JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 19, No. 8 : 1818~1824 Aug. 2015

클라우드 컴퓨팅에서 동적 데이터 무결성을 위한 개선된 감사 시스템

김태연[1] · 조기환[2*]

A Novel Auditing System for Dynamic Data Integrity in Cloud Computing

Tae-yeon Kim[1] · Gi-hwan Cho[2*]

[1]Department of Computer and Information, Seonam University, Chungnam 31556, Korea
[2*]Division of Computer Science and Engineering, Chonbuk National University, Jeonbuk 54896, Korea

요 약

응용 프로그램과 데이터, 파일 스토리지를 위한 동적 확장이 가능한 하부구조를 제공하는 응용으로서 클라우드 컴퓨팅에 대한 관심이 날로 증가하고 있다. 그러나 데이터 보호 측면에서 신뢰성 없는 원격 서버가 다양한 문제들을 야기할 수 있다. 그 중 사용자의 데이터를 고의적 또는 무의식적으로 연산(수정, 삽입, 삭제)을 하거나 데이터의 무결성을 감사하는 과정에서 자신의 잘못을 감추기 위해 거짓 정보를 제공할 수 있다. 본 논문에서는 서버가 악의적인 행위를 했는지를 확인할 수 있는 새로운 데이터 감사 시스템을 제안한다. 그리고 시스템의 성능과 보안 분석을 통해 제안된 구조가 클라우드 컴퓨팅 환경에 적합함을 증명한다.

ABSTRACT

Cloud computing draws attention as an application to provide dynamically scalable infrastructure for application, data and file storage. An untrusted remote server can cause a variety of problems in the field of data protection. It may process intentionally or involuntarily user's data operations(modify, insert, delete) without user's permission. It may provide false information in order to hide his mistakes in the auditing process. Therefore, it is necessary to audit the integrity of data stored in the cloud server. In this paper, we propose a new data auditing system that can verify whether servers had a malicious behavior or not. Performance and security analysis have proven that our scheme is suitable for cloud computing environments in terms of performance and security aspects.

키워드 : 클라우드 컴퓨팅, 감사, 데이터 무결성, 인증

Key word : cloud computing, auditing, data integrity, authentication

Received 08 June 2015, **Revised** 09 July 2015, **Accepted** 24 July 2015

*** Corresponding Author** Gi-Hwan Cho(E-mail:ghcho@chonbuk.ac.kr, Tel:+82-63-270-3437)
Division of Computer Science and Engineering, Chonbuk National University, Jeonbuk 54896, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2015.19.8.1818 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

소형 컴퓨터나 스마트패드, 스마트폰과 같은 모바일 기기의 사용이 보편화되고, 이러한 기기에서 나오는 데이터의 량이 폭발적으로 늘어남에 따라 클라우드 컴퓨팅에 대한 관심도이 날로 증가하고 있다. 하지만 많은 사용자들은 원격지 스토리지에 저장된 데이터의 비밀성과 무결성 등과 같은 보안 이슈들로 인해 스토리지의 사용을 주저하는 경향이 있다[1]. 클라우드 시스템 환경에서의 보안 취약점은 악의적인 공격자나 서버가 원격지에 저장된 데이터의 불법 수정이나 삭제하거나, 사용자의 데이터가 부주의로 삭제된 사실을 통보하지 않거나, 정당한 사용자가 요청한 데이터 연산(수정, 삽입, 삭제)을 실행하지 않거나, 데이터의 무결성을 감사(auditing)하는 과정에서 자신의 잘못을 감추기 위해 거짓 정보를 제공하는 행위 등이 있다. 따라서 정기적 또는 비정기적으로 원격지 스토리지에 저장된 데이터가 안전하게 관리되고 있는지를 감사하는 시스템이 필요하다. 이러한 무결성 감사는 모바일 기기의 특성을 고려하여 데이터의 비밀성은 최대한 보장되면서 사용자의 관여는 최소화하는 방식으로 이루어져야 한다.

지금까지 클라우드 컴퓨팅 환경에서 데이터의 무결성 감사를 위한 다양한 방법들이 제안되었다. 제안된 방법들의 대부분은 사용자와 검증자(TPA: Third Party Auditor, verifier), 신뢰하지 않은 CSP(Cloud Service Provider, server)간에 감사가 이루어진다는 가정을 두고 있다. Ateniese et al.[2]는 공개 검증의 특성을 가진 PDP(Provable Data Possession) 모델을 제안하였다. PDP 모델은 처리 비용을 최소화하기 위해 하나의 파일을 여러 개의 블록(block)으로 분할하여 저장하는 방식이다. 다시 말해서 서버 스토리지에 저장된 모든 데이터를 자신의 컴퓨터로 다운로드하지 않고 데이터의 무결성을 검증할 수 있는 확률적 증명 기법이다. 이를 위하여 여러 개의 블록 중에서 몇 개의 블록만을 랜덤하게 샘플링하여 검증하기 위해 RSA 기반 HLA (Homomorphic Linear Authenticators)을 적용하였다. 하지만 이 모델은 검증자(TPA)에게 임의로 샘플링된 블록들의 선형 조합을 노출하기 때문에 사용자의 데이터 프라이버시가 침해된다. 게다가 데이터 암·복호화에 따른 계산 비용을 무시할 수 없으며 동적인 데이터 연산을 지원하지 않는다.

G. Wang et al.[3]과 C. Wang et al.[4]는 Ateniese et al.[2]의 구조에 발생하는 몇 가지 문제들 보완하기 위해 MHT(Merkle Hash Tree)와 HLA을 기반으로 하는 DPDP(Dynamic PDP) 구조를 제안하였다. 이들의 구조는 완전한 동적 데이터 연산을 지원하지만 특정 데이터 블록을 찾기 위해서는 순차적으로 검색해야 하고, 특정 데이터 블록을 삽입하거나 삭제하기 위해서는 특정 블록들을 좌우로 이동($O(n)$)해야 하고, 블록의 삽입이나 삭제 연산을 하는 경우에 MHT의 단 노드들의 레벨 위치가 동일하지 않기 때문에 서버가 전송한 정보만을 가지고는 정확한 블록의 위치를 알아내기가 쉽지 않는 문제들이 있다. 이러한 문제를 해결하기 위해 Chen et al.[5]는 변형된 DMHT(Dynamic MHT)를 적용하여 데이터 블록을 효율적으로 관리할 수 있는 동적 데이터 무결성 검사 프로토콜을 제안하였다.

최근에는 멀티 클라우드 환경에서 사용자 파일을 분산시켜 저장하는 방법이나 하나의 파일만을 감사하는 것이 아니라 여러 개의 파일을 동시에 감사하는 방법들이 제안되었다[6,7]. 하지만 위에서 기술한 모든 구조는 서버인 CSP들은 고성능이지만 신뢰할 수 없는 노드이고, 검증자인 TPA는 고성능이면서 전적으로 신뢰하는 노드라는 가정을 두고 있다. 그러나 모든 사용자들의 데이터와 감사에 필요한 검증 정보들을 신뢰하지 않는 서버들로 하여금 관리하도록 하는 것은 타당하지 않다. 이러한 구조에서 발생할 수 있는 문제점들은 다음과 같다. 첫째, 사용자가 서버에 저장된 자신의 데이터에 대해 연산을 요청했음에도 불구하고 제대로 실행을 하지 않거나 잘못 실행했을 경우 검증자가 감사에 필요한 정보를 요청하면 과거의 데이터를 전송해서 데이터 무결성 감사를 무사히 통과할 수 있다. 특히, 수정 연산만을 한 경우에 그 가능성은 훨씬 높아진다. 둘째는 하나의 파일을 여러 서버에 분산시켜 저장할 때 특정 서버에 하나의 블록만이 저장될 수 있다. 이 경우에 데이터 무결성 감사를 하는 과정에서 중간 노드인 관리자나 TPA는 서버가 전송한 메시지를 분석하여 실제 데이터를 추출하는 문제가 발생할 수 있다.

따라서 본 논문에서는 서버가 악의적인 행위를 했는지를 확인할 수 있는 새로운 데이터 감사 시스템을 제안한다. 본문의 구성은 다음과 같다. 1장 서론에 이어 2장에서는 관련연구를 기술한다. 3장과 4장에서는 본 논문에서 개선된 데이터 무결성 감사 프로토콜과 보안 분

석에 관해 설명한다. 마지막으로 5장에서는 결론과 향후 연구방안에 대하여 기술한다.

Ⅱ. 관련연구

2.1. 클라우드 컴퓨팅 구조

본 논문에서 제안된 클라우드 컴퓨팅 구조는 (그림 1)과 같이 사용자들과 서버들, TPA, 관리자(manager)로 구성된다. 사용자는 다중 서버에 대규모 데이터 파일을 분산 저장해 두고, 필요할 때마다 저장된 데이터를 읽어오거나 연산을 요청하는 노드이다. 대용량의 스토리지가 장착된 서버는 사용자들의 실제 데이터를 저장하고 관리하는 노드이다. 검증자인 TPA는 서버에 저장된 데이터의 무결성을 정기적 또는 비정기적으로 감사하는 노드이다. TPA를 두는 이유는 사용자가 서버에 저장된 데이터가 안전하게 관리되고 있는지를 감사할 시간이나 자원이 없을 경우에 대신 위임받아 감사 기능을 수행하도록 하기 위함이다. 마지막으로 어느 서버가 어떤 사용자의 데이터를 관리하는지에 관한 정보가 외부 사용자(TPA, 권한이 없는 사용자, 침입자)들에게 노출되는 것을 방지하고 감사가 효율적으로 수행될 수 있도록 서버들중에서 특정 서버를 관리자로 지정하여 운영한다. 그리고 각 노드 간의 통신은 불법 접근을 할 수 없도록 안전한 채널을 이용한다고 가정한다.

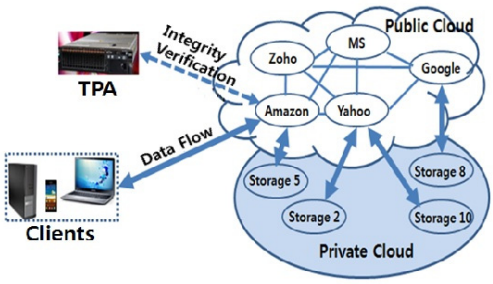

그림 1. 클라우드 컴퓨팅 구조
Fig. 1 Cloud Computing Structure

2.2. 이중 선형 맵(bilinear map)

이중 선형 맵은 $e : G \times G \rightarrow G_T$이다. 여기에서 G는 Gap Diffie_Hellman 그룹이고, G_T는 큰 소수 p의 또 다른 곱 순환 그룹이다[3]. 맵은 다음 3가지의 성질을 만족한다. (a) 계산 가능성: e을 계산할 수 있는 효율적인 계산 알고리즘이 존재한다. (b) 이중 선형: $e(h_1^a, h_2^b) = e(h_1, h_2)^{ab}$가 성립한다. 여기에서 모든 $h_1, h_2 \in G$이고 $a, b \in Z_p$이다. (c) 비 퇴보: $e(g, g) \neq 1$이다. 여기에서g는 G의 생성기이다.

2.3. DMHT

DMHT의 각 노드는 MHT와는 달리 서브 트리의 해쉬 값과 인덱스(서브 트리에 연결된 전체 단 노드의 수)를 나타내는 2개의 보조 정보를 관리한다[5]. 트리의 단 노드는 데이터의 해쉬값 T_i ($T_i = H(m_i)$)과 보조 정보 1로 지정한다. 즉, 단 노드의 값은 $h_1 = H(T_i \| 1)$이다. 여기에서 해쉬 함수 $H : \{0,1\}^* \rightarrow G$이다. 이와 같이 중간 노드의 값을 계산 한 다음에 최종적으로 근 노드 R의 좌측 자식 노드 a와 우측 자식 노드 b의 보조 정보인 해쉬 값과 인덱스가 각각 (h_a, n_a)와 (h_b, n_b)인 경우에 노드w의 해쉬 값은 $h_R = H(h_a \| n_a \| h_b \| n_b)$이고 인덱스는 $n = n_a + n_b$가 된다. 그리고 DMHT에 대한 보조 정보 AAI(Ω_i)는 단 노드 i로부터 근 노드까지의 경로를 나타낸 것으로 $\Omega_5 = \langle h(x_6,1,r), h(x_7,1,r), (h_a,4,l) \rangle$인 경우에 트리내의 노드 x_5와 x_6, x_7, x_a의 위치와 자식 노드의 연결된 단 노드의 수를 알 수 있다. 여기에서 r과 l은 각각 우측 노드와 좌측 노드라는 것을 나타낸다. 그리고 인덱스 정보를 사용하기 때문에 DMHT내의 특정 노드를 찾는데 더 효율적이다.

Ⅲ. 개선된 데이터 무결성 감사 프로토콜

3.1. 초기화 단계

사용자 A는 파일 $File$을 $n(\geq 2)$개의 블록 $(m_1, m_2, ..., m_n)$으로 분할하고, $KeyGen(1^k)$을 통해서 랜덤 키 쌍(sk_A, pk_A)과 $\alpha_A \leftarrow Z_P$, $\beta_A \leftarrow g^{\alpha_A}$을 생성하여 (sk_A, α_A)은 개인키로 관리하고 (pk_A, β_A)은 공개키로 사용한다. 여기에서 $m_i \in Z_p$이고 p는 큰 소수이다. 그리고 하나의 랜덤 값 $u (\leftarrow G)$과 현재 시간 t을 개인키로 암호화하여 파

일의 태그 $Tag_{file-name}(=F_{name}\|n\|u\|t)\|\|\left[F_{name}\|n\|u\|t\right]_{sk_A}$을 생성한다. 그 다음에 사용자는 파일의 각 블록 $m_i\,(i=1,2,\cdots,n)$에 대한 서명 $\sigma_i\left(=H\left(m_i \bullet u^{m_i}\right)^{\alpha_A}\right)$을 생성한다. 이 서명들의 집합을 $\Phi_{file-name}=\{\sigma_i\}, 1\le i\le n$로 표시한다. 그리고 DMHT의 구조를 기반으로 하여 $DMHT_{file}$을 생성한 다음, 개인키 α_A을 사용하여 근 노드 값(R)에 대한 서명 $File_R(=(H(R))^{\alpha_A})$을 생성한다.

사용자는 위와 같은 초기 작업을 완료하면 TPA에게 F_{name}와 $File_R$, t을, 관리자에게 $Tag_{file-name}$와 $DMHT_{file}$을 안전한 채널을 통해 전송한다. 그리고 파일을 분산 저장하기 위해 원하는 서버들에게 F_{name}과 2개 이상의 데이터 블록들($\{m_i, \sigma_i\}$)을 전송하고, 자신의 로컬 스토리지에는 $Tag_{file-name}$만을 저장해 두고, 나머지 모든 정보들은 삭제한다. 사용자가 새로운 블록을 삽입하거나 저장된 블록을 수정, 삭제하는 동적 데이터 연산을 수행하는 경우에도 위에서 기술한 방식과 동일하게 처리한다.

3.2. 데이터 무결성 감사 단계

TPA는 감사하고자 하는 사용자의 파일명 F_{name}을 관리자에게 전송한다. 메시지를 받은 관리자가 해당하는 파일 태그($Tag_{file-name}$)를 TPA에게 전송하면 TPA는 수신한 파일 태그를 사용자의 공개키로 복호화하여 파일 태그의 내용을 검사한다. 태그의 내용이 타당하지 않으면 사용자에게 보고하고, 그렇지 않으면 도전 메시지(F_{name}, $Q=(i,v_i)_{s_1\le i\le s_c\le n}$)를 생성하여 관리자에게 전송한다. 여기에서$v_i\subseteq Z_p$, $|Q|\le n$이다. 도전 메시지를 받은 관리자는 모든 서버들에게 방송한다. F_{name}에 해당하는 블록 i들을 자신의 스토리지에 저장하고 있는 서버는 Algorithm 1과 같이 $\mu_i^{'}$와 $\sigma_c^{'}$을 계산한다. 기호 $m_i\Rightarrow P_c$는 블록 $m_{i\in[s_1,s_c]}$이 서버 P_c에 저장되어 있다는 것을 의미한다.

Algorithm 1: computes $\mu_i^{'}$ and $\sigma_c^{'}$

$k=|m_i\Rightarrow P_c|$
if $k==1$ then
  $v_{s_c}^{'}=v_{s_c}$
  $\exists\, m_j\Rightarrow P_c, i\ne j$
  $\mu_c^{'}=\sum_{i\in[s_1,s_c]} v_i m_i + v_{s_c}^{'} m_j,$
  $\sigma_c^{'}=\prod_{i\in[s_1,s_c]} \sigma_i^{v_i} \bullet \sigma_j^{v_{s_c}^{'}}$
  $\mu_c^{'}=0, \sigma_c^{'}=1, i=s_1$
else
  $v_{s_c}^{'}=0$
  while($k>=1$) do
    if($\exists\, m_i\Rightarrow P_c$) then
      $\mu_c^{'}=\sum_{i\in[s_1,s_c]} v_i m_i + v_{s_c}^{'} m_j,$
      $\sigma_c^{'}=\prod_{i\in[s_1,s_c]} \sigma_i^{v_i} \bullet \sigma_j^{v_{s_c}^{'}}$
      $k=k-1$
    end if
    $i=i+1$
  end while;
end if

각 서버 P_c는 $\{\mu_c{}', \sigma_c^{'}, \{v_i, H(m_i)\}, H(m_j)\}$나 $\{\mu_c{}', \sigma_c^{'}, \{v_i, H(m_i)\}_{i\in[s_1,s_c]}\}$ 을 관리자에게 전송한다. 관리자는 서버들로부터 s_c개의 정보를 모두 수신하면 $\mu=\sum_{P_c\in P}\mu_c^{'}\in Z_P$와 $\sigma=\prod_{P_c\in P}\sigma_c^{'}\in G$을 계산한다. 그리고 응답 메시지로 $\{\mu, \sigma, \{H(m_i), \Omega_i\}_{s_1\le i\le s_c}, \{H(m_j)\}^{*}_{j\notin[s_1,s_c]}\}$을 TPA에게 전달한다. Ω_i는 단 노드 i로부터 근 노드까지의 경로를 나타낸 노드 정보들의 리스트이다.

Algorithm 2 : verifies data integrity

$k=1, i=s_1$
while($i<=s_c$) do
  $h(R^{'}) : H(m_i)\,\text{and}\,\Omega_i$
  $k=k+1, i=s_k$
end while;
$H(R)=h(R^{'})$

if $e(File_R, g) \neq e(H(R), \beta_A)$ then
  notify the integrity violations
else
  if not existed $H(m_j)$
    $A = \prod_{i=s_1}^{s_c} H(m_i)^{v_i}$
    if $e(\sigma, g) = e(A \cdot u^{\mu}, \beta)$ then
      verification terminated
    else
      notify the integrity violations
    end if
  else
    $A = \prod_{i=s_1}^{s_c} H(m_i)^{v_i}$, $B = \prod_{P_c \in P}^{j^*} H(m_j)^{v_{s_c}}$
    if $e(\sigma, g) = e(A \cdot B \cdot u^{\mu}, \beta)$ then
      verification terminated
    else
      notify the integrity violations
    end if
  end if
end if;

TPA는 Algorithm 2와 같이 수신한 메시지내의 $\{H(m_i), \Omega_i\}_{s_1 \le i \le s_c}$을 이용하여 근 노드의 값에 대한 해쉬 값인 $H(R)$을 계산한 다음에 $e(File_R, g) \stackrel{?}{=} e(H(R), \beta_A)$ 의 관계가 성립하는지를 확인한다. 성립이 확인되면 서버에 대한 인증이 성공한 것이고 그렇지 않으면 감사 절차를 중지하고 사용자에게 통보한다.

$$e(\sigma, g) \stackrel{?}{=} e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{P_c \in P}^{j^*} H(m_j)^{v_{s_c}} \cdot u^{\mu}, \beta) \quad (1)$$

그리고 서버에 대한 인증이 성공하면 식 (1)을 비교하여 데이터의 무결성을 검사한다. 만일 좌우 값이 서로 같으면 서버에 저장된 데이터의 무결성이 보장된 것으로 간주한다. 그렇지 않으면 무결성이 보장되지 않음을 사용자에게 통보한다. 식 (1)은 아래와 같이 (2)와 (3)이 서로 일치함이 증명된다.

$$\begin{aligned}
&e(\sigma, g) \\
&= e(\prod_{P_c \in P} \sigma_c', g) \\
&= e(\prod_{P_c \in P} (\prod_{i \in [s_1, s_c]} \sigma_i^{v_i} \cdot \sigma_j^{v_{s_c}}), g) \\
&= e(\prod_{P_c \in P} (\prod_{i \in [s_1, s_c]} \sigma_i)^{v_i} \cdot \prod_{P_c \in P}^{j^*} \sigma_j^{v_{s_c}}), g) \\
&= e(\prod_{P_c \in P} (\prod_{i \in [s_1, s_c]} H(m_i \cdot u^{m_i})^{\alpha_A})^{v_i} \cdot \\
&\quad \prod_{P_c \in P}^{j^*} (H(m_j \cdot u^{m_j})^{\alpha_A})^{v_{s_c}}, g) \\
&= e(\prod_{P_c \in P} (\prod_{i \in [s_1, s_c]} H(m_i \cdot u^{m_i})^{v_i}) \cdot \\
&\quad \prod_{P_c \in P}^{j^*} (H(m_j \cdot u^{m_j})^{v_{s_c}}), g^{\alpha_A}) \\
&= e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{i=s_1}^{s_c} u^{v_i m_i} \cdot \\
&\quad \prod_{P_c \in P}^{j^*} H(m_j)^{v_{s_c}} \cdot \prod_{P_c \in P}^{j^*} u^{v_{s_c} m_j}), g^{\alpha_A})
\end{aligned} \quad (2)$$

$$\begin{aligned}
&e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{P_c \in P}^{j^*} H(m_j)^{v_{s_c}} \cdot u^{\mu}, \beta) \\
&= e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{j=?}^{j^*} H(m_j)^{v_{s_c}} \cdot u^{\sum_{P_c \in P} \mu_c'}, \beta) \\
&= e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{j=?}^{j^*} H(m_j)^{v_{s_c}} \cdot \\
&\quad (u^{\sum_{P_c \in P} (\sum_{i \in [s_1, s_c]} v_i m_i + v_{s_c} m_j)}), \beta) \\
&= e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot u^{\sum_{P_c \in P} \sum_{i \in [s_1, s_c]} v_i m_i} \cdot \\
&\quad \prod_{j=?}^{j^*} H(m_j)^{v_{s_c}} \cdot u^{\sum_{P_c \in P} v_{s_c} m_j}, \beta) \\
&= e(\prod_{i=s_1}^{s_c} H(m_i)^{v_i} \cdot \prod_{i=s_1}^{s_c} u^{v_i m_i} \cdot \\
&\quad \prod_{P_c \in P}^{j^*} H(m_j)^{v_{s_c}} \cdot \prod_{P_c \in P}^{j^*} u^{v_{s_c} m_j}, g^{\alpha_A})
\end{aligned} \quad (3)$$

Ⅳ. 성능 및 보안 분석

4.1. 성능 분석

기존에 제안된 대부분의 구조들[3-7]은 데이터 무결성 감사에 필요한 정보들을 서버가 관리하도록 하지만 제안한 구조에서는 인증 정보인 $File_R$을 서버가 아닌 TPA가 관리하도록 한다. 멀티 클라우드 환경에서 여러 개의 파일을 동시에 감사하는 구조[7]에서 제안된 방식을 적용할 수 있으므로 기본적인 성능과 보안을 분석한다. 사용자의 id와 전체 사용자를 각각 u와 a, 파일명과 전체 파일의 수를 각각 f와 b, 파일 태그인 $Tag_{file-name}$을 t, 파일 인증 값인 $File_R$을 r, 파일 트리 구조인 $DMHT_{file}$을 d, 각 서버에 데이터 쌍 $\{m_i, \sigma_i\}_{i\in[s_1,s_c]}$이 저장된 수의 평균을 m이라고 가정한다. Sun[6]의 구조와 제안된 구조에서 TPA가 필요한 기억공간은 각각 $O(au+bt)$와 $O(au+bt+br)$이고, 관리자가 필요한 기억공간은 각각 $O(bt+br+bd)$와 $O(bt+bd)$이다. 반면에서 두 구조에서 서버가 필요한 공간은 $O(bf+m)$이다. 감사에 필요한 정보를 서버가 아닌 TPA가 관리함에 따라 TPA는 기억공간을 약간 더 필요하지만 관리자는 필요한 기억 공간이 줄어든다는 것을 알 수 있다. 하지만 각 서버는 사용자의 데이터를 저장할 때는 반드시 2개 이상의 블록을 저장해야 한다는 제약이 있는 구조이다. 그리고 통신비용은 관리자와 TPA 간에 교환되는 메시지의 수는 변함이 없지만 관리자가 마지막에 보낸 메시지의 크기가 인증 정보인 $File_R$의 항목만큼 작아진다는 차이점만 있다.

마지막으로 계산 비용은 두 구조에서 커다란 차이점은 없지만 서버가 관리하고 있는 블록을 관리자에게 전송했을 때 실제 블록 내용이 노출되는 것을 방지하기 위해 사용하는 블라인드(blinded) 데이터를 생성하는 과정이 Sun 구조에서는 필요하지만 제안된 구조에서는 이런 부과적인 처리 과정이 필요하지 않기 때문에 처리 시간이 줄어들 것으로 예상된다.

4.2. 보안 분석

기존의 구조들에서는 모든 사용자의 파일들과 감사에 필요한 검증 정보들을 관리자가 저장하고 관리하기 때문에 최신의 정보가 아닌 과거의 정보를 사용하여 무결성 감사를 통과할 수 있다. 그리고 Sun[6]을 제외한 몇몇 구조에서는 특정 서버에 저장된 데이터가 중간 노드인 관리자나 TPA에 노출되는 문제가 발생할 수 있다. 따라서 본 논문에서 제안한 구조에서는 TPA가 $File_R$을 직접 관리할 뿐만 아니라 파일 태그에 타임 스템프(t)을 첨가하기 때문에 서버들이 과거의 정보를 사용하여 감사를 통과하기가 어렵다. 또한 Sun[6]이 제안한 구조에서는 서버가 저장하고 있는 데이터의 노출을 방지하기 위해 블라인드 데이터를 사용하기 때문에 부수적인 정보를 전송해야 한다. 게다가 기존의 구조들에서는 선택된 블록에 대해서만 감사를 수행하지만 제안된 구조에서는 선택되지 않는 블록에 대한 무결성 감사를 할 수도 있기 때문에 더 강한 데이터 무결성을 보장할 수 있다.

Ⅴ. 결 론

본 논문은 클라우드 컴퓨팅 환경에서 데이터 보안과 프라이버시 보호의 측면에서 데이터 무결성을 감사하는 개선된 구조를 제안하였다. 감사에 필요한 정보를 서버가 아닌 TPA가 관리하도록 함에 따라 TPA의 기억공간이 더 필요하지만 과도한 오버헤드를 유발하지 않음을 확인되었다. 계산 비용과 통신비용에 있어서는 기존의 구조들과 큰 차이는 없다. 하지만 본 논문에서 재안한 구조는 각 서버가 사용자의 데이터를 저장할 때는 반드시 2개 이상의 블록을 저장해야 한다는 제약을 두고 있다.

보안 분석 측면에서, 인증 정보인 $File_R$의 저장 위치를 변경하고 파일 태그에 타임 스템프(t)을 첨가하여 신뢰할 수 없는 서버가 최신의 정보가 아닌 과거의 정보를 사용해서 무결성 감사를 통과하는 것을 어렵게 하였다. 그리고 특정 서버에 저장된 데이터가 관리자나 TPA에 노출되는 문제를 개선함에 따라 더 강한 데이터 무결성을 보장할 수 있다. 지금까지 시스템의 성능과 보안 분석 결과를 살펴본 결과 제안된 구조가 기본의 구조에 비해 더 우수함을 보여주고 있다. 향후 연구과제는 제안된 알고리즘을 실제 환경에 구현하여 성능 평가를 실시하는 것이다.

REFERENCES

[1] L. M. Kaufman, "Data Security in the World of Cloud Computing," *IEEE Security & Privacy,* vol 7, no. 4, pp.61-64, 2009.

[2] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and S. Song, "Provable Data Possession at Untrusted Stores," 14th ACM conference on Computer and Communication Security, pp.598-609, 2007.

[3] G. Wang, C. Wang, K. Ren, W. Lou, and J. Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp.847-859, 2011.

[4] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving Public Auditing for Secure Cloud Storage," *IEEE Transactions Computer,* vol. 62, no. 2, pp.862-375, 2013.

[5] L. Chen and H. Chen, "Ensuring Dynamic Data Integrity with Public Auditability for Cloud Storage," *2012 International Conference on Computer Science and Service System*, pp.711-714, 2012.

[6] X. Sun, L. Chen, Z. Xia, and Y. Zhu, "Cooperative Provable Data Possession with Stateless Verification in Multicloud Storage," *Journal of Computational Information Systems*, vol. 10, no. 8, pp.1-9, 2014.

[7] X. Liu and Y. Jiang, "Batch Auditing for Multi-client Dynamic Data in Multi-cloud Storage," *International Journal of Security and Its Applications*, vol. 8, no. 6, pp.197-210, 2014.

김태연(Tae-yeon Kim)

1988년 전남대학교 대학원 계산통계학과 졸업 (석사)
1996년 전남대학교 대학원 전산통계학과 졸업 (박사)
1996년 ~ 현재 서남대학교 컴퓨터정보학과 조교수
관심분야 : 네트워크 보안, 이동컴퓨팅, 네트워크 관리, 센서 네트워크

조기환(Gi-hwan Cho)

1987년 서울대학교 대학원 계산통계학과 졸업 (석사)
1996년 영국 Newcastle대학교 대학원 전산학과 졸업 (박사)
1987년 ~ 1997년 한국전자통신연구원 선임연구원
1997년 ~ 1999년 목포대학교 컴퓨터과학과 전임강사
1999년 ~ 현재 전북대학교 컴퓨터공학부 부교수
관심분야 : 이동컴퓨팅, 컴퓨터통신, 무선 네트워크 보안, 센서 네트워크, 분산처리 시스템