ETRI

2018
Electronics and
Telecommunications
Trends

암호데이터 중복처리 기술

Deduplication Technologies over Encrypted Data

김건우 (Keonwoo Kim, wootopian@etri.re.kr) 지능보안연구그룹 책임연구원
장구영 (Ku-Young Chang, jang1090@etri.re.kr) 지능보안연구그룹 책임연구원
김익균 (Ik-Kyun Kim, ikkim21@etri.re.kr) 지능보안연구그룹 책임연구원/그룹장

Data deduplication is a common used technology in backup systems and cloud storage to reduce storage costs and network traffic. To preserve data privacy from servers or malicious attackers, there has been a growing demand in recent years for individuals and companies to encrypt data and store encrypted data on a server. In this study, we introduce two cryptographic primitives, Convergent Encryption and Message-Locked Encryption, which enable deduplication of encrypted data between clients and a storage server. We analyze the security of these schemes in terms of dictionary and poison attacks. In addition, we introduce deduplication systems that can be implemented in real cloud storage, which is a practical application environment, and describes the proof of ownership on client-side deduplication.

* DOI: 10.22648/ETRI.2018.J.330107

* 본 연구는 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No.2017-0-00364, 임베디드 보안장치의 비밀 데이터 누출 안전성 검증을 위한 학습기반 템플릿 분석 소프트웨어 개발(T&HO 프로젝트)].

4차 산업혁명 사회의 초연결 지능과 신뢰 인터넷 기술 특집

Ⅰ. 서론

데이터 중복처리(중복제거) 기술은 두 개 이상의 중복 데이터가 있으면 하나만 저장하고 나머지는 포인터로 대체하여, 동일한 데이터를 반복해서 저장하지 않고 중복되는 부분을 제거하는 기술이다. 이러한 중복처리 기술은 스토리지 비용 감소, 데이터 백업 시간 단축, 네트워크로의 전송량 감소 등 IT 비용 절약을 위해 사용된다. 중복 처리를 통해 최대 90% 정도의 스토리지 공간을 절약할 수 있기 때문에, 백업용 스토리지뿐만 아니라 DropBox나 Google-Drive 같은 클라우드에도 적용되고 있다. 최근에는 스토리지 서버에 저장된 데이터가 서버나 제 3자에 의해 노출되는 것을 방지하기 위하여, 클라이언트는 데이터를 암호화하고 암호된 데이터를 서버에 저장하려고 한다. 따라서, 클라이언트와 서버에서 암호 데이터에 대한 안전한 중복 처리 기술이 필요하다.

중복제거가 가능한 동일한 평문이라 할지라도 이를 암호하려고 하는 사용자마다 서로 다른 비밀키를 가진다면, 하나의 평문에 대하여 다수개의 서로 다른 암호문이 생성된다. 따라서 일반적인 대칭키 암호 방식으로는 다른 사용자가 암호화한 암호 데이터의 중복제거는 불가능하다. 이러한 문제를 해결하기 위해 사용자마다 다른 비밀키를 사용하는 것이 아니라, 데이터에 의해 키가 결정되는 수렴 암호화(Convergent Encryption) 방식[1]이 제안되었다. Convergent Encryption은 데이터를 해시한 값을 키로 사용하기 때문에, 동일한 평문에 대하여 동일한 암호문을 생성한다. 그래서 다중 사용자에 대한 암호 데이터 중복 제거가 가능하다.

본고에서는 암호데이터 중복처리를 가능하게 하는 프리미티브 기술을 소개하고, 이때 발생하는 문제점에 대하여 살펴본다. 그리고 메시지 기반 암호화[2]에 기반한 데이터 중복처리 방식을 살펴보고, 현실에 구현될 수 있는 암호데이터 중복처리 시스템에 관하여 기술한다.

Ⅱ. 배경 지식

1. 수행 주체에 따른 데이터 중복처리 기법

가. Server-Side 중복처리

클라이언트는 저장하고자 하는 데이터를 서버에 업로드하고, 서버는 데이터를 수신한 후 기존 데이터와 비교하여 중복된 데이터를 제거한다. 이 방식은 포이즌 공격(Poison attack)에 안전하고 프라이버시 침해가 적지만, 클라이언트는 서버 내의 데이터 중복 여부와 관계없이 데이터를 항상 업로드하므로 네트워크 트래픽이 많아지는 단점이 있다.

나. Client-Side 중복처리

클라이언트는 서버에 저장되어 있는 데이터의 존재 여부를 확인한 후 데이터 업로드 여부를 결정하는 중복처리 수행 주체이다. 구체적으로 클라이언트는 데이터의 태그값을 계산하고 이를 서버로 전송하면, 서버는 수신한 태그 값을 자신이 가지고 있는 태그 리스트와 비교한다. 만약 수신 태그 값과 동일한 태그가 존재하면, 이 태그에 해당하는 데이터는 이미 서버에 저장되어 있음을 의미하고 클라이언트는 동일한 데이터를 업로드하지 않는다. 대신 서버는 해당 클라이언트에게 데이터 다운로드 권한을 주기 위한 메타데이터 정보만 업데이트한다. 반면에 수신 태그값과 동일한 태그가 서버에 존재하지 않으면, 클라이언트는 데이터를 서버에게 업로드한다. 이 방식에서는 중복되지 않는 데이터만 전송되므로 네트워크 트래픽 절감이 가능하지만, 상대적으로 크기가 작은 해시값으로 전체 데이터의 중복 여부를 결정하기 때문에 데이터 소유권 문제가 발생할 수 있다.

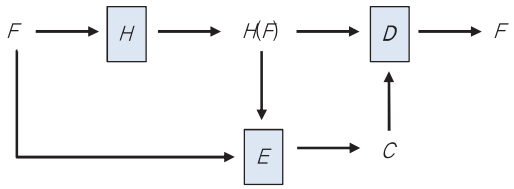
중복 처리 방식을 실제 응용에 적용함에 있어서, 중복처리 비용보다 업로드 비용이 저렴하다고 판단되는 크기가 작은 파일은 Server-side 중복 처리하고, 업로드 비용 절감이 더 중요한 크기가 큰 파일은 Client-side

중복 처리를 활용할 수 있다. 또한, 파일에 대한 카운터 값을 미리 설정하여 파일을 업로드할 때마다 카운트를 증가시켜, 카운터 값이 미리 정의된 기준치보다 작으면 Server-side 중복 처리하고, 카운터 값이 기준치보다 크면 Client-side 중복 처리를 수행할 수 있다.

2. 수렴 암호화(CE: Convergent Encryption)

(그림 1)은 수렴 암호화를 사용한 데이터 암복호 과정을 나타낸 것이다. 평문 파일 F로부터 키 $K=H(F)$가 유도되고, F에 대한 암호 파일 C가 생성된다. 또 다른 사용자가 동일한 파일 F를 암호화하더라도 동일한 암호 파일 C가 생성되어, 다중 사용자에 대한 중복처리가 가능하다. 복호화 과정은 역으로 진행된다. 이때, H는 암호학적 해시 함수이고, E와 D는 결정론적 대칭키 암호 알고리즘이다.

$$C=E(K, F)=E(H(F), F),$$
$$F=D(K, C)=D(H(F), C).$$



(그림 1) Convergent Encryption

실제 응용에서는 파일로부터 유도된 키 K는 사용자마다 고유한 비밀키 또는 패스워드를 이용하여 암호화되어 서버에 저장한다.

3. 암호데이터 중복처리 취약점

암호 데이터 중복 제거에서 발생할 수 있는 대표적인 공격 기법과 취약점은 다음과 같다.

가. 사전 공격(Brute-Force Dictionary Attack)

Convergent Encryption은 짧은 길이의 $H(F)$를 키로 사용하므로 사전 공격에 취약하다. 즉, 타켓 암호 파일 C에 대한 평문 파일 F가 n개의 단어를 가진 사전 $S=\{F_1, F_2, \ldots, F_n\}$ 로부터 유도된다는 것을 아는 공격자는 C에 대해서 $n=|S|$번의 오프라인 암호화를 시도함으로써 F를 얻을 수 있다. 공격자가 매번의 암호 시도에서 F_i를 선택해서 수렴 암호화 방식으로 C_i를 구하고, $C=C_i$인 F_i를 찾는 것은 간단한 방법이다. 그래서 수렴 암호화 방식은 전수 조사가 어려운 Unpredictable 데이터에 대해서만 안전성을 제공하고, 낮은 엔트로피를 가지는 데이터에 대해서는 안전성을 보장하기 어렵다.

오프라인 사전 공격은 이론적인 개념이 아니라 실제 스토리지 서버 시스템의 Convergent Encryption에 매우 중요한 위협이다[3]. 이는 보안에 민감한 데이터를 클라우드 스토리지에서 중복으로 처리하는데 큰 장애물이 되고 있다.

나. 포이즌 공격(Poison Attack)

Client-side 중복처리에서, 악의적인 목적을 가진 공격자가 원본 파일 대신 위조된 다른 파일(Poisoned file)을 업로드 할 수 있다면, 이후의 다른 정당한 사용자는 자신의 원본 파일을 잃고 다른 파일을 다운로드 받는 경우가 발생한다[4]. 개념적으로 포이즌 공격은 데이터 위조 공격(Duplicate-faking attack)과 데이터 삭제 공격(Erasure attack)을 포함한다.

중복처리 프로토콜이 데이터 위조 공격에 취약하면, 원본 데이터는 합법적인 소유자도 모르는 채 가짜 위조 데이터로 대체 될 수 있다. 다음과 같은 시나리오를 가정해보자[5]. 공격자는 암호 파일 C를 식별하는 태그를 보내고 서버로부터 파일 업로드 요청을 받은 후, 정상 파일 C 대신에 위조된 다른 파일 C'를 업로드한다. 이후 합법적인 클라이언트가 정상 파일 F를 암호화한 C를 업로드하려고 시도할 때, 서버는 C와 연관된 태그가 C'과 연관된 태그와 동일한 것으로 보고 C를 저장하지 않고 합법 클라이언트에게는 C'에 대한 소유권을 부여한

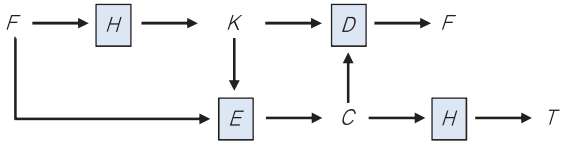
다. 그러면, 이 클라이언트는 C가 서버에 저장되어 있다고 여기고 파일 C를 삭제한다. 나중에 합법적인 사용자는 C 대신 C'을 다운로드하고 복호한다. 클라이언트는 정상 평문 파일 F를 복구할 것으로 예상하지만, 성공적인 데이터 위조 공격으로 인해 클라이언트는 위조된 파일 F'을 복구하게 된다. 만약 데이터 위조 공격에 안전한 중복 제거 방식이 사용된다면, 클라이언트는 파일의 위조를 발견할 수 있고 공격자는 합법 클라이언트에게 위조된 파일을 받아들이게 하는 공격에 실패한다. 그럼에도 불구하고, 클라이언트는 공격자에 의한 원본 파일의 삭제를 막지 못하고 더 이상 정상 파일 F를 복구할 수 없다.

다. 작은 크기 태그 사용의 취약점

Client-side 중복처리 기법은 전체 데이터를 식별하기 위한 태그로 크기가 작은 해시값을 사용하기 때문에, 짧은 길이의 해시값이 위협을 받는다면 실제 데이터의 소유주가 아닌 사용자도 데이터에 대한 접근이 가능하다. 예를 들어, 실제 파일을 소유하지 않는 어떤 사용자가 아웃소싱된 데이터 파일에 대한 짧은 해시값을 가지고 있다면, 이 사용자는 서버에게 해시값을 제시함으로서 파일에 대한 소유를 주장하고 파일을 다운로드할 수 있다. 이를 방지하기 위해, Client-side 중복처리는 비인가 사용자의 데이터 접근을 불허하고 사용자가 정말로 파일을 소유하고 있는 정당한 사용자인지 확인하기 위한 소유권 증명 과정을 필요로 한다[4].

Ⅲ. MLE에 기반한 데이터 중복처리

메시지 기반 암호화(MLE: Message-Locked Encryption)는 암호데이터 중복제거를 가능하게 하는 대표적인 암호학적 프리미티브이다[2]. Convergent Encryption과 마찬가지로, 암호와 복호를 위한 키가 메시지로부터 유도된다. 또한, (그림 2)에서와 같이 MLE는 데이터의 무결성 체크를 위한 태그 일치성(Tag consistency)을 제공한다.



(그림 2) Message-locked encryption

MLE의 4가지 실용적인 암호데이터 중복처리 방식은 다음과 같다.

- CE(Convergent Encryption)
- HCE1(Hash and CE without Tag Check)
- HCE2(Hash and CE with Tag Check)
- RCE(Randomized Convergent Encryption)

안전성 측면에서, MLE 역시 Convergent Encryption과 마찬가지로 낮은 엔트로피를 가지는 데이터에 대한 사전 공격에 취약하다. 특히, Client-side MLE 스킴은 포이즌 공격에 취약하다. 본 장에서는 Server-side CE 방식과 Client-side HCE1, HCE2, RCE 방식을 소개하고 포이즌 공격 관점에서 안전성을 분석한다[2], [4].

1. Server-Side CE를 이용한 중복처리

가. 데이터 업로드 및 중복제거[(그림3) 참조]

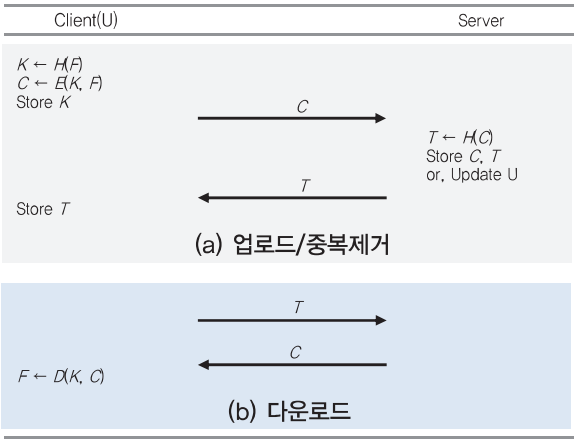
클라이언트(U)는 평문 파일 F에 대한 암호 파일 C를 서버에 업로드 하기 위해 다음과 같은 과정을 거친다.

① U는 F로부터 암호 키 $K \leftarrow H(F)$를 생성한다.

② U는 K를 이용하여 암호 파일 $C \leftarrow E(K, F)$를 출력한다.

③ U는 C를 서버에 업로드하고, K를 저장한다. 그리고, M과 C를 삭제한다.

④ 서버는 수신한 C로부터 태그 $T \leftarrow H(C)$를 생성하여, 자신이 보관 중인 태그 리스트와 비교한다. 이때, T는 C의 Uniqueness를 확인하기 위한 식



(그림 3) CE를 이용한 데이터

별자로 사용된다.

⑤ T가 서버에 존재하지 않는 값이라면, C는 중복되지 않은 데이터를 의미하기 때문에 서버는 C와 T를 저장한다. 만약 T가 서버에 존재하는 값이라면, C는 중복된 데이터를 의미하기 때문에 서버는 C를 중복 저장하지 않고 U가 C의 소유자임을 가리키는 메타데이터만 업데이트한다.

⑥ 나중에 U가 C를 다운로드 받게 하기 위해 서버는 T를 U에게 보낸다.

나. 데이터 다운로드[(그림 3) 참조]

① U는 T를 이용해서 C의 다운로드를 요청한다.

② 서버는 T에 해당하는 C를 찾아서 U에게 전송한다.

③ U는 보관하고 있는 K를 이용하여 C를 복호하고, $F \leftarrow D(K, C)$를 얻는다.

다. 안전성 및 효율성

서버가 업로드된 파일에서 태그를 계산하고 암호 파일과 태그는 원본 평문 파일로부터 생성되었다는 것을 검증하므로, CE 방식은 포이즌 공격에 안전하다. 하지만, 다수의 사용자가 동일한 파일을 반복해서 업로드 하기 때문에 네트워크 효율성이 떨어지고, 서버는 중복 여부와 관계없이 모든 C에 대한 해시 연산을 해야 한다.

2. Client-Side HCE2를 이용한 중복처리

가. 데이터 업로드 및 중복제거[(그림4) 참조]

① U는 F로부터 $K \leftarrow H(F)$를 생성한다.

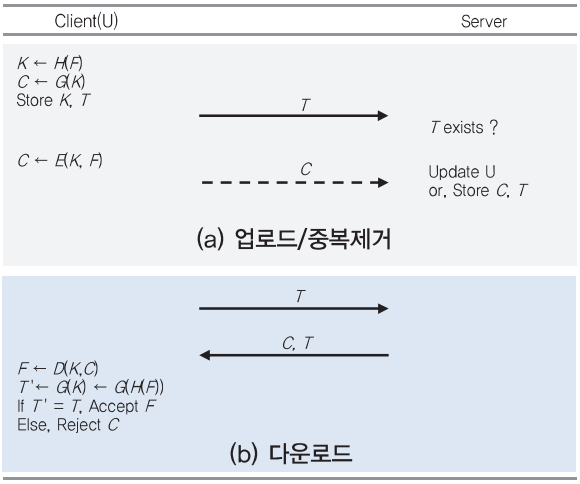
② U는 태그 $T \leftarrow G(K)$를 생성해서 서버에게 보낸다. G는 암호학적 해시 함수로, H와 같은 함수일 수도 있다.

③ 서버는 수신한 T와 동일한 태그가 있는지 검색하여 C를 요청하거나, 메타데이터 정보만 업데이트 한다.

④ C의 요청을 받은 경우, U는 C를 생성하여 업로드하고 서버는 C와 T를 저장한다.

⑤ U는 C를 다운로드하고 복호하기 위해 T와 K를 저장한다.

CE 방식에서는 서버가 상대적으로 크기가 큰 C에 대한 태그 $T \leftarrow H(C)$를 생성하지만, HCE2 에서는 클라이언트가 크기가 작은 K에 대한 태그 $T \leftarrow G(K)$를 생성한다. 또한, 서버에서 동일한 C가 있으면 클라이언트는 C를 출력할 필요가 없다.



(그림 4) HCE2를 이용한 데이터

나. 데이터 다운로드[(그림 4) 참조]

파일 다운로드 과정에서 클라이언트가 태그를 검증하

는 단계를 포함한다.

① C와 T를 다운로드한 U는 K를 이용하여 $F \leftarrow D(K, C)$를 복호한다.

② U는 태그 $T' \leftarrow G(H(F))$를 계산해서 보관하고 있는 T와 비교한다.

③ 두 값이 같으면 복호한 F를 정상 파일로 받아들이고, 두 값이 다르면 다운로드한 C는 정상 파일이 아니므로 C를 버린다.

다. 안전성

클라이언트는 다운로드한 파일로부터 태그를 다시 계산하여 T의 무결성을 확인하므로, 원본 파일이 위조 파일로 대체되었는지 발견할 수 있다. 그럼에도 클라이언트는 공격자에 의해 자신의 정상 파일 C를 삭제 당하는 것을 막을 수 없다. 만약 서버가 C의 무결성을 검증한다면 데이터 삭제 공격도 막을 수 있다.

3. Client-Side HCE1과 RCE를 이용한 중복처리

HCE1을 이용한 업로드 및 데이터 중복제거 과정은 HCE2 방식과 동일하지만, 다운로드 단계에서 클라이언트의 태그 검증 과정이 없다. 클라이언트는 C 대신 C'을 받더라도, C'을 복호하여 F'을 실행하고 나서야 정상 파일이 아님을 알게 된다. 물론 자신의 원본 파일도 공격자에 의해서 대체되어서 복구할 수 없다. 만약 F'이 바이러스 파일이라면 포이즌 공격은 치명적일 수 있다. HCE1에서는 원본 파일 C의 무결성을 클라이언트와 서버 둘다 검증하기 않기 때문에, 데이터 위조 공격과 데이터 삭제 공격에 취약하다.

RCE는 HCE2에 랜덤성을 적용한 버전으로, 키 생성, 파일 암호, 태그 생성을 한 패스에 수행할 수 있다. 클라이언트는 랜덤 대칭 암호키 L을 선택하여 F를 암호하고, MLE 키 K를 생성한다. 그리고, 클라이언트는 one-time pad로서 K를 사용하여 L을 암호하고 태그 T를 생성한다. 클라이언트는 C_1과 C_2를 함께 업로드한다.

$$C_1 \leftarrow E(L, F),\ C_2 \leftarrow K \oplus L.$$

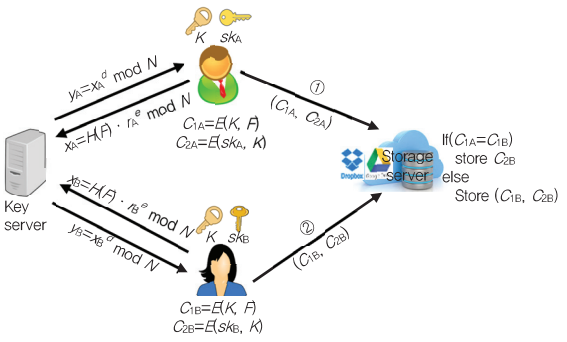
다운로드 단계에서 클라이언트는 C_2로부터 L을 복호한 후, L과 C_1으로부터 F를 얻는다. 태그 검증 방법은 HCE2와 동일하다. 그래서 RCE는 데이터 위조 공격에는 안전하지만 데이터 삭제 공격에는 취약하다.

Ⅳ. 암호데이터 중복처리 시스템

1. DupLess

Convergent Encryption이나 Message-Locked Encryption은 근본적으로 사전 공격에 취약해서 공격자가 암호 데이터만 가지고 평문 데이터를 추측할 수 있는 문제점이 있었다. 이에 반해, DupLess[6]는 키 서버를 추가적으로 사용하여 전수 조사에 저항성을 가지는 안전한 암호데이터 중복처리 시스템으로, DropBox와 Google Drive에서 제공하는 API를 이용하여 구현 가능하다.

(그림 5)에서 사용자는 RSA-OPRF 프로토콜을 통해 키 서버와 협력하여 키 K를 생성한다. 두 개체간의 통신 과정에서 사용자는 키 서버의 비밀 정보인 개인키를 알 수 없고, 키 서버는 사용자로부터 생성된 비밀 정보인 메시지 기반 암호키를 알 수 없다. RSA blind signature[7]를 이용한 실제 키는 $K \leftarrow G(H(F)^d \bmod N)$의 형태이다. 이때 d는 키 서버의 RSA 개인키이고, G와



(그림 5) DupLess 아키텍츠

H는 해시 함수이다. 결과적으로 공격자는 키 서버의 개인키를 알 수 없으므로, 오프라인 전수 조사를 통해 암호 파일로부터 평문 파일을 얻는 일에 성공할 수 없다.

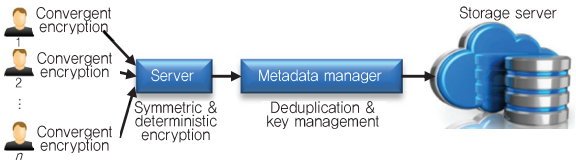
사용자 A, B가 암호 파일 C_{1A}, $C_{1B} \leftarrow E(K, F)$을 생성하여 서버에 업로드하고, Server-side에서 중복처리하는 방식은 CE와 동일하다. Dupless에서는 사용자 마다 고유한 비밀키 sk를 사용하여 CE 암호키 K를 암호 ($C_2 \leftarrow E(sk, K)$)하여 서버에 저장한다.

DupLess는 사용자의 키 서버에 대한 키 요청 쿼리 횟수를 제한함으로써 온라인 전수 조사 공격에도 안전하다. 최악의 경우, 공격자가 키 서버와 공모하여 키 서버의 비밀키 정보를 안다면, 엔트로피가 낮은 데이터는 사전 공격에 취약해지고 DupLess의 안전성은 Convergent Encryption의 안전성과 동일하다.

2. ClouDedup

Cloudedup[8]은 블록 단위 중복 제거와 데이터 기밀성을 제공하는 암호데이터 중복 제거 시스템이다. (그림 6)과 같이, Convergent Encryption에 기반하지만 부가적인 암호 기능을 수행하는 서버의 도입으로 사전 공격에 안전하다. 파일 단위 중복 제거와 달리 블록 단위 중복 제거는 각 블록에 대한 CE 키를 관리해야 하고, ClouDedup에서는 메타데이터 매니저가 키 관리와 중복 제거를 수행한다.

클라이언트는 파일을 블록으로 분할하고 각 블록을 Convergent Encryption으로 암호화한다. 다음 블록의 CE 암호키는 이전 블록의 CE 암호키로 암호되고, 첫번째 블록키를 제외한 나머지 암호화된 블록키는 클라이언트 비밀키를 사용하여 다시 암호한다. 그리고 암호화



(그림 6) ClouDedup high-level view

된 블록에 대한 서명값을 생성한다. 클라이언트는 첫번째 블록에 대한 암호키만 저장한다.

업로드 과정에서 서버는 클라이언트로부터 받은 CE로 암호화된 블록 리스트, 암호화된 블록키 리스트, 블록에 대한 서명값 리스트를 각각 서버의 비밀키를 사용하여 재암호한다. 또한, 다운로드 과정에서 각 블록을 복호하고, 사용자의 공개키를 이용하여 블록의 서명값을 검증하는 역할을 한다. 실제 서버는 사용자 그룹의 도메인에 존재하며, Luna SA HSM 등으로 구현될 수 있다.

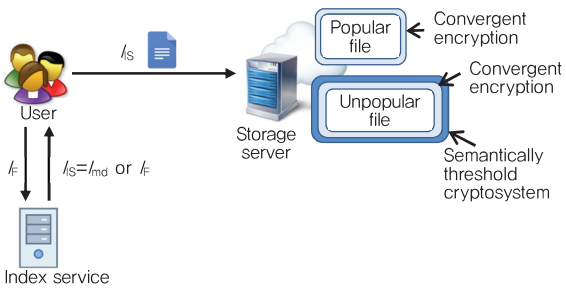
메타데이터 매니저(MM)는 암호된 키와 블록 서명값을 저장하고 중복된 블록을 제거한다. 다수의 블록으로부터 파일을 구성하고 파일 소유권을 관리하기 위해, 블록 id와 파일 id 등이 포함된 linked list 뿐만 아니라 파일 테이블, 포인터 테이블, 서명값 테이블을 관리한다. 실제 응용에서 MM은 Amazon EC2와 같은 클라우드에서 가상 머신 형태로 동작할 수 있으며, 서버의 HSM (Hardware Security Module)이 공격당하는 경우를 대비에 MM에 부가적인 HSM을 두거나 CloudHSM을 사용하기도 한다.

스토리지 서버는 MM으로부터 수신한 블록을 저장하고, 중복제거에 관한 어떠한 역할도 하지 않는다. 그래서, Amazon S3나 Google Drive 와 같은 클라우드 서비스를 사용하여 CloudDedup을 실현하는 것이 가능하다.

3. 다중 계층 암호를 사용한 중복처리 시스템

IBM은 많은 사용자들에 의해서 업로드 되는 인기 데이터(Popular data)에 대해서는 Convergent Encryption을 적용하고, 사용자에 의해서 생산된 파일일 가능성이 많은 비인기 데이터(Unpopular data)에 대해서는 semantic security를 보장하는 암호데이터 중복처리 시스템을 제안하였다[9].

(그림 7)과 같이, 사용자는 파일을 Convergent Encryption으로 암호한 뒤 이 암호 파일의 인덱스를 인



(그림 7) 이중 계층 암호를 사용한 중복처리 시스템

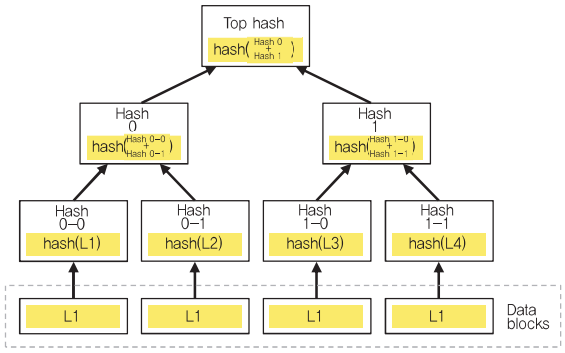
덱스 서버(Index server)에게 보내면, 인덱스 서버는 인덱스 카운터를 비교한다. 인덱스 카운터가 Popularity 기준치 보다 작으면 Unpopular file을 의미하고 인덱스 서버는 새로운 인덱스를 발급한다. 그렇지 않으면, 인덱스에 해당하는 파일은 Popular file을 의미하고 중복처리를 위해 사용자가 생성한 인덱스를 그대로 사용한다.

Popular file은 서버에 중복된 파일이 있으므로 업로드 되지 않고 사용자에게 파일의 소유권만 부여한다. Unpopular file인 경우, 사용자는 CE 파일을 Threshold 공개키 암호 시스템을 이용하여 다시 한번 더 암호한다. 이중 계층 암호 시스템으로 보호된 파일은 스토리지에 저장되고, 동일한 파일의 업로드가 Popularity 기준에 도달하면 Threshold 암호는 복호된다. 즉, 초기에는 모든 파일이 이중 계층 암호로 보호되어 Sematic security가 보장되고, 빈번히 업로드되는 파일은 CE로만 암호화된 파일이 되어서 이후에 동일 파일은 중복 처리된다.

4. PoW를 이용한 중복처리 시스템

가. 소유권 증명

Client-side 중복처리에서는, 실제 파일의 소유없이 태그만 가지고 있는 사용자가 해당 파일의 소유권을 획득하는 것을 방지할 수 있는 소유권 증명 과정을 필요로 한다. PoW(Proof of Ownership)[10]는 머클 트리(Merkle-Tree)[11] 기반의 소유권 증명 프로토콜로, 실제 소유하고 있는 파일의 일부를 서버에게 제시함으로써 사용자가 파일에 대한 소유권을 서버에게 증명하는



(그림 8) Binary merkle-tree

[출처] https://en.wikipedia.org/wiki/Merkle_tree, CC-BY-SA 3.0

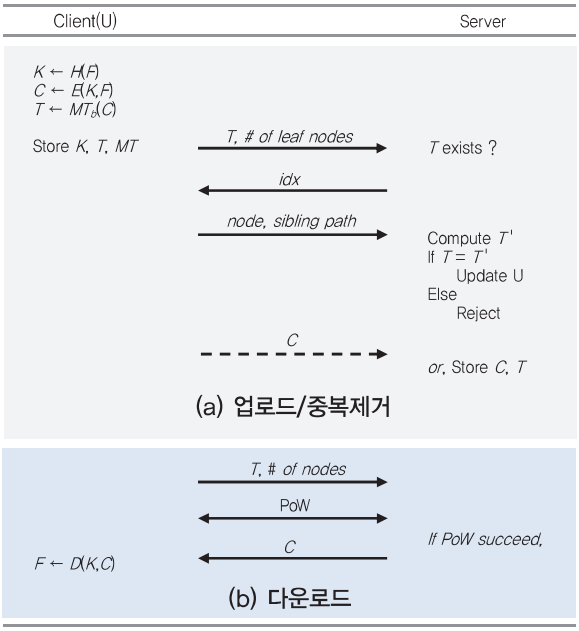
방법이다.

머클 트리를 구성하기 위해 우선 데이터를 여러 블록(leaf node)으로 나눈 뒤, 최하위 leaf 노드부터 인접한 두 블록에 대한 해시값을 계산한다. 상위 노드는 두개의 자식 노드의 해시값들을 이용해 새로운 해시값을 생성한다. 최종적으로 하나의 루트 해시 값을 생성할 때까지 반복한다. 예를 들어, (그림 8)은 데이터를 4개의 블록으로 분할했고 최하위 leaf 노드 개수는 4이며, 이 데이터에 대한 머클 트리 MT는 {Hash 0-0, Hash 0-1, Hash 1-0, Hash 1-1, Hash 0, Hash 1, Top Hash} 이다. 또한, 최하위 leaf 노드부터 루트 노드까지의 경로 중 형제 노드들을 형제 경로(sibling path) 라고 한다. L1의 sibling path는 {Hash 0-1, Hash 1} 이다.

이 밖의 소유권 검증 관련 기술로서, 블룸 필터(Bloom filter)를 활용한 소유권 증명 기술[12], 서버에 저장된 데이터의 무결성 및 복구 가능성을 검증하기 위한 PoR(Proof of Retrievability) 복구 증명 기술[13], 클라이언트와 서버 둘다 소유권을 확인하는 상호 소유권 검증 기술 등이 있다.

나. PoW와 결합된 중복처리 프로토콜

(그림 9)와 같이, 머클 트리 기반의 데이터 소유권 증명 과정에서 클라이언트는 암호 파일에 대한 머클 트리를 구성한 후, 루트 해시 값과 최하위 leaf 노드 개수를



(그림 9) PoW와 결합한 데이터

서버에게 전송한다. 루트 해시 값과 동일한 태그가 서버의 DB에 없으면 서버는 파일 업로드를 요청하고, 동일한 태그가 있으면 서버는 해당 루트 해시 값 사용자에게 파일의 소유권 증명을 요청한다. 클라이언트는 서버로부터 요구되는 랜덤 색인(leaf 노드 개수 이내의 수)에 대하여 해당 노드와 올바른 Sibling path를 답해야 한다. 서버는 응답받은 값으로부터 머클 트리 탑 해시 값을 계산하고 클라이언트로부터 받은 루트 해시 값과 비교함으로써, 클라이언트의 데이터 소유 여부를 검증한다. PoW가 성공적으로 검증되면 클라이언트에 대한 메타데이터 정보를 갱신하지만, 검증되지 않으면 올바른 태그를 가진 클라이언트라 할지라도 서버에 저장된 파일에 대한 접근을 허락하지 않는다.

다운로드 단계에서도 PoW 과정을 거치므로, 정상 파일의 머클 트리없이 태그만 소유한 부당 사용자는 서버의 PoW 요청에 올바른 응답을 할 수 없어서 파일을 다운로드 받을 수 없다. 한편, 서버는 PoW 과정에서 복수개의 색인을 보낼 수 있는데, 데이터를 소유하지 않는 사용자가 올바른 응답을 하는 것은 더욱 불가능하다.

V. 결론

데이터 중복 제거는 스토리지 비용과 네트워크 트래픽을 줄이기 위해 이미 백업 시스템과 클라우드 스토리지에서 널리 사용되는 보편적인 기술이다. 최근에는 해킹으로 인한 데이터 노출이나 프라이버시 이슈 등의 문제로 데이터를 암호하여 서버에 보관하고 하고자 하는 개인과 기업의 요구가 증가하고 있다.

본고에서는 암호데이터 중복처리를 위한 기본적인 프리미티브 기법과 실제 응용에 활용할 수 있는 중복처리 시스템을 소개하였다. 아직까지는 안전성과 효율성을 모두 만족하는 암호 데이터 중복제거 기술의 완성도는 높지 않는 편이다. 하지만, 프라이버시 보호에 관한 강화된 법규와 기하급수적으로 증가하는 데이터로 인한 비용 절감의 목적으로, 클라이언트와 스토리지 서버간 암호데이터 중복처리 기술이 곧 도입될 것으로 예상된다.

약어 정리

CE	Convergent Encryption
MLE	Message-Locked Encryption
HCE1	Hash and CE without Tag Check
HCE2	Hash and CE with Tag Check
RCE	Randomized Convergent Encryption
HSM	Hardware Security Module
PoW	Proof of Ownership

참고문헌

[1] J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer, "Reclaiming Space from Duplicate Files in a Serverless Distributed File System," In *Proc. Int. Conf. Distr. Comput. Syst.*, Vienna, Austria, July 2-5, 2002, pp. 617-624.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Message-Locked Encryption and Secure Deduplication," In *Adv. Cryptology-Eurocrypt*, Athens, Greece, May 2013, pp. 296-312.

[3] Z. Wilcox-O'Hearn et al., "Confirmation of a File Attack," Accessed 2018. https://tahoe-lafs.org/hacktahoelafs/drew_perttula.html

[4] N. Kaaniche and M. Laurent, "A Secure Client Side

Deduplication Scheme in Cloud Storage Environments," In *Proc. Int. Conf. New Technol., Mobility Security*, Dubai, United Arab Emirates, Mar. 30–Apr. 2, 2014, pp. 1–7.

[5] K. Kim, T.-Y. Youn, N.-S. Jho, and K.-Y. Chang, "Client-Side Deduplication to Enhance Security and Reduce Communication Costs," *ETRI J.*, vol. 39, no. 1, Feb. 2017, pp. 116–123.

[6] M. Bellare, S. Keelveedhi, and T. Ristenpart, "DupLESS: Server-Aided Encryption for Deduplicated Storage," *Proc. USENIX Conf. Security,* Washington, DC, USA, Aug. 14–16, 2013, pp. 179–194.

[7] D. Chaum, "Blind Signatures for Untraceable Payments," In *Advances in Cryptology*, Boston, MA, USA: Springer, 1983, pp. 199–203.

[8] P. Puzio, R. Molva, M. Onen, and S. Loureiro, "ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage," In *IEEE Int. Conf. Cloud Comput. Technol. Sci.*, Bristol, UK, Dec. 2–5, 2013, pp. 363–370.

[9] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A Secure Data Deduplication Scheme for Cloud Storage," IBM Technical Report 2013.

[10] S. Halevi, D. Harnik, B PinKas, and A. Shulman-Peleg, "Proofs of Ownership in Remote Storage Systems," In *Proc. ACM Conf. Comput. Commun, Security*, Chicago, IL, USA, Oct. 2011, pp. 491–500.

[11] Wikipedia, Merkle Tree, Accessed 2018. https://en.wikipedia.org/wiki/Merkle_tree

[12] J. Blasco, R. di Pietro, A. Orfila, and A. Sorniotti, "A Tunable Proof of Ownership Scheme for Deduplication Using Bloom Filters" In *IEEE, Conf. Commun. Netw. Security*, San Francisco, CA, USA, Oct. 29–31, 2014, pp. 481–489.

[13] K.D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," In *Proc. ACM Conf. Comput. Commun. Security*, Chicago, IL, USA, Nov. 13, 2009, pp. 43–54.