 $2r$ 个随机数,奇数下标的数值进行 mod n 计算,偶数下标进行 mod m 计算。然后每两个数值为一数据对。所以需要进行 r 次随机数计算。

(3) $f(K_{prv},i,j,l)$ 公式的主要运算为计算 l 次哈希 $h(\cdot)$ (例如 SHA-1)。

(4)在计算 $f(K_{prv},i,j,l)$ 的同时,也顺带计算出了 $f'(K_{prv},i,j,K_{i,j,w},p)$ 。

(5)最终的检测对象包含 r 块数据,所以需要计算 $f(K_{prv},i,j,l)$ 总的 r 次,即计算 $h(\cdot)$ 总的 $r*l$ 次。

(6)其他运算,例如窗口的滑动,相对上面核心计算来说均为轻量级运算。

服务端:只需要按照 (s,e,i,j) 传输数值,不需要其他额外计算。

2)通信量。

每次检测阶段,建议的协议仅仅需要传输一个窗口内的数据,即 $r \times l'$ bits,这里 l' 为窗口步长,初始为 $\text{len}(a_{ij}^{'})/l$ 。

3)存储量。

(1)检测时的存储量:用户需要存储 K_{prv}、l、p、n、m、$\text{len}(a_{ij}^{'})$、(s,e) 以及所含"哨兵"的编号 (i,j,w)、单个"窗口"大小、单个"哨兵"大小 p,r 个 $K_{i,j,w}^{'}$ 和一张 $\text{len}(a_{ij}^{'})$ 个项的统计表。

(2)平时需要存储的数据:对称加密密钥 k 和一对公私钥 PK、SK。

4)与其他协议的比较

将建议的协议和已有的部分优秀协议进行比较。比较的这两个协议对于数据的检测方式都是通过抽取数据样本进行检测,这种方式比较适合大数据的检测。主要对计算量、通信量、能否重复使用,以及存储量方面进行对比,结果如表 1 所示。为了使协议间的比较更加清晰,规定数据分块后的数据块 $a_{ij}^{'}$ 大小为 x_1,总共拥有 n 个数据块,每块中每个"哨兵"的大小为 x_2,个数为 x_3,检测的块数为 x_4,则"窗口"的步长为 x_1/x_3。

表 1　比较结果

协议	检测次数	通讯量	计算量	存储量(本地)
Juels	有限次	$x_2 \times x_4$	比较 x_4 个块中的"哨兵"是否全部相等	$x_2 \times x_3 \times n$
Shacham	无限次	$x_1 \times x_5 (1 < x_5 < x_4)$	x_4 次数据块级别的乘法操作	两个核心参数 α 和 s
文中协议	无限次	$x_4 \times x_1/x_3$	$x_3 \times x_4$ 次哈希运算和对"窗口"内"哨兵"的对比	一对密钥 PK、SK 和对称加密密钥 k

从表 1 中可以看出,文中协议和 Shacham 的协议均可以实现无限次数的重复检测。

其次,从通讯量中可以看出 Juels 的协议需要传输 x_4 个哨兵,而 Shacham 的协议需要传输 x_4 个数据块乘以一个随机数后的加和,即 $\sum v_i m_i$ 。文中协议则是需要传输 x_4 个"窗口"大小的数据。文中协议中的"窗口"大小是由每块数据中的"哨兵"个数决定的,当"哨兵"数量越多时,"窗口"就会越小,所以通讯量会根据具体情况有所改变。以数据分块后的大小为 10 MB,总共拥有 1 024 个数据块,每块中每个"哨兵"的大小为 16 B,个数为 64,检测的块数为 10 为例,这里需要说明的是对于大数据的完整性检测样本的抽取不可能

占整体数据的比例很大,所以选择 10 块作为例子进行说明,同时将例子中数据的数量级进行了压缩。从上面的例子可以看出文中协议的通讯量是 1 600 kB,而 Shacham 协议的通讯量最少为 $x_5 \times 10$ MB,其中 $x_5 \in [1,10]$。

计算量上,Juels 的计算量最小,只需要比较哨兵是否一致即可;Shacham 的协议中,主要的计算量是计算 $\sum v_i m_i$ 和 $\sum \sigma_i m_i$,其中 i 的个数为 x_4,v_i 为随机数,σ_i 为 a'_{ij} 对应的特征值。文中协议主要计算量在于 $x_4 \times x_3$ 次哈希运算和"哨兵"的比对。可以看出文中协议在计算量上比 Shacham 协议中的计算量要小。同样以上面的数据为例,Shacham 协议的计算量为 10 次 10 MB 级别的数据块的乘法;而文中协议需要 640 次哈希计算(此处计算的是小整数的哈希值,非整块数据的哈希值)。

在本地存储中,文中协议和 Shacham 的协议都只需要存储一些参数,而 Juels 的协议会随着数据量的增大而增大。

现有的大数据的大部分都是一些随着时间的增加而增加的数据,比如消费记录、浏览记录等等,而人们对于大数据的使用也是对其部分数据的分析和使用,所以编码和加密技术的使用只会在解码方面消耗一定的资源,但是为了保证资源的可恢复性这些消耗是值得的。

通过以上计算、通信、存储开销的分析,以及各主流协议的比较,文中协议在存储、通信开销以及重复检测次数上有着显著的优点。而对于计算开销,最主要的运算是加密与哈希函数的运算,对于目前的计算能力和计算设备来说都是可行的。

3 结束语

通过对 Juels 协议的分析,对不能重复检测的缺点进行了改进,实现了对存储在云服务器端大数据完整性的无限次检测,即检测次数不再受"哨兵"数量的限制。而且避免了本地存储大量的"哨兵"和其插入的位置,减轻了本地存储开销。另外,使用纠删码替代了"哨兵"对数据的恢复功能。该方案可以很好地应用于大数据的完整性检测,而且用于检测的计算开销低,可以在手持终端或者其他计算能力弱的终端实现数据完整性检测。今后进一步努力的方向是解决大数据的恢复方法,提出更加安全实用的大数据恢复方案。

参考文献:

[1] 冯登国,张 敏,张 妍,等. 云计算安全研究[J]. 软件学报,2011,22(1):71-83.

[2] 杨 健,汪海航,王 剑,等. 云计算安全问题研究综述[J]. 小型微型计算机系统,2012,33(3):472-479.

[3] 王 珊,王会举,覃雄派,等. 架构大数据:挑战、现状与展望[J]. 计算机学报,2011,34(10):1741-1752.

[4] Lillibridge M, Elnikety S, Birrell A, et al. A cooperative internet backup scheme[C]//Proceedings of the annual conference on USENIX. Berkeley, CA, USA: USENIX Association, 2004: 29-41.

[5] Deswarte Y, Quisquater J J, Saidance A. Remote integrity checking[C]//Proc of the sixth working conference on integrity and internal control in information systems. Netherlands: Kluwer Academic Publishers, 2004: 1-11.

[6] Naor M, Rothblum G N. The complexity of online memory checking[C]//Proceedings of the 46th annual IEEE symposium on foundations of computer science. Washington, DC, USA: IEEE Computer Society, 2005: 573-584.

[7] Filho D L G, Barreto P S L M. Demonstrating data possession and uncheatable data transfer[R]. [s. l.]: [s. n.], 2006.

[8] Ateniese G, Burns R, Curtmola R, et al. Provable data possession at untrusted stores[C]//Proceedings of the 14th ACM conference on computer and communications security. New York, NY, USA: ACM, 2007: 598-609.

[9] Shacham H, Waters B. Compact proofs of retrievability[C]//Proceedings of the 14th international conference on the theory and application of cryptology and information security: advances in cryptology. Berlin: Springer, 2008: 90-107.

[10] Sebe F, Domingo-Ferrer J, Martinez-Balleste A, et al. Efficient remote data possession checking in critical information infrastructures[J]. IEEE Trans on Knowledge and Data Engineering, 2008, 20(8): 1034-1038.

[11] Zhu Y, Wang H, Hu Z. Cooperative provable data possession[R]. [s. l.]: [s. n.], 2010.

[12] 林果园,贺 珊,黄 皓,等. 基于行为的云计算访问控制安全模型[J]. 通信学报,2012,33(3):59-66.

[13] 王正飞. 云中数据正确性和完整性的高效验证[J]. 计算机工程与科学,2012,34(4):167-170.

[14] 陈 康,郑纬民. 云计算:系统实例与研究现状[J]. 软件学报,2009,20(5):1337-1348.

[15] Juels A, Kaliski J B S. Proofs of retrievability for large files[C]//Proceedings of the 14th ACM conference on computer and communications security. New York, NY, USA: ACM, 2007: 584-597.

[16] Kumar R S, Saxena A. Data integrity proofs in cloud storage[C]//Proc of third international conference on communication systems and networks. Bangalore: IEEE, 2011: 1-4.

[17] Yang Kan, Jia Xiaohua. Data storage auditing service in cloud computing: challenges, methods and opportunities[J]. World Wide Web, 2012, 15(4): 409-428.

[18] Wang Cong, Wang Qian, Ren Kui. Privacy-preserving public auditing for data storage security in cloud computing[C]//Proceedings of the 29th conference on information communications. Piscataway, NJ, USA: IEEE Press, 2010: 525-533.

[19] 田 敬,代亚非. P2P 持久存储研究[J]. 软件学报,2007,18(6):1379-1399.

一种面向云的大数据完整性检测协议

作者： 张也弛， 周文钦， 石润华， ZHANG Ye-chi， ZHOU Wen-qin， SHI Run-hua
作者单位： 安徽大学 计算智能与信号处理教育部重点实验室，安徽 合肥 230039；安徽大学 计算机科学与技术学院，安徽 合肥 230601

刊名： 计算机技术与发展 ISTIC
英文刊名： Computer Technology and Development
年，卷(期)： 2014(9)