분산 저장소 구조에서 안전한 데이터 접근을 위한 키 관리 메커니즘

이명진° 채기준
이화여자대학교
maya012°@ewhain.net, kjchae@ewha.ac.kr

Key management for Secure Data Access in Distributed Storage

Myungjin Lee° Kijoon Chae
Ewha Womans University

요 약

분산 저장소의 데이터를 여러 사용자가 공유하는 시스템에서는 데이터 보호를 위해 데이터를 암호화하여 저장한다. 사용자가 데이터를 서비스받기 원하면 사용자에게 데이터를 암호화한 키를 공개한다. 이러한 시스템에서는 기존의 사용자가 시스템을 탈퇴할 경우 그 사용자가 접근 가능했던 데이터가 위험에 노출되게 된다. 이를 막기 위해서 시스템을 탈퇴한 사용자가 접근 가능했던 모든 데이터 각각에 대해 다른 키를 사용하여 암호화를 해야 한다. 이 과정은 데이터의 크기가 기가바이트 혹은 테라바이트에 달할 경우 막대한 오버헤드를 수반한다. 이러한 문제점을 해결하기 위해 본 연구에서는 사용자에게 데이터 키를 노출하지 않고 사용자가 데이터를 서비스 받을 수 있는 메커니즘을 제안한다.

1. 서 론

안전하지 않은 네트워크상에서의 안전한 데이터 전송을 위해 IPSec, SSH, TLS 와 같은 프로토콜들이 개발되었다. 그러나 접근권한이 없는 악의적인 사용자에게는 단시간 노출되는 전송중인 데이터보다 언제든지 접근을 시도해 볼 수 있는 저장소에 저장되어 있는 데이터가 더 공격하기 쉬운 목표물이다. 그래서 대부분의 스토리지 시스템에서는 데이터를 암호화하여 저장한다. 이러한 시스템에도 다른 암호화 시스템에서 항상 문제가 되는 키 관리가 필요하다. 기존의 사용자가 시스템 그룹에서 탈퇴할 경우 그 사용자에게 주어졌던 키로 암호화한 데이터는 새로운 키로 암호화 하는 과정이 필요하다. 이러한 과정은 데이터의 용량이 수 기가바이트 혹은 수 테라바이트에 달하는 경우에는 엄청난 오버헤드가 된다. 이러한 문제를 해결하고자 본 연구에서는 이러한 오버헤드 없이 암호화한 데이터와 키를 관리하는 메커니즘을 제안하고자 한다.

2. 분산 저장 시스템 구조

스토리지 시스템은 [그림 1]과 같이 User, Admin, Key Server, Integrity Engine, 그리고 Storage Server로 구성된다. 이러한 시스템 구조는 기존 연구[1]에서 제안한 구조에 Integrity Engine을 추가한 것이다. 물론 Storage Server에 데이터를 암호화하여 저장하지만 저장되어 있는 동안 악의적인 User에 의해 암호화된 데이터도 변질될 수 있다. 데이터의 무결성(integrity)을 확인하기 위해 Integrity Engine을 Storage Server 전에 위치시켜서 데이터가 Storage Server 내에서 수정, 변경되었는지를 확인할 수 있다.

시스템을 구성하는 각각의 객체가 가지는 정보와 기능을 [표 1]에 정리하여두었다.

[표 1] 객체 소유 정보

객체	소유정보	기능
User	미리 부여된 고유한 ID	Data 읽기
	Admin과의 키	
Admin	데이터 접근권한 테이블	Data 쓰기 User 인증 데이터 접근권한 테이블 관리
	User와의 키	
	Key Server와의 키	
	Integrity Engine과의 키	
Key Server	Data를 암호화한 키	데이터를 암호화한 키 저장
	Admin과의 키	
Integrity Engine	암호화 된 Data의 MAC 값	암호화 된 데이터의 무결성 검사 및 데이터 조각화
	Admin과의 키	
Storage Server	암호화 된 Data의 조각	암호화 된 데이터의 조각을 분산 저장
	Admin과의 키	

User는 고유한 식별자를 가지고 있고, Admin으로부터 데이타에 대한 접근권한을 얻어 Key Server와 Integrity Engine과의 통신을 통해 저장되어 있는 데이터를 얻을 수 있다. 이때 User는 데이타에 대한 읽기 권한만이 주어진다고 가정한다.

Admin은 신뢰되는 객체(trusted entity)이며 새로운

본 논문은 정보통신부 및 정보통신 연구진흥원의 대학 IT연구센터 지원 사업 및 한국전자통신연구원 정보 보호 연구단 위탁연구과제의 연구결과입니다.

User가 가입 시 User에 대한 인증이 가능하다고 가정한다. Admin은 User, Key Server 그리고 Integrity Engine과 각각의 대칭키를 저장하고 있어서 각 객체와의 안전한 통신이 가능하다. 각 데이터에 대한 접근권한 테이블을 관리하고 있어서 User가 데이터에 대한 요청을 하면, 테이블을 검색하여 데이터에 대한 접근권한을 확인할 수 있다. 그리고 Admin은 데이터를 Storage Server에 저장할 수 있다. 데이터에 대한 키를 생성하여 데이터를 암호화한 뒤 데이터는 Integrity Engine에게, 데이터를 암호화한 키는 Key Server에 전송하고 이 후에 데이터와 키는 삭제한다.
Key Server는 데이터를 암호화한 키를 저장하고 Integrity Engine은 데이터가 Storage Server에 저장된 동안의 수정, 변경되지 않았는지에 대한 무결성 검사를 한다. 데이터에 이상이 없는 경우 Integrity Engine은 Storage Server에는 Key Server에 저장된 키로 암호화된 데이터를 저장한다.

3. 제안된 키 관리 프로토콜

기존 연구에서는 데이터에 대한 권한을 가졌던 User가 권한을 잃어버리면 그 User가 접근 가능했던 모든 데이터 각각에 대해 다른 키를 사용하여 암호화를 해야 하는 문제점이 있었다. 이러한 문제점을 해결하기 위해 본 연구에서는 User에게 데이터 키를 노출하지 않고 User가 데이터를 서비스 받을 수 있는 메커니즘을 제안한다. 앞으로 사용될 표기법에 대해 아래 [표 2] 에 정리하여 두었다.

[표 2] 표기법

표기	설명
$E_K(Data)$	key로 Data를 암호화
K_D	데이터를 암호화한 키
K_{AU}	Admin과 User사이의 대칭키
K_{AK}	Admin과 Key server사이의 대칭키
K_{AI}	Admin과 Integrity engine사이의 대칭키
S_{UK}	User와 Key Server사이의 Session Key
S_{UI}	User와 Integrity Engine사이의 Session Key
S_{KI}	Key Server와 Integrity Engine사이의 Session Key
$\|$	스트링 연결 (Concatenate)

3.1 새로운 User 인증

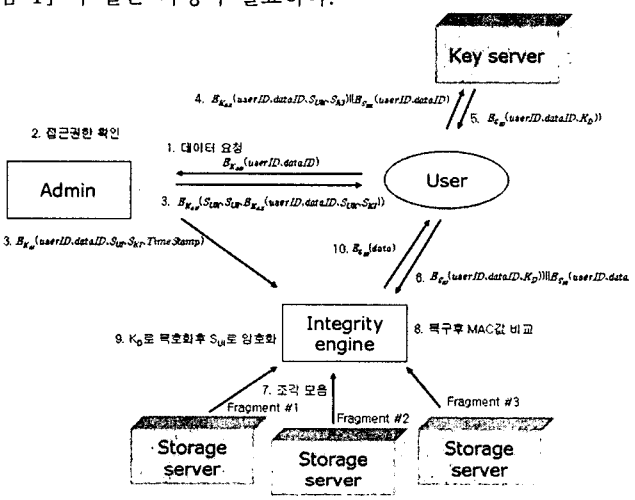
Admin은 시스템에 새로 가입한 User에 대한 인증을 하고 데이터 접근권한 테이블을 업데이트한다. 새로운 User에게 Admin과의 키(K_{AU})를 부여한다. User Admin에게 받은 키를 이용하여 이용하고자 하는 데이타를 요청할 수 있다.

3.2 Admin이 새로운 데이터를 저장

새로운 데이터를 Storage Server에 저장하고자 할 때 암호화할 키(K_D)를 생성하고 데이터를 암호화한 뒤에 K_D는 Key Server와의 키(K_{AK})로 암호화하여 Key Server에 저장하고 암호화한 데이터는 Integrity Engine과의 키(K_{AI})로 다시 암호화하여 Integrity Engine에 전송한다. 이러한 과정이 끝난 뒤에 Admin은 메모리에서 데이터 키와 데이터를 영구 삭제한다. Integrity Engine은 Admin으로부터 받은 $E_{K_{AI}}(E_{K_D}(data))$를 K_{AI}로 복호화하고 $E_{K_D}(data)$에 대한 MAC값을 계산하여 저장한 뒤 secret sharing[2]과 같은 알고리즘을 적용하여 n조각으로 나눈 뒤에(fragmentation) 각각 Storage Server에 저장한다.

3.3 User의 데이터 요청

데이터를 Storage Server로부터 얻어오기 위해서는 [그림 1] 과 같은 과정이 필요하다.



[그림 1] 시스템 구조

3.3.1 Admin에게 데이타 요청

$User \rightarrow Admin$

$$E_{K_{AU}}(userID, dataID)$$

3.3.2 User의 사용권한 확인

Admin은 데이터 접근권한 테이블을 참조하여 User가 요청한 데이터에 대한 접근권한을 갖는지 확인한다.

3.3.3 Session key 전송

$Admin \rightarrow User$

$$E_{K_{AU}}(S_{UK}, S_{UI}, E_{K_{AK}}(userID, dataID, S_{UK}, S_{KI}))$$

$$Admin \rightarrow \int egrityEngine \quad E_{K_{AI}}(userID, dataID, S_{UI}, S_{KI}, TimeStamp)$$

먼저 User에게 이번 Session동안 Key server와 사용할 키(S_{UK})와 Integrity Engine과 사용할 키(S_{UI})를 전송한다.
Integrity Engine에 전송하는 메시지 중 TimeStamp는 User가 Admin과 Key Server와의 통신을 한 뒤 Integrity Engine에게 요청할 때까지 걸리는 시간의 상한선 역할을 한다. TimeStamp를 사용함으로써 Integrity Engine이 일정 시간 후에 도착하는 User 요청에 대해 무시할 수 있어 reply 공격을 막을 수 있다.

3.3.4 Key Server에 데이터 키 요청

$User \rightarrow KeyServer$

$E_{K_{AK}}(userID, dataID, S_{UK}, S_{KI}) || E_{S_{UK}}(userID, dataID)$

User는 Admin으로부터 받은 메시지 중 K_{AK}로 암호화된 부분에 S_{UK}로 자신의 ID와 데이터 ID를 암호화한 메시지를 붙여서 Key Server에 전달한다. Key Server는 K_{AK}키를 Admin과 공유하므로 K_{AK}로 암호화한 메시지를 복호화 할 수 있고 그러면 S_{UK}를 알게 되므로 User가 S_{UK}로 암호화한 메시지도 풀어 볼 수 있다. 두 메시지에 들어있는 User ID와 데이터 ID가 서로 같은 지 비교하여 중간단계에게 변형이 있었는지 확인할 수 있다.

3.3.5 데이터 키 전송

$KerServer \rightarrow User$

$E_{S_{KI}}(userID, dataID, K_D))$

Key Server는 데이터 ID에 대한 데이터 키(K_D)를 찾아서 S_{KI}으로 암호화하여 User에게 전송한다. 이때 User는 S_{KI}을 모르기 때문에 K_D를 알 수 없다.

3.3.6 Integrity Engine에게 데이터 요청

$User \rightarrow \int egretyEngine$

$E_{S_{KI}}(userID, dataID, K_D)) || E_{S_{UI}}(userID, dataID)$

User는 Key Server로 받은 메시지에 S_U로 자신의 ID와 데이터 ID를 암호화한 메시지를 붙여서 Integrity Engine에 전달한다.
Integrity Engine은 TimeStamp와 User로부터의 메시지가 도착한 시간을 비교하여 보고 합당하다고 판단되면 Admin으로부터 받았던 S_U와 S_{KI}으로 User에게 받은 메시지를 복호화한다. Admin, Key Server, User로부터 받은 세 개의 User ID와 데이터 ID를 비교하여 중간과정에서의 위조가 있었는지 판단할 수 있다.

3.3.7 데이터 조각 모음
Storage Server에서 데이터 ID에 해당하는 조각을 요청한다. k개의 조각을 모은 뒤에 $E_{K_D}(data)$로 복구한다.

3.3.8 다시 MAC 값을 계산하여 저장되어 있는 MAC 값과 비교하여 Storage Server에 저장된 동안 수정, 변경되지 않았는지에 대한 무결성 검사를 한다.

3.3.9 이상이 없으면 D_K 로 복호화 한 뒤 S_U로 암호화하여 User에게 전송한다.

$\int egrityEngine \rightarrow User$

$E_{S_{UI}}(data)$

4. 보안 분석

제안한 메커니즘은 공격자가 Admin을 제외한 객체를 공격하여도 얻고자 하는 민감한 데이터(sensitive data)를 얻을 수 없는 안전한 시스템을 보장한다.

4.1 Key Server가 공격당한 경우
공격자가 Key Server를 가장하여 Integrity Engine에 접근해서 데이터를 가져오려 한다고 가정해 보자. Integrity Engine에게 데이터를 요청하기 위해서는 Admin이 결정한 Key Server와 Integrity Engine 사이의 키(S_{KI})가 필요하다. 만약 Admin을 가장하여 S_{KI}을 생성하여 보내려 하여도 Admin과 Integrity Engine 사이의 키(K_{AI})를 알지 못하기 때문에 불가능하다.

4.2 Integrity Engine이 공격당한 경우
공격자가 Integrity Engine을 공격하여 성공한 경우에도 데이터를 암호화한 K_D를 모르기 때문에 데이터를 복호화할 수 없다. 키 서버에 접근하여 K_D를 알아내려 하는 경우에도 Admin으로부터 받아야 하는 S_{KI}를 모르기 때문에 위의 경우와 비슷하게 공격에 성공할 수 없다. 공격자는 암호화된 데이터는 볼 수 있지만 암호화한 키를 모르기 때문에 역시 민감한 데이터를 얻을 수 없다.

4.3 접근권한을 잃어버린 악의적인 사용자가 시스템을 공격할 경우
제안한 메커니즘은 관련 연구에서 데이터에 대한 접근권한이 있던 User가 접근 권한을 잃어버렸을 때 생기는 문제도 해결할 수 있다. 기존연구와 달리 데이터를 암호화한 키가 User에게 노출되지 않기 때문에 접근권한이 없는 User는 데이터를 볼 수 없다.

4.4 Storage Server가 공격당한 경우
이 외에도 n-k개 이하의 Storage Server가 고장이 나거나 공격당하여도 k개의 조각을 이용하여 복구 할 수 있는 secret sharing의 성질에 의해서 User에게 서비스가 가능하여 가용성(availability)을 높일 수 있다.

5. 결론

제안한 메커니즘은 User에게 데이터를 암호화한 키를 노출시키지 않기 때문에 기존 User의 탈퇴 시 그 User에게 주어졌던 키에 대응하는 데이터에 대해 새로운 키로 암호화 하는 오버헤드가 없다. 또한 기존에 제안된 구조에 Integrity Engine을 추가함으로써 Storage Server에 저장되어 있는 동안 데이터의 수정, 변경 여부를 확인할 수 있다. 이처럼 제안한 메커니즘은 기존연구의 취약했던 부분을 보강하여 더 안전한 시스템을 보장한다.

6. 참고문헌

[1] Ludwig Seitz, Jean-Marc Pierson, Lionel Brunie, "Key management for encrypted storage in distributed systems," Second IEEE International Security in Storage Workshop, 2003.
[2] A. Shamir. "How to Share a Secret," communication of the ACM, Vol. 22, No. 11, pp.612-613, 1979.