클라우드 스토리지에서 암호화된 데이터의 소유권 증명 기법

신영주, 김광조

한국과학기술원

Proof of Ownership for Encrypted Outsourced Data in the Cloud Storage

Youngjoo Shin and Kwangjo Kim

Korea Advanced Institute of Science and Technology

요 약

클라우드 스토리지 서비스는 IT 자원의 효율적 활용을 위해 데이터 중복 제거 기술을 사용한다. 데이터 중복 제거 기술이란 동일한 파일을 각각 저장하는 대신 파일의 논리적 링크만 저장하여 저장 공간과 네트워크 대역폭을 절약하는 기술을 말한다. 그러나, 데이터 중복 제거 기술은 저장 데이터의 기밀성과 비인가 사용자의 접근 제어 측면에서 보안 위협을 안고 있다. 본 논문은 신뢰할 수 없는 스토리지 서버로부터 아웃소싱 데이터의 기밀성을 보장하면서 비인가 사용자의 데이터 접근을 제어할 수 있는 효율적 데이터 중복 제거 방법을 제안한다.

I. 서론

빅 데이터의 규모가 폭발적으로 증가하면서 기존의 인터넷 기반 스토리지 기술을 클라우드 컴퓨팅 패러다임에 접목한 클라우드 스토리지 서비스가 최근 각광을 받고 있다. 이미 DropBox[1], Mozy[2]와 MemoPal[3]등 상용 서비스가 존재하며, 최근 구글의 G-드라이브[4], 마이크로소프트의 스카이드라이브[5] 등 새로운 클라우드 스토리지 서비스도 빠른 속도로 생겨나고 있다.

대부분의 클라우드 스토리지 서비스는 스토리지 저장 용량과 네트워크 대역폭 등 IT 자원을 효율적으로 사용하기 위해 데이터 중복 제거 기술을 사용하고 있다[6]. 데이터 중복 제거 기술이란 여러 사용자가 동일한 파일을 스토리지에 아웃소싱 하게 되는 경우 물리적으로 하나의 파일을 제외하고 나머지는 논리적인 링크로 대체하는 기술을 말한다. IT 자원의 효율적 이용이 금전적 비용의 절감과 직결되면서 데이터 중복 제거 기술은 클라우드 스토리지 서비스 사업자에게는 반드시 필요한 기술로 받아들여지고 있다 [6].

그런데 데이터 중복 제거 기술은 클라우드 스토리지 서비스에 두 가지 측면에서 심각한 보안 위협을 초래하는 것으로 알려져 있다 [7][8]. 첫째, 신뢰할 수 없는 스토리지 서버로부터 아웃소싱 데이터의 기밀성 보장이 어렵게 된다. 데이터 암호화는 데이터 중복 제거 기술과 동시에 사용될 수 없기 때문이다[7]. 둘째, 데이터 중복 제거 기술을 이용하여 비인가 사용자가 파일을 획득 할 수 있는 보안 취약점이 있다[8]. 이는 사용자가 제시한 파일의 해쉬 값만을 가지고 해당 파일의 실제 소유 여부를 결정하기 때문이다.

데이터 중복 제거 기술은 효율적인 IT 자원의 활용을 위해 반드시 필요하므로 이러한 보안 취약점이 해결되어야 한다. 최근에 데이터 중복 제거 기술에서 야기되는 보안 문제를 해

결하기 위한 몇 가지 방법들이 제안 되었다[7][8][9]. 그러나 앞서 기술한 두 가지 이슈를 효율적으로 해결한 방법은 아직 제시되지 않고 있다.

본 논문에서는 효율적이고 안전한 데이터 중복 제거 기술을 위한 방법을 제안한다. 이 방법은 두 가지 측면에서 안전성을 제공한다. 첫째, 아웃소싱 데이터의 기밀성을 보장하기 위해 데이터 파일을 결정적 대칭키 암호 알고리즘으로 암호화하는 방법을 제시한다. 제시한 암호 방식은 동일한 파일에 대해 항상 동일한 암호문을 생성하므로 스토리지 서버는 암호화된 파일에 대해서도 데이터 중복 제거 실행이 가능하다. 이 방법은 동일한 파일 소유자 간에 그룹 키 관리 시스템 등 별도의 키 관리 메커니즘을 요구하지 않아 효율성 측면에서 부가적인 이점을 제공한다. 둘째, 파일을 소유하고 있지 않는 사용자가 해당 파일의 소유권을 주장하는 것을 방지할 수 있는 방법을 제시한다. 구체적으로, 작은 크기의 파일 해쉬 값 대신 실제 보유하고 있는 파일의 일부를 서버에게 제시함으로써 사용자가 파일에 대한 소유권을 서버에게 증명할 수 있는 소유권 증명 프로토콜(Proof of Ownership)을 제시한다.

본 논문의 구성은 다음과 같다. II장에서 배경이 되는 내용들을 기술하고 III장에서 본 논문의 제안 방법을 기술한다. 그리고 IV장에서 제안 방법의 성능 및 안정성을 분석하며 V장에서 결론을 맺는다.

1.1 관련 연구

Storer 등은 스토리지 서버가 암호화된 파일에서 데이터 중복 제거를 실행할 수 있는 구체적 방법을 처음으로 제안하였다[7]. Storer 등의 방법은 스토리지 서버로부터 아웃소싱 데이터의 기밀성 보장이 가능하나 파일의 해시 값을 소유권 증명에 사용하기 때문에[8] 이미 지적한 보안 취약점을 해결하지 못하고 비인가 사용자의 파일 접근을 막지 못한다. 한편 [7]의 방법은 별도의 공개키 시스템을 필요로 한다.

Halevi 등[8] 과 Wee 등[9]은 Merkle 트리 기반의 소유권 증명 프로토콜을 제안하였다. 이들의 제안 방법은 스토리지 서버에 저장된 데이터의 무결성 및 복구 가능성을 검증하기 위한 복구 증명 프로토콜 (Proof of Retrievability)[10] 과는 검증자와 증명자의 역할이 바뀐 것을 제외하고는 유사하다. [8]의 방법은 암호화된 파일에서는 소유권 증명이 불가능하며, [9]의 방법은 암호화된 상태에서도 사용이 가능하지만 그룹 키 관리 시스템 등 별도의 키 공유 프로토콜이 필요하고 프로토콜 실행 시 많은 양의 연산이 요구된다.

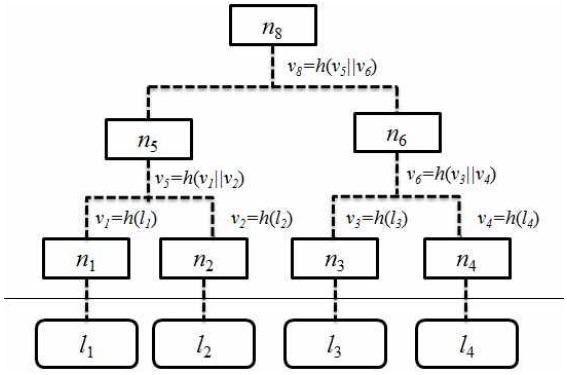
II. 배경 지식

이 장에서는 본 논문의 제안 방법을 구현하는 도구로 사용된 몇 가지의 배경 지식들에 대한 내용을 간략히 기술한다.

2.1 Merkle 트리

Merkle 트리는 원격에 있는 대용량의 데이터 버퍼를 전부 다운로드 하지 않고서도 데이터 버퍼에 대한 무결성을 검증할 수 있도록 고안된 자료구조이다[11]. Merkle 트리를 구성하는 방법은 우선 버퍼를 고정 길이를 갖는 블록들로 나눈 뒤, 인접한 두 블록 쌍에 대한 해쉬 값을 계산한다. (여기서 블록의 개수는 2^h 개로 가정한다. 2^h 개가 아닌 경우는 패딩 등의 방법으로 보정이 가능하다.) 이 해쉬 값은 다시 인접한 다른 해쉬 값과 함께 해쉬 알고리즘의 입력으로 들어가 새 해쉬 값이 계산된다. 이 과정이 반복되어 최종적으로 하나의 해쉬 값이 남게 된다. 이러한 과정은 결국 리프 노드는 파일의 블록들이고 중간 노드는 해쉬 값을 갖는 높이 h 의 이진 트리를 생성하게 된다.

그림 1은 4개의 블록 $l_1,...,l_4$ 으로 이루어진 Merkle 트리의 예를 보여준다. 리프 노드 n_1, ...n_4 는 각각 블록 $l_1,...,l_4$ 의 해쉬 값 $v_1=h(l_1),...,$ $v_4=h(l_4)$ 을 가지고 있다. 중간 노드 n_5 는 n_1 과 n_2 의 해쉬 값을 다시 해쉬 한 값



(그림 1) 4개의 블록을 갖는 Merkle 트리

$v_5=h(v_1||v_2)$를 가지고 있으며 n_6 와 루트 노드 n_8 도 마찬가지 방법으로 해쉬 값 v_6 과 v_8을 계산하여 갖게 된다.

$MT_{h,b}(X)$ 는 데이터 버퍼 X를 길이 b 비트의 블록으로 나누고, 해쉬 함수 h를 사용하여 생성한 Merkle 트리를 나타낸다. 리프 노드 l에 대해, l 에서 루트 까지의 경로에서 그 형제 노드들을 포함한 경로를 형제 경로라고 부른다. 예를 들어, (그림 1)에서 l_1 의 형제 경로는 {n_1, n_2, n_5, n_6, n_8} 이다.

2.2 소유권 증명 프로토콜

소유권 증명 프로토콜 (Proof of Ownership)은 공통의 입력으로 주어지는 데이터 파일 F를 대상으로 하여 검증 알고리즘 V 와 증명 알고리즘 P 상호간에 실행되는 프로토콜이다. 프로토콜이 실행되기에 앞서 V 는 F를 전 처리하여 F를 대신할 수 있으면서 F의 크기보다 작은 요약 문자열 (또는, 해쉬 값) v 를 생성해야한다. P 는 전체 파일 F를 가지고 있고, V 는 F 에 대한 요약 문자열 v를 가진 상태에서 상호간에 프로토콜이 실행되며, 프로토콜의 실행 결과로 V 는 "**accept**" 또는 "**reject**"을 출력하게 된다.

좀 더 엄밀하게 기술하면, 소유권 증명 프로토콜 P=(S, Π) 는 검증 알고리즘 V 와 증명 알고리즘 P 사이의 상호 프로토콜 Π와 F 의 요약 함수 S(·) 로 구성된다. P 는 다음의 두 가지 조건을 만족한다.

- 요약 함수 S(·) 와 프로토콜 Π 는 안정성 파라미터 l 에 대해 효율적이다.
- 모든 입력 파일 F ∈ {0, 1}* 과 모든 안정성 파라미터 l 에 대해 다음과 같은 확률식이 성립 한다.

Pr{ (Π(P(F, 1^l) ↔ V(S(F, 1^l))) ⇒ **accept** } ≥ 1 − $neg(1^l)$,

여기서, $neg(1^l)$ 은 안정성 파라미터 l 에 대해 무시할 수 있는 함수(negligible function)이다.

2.3 에러 정정 코드

[n, k, d] 에러 정정 코드는 다음과 같은 두 개의 속성을 갖는 코드를 말한다.

- 인코딩 함수 E : {0, 1}^{k} → {0, 1}^{n} 을 이용하여 k 비트 데이터를 n 비트 코드워드로 변환한다.
- 서로 다른 두 값 x, y ∈{0, 1}^{k} 에 대해서 E(x) 와 E(y) 의 해밍 디스턴스는 최소 d 보다 크다.

위의 속성을 만족하는 [n, k, d] 에러 보정 코드는 d-1 만큼의 손실이 발생하여도 원래의 데이터를 복구할 수 있다.

III. 소유권 증명 기법

본 논문에서는 신뢰할 수 없는 클라우드 스토리지 서버에 저장된 데이터에 대한 안전한 소유권 증명 기법을 제안한다. 본 제안 방법에서 클라우드 스토리지 서버와 데이터 소유자간에 실행되는 프로토콜은 암호화된 데이터를 기반으로 동작한다. 데이터의 암호화는 신뢰할 수 없는 서버에 데이터가 노출되는 것을 방지하며, 동시에 소유권 증명 프로토콜은 접근 권한이 없는 사용자가 데이터 중복 제거 기술의 취약점을 이용하여 불법으로 데이터에 접근하는 것을 막는 것이 가능하다.

3.1 시스템 모델

본 논문의 제안 방법이 동작하기 위한 시스템 구성과 공격 모델, 그리고 안정성 요구사항을 이 장에서 기술한다.

제안 방법은 다음과 같은 개체들로 이루어진 시스템에서 동작한다.

- 스토리지 서버 : 스토리지 서버는 데이터 소유자가 아웃소싱한 데이터 파일을 저장하여 보관한다. 일반적으로 스토리지 서버와 데이터 소유자(또는 사용자)는 서로 다른 도메인에 속하여 있으며 스토리지 서버는 신뢰할 수 없다고 가정한다.
- 데이터 소유자 : 데이터 소유자는 스토리지 서버에 데이터 파일을 직접 업로드 하였거나, 업로드하지 않고 소유권 증명을 통해 데이터에 대한 소유권을 가지고 있는 사용자이다.
- 사용자 : 사용자는 보통 데이터 소유자를 포함하여 클라이언트의 역할로서 스토리지 서버를 이용하는 모든 사람을 통칭한다.

본 논문에서는 2가지의 공격 모델을 가정한다. 첫째, 스토리지 서버는 사용자와의 프로토콜을 정직하게 실행하지만 데이터 소유자가 아웃소싱한 데이터 파일의 내용에 관심이 있다고 가정한다(Honest-but-curious 모델). 즉, 스토리지 서버는 주어진 컴퓨팅 자원 내에서 데이터 소유자가 저장한 파일의 내용을 알아내고자 노력한다. 둘째, 일부 사용자는 소유권이 없는 데이터에 대해서 그 내용에 관심이 있으며 그 데이터를 획득하고자 노력한다고 가정한다. 사용자(공격자)는 스토리지 서버에 무단으로 접근하거나 데이터 소유자와 서버사이의 프로토콜을 모니터링하면서 최대한 정보를 얻어내려고 노력한다.

이와 더불어 본 논문에서는 스토리지 서버와 사용자간의 공모는 이루어지지 않는다고 가정한다. 한편, 사용자(공격자)는 스토리지 서버의 구현 취약점등을 이용하여 서버 내부에 침투할 수 있으며 네트워크 대역폭의 제약 등을 고려하여 서버로부터 최대 s 비트 크기만큼의 데이터를 획득할 수 있다고 가정한다.

3.2 설계 목표

앞 서 기술한 시스템 구성과 공격 모델에 대하여 다음과 같은 안정성을 제공하는 소유권 증명 프로토콜 P=(S, Π) 을 설계하는 것을 목표로 한다.

- *스토리지 서버로부터의 기밀성* : 스토리지 서버는 데이터 소유자가 저장한 파일의 내용을 접근할 수 없어야한다.
- *비인가 사용자에 대한 접근 제어* : 데이터 소유권이 없는 사용자는 스토리지 서버에 저장된 데이터에 접근할 수 없어야 한다. 즉, t 비트의 최소 엔트로피(min-entropy)를 갖는 데이터 파일 F 에 대해 최대 s 비트만큼 (t>s) 획득한 공격자가 파일 F 전체를 가지고 있다고 주장하여 스토리지 서버를 속일 수 있는 확률이 s 에 대해 무시되는 수준이어야 한다.

3.2 파일 암호화 방식

제안 프로토콜 P 는 암호화된 데이터 파일을 가지고 실행된다. 본 논문에서 사용된 파일 암호화 방식을 이 장에서 기술한다. 이에 앞서 몇 가지 표기를 다음과 같이 사용하기로 한다. 대칭키 암호알고리즘 $E=(ENC, DEC)$ 가 있다고 하자. E 는 AES, DES 또는 SEED 등의 대칭키 암호알고리즘이 될 수 있으며 키의 크기 len 은 현재 비교적 안전하다고 인정되는 수준 (예. len=128)에서 정할 수 있다 H 는 다음과 같이 정의되는 해쉬 함수이다.

$$H : \{0,1\}^* \rightarrow \{0,1\}^{len}.$$

F 는 임의의 크기를 갖는 데이터 파일의 이진 표기라고 가정하자. 즉, $F \in \{0,1\}^*$ 이다. 암호알고리즘 E 에서 하나의 블록의 길이를 b 라고 할 때 F 는 다음과 같이 길이가 b 인 n 개의 블록 F_i $(1 \le i \le n)$로 나눌 수 있다.

$$F=\{F_1,F_2,...,F_n\},\ |F_i|=b.$$

여기서 마지막 블록 F_n 의 남는 부분은 0 으

로 패딩된다. 제안 프로토콜 P를 위한 파일 암호 알고리즘은 다음과 같이 정의 한다.

정의1 제안 프로토콜 P를 위한 파일 암호알고리즘 *FE=(FENC, FDEC)* 는 2개의 알고리즘으로 구성된다.

FENC(F) : 이 알고리즘은 파일 F를 n 개의 블록 F_i, 여기서 $|F_i|=b$ 로 나누고 각 블록에 대해 다음과 같이 암호화한다.

$$C_i = ENC(\ H(F),\ F_i\ ).$$

알고리즘의 계산 결과로 $C=\{C_1,...,C_n\}$을 반환한다.

FDEC(F) : 이 알고리즘은 암호화된 파일 C를 다시 원래의 파일로 복호화 한다. 즉, 암호화된 블록 C_i 에 대해,

$$F_i = DEC(\ H(F),\ C_i\ )$$

를 계산하고 그 결과로 $F = \{F_1,...,F_n\}$ 을 반환한다.

3.3 Merkle 트리 기반 제안 프로토콜

본 제안 방법은 다음과 같이 두 개의 프로토콜로 구성된다.

데이터 업로드 프로토콜 : 사용자가 데이터 파일 F를 스토리지 서버에 업로드 하고자 할 때 이 프로토콜을 실행한다. (즉, 스토리지 서버는 F를 아직 가지고 있지 않은 상태이다.)

1. 사용자는 파일 F를 다음과 같이 암호화 한다.
$$C=FENC(F),\ F\in\{0,\ 1\}^*.$$
2. 사용자는 $[M,\ M',\ a]$ 에러정정코드를 이용하여 암호화된 파일 C를 인코딩 한다. (C 의 크기는 M 이며, 인코딩 함수는 $E : \{0,\ 1\}^M \rightarrow \{0,\ 1\}^{M'}$ 이라 가정한다.)
3. 사용자는 인코딩 된 값 $X=E(C)$를 스토리지 서버에게 전송한다.
4. 스토리지 서버는 X 에 대해 $MT_{h,b}(X)$ Merkle 트리를 생성 하고 루트 노드의 값 v와 함께 X 를 스토리지 공간에 저장한다.

데이터 중복 확인 프로토콜 : 스토리지 서버에 이미 저장된 파일 F 가 존재하며, 사용자가 동일한 파일 F를 업로드 하고자 할 때 이 프로토콜이 실행된다. (사용자는 프로토콜 실행 전에 파일 F 에 대한 암호화 및 인코딩을 앞서 기술한 바와 같이 실행하였다고 가정한다.)

1. 스토리지 서버는 X (=E(C)) 의 블록 $l_1,...l_n$ 중에서 임의로 u 개의 블록을 선택하고, 선택된 블록의 인덱스 집합 U를 사용자에게 전달한다.
2. 사용자는 X 에 대한 Merkle 트리 $MT_{h,b}(X)$ 를 생성하고, 모든 $l_i \in$U 에 대해 l_i 부터 시작되는 형제 경로들을 계산하여 스토리지 서버에게 전달한다.
3. 스토리지 서버는 사용자로부터 수신한 형제 경로 집합을 가지고 리프 노드에서부터 차례로 해쉬 계산을 하여 루트 노드의 해쉬 값을 계산한다. v를 루트 노드 해쉬 값과 비교 하여 두 값이 서로 동일하면 "**accept**"를 출력하고, 그렇지 않으면 "**reject**"를 출력한다.

사용자가 데이터 업로드 프로토콜을 통하여 파일을 업로드 하였거나, 데이터 중복 확인 프로토콜을 실행하여 "**accept**" 결과를 얻게 되면 이 사용자는 파일에 대한 소유권을 갖는 데이터 소유자가 된다. 데이터 소유자는 이 후에 스토리지 서버로부터 파일을 다운로드 할 수 있다. 이 때 다운로드된 파일은 인코딩 및 암호화된 상태이다. 따라서 데이터 소유자는 우선 파일을 디코딩 후 다시 복호화하여 원래의 파일에 접근할 수 있다.

<표 1> 본 논문의 제안 방법과 타 방식의 비교

	Storer 등 [7]	Halevi 등 [8]	Wee 등 [9]	제안 방식
데이터 암호화	O	X	O	O
소유권 증명	X	O	O	O
키 관리	공개키 시스템	N/A	그룹 키 관리 스킴 필요	필요 없음
연산량	nE	$n(R+H\log n)$	$n(R+H\log n+P)$	$n(E+R+H\log n)$

n: 파일의 블록 개수, E: 암호화, R: 인코딩, H: 해쉬, P: 지수 연산

IV. 분석

이 장에서는 본 논문의 제안 방법을 성능과 안정성 측면에서 각각 분석한 내용을 기술한다.

4.1 성능 분석

본 논문의 제안 방법은 기본적으로 파일의 해쉬 값을 비밀키로 사용하여 파일을 암호화하고 있다. 이로 인해 스토리지 서버는 서로 다른 사용자들이 암호화화여 업로드 한 파일들에 대해서도 중복 제거를 수행하는 것이 가능하다. [7]에서도 동일한 기능을 제공하나 [7]의 제안 방법은 데이터 소유자의 소유권 증명 방법을 제공하지 않으며 별도의 공개키 시스템을 필요로 한다.

Halevi 등의 방법 [8] 과 Wee 등의 방법 [9] 은 본 논문의 제안 방법과 같이 Merkle 트리 기반의 프로토콜을 제안하고 있다. 그러나 [8]은 암호화를 지원하지 않아 스토리지 서버를 완전히 신뢰하거나 사용자가 동일한 도메인에 있는 경우에만 사용이 가능하다. 그리고 [9] 은 암호화된 파일에서도 소유권 증명이 가능하지만 프로토콜 실행 시 많은 양의 지수 연산이 요구되고 동일한 파일을 소유한 사용자 간에 별도의 키 공유 프로토콜이 필요하다. <표 1> 은 각 제안 방법 별 비교 분석 결과를 보여준다.

4.2 안정성 분석

본 논문에서 제안한 방법에 대한 안정성을 두 가지 요구조건 별로 분석한다. 우선 스토리지 서버로부터의 기밀성 측면에서 살펴보자. 사용자는 데이터 파일을 인코딩 하기 전에 암호화를 수행한다. 따라서 스토리지 서버는 항상 암호화된 파일 만을 저장하게 된다. 파일 암호화에는 대칭키 알고리즘이 사용되는데, 이 때 비밀키는 파일의 해쉬 값, 즉 파일로부터 결정적으로 계산되는 값을 사용한다. 그러므로 파일의 내용이 같으면 동일한 암호문이 생성되는 결정적 특성을 갖는다. 결정적 암호 알고리즘에 대한 안정성은 Bellare 등이 분석한[12] 대로 평문 분포, 즉 파일 분포의 최소 엔트로피가 충분히 크다면 확률적 암호 방식에서 얻을 수 있는 의미론적 안정성 (semantic security) 보다 약하지만 어느 정도의 안정성의 보장이 가능하다.

다음으로 비인가 사용자에 대한 접근제어 측면에서 살펴보자. 본 논문의 데이터 중복 확인 프로토콜은 스토리지 서버와 사용자 상호 간의 Merkle 트리 기반 프로토콜이다. 이 때 스토리지 서버는 검증자의 역할을, 그리고 사용자는 증명자의 역할을 하게 된다. Merkle 트리 보조 정리[8] 에 따르면 검증자는 데이터 중복 확인 프로토콜을 이용하여 증명자로부터 파일의 내용을 전부 복원 하는 것이 가능하다. Halevi 등은 Merkle 트리 보조 정리를 이용하여 비인가 사용자(공격자)가 최소 엔트로피 t 인 파일 F 에서 최대 s 비트의 정보를 확보하여도 t-s 가 충분히 크다면 비인가 사용자는 스토리지 서버를 속일 수 없음을 증명하였다 [8].

V. 결론

본 논문에서는 클라우드 스토리지 서비스의 데이터 중복 제거 기술 적용 시 발생하는 보안 문제들을 해결하기 위한 방법을 제안하였다. 본 논문의 제안 방법은 신뢰할 수 없는 스토리지 서버로부터 아웃소싱 데이터의 기밀성을 보장하면서 비인가 사용자의 데이터 접근을 제어할 수 있다. 제안 방법의 안정성과 성능을 분석하였으며 분석 결과 높은 효율성과 함께 제기된 보안 문제를 모두 해결한 안전한 데이터 중복 제거 기술의 구현이 가능함을 알 수 있다.

[참고문헌]

[1] DropBox, http://www.dropbox.com

[2] Mozy, http://www.mozy.com

[3] MemoPal, http://www.memopal.com

[4] G-Drive, http://drive.google.com

[5] SkyDrive, http://skydrive.com

[6] D. Russell, "Data deduplication will be even bigger in 2010", Gartner, Feb., 2010.

[7] M. W. Storer, K. Greenan, D. D. Long, and E. L. Miller, "Secure data deduplication", Proc. ACM Int'l Workshop on Storage security and survivability (StorageSS'08), pp. 1-10, 2008.

[8] Halevi, S., Harnik, D., Pinkas, B., & Shulman-Peleg, A., "Proofs of ownership in remote storage systems", Proc. ACM Conf. Computer and Comm. Security (CCS'11), pp. 491-500, 2011

[9] Wee, N., Yongnang, W., Huafei, Z., "Private Data Deduplication Protocols in Cloud Storage", Proc. ACM Symp. Applied Computing (SAC'12), 2012

[10] Bowers, K.D., Juels, A., Oprea, A., "Proofs of retrievability: Theory and implementation", Proc. ACM Conf. Computer and Comm. Security (CCS'09), pp. 43-53, 2009

[11] Merkle Hash Tree, Wikipedia, http://en.wikipedia.org/wiki/Hash_tree

[12] M. Bellare, A. Boldyreva, A. O'Neill, "Deterministic and efficiently searchable encryption", Proc. CRYPTO'07, pp.535-552, 2007.