Journal of The Korea Institute of Information Security & Cryptology(JKIISC) ISSN 1598-3986(Print)
VOL.23, NO.5, October 2013 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2013.23.5.803

Kerberos 기반 하둡 분산 파일 시스템의 안전성 향상방안*

박 소 현,† 정 익 래‡
고려대학교 정보보호대학원

A Study on Security Improvement in Hadoop Distributed File System Based on Kerberos*

So Hyeon Park,† Ik Rae Jeong‡
Graduate School of Information Security, Korea University

요 약

최근 스마트 기기 및 소셜 네트워크 서비스의 발달로 인해 데이터가 폭증하며 세계는 이른바 빅데이터 시대를 맞고 있다. 이에 이러한 데이터를 처리할 수 있는 새로운 기술인 빅데이터 처리기술은 클라우드 컴퓨팅 기술과 함께 주목받고 있으며, 가장 대표적인 기술이 바로 하둡이다. 하둡 분산 파일 시스템은 상용 리눅스 서버에서 실행되도록 설계된 오픈소스 프레임워크로서 수백 테라바이트 크기의 파일을 저장할 수 있다. 초기 하둡은 빅데이터 처리에 초점을 맞추어 보안이 거의 도입되지 않은 상태였으나 사용자가 빠르게 늘어남에 따라 하둡 분산 파일 시스템에 개인정보를 포함한 민감한 데이터가 많이 저장되면서, 2009년 커버로스와 토큰 시스템을 도입한 새로운 버전을 발표하였다. 그러나 이 시스템은 재전송 공격, 가장 공격 등이 가능하다는 취약점을 가진다. 따라서 본 논문에서는 하둡 분산 파일 시스템 보안 취약점을 분석하고, 이러한 취약점을 보완하면서 하둡의 성능을 유지할 수 있는 새로운 프로토콜을 제안한다.

ABSTRACT

As the developments of smart devices and social network services, the amount of data has been exploding. The world is facing Big data era. For these reasons, the Big data processing technology which is a new technology that can handle such data has attracted much attention. One of the most representative technologies is Hadoop. Hadoop Distributed File System(HDFS) designed to run on commercial Linux server is an open source framework and can store many terabytes of data. The initial version of Hadoop did not consider security because it only focused on efficient Big data processing. As the number of users rapidly increases, a lot of sensitive data including personal information were stored on HDFS. So Hadoop announced a new version that introduces Kerberos and token system in 2009. However, this system is vulnerable to the replay attack, impersonation attack and other attacks. In this paper, we analyze these vulnerabilities of HDFS security and propose a new protocol which complements these vulnerabilities and maintains the performance of Hadoop.

Keywords: Hadoop distributed file system(HDFS), Hadoop, Big Data, Cloud computing, Authentication, Kerberos

접수일(2013년 5월 8일), 수정일(2013년 7월 24일), 게재확정일(2013년 7월 24일)

* 본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(2012000 7037)과 미래창조과학부 및 정보통신산업진흥원의 IT융합 고급인력과정 지원사업(NIPA-2013- H0301-13-30 07)의 연구결과로 수행되었음

† 주저자, shsoonie@naver.com
‡ 교신저자, irjeong@korea.ac.kr(Corresponding author)

I. 서 론

IDC[10]는 2011년 전 세계에서 생성된 디지털 정보량이 1.8제타바이트(1.8조 기가바이트)에 달하였으며 이 데이터의 양은 매 2년마다 2배씩 증가할 것으로 예측했다. 특히 스마트 기기의 발달과 소셜 미디어(SNS)의 등장으로 인해 생산되는 데이터의 대부분이 비정형 데이터이며 실제로 빅데이터는 비정형 데이터와 정형 데이터를 모두 포괄한다. 세계는 이른바 빅데이터(big data) 시대를 맞고 있다. 이 빅데이터를 어떻게 활용하느냐에 따라 기업의 이익이 판가름 날수도 있고 빅데이터 분석 결과가 국가 안보에 도움이 될 수도 있다. 그러나 문제는 이러한 빅데이터를 저장하고 처리하는 방법이 기존 방식인 데이터 웨어하우스(DW, Data Warehouse)와 관계형 데이터베이스 관리 시스템(RDBMS, Relational Data Base Management System)으로는 거의 불가능하다는 점이다. 또한 기존 시스템들은 대부분 고가의 장비로 이루어져 있어 실제로 구축하더라도 수용범위를 넘어서는 비용이 발생할 것이다. 따라서 저렴한 비용이면서 대용량 데이터를 분산 처리할 수 있는 새로운 기술이 필요하게 되었고 이렇게 등장하게 된 것이 바로 하둡(hadoop)이다.

하둡[5]은 오픈소스 웹 검색엔진인 아파치 너치(nutch)의 분산처리를 지원하기 위해 개발된 것으로 구글의 빅데이터 관리 솔루션인 구글 파일 시스템(GFS, Google File System) 논문[6] 기반으로 구현된 GFS의 오픈소스 버전이라고 할 수 있다. 하둡은 기존의 고가의 서버장비가 아닌 저렴한 일반 x86 CPU의 리눅스 장비에서 수 백 페타바이트 이상의 데이터를 관리할 수 있도록 설계되었기 때문에 비용 부담이 적어 사용자가 빠르게 늘고 있으며 실제로 페이스북(facebook), 트위터(twitter), 아마존(amazon), 이베이(ebay)를 포함한 수백 개의 기업이 하둡을 사용하고 있다[7].

초기에는 소수의 기업들만이 하둡을 사용하였으며 저장되는 데이터에 민감한 데이터의 비중이 매우 적었기 때문에 보안이 거의 도입되지 않았다. 하지만 사용자가 늘어남에 따라 하둡 분산 파일 시스템(HDFS, Hadoop Distributed File System)에 개인정보와 같이 보안을 필요로 하는 데이터의 양이 급격히 증가하기 시작하였고, 이에 하둡은 자신들의 기술에 보안을 도입할 필요성을 인식하여 2009년 사용자와 서버간의 강력한 상호 인증을 갖춘 새로운 버전[4]을 발표하였다.

하둡은 보안과 성능, 비용 모두를 만족하기 위하여 공개키 시스템은 배제하였고, 기존 하둡에 대칭키 시스템인 커버로스(kerberos)를 결합하는 방법을 선택하였다. 하지만 네임노드와 데이터노드, 하둡 클라이언트가 모두 커버로스를 통해 인증하게 되면 커버로스의 키 분배 센터에 병목현상이 발생하기 때문에, 하둡은 성능 향상을 위해 자체적으로 개발한 대칭키 기반 토큰 시스템을 함께 도입하였다[4].

하지만 이렇게 보안이 도입된 하둡 분산 파일 시스템 또한 재전송 공격, 가장 공격 등이 가능하다는 문제점이 존재하였다. 본 논문에서는 하둡 분산 파일 시스템이 가지는 취약점에 대해 분석하고, 이러한 취약점을 개선하면서 하둡의 성능적인 면도 유지할 수 있는 새로운 프로토콜을 제안하고자 한다.

II. 배경 지식

2.1. 하둡 분산 파일 시스템

하둡은 대용량 데이터를 처리하는 분산 응용 프로그램을 작성하고 실행시키기 위한 오픈 소스 프레임워크로서 네트워크로 연결된 서버들의 스토리지상에 파일을 저장하고 이를 관리하는 하둡 분산 파일 시스템과 대용량 데이터를 분석할 수 있는 하둡 맵리듀스(hadoop mapreduce)[12]로 구성된다.

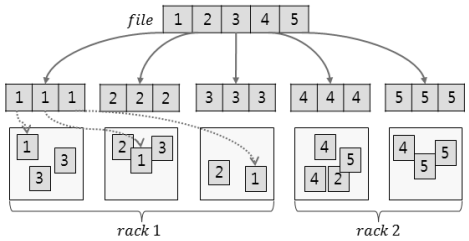
하둡 분산 파일 시스템[1]은 기존의 고가의 서버 장비가 아닌 노드 장애가 발생할 확률이 높은 리눅스 서버에서 실행되도록 설계되어 기계와 디스크 장애에도 사용자가 중단 없이 작업을 수행 할 수 있다. 또한 수백 테라바이트 크기의 파일을 저장할 수 있으며 확장성이 높고, 낮은 데이터 접근 시간보다는 높은 데이터 처리량에 중점을 두어 데이터에 대한 액세스를 스트리밍 방식으로 지원한다. 데이터의 수정 연산은 지원하지 않으며 다중 읽기와 쓰기 연산, 저장된 파일에 대한 덧붙이기(append) 연산을 지원한다.

2.1.1 하둡 분산 파일 시스템 구조

하둡 분산 파일 시스템은 모든 클러스터의 로컬 스토리지(local storage)들을 하나의 네임스페이스로 결합하였다. 하둡에 저장되는 모든 파일들은 [그림 1]과 같이, 기본적으로 64메가바이트(MB)의 크기로 쪼개진 후 3개씩 복제되어 분산된 서버에 저장된다. 따

라서 로컬 서버의 하드디스크보다 큰 사이즈의 데이터를 저장할 수 있으며 특정 디스크나 서버에 장애가 발생하더라도, 다른 정상 서버에 저장된 복제본에 접근하여 작업을 수행함으로써 효율적으로 대응할 수 있다. 또한 높은 내고장성(fault tolerance)과 가용성(availability)을 지원한다.

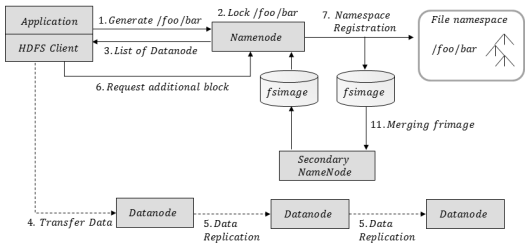
이때 나누어진 64메가바이트의 파일조각을 블록(block)이라고 정의하며, 블록의 크기나 복제본의 개수는 사용자 임의로 수정 가능하다[9].



[그림 1] 하둡 분산 파일 시스템에서의 파일 저장

하둡 분산 파일 시스템은 마스터(master) 역할을 하는 한 대의 네임노드(namenode) 서버와 슬레이브(slave) 역할을 하는 여러 대의 데이터노드(datanode) 서버로 구성된다. 데이터노드는 하둡 클라이언트(이하 클라이언트)나 네임노드의 요청이 있을 때 블록들을 저장 및 탐색한다. 또한 자신의 상태 정보 등을 담은 하트비트(heartbeat) 메시지와 블록의 목록이 저장된 블록 리포트(block report)를 네임노드에게 주기적으로 보낸다. 네임노드는 파일 시스템의 트리와 그 트리상의 모든 파일과 디렉터리, 즉 전체 파일 시스템의 네임스페이스를 관리하는 역할을 한다. 이 정보는 네임스페이스 이미지(namespace image)와 에디트 로그(edit log) 형태의 파일로 로컬 디스크에 지속적으로 저장된다[7].

이때 네임노드는 한 대로 구성되어 있기 때문에 이를 실행중인 장비가 손상되면 파일 시스템 전체를 사용할 수 없게 된다. 이러한 이유로 네임노드는 장애 복구 능력을 갖추는 것이 중요하다. 하둡은 이를 위한 메커니즘으로 보조 네임노드(secondary namenode)를 제공하고 있다. 보조 네임노드의 역할은 주 네임노드가 손상되었을 때 사용될 네임스페이스의 복제본을 유지하는 것이다. 하지만 보조 네임노드의 상태는 주 네임노드의 상태에 뒤처지기 때문에, 주 네임노드 데이터에 총체적 장애가 발생하면 데이터손실이 발생할 수 있다[7].



[그림 2] 하둡 분산 파일 시스템의 동작 방식

전체적인 하둡 분산 파일 시스템의 동작 방식은 [그림 2]와 같다[13]. 이때 모든 서버들은 완전 연결(fully connected)되어 있으며 TCP(Transmission Control Protocol)기반 프로토콜을 이용하여 통신한다.

2.2 하둡 보안

초기에는 소수의 기업들만이 하둡을 사용하였고 저장되는 데이터에 개인정보와 같은 민감한 데이터의 양이 매우 적었기 때문에 보안이 거의 도입되어있지 않았다. 하지만 사용자가 늘어남에 따라 하둡 상에 개인정보와 같이 보안을 필요로 하는 데이터들이 많이 저장되기 시작하였고, 이에 하둡은 자신들의 기술에 보안을 도입할 필요성을 인지하여 2009년, 클라이언트와 서버간의 강력한 상호 인증을 갖춘 버전을 발표하였다[4].

하둡은 먼저 SSL(Secure Socket Layer)을 사용하였지만 이는 공개키 시스템이기 때문에 대칭키 시스템에 비해 느리고 계산량이 많다. 따라서 하둡은 대칭키 시스템에 기반을 둔 새로운 보안체계를 도입하였으며 현재 SSL은 거의 사용하지 않고 있다.

클라이언트는 하둡의 RPC(Remote Procedure Call) 라이브러리를 통하여 하둡 서비스에 접근할 수 있다. 하둡은 보안을 위해 RPC 레벨에서 구현되고 있는 SASL(Simple Authentication and Security Layer)[11]을 채택하였다. 하둡에서의 SASL은 커버로스와 DIGEST-MD5를 통한 클라이언트 인증을 지원한다[4].

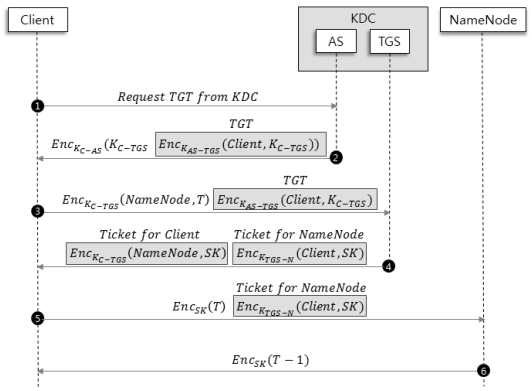
커버로스는 분산 컴퓨팅 환경에서 대칭키를 이용하여 사용자 인증을 제공하는 중앙 집중형 방식이며 하둡 분산 파일 시스템의 데이터노드는 수백에서 많게는 수천대로 구성되어있다[2]. 따라서 네임노드와 데이터노드, 클라이언트가 상호인증을 위해 매번 커버로스를 사용하면 커버로스의 키 분배센터(KDC, Key

Distribution Center)에 병목현상이 발생한다. 이러한 문제를 막기 위하여 하둡은 자체적인 토큰 시스템을 함께 도입하였다[4]. 더 자세한 사항은 2.3절과 2.4절에서 설명한다.

2.3 커버로스(kerberos)

커버로스는 인증 프로토콜인 동시에 키 분배센터로 서비스를 제공하는 서버와 서비스를 제공받는 사용자가 신뢰할 수 있는 제3자(TTP, Trusted Third Party)인 KDC를 통하여 서로를 인증하는 프로토콜이다[3]. KDC는 인증 서버(AS, Authentication Server)와 티켓 발급 서버(TGT, Ticket Granting Server)로 구성되어있다.

커버로스의 사용자는 하둡의 클라이언트이고 커버로스 서버는 하둡의 데이터노드와 네임노드이다. 하지만 실제로 데이터노드는 커버로스를 사용하지 않고 네임노드와 클라이언트만 최초 인증 시에 커버로스를 사용한다[14]. 클라이언트가 네임노드와 커버로스를 통하여 인증하는 과정은 [그림 3]과 같다[8].



[그림 3] 하둡 파일 시스템에서의 커버로스 인증과정

먼저 모든 커버로스 사용자는 인증서버에 등록을 하고 아이디와 비밀번호를 발급한다. 이로써 AS는 클라이언트와 네임노드의 아이디와 대응되는 패스워드에 대한 데이터베이스를 가지고 있는 상태이다. 여기서 K_{x-y}는 x와 y가 공유한 비밀키, C는 클라이언트, N은 네임노드, SK는 클라이언트와 네임노드가 공유할 세션키, T는 타임스탬프를 의미한다.

① 클라이언트는 자신의 등록된 아이디를 이용해 평문으로 AS에 요청을 보낸다.

② AS는 K_{C-TGS}와 TGT(Ticket Granting Ticket)를 클라이언트와 공유한 키 K_{C-AS}로 암호화하여 클라이언트에게 보내는데 이때 TGT는 클라이언트가 TGS에 접속 시 사용할 티켓이다.

③ 클라이언트는 이를 복호화하여 K_{C-TGS}와 TGT를 얻는다. 이후 네임노드와 상호인증을 원할 때, 인증요청과 타임스탬프를 키 K_{C-TGS}로 암호화하여 TGT와 함께 TGS에게 보낸다.

④ TGS는 클라이언트를 위한 티켓과 네임노드를 위한 티켓을 생성하여 클라이언트에게 보내는데, 이 티켓 안에는 클라이언트와 네임노드가 공유할 세션키 SK가 암호화된 상태로 포함되어 있다.

⑤ 클라이언트는 이를 복호화하여 SK를 얻는다. 다음 SK로 타임스탬프를 암호화하고 TGT로부터 받은 네임노드를 위한 티켓과 함께 네임노드에게 보낸다.

⑥ 네임노드는 키 TGT와 미리 공유한 K_{TGT-N}로 티켓을 복호화하여 SK를 얻고, 그 키로 타임스탬프 T를 확인하여 클라이언트를 인증한다. 다음 $T-1$값을 SK로 암호화하여 클라이언트에게 보내면 클라이언트는 $T-1$값을 확인하고 네임노드를 인증한다.

이 과정을 마친 후 세션키 SK는 삭제된다.

2.4 토큰 시스템

하둡 분산 파일 시스템에서 사용되는 토큰은 위임 토큰(delegation token)과 블록 접근 토큰(block access token)이다. 위임 토큰은 커버로스 인증 이후 네임노드와 클라이언트의 상호인증을 제공하고 블록 접근 토큰은 데이터노드에서 클라이언트에 대한 접근 제어 및 인증을 제공한다. 토큰의 구조는 HMAC-SHA1에 기반을 두는데 이는 해쉬함수 SHA1을 이용하여 구한 메시지 인증 코드(HMAC, Hash-based Message Authentication Code)를 의미한다.

2.4.1 위임 토큰

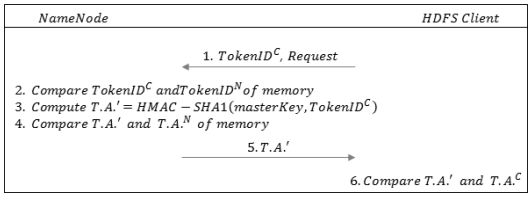
위임 토큰은 네임노드와 클라이언트가 커버로스 인증 없이도 서로의 즉각적인 인증을 위하여 공유한 비밀 값이다. 네임노드와 클라이언트는 최초에 반드시 커버로스를 통해 인증을 받아야 하며, 클라이언트는

커버로스를 통하여 네임노드에게 자신을 인증한 후에만 위임 토큰을 발급받을 수 있다. 위임 토큰은 기본적으로 10시간동안 유효하며 만기기간 내에 갱신 가능하다. 갱신하지 않은 경우엔 커버로스로 다시 인증한 후 위임 토큰을 발급받을 수 있다. 네임노드는 발급한 모든 위임 토큰을 메모리에 유지하며 만료되거나 클라이언트가 갱신하지 않은 토큰들은 삭제한다. 위임 토큰의 구조는 〔표 1〕과 같다[4].

〔표 1〕 위임 토큰의 구조

Delegation Token	TokenID, TokenAuthenticator
TokenID	ownerID, renewerID, issueDate, maxDate, sequenceNumber
Token Authenticator	HMAC-SHA1 (masterKey, TokenID)

위임 토큰은 토큰 아이디(TokenID)와 토큰 인증자(TokenAuthenticator)로 이루어져있다. 토큰 아이디는 만기일을 나타내는 issueDate, maxDate와 토큰을 소유한 클라이언트의 식별자인 소유자 아이디(ownerID), 갱신 시 필요한 갱신 아이디(renewerID), 해당 토큰만의 고유 전역 카운터인 시컨스 넘버(sequenceNumber)로 구성된다. 네임노드는 위임 토큰을 생성하고 검증하는데 사용되는 마스터키(masterKey)를 생성하고 주기적으로 갱신하는데 이 마스터키로 토큰 아이디에 대한 HMAC-SHA1값을 계산한 것이 토큰 인증자이다. 위임 토큰을 사용한 인증과정은 다음 〔그림 4〕와 같다[4].



〔그림 4〕 위임 토큰 인증과정

클라이언트는 최초 커버로스 인증 이후 네임노드로부터 위임 토큰을 발급받은 상태이다. 여기서 클라이언트가 가진 토큰 아이디와 토큰 인증자는 $TokenID^C$와 $T.A.^C$이고 네임노드가 발급한 후 메모리에 유지하고 있는 토큰 아이디와 토큰 인증자는 $TokenID^N$와 $T.A.^N$이다.

① 클라이언트는 블록 연산을 원할 때, 요청 내용과 자신이 가진 토큰 아이디 $TokenID^C$를 네임노드에게 보낸다.
② 네임노드는 $TokenID^C$의 내용을 확인하고 자신의 메모리상의 $TokenID^N$와 비교한다.
③ ②에서 비교한 값이 같다면 masterKey를 사용하여 $TokenID^C$에 대한 토큰 인증자 $T.A.'$를 계산한다.
④ ③에서 계산한 $T.A.'$와 메모리상의 $T.A.^N$를 비교하여 클라이언트를 인증한다. 이 때 클라이언트와 네임노드는 서로에게 자신들이 가진 토큰 인증자를 드러내지 않는다.
⑤ $T.A.'$를 클라이언트에게 보낸다.
⑥ 클라이언트는 자신이 가진 $T.A.^C$와 ⑤에서 받은 $T.A.'$를 비교하여 네임노드를 인증한다.

클라이언트는 한번 위임 토큰을 발급받으면, 토큰이 만기되기 전에는 다른 인증절차 없이 네임노드에게 간편하게 자신을 인증할 수 있으며 블록 접근 권한을 요청할 수 있다.

2.4.2 블록 접근 토큰

블록 접근 토큰은 데이터노드에서 클라이언트에 대한 접근 제어 및 인증을 제공한다. 네임노드에 의하여 발급되며 데이터노드에서 사용되고 검증된다. 네임노드는 마스터키와는 다른 랜덤한 키를 생성하고 이 키를 모든 데이터노드와 하트비트 메시지를 통하여 공유한다. 네임노드는 이 키를 이용하여 토큰을 생성하고 데이터노드는 같은 키로 토큰을 검증한다. 볼록 접근 토큰의 구조는 〔표 2〕와 같다[4].

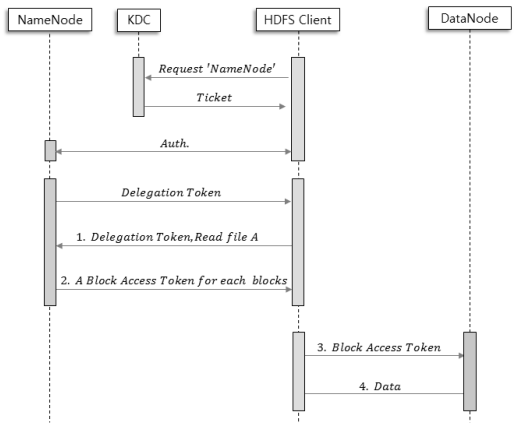
〔표 2〕 블록 접근 토큰의 구조

Block Access Token	TokenID, TokenAuthenticator
TokenID	expirationDate, keyID, ownerID, blockID, accessModes
Token Authenticator	HMAC-SHA1 (key, TokenID)

블록 접근 토큰은 토큰 아이디와 토큰 인증자로 이루어져 있다. 토큰 아이디는 해당 토큰의 만기일(expirationDate)과 토큰을 생성할 때 사용한 키의 식별자(keyID), 토큰을 소유한 클라이언트의 식별자(ownerID), 요청한 블록의 위치정보인 블록 아이디

(blockID), 접근 모드(accessModes)로 구성된다. 접근모드는 파일에 대한 연산 즉 읽기, 쓰기, 복제, 재배치들의 결합이다. 토큰 인증자는 네임노드와 데이터노드가 공유한 key를 사용하여 토큰 아이디에 대한 HMAC-SHA1값을 계산한 것이다.

클라이언트가 파일 A에 대한 읽기요청을 보내는 전체 과정은 다음 〔그림 5〕와 같다.



〔그림 5〕 하둡 분산 파일 시스템 전체 인증 과정

① 클라이언트는 발급받은 위임 토큰과 함께 파일 A에 대한 읽기요청을 네임노드에게 보낸다.
② 네임노드는 위임 토큰으로 클라이언트를 인증한다. 인가된 클라이언트라면 파일 A에 대한 블록 위치를 확인하고 해당 블록에 대한 블록 접근 토큰을 생성하여 클라이언트에게 보낸다.
③ 클라이언트는 블록 접근 토큰을 받아서 해당 데이터노드에게 보낸다.
④ 이를 받은 데이터노드는 키 key를 이용하여, 토큰 아이디에 대한 HMAC-SHA1값을 구한다. 이 값을 클라이언트로부터 받은 블록 접근 토큰의 토큰 인증자와 비교하여, 값이 같으면 클라이언트에게 데이터를 전송하고, 다르면 블록 접근을 통제한다.

2.4.3 키 생성

2.4.2절에서 네임노드는 블록 접근 토큰의 생성과 검증을 위하여 랜덤한 키를 생성하여 사용하였다. 이때 안전성을 위해 하나의 키를 계속 사용하는 것이 아니라 주기적으로 업데이트한다. 자세한 키 롤링 메커니즘(key rolling mechanism)은 다음과 같다.

네임노드는 하둡이 처음 시작할 때 하나의 랜덤한 키를 생성한다. 이 키를 현재키(current key)라고 부른다. 현재키는 일정 시간동안 블록 접근 토큰 발급 요청 발생 시 블록 접근 토큰을 생성하고, 이 키로 생성된 토큰을 검증하는 역할을 한다. 그리고 일정한 시간이 지나면 새로운 키를 생성하고, 새롭게 생성된 키가 현재키가 되며 이전의 키는 준 만료키가 된다. 준 만료키는 바로 삭제되는 것이 아니라 이 키로 발급했던 블록 접근 토큰을 검증하는데 사용된다. 따라서 준 만료키는 이 키로 생성한 모든 토큰들이 만기될 때까지 메모리상에 유지된다. 네임노드는 현재키와 준 만료키들을 메모리상에 유지하며, 이 때 오직 현재키만이 블록 접근 토큰을 생성하는데 사용된다. 블록 접근 토큰에서 keyID란 토큰을 생성할 때 사용한 키의 식별자로 토큰을 검증할 때 keyID를 통해 메모리에서 해당키를 찾는데 사용된다.

네임노드는 데이터노드가 시작할 때 유효한 모든 키를 전송하며 이후 주기적으로 만기된 키 정보와 새로 생성한 키를 업데이트 한다〔4〕.

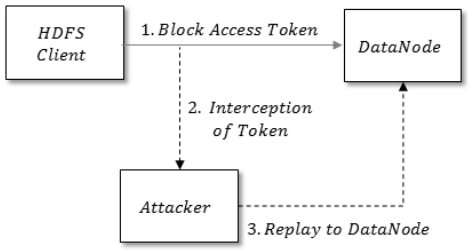
III. 하둡 분산 파일 시스템의 취약점

하둡은 서버와 클라이언트간의 인증을 위해 커버로스와 토큰 시스템을 도입하였다. 하지만 이 시스템에는 재전송 공격과 가장 공격 등이 가능하다는 취약점이 존재한다. 본 3장에서는 이러한 하둡 분산 파일 시스템의 취약점에 대하여 분석한다.

3.1 재전송 공격

하둡 토큰시스템의 큰 문제점중 하나는 토큰에 있는 소유자 아이디(ownerID)이다. 현재 토큰 인증과정에서 토큰 아이디에 대한 해쉬값을 계산하여 값을 비교하는 것 외에 해당 토큰이 실제 소유자로부터 보내졌는지에 대한 검증과정이 없다〔4〕. 따라서 블록 접근 토큰에 대하여 〔그림 6〕과 같은 재전송 공격이 가능하다.

클라이언트가 발급받은 블록 접근 토큰을 데이터노드에게 전송할 때, 공격자가 중간에서 블록 접근 토큰을 가로챈다. 이후 이 공격자 인증된 클라이언트인척 가장하여 가로챈 블록 접근 토큰을 데이터노드에게 보내게 되면 아무런 제지 없이 데이터노드로부터 파일을 전송받게 된다. 이러한 공격은 위임 토큰에도 발생할 수 있다.



〔그림 6〕 하둡 분산 파일 시스템상의 재전송 공격

3.2 가장 공격(impersonation attack)

하둡 분산 파일 시스템은 한 개의 네임노드와 여기에 연결된 수많은 데이터노드로 구성된다. 수천 대의 데이터노드를 모두 안전하게 보관할 수 없기 때문에, 데이터노드가 해킹되었을 때도 안전한 시스템을 만들어야 한다. 따라서 네임노드는 TTP로 가정하고, 데이터 노드는 자체로 공격자가 되거나 공격당할 수 있다고 가정한다.

한 대의 데이터노드가 해킹되었다고 가정하면 다음과 같은 가장 공격이 가능하다. 여기서 vk는 블록 접근 토큰을 생성하고 검증하는 키로, 네임노드와 데이터노드가 사전에 공유한 비밀키이다. 클라이언트가 회사 직원들의 급여 내역이 적힌 문서 파일의 읽기 연산 요청과 위임 토큰을 네임노드에게 보낸다고 하자. 악의적인 데이터노드는 위임 토큰과 블록 연산 요청을 보낼 때 이를 가로챈다. 그리고 자신이 미리 생성해 놓은 변조된 급여 내역 파일에 대한 블록 접근 토큰을 생성한다. 이때 하둡 분산 파일 시스템 전체의 블록 아이디는 네임노드가 관리하지만 데이터노드는 자신에게 저장된 블록들의 블록 아이디를 소유하고 있고 키 vk를 소유하고 있으므로 변조된 파일에 대한 블록 접근 토큰 생성이 가능하다. 이렇게 생성한 토큰을 자신이 네임노드인척 가장하여 클라이언트에 보내면 클라이언트는 블록 아이디에 대한 정보는 알 수 없으므로 정당하게 발급된 토큰이라고 인지하여 잘못된 급여 내역을 보게 된다.

이처럼 커버로스를 통한 인증 이 후, 토큰 시스템 사용 시에는 데이터노드가 클라이언트에게 네임노드인척 가장하는 것이 가능하다. 따라서 토큰을 사용할 때도 네임노드와 클라이언트간의 간단한 상호인증이 필요하다.

3.3 암호화 통신 및 키 교환 문제

하둡 사용 시 네임노드와 데이터노드에 SSH (Secure SHell)를 설치하고 비밀번호를 설정하지 않으면, 하둡이 동작하지 않는다. 따라서 네임노드와 데이터노드가 주고받는 모든 메시지는 암호화되어있으며 상호 인증을 제공받는다. 하지만 클라이언트와 네임노드, 클라이언트와 데이터노드 사이에는 암호화 통신이 제공되지 않는다. 하둡은 성능 향상과 암호화의 비용적인 문제 때문에 SASL(Simple Authentication and Security Layer)의 정보보호 수준(QOP, Quality Of Protection)의 기본 설정(default)이 매우 취약한 상태이다〔2〕. 또한 하둡이 고려한 것처럼 암호화 통신을 하게 되면 다음과 같은 하둡의 성능 저하가 예상되며 많은 계산량과 더불어 고비용이 발생할 것이다.

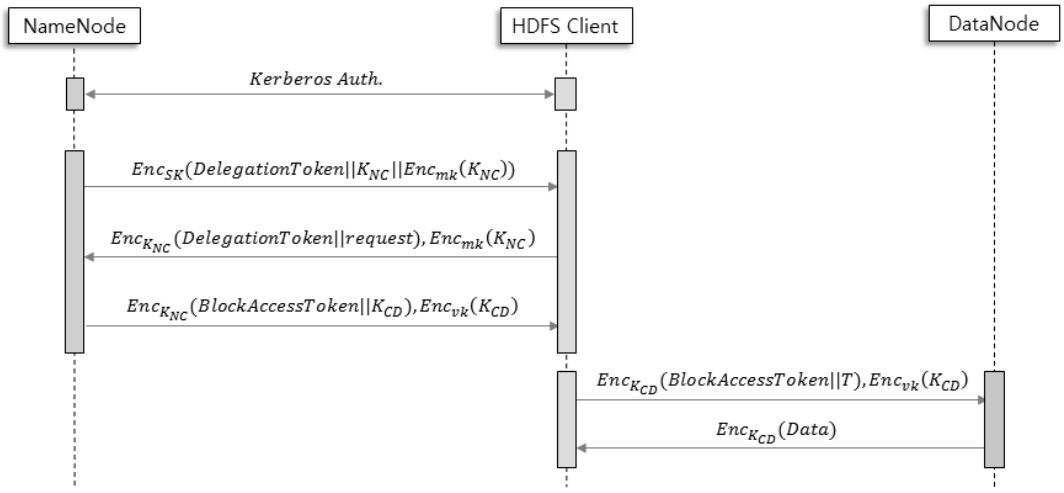
암호화 통신을 위해서는 모든 데이터노드와 클라이언트, 네임노드가 사전에 키를 교환해야 한다. 하지만 현재 사용자가 설정하여 사용할 수 있는 키 교환 시스템은 공개키 키 교환 시스템뿐이다. 따라서 모든 클라이언트와 데이터노드가 암호화 통신을 위하여 공개키 시스템으로 키 교환을 해야만 한다. 네임노드가 1대, 데이터노드가 n대, 클라이언트가 m명이라고 가정하자. 네임노드와 클라이언트의 암호화 통신을 위해서는 m번의 키 교환이 일어나야 하며, 모든 데이터노드와 클라이언트가 암호화 통신을 하기 위해서는 $n \times m$번의 키 교환이 일어나야 한다. 실제로 데이터노드가 수백에서 수천대로 구성되어 있는 것을 고려하면 계산량이 매우 많고, 비용 또한 상당하다. 게다가 키 교환 후에 각각의 데이터노드는 클라이언트 수 이상의 키를 저장해야만 하며 클라이언트는 데이터노드의 수 이상의 키를 저장해야만 한다.

IV. 제안하는 프로토콜

4.1 목적

제안하는 서버-클라이언트 인증 프로토콜은 다음과 같은 성질을 만족한다.

- 오직 토큰을 발급받은 클라이언트만이 블록 접근 권한을 갖도록 한다.
- 토큰 사용 시에도 클라이언트와 데이터노드, 클라이언트와 네임노드 간의 인증이 가능하도록 한다.



〔그림 7〕 제안하는 프로토콜

- 악의적인 데이터노드가 네임노드인척 가장할 수 있는 문제점을 해결한다.
- 암호화 통신을 위한 키 교환 시스템을 제안한다. 이때 암호화 통신은 사용자 선택사항으로 둔다.

단 이러한 조건을 만족하되 하둡의 성능적인 부분과 비용을 위하여 공개키 시스템이 아닌 대칭키 시스템만을 이용한다.

4.2 서버-클라이언트 인증 프로토콜

본 장에서는 제안하는 서버-클라이언트 인증 프로토콜은 〔그림 7〕과 같으며 다음과 같은 단계로 구성되어 있다.

4.2.1 초기화

네임노드와 클라이언트는 커버로스를 통해 상호인증을 받는다. 네임노드는 클라이언트와 커버로스 인증을 통해 공유한 세션키를 대칭키 SK의 값으로 저장하고 위임 토큰을 생성 및 검증할 마스터키 mk를 설정한다. 이때 mk는 네임노드만 소유한다. 클라이언트 또한 네임노드와 커버로스 인증을 통해 공유한 세션키를 대칭키 SK의 값으로 저장한다. 데이터노드는 안전한 대칭키 암호화 기법 Enc를 설정한다.

① 네임노드는 클라이언트에 대한 위임 토큰을 발급하고 클라이언트와 통신 시 사용할 키 K_{NC}를 생성한 후 다음과 같이 암호화한다.
$Enc_{mk}(K_{NC})$

② 위임 토큰과 K_{NC}, $Enc_{mk}(K_{NC})$를 대칭키 SK로 다음과 같이 암호화하여 클라이언트에게 전송한다.
$Enc_{SK}(DelegationToken||K_{NC}||Enc_{mk}(K_{NC}))$

③ 클라이언트는 네임노드로부터 받은 암호문을 대칭키 SK로 복호화하여 위임 토큰과 키 K_{NC}, $Enc_{mk}(K_{NC})$를 얻는다.

이 과정을 마친 후 네임노드와 클라이언트는 대칭키 SK를 삭제한다.

4.2.2 네임노드 - 클라이언트 인증

네임노드와 클라이언트 인증과정은 다음과 같다.

① 클라이언트는 블록 연산을 원할 때, 위임 토큰과 요청을 K_{NC}로 암호화하여 $Enc_{mk}(K_{NC})$와 함께 네임노드에게 전송한다.

② 네임노드는 $Enc_{mk}(K_{NC})$를 마스터키 mk로 복호화하여 키 K_{NC}를 얻고, 이 키로 위임 토큰과 요청의 원문을 얻는다.

③ 네임노드는 ②에서 얻은 위임 토큰을 검증하고, 인가된 클라이언트라면 해당 파일에 대한 블록 접근 토큰과 클라이언트와 데이터노드가 공유할 비밀키 K_{CD}를 생성한다.

④ 비밀키 K_{CD}를 다음과 같이 암호화한다.
$Enc_{vk}(K_{CD})$

⑤ 키 K_{NC}로 ②~③에서 생성한 블록 접근 토큰, K_{CD}, $Enc_{vk}(K_{CD})$을 다음과 같이 암호화하여 클라이언트에게 전송한다.

$Enc_{K_{NC}}(BlockAccessToken \| K_{CD}), Enc_{vk}(K_{CD})$

⑥ 클라이언트는 네임노드로부터 받은 암호문을 키 K_{NC}로 복호화하여 블록 접근 토큰과 K_{CD}, $Enc_{vk}(K_{CD})$를 얻는다.

클라이언트는 위임 토큰과 함께 $Enc_{mk}(K_{NC})$값을 보관하고 있으며 위임 토큰이 만기되어 폐기할 때 $Enc_{mk}(K_{NC})$도 함께 폐기한다. 이후 커버로스 재인증 후 위임 토큰을 발급받을 때 $Enc_{mk}(K_{NC})$도 새로 발급받는다.

4.2.3 클라이언트 - 데이터노드 인증

클라이언트와 데이터노드 인증과정은 다음과 같다.

① 클라이언트는 블록 접근 토큰과 타임스탬프를 다음과 같이 암호화한다.
$Enc_{K_{CD}}(BlockAccessToken, T)$

② 네임노드로부터 받은 $Enc_{vk}(K_{CD})$와 ①에서 구한 암호문을 데이터노드에게 보낸다.

③ 데이터노드는 네임노드와 공유한 키 vk로 $Enc_{vk}(K_{CD})$를 복호화하여 비밀키 K_{CD}를 얻는다.

④ 클라이언트와의 공유키 K_{CD}를 사용하여 암호문 $Enc_{K_{CD}}(BlockAccessToken, T)$를 복호화하고, 타임스탬프를 확인한 후, 블록 접근 토큰을 인증한다.

⑤ 인가된 사용자라면, 데이터를 K_{CD}로 다음과 같이 암호화하여 클라이언트에게 보낸다.
$Enc_{K_{CD}}(Data)$

⑥ 클라이언트는 비밀키 K_{CD}로 원본 데이터를 얻는다.

암호화 통신을 사용하지 않는 경우, 과정 ⑤~⑥은 기존의 시스템처럼 원본 데이터를 전송한다.

V. 분석

5.1 안전성 분석

제안된 프로토콜에서는 네임노드와 클라이언트, 데이터노드와 클라이언트간의 즉각적인 인증이 가능하다. 세션키 SK는 커버로스 인증을 통하여 네임노드와 클라이언트가 나누어가진 키이기 때문에, 오직 네임노드만이 SK로 암호화할 수 있다. 따라서 SK로 암호화한 값을 클라이언트에게 보내면, 클라이언트는 SK를 통해 네임노드를 인증할 수 있으며 데이터노드나 공격자가 네임노드인척 가장할 수 없다. 또한 인증된 클라이언트만이 메시지의 원본을 얻을 수 있으므로 클라이언트 인증이 가능하다.

마찬가지로 네임노드만 가지고 있는 마스터키로 $Enc_{mk}(K_{NC})$를 생성하기 때문에, 이를 통해 클라이언트는 네임노드를 인증할 수 있으며, 커버로스를 통해 발급받은 세션키 SK를 가진 클라이언트만이 K_{NC}를 얻을 수 있기 때문에 인가된 클라이언트만이 네임노드에게 위임 토큰을 전송할 수 있다.

K_{CD}는 네임노드로부터 인가된 클라이언트만이 얻을 수 있는 키이므로, 이 키로 블록 접근 토큰과 타임스탬프를 암호화함으로써 데이터노드가 클라이언트를 인증할 수 있으며, 다른 공격자가 클라이언트인척 가장할 수 없다.

4장에서 분석한 취약점의 안전성에 대한 자세한 사항은 다음과 같다.

5.1.1 재전송 공격

제안하는 프로토콜에서 공격자가 클라이언트가 네임노드에게 보내는 암호문 $Enc_{K_{NC}}(D \| request)$과 $Enc_{mk}(K_{NC})$를 가로챘다고 가정하자. 이때 D는 위임 토큰, B는 블록접근이다. 공격자가 자신이 정당한 클라이언트인척 가장하여 네임노드에게 이 값을 보내면 공격자는 네임노드로부터 $Enc_{K_{NC}}(B \| K_{CD})$, $Enc_{vk}(K_{CD})$을 받을 것이다. 하지만 공격자는 키 K_{NC}가없으므로 이 값을 복호화할 수 없다. 따라서 키 K_{CD}를 얻을 수 없고 데이터노드에게 보낼 값 $Enc_{K_{CD}}(B \| T)$도 만들 수 없으므로 재전송 공격이 불가능하다.

또한 공격자가 클라이언트가 데이터노드에게 보내는 $Enc_{K_{CD}}(B \| T), Enc_{vk}(K_{CD})$값을 가로챘다고 가정하자. 공격자가 자신이 정당한 클라이언트인척 가장하여 데이터노드에게 이 값을 보내면, 데이터노드는 타임스탬프 T값을 확인하고, 정당하지 않은 요청이라는 것을 알게 되어 재전송 공격이 불가능하다. 또한 공격자는 비밀키 K_{CD}를 모르기 때문에 타임스탬프를 수정할 수도 없다.

5.1.2 가장 공격

제안하는 프로토콜에서 데이터노드가 공격당하더라

〔표 3〕 HDFS, 공개키 시스템을 사용하는 HDFS, 제안하는 프로토콜의 성능과 안전성 비교

n: 데이터 노드 수 m: 클라이언트 수		HDFS	HDFS with public key system	Proposed protocol
총 라운드 수		6	$6+\alpha$	6
저장하는 키	네임노드	세션키, mk, vk	mk, vk + m개의 키	세션키, mk, vk
	클라이언트	세션키	세션키 + 네임노드와 공유키 + n개의 키	세션키 K_{CD}, $Enc_{vk}(K_{CD})$ K_{NC}, $Enc_{mk}(K_{NC})$
	데이터노드	vk	vk + m개의 키	vk
키 교환 수	네임노드 - 클라이언트	0	m	0
	데이터노드 - 클라이언트	0	$n\times m$	0
안전성 분석	재전송 공격 내성	×	O	O
	가장 공격 내성	×	O	O
	데이터 암호화	×	O	O

도 클라이언트에게 네임노드인척 가장하는 것이 불가능하다.

하둡 분산 파일 시스템 상에 하나의 악의적인 데이터노드가 존재한다고 가정하자. 클라이언트가 네임노드에게 보내는 요청을 가로채고 자신이 네임노드인척 가장하려 한다. 하지만 데이터노드는 마스터키 mk가 없으므로 K_{NC}를 얻을 수 없으며 클라이언트가 보내는 요청내용을 알 수 없다. 또한 vk로 블록 접근 토큰을 생성할 수 있다고 하더라도 클라이언트에게 전송할 포맷 $Enc_{K_{NC}}(B \| K_{CD})$을 생성할 수 없다. 따라서 가장 공격이 불가능하다.

5.1.3 암호화 통신 및 키 교환 문제

제안하는 프로토콜에서는 사용자의 선택에 따라 클라이언트와 데이터노드 간에 암호화 통신을 가능하게 한다. 또한 암호화 통신을 하더라도 기존 프로토콜과 같은 키 교환 문제가 발생하지 않는다. 자세한 사항은 5.2절에서 설명한다.

5.2 성능 분석

〔표 3〕은 하둡 파일 시스템과 안전성을 위하여 공개키 시스템을 사용한 하둡 파일 시스템 그리고 제안한 프로토콜의 성능과 안전성을 비교한 것이다.

먼저 키와 토큰 저장을 위해 사용하는 메모리에 대한 분석은 다음과 같다. 네임노드는 기존과 같이 vk만 유지하고 있으며, 기존에 커버로스 인증시 사용하고 삭제되는 세션키는 SK에 저장시켜 위임 토큰을 발행할 때 까지 추가적으로 유지한다. 또한 클라이언트가 $Enc_{mk}(K_{NC})$값을 보내주기 때문에 클라이언트에 대한 키를 따로 저장하지 않는다. 마찬가지로 데이터노드도 클라이언트가 $Enc_{vk}(K_{CD})$값을 보내주기 때문에 클라이언트에 대한 어떤 키도 저장하고 있지 않는다. 기존 프로토콜에서 각각의 클라이언트는 블록 접근 토큰을 메모리에 유지하고 있는데, 이와 함께 $Enc_{vk}(K_{CD})$와 K_{CD}를 테이블 형태로 저장한다. 마찬가지로 K_{NC}와 $Enc_{mk}(K_{NC})$도 위임 토큰과 함께 테이블 형태로 저장한다. 이 값들은 토큰이 만기될 때 함께 폐기한다. 따라서 제안하는 프로토콜의 사용 메모리는 기존 시스템과 비슷하며, 공개키 시스템을 사용하는 하둡 파일 시스템과 비교하면 매우 적은 메모리를 사용한다.

또한 암호화 통신시 공개키 시스템을 사용하는 하둡 파일시스템과 제안하는 프로코톨의 키 교환 횟수를 비교하면 다음과 같다. 공개키 시스템 하둡 파일시스템은 기존 6라운드의 프로토콜 이외에 네임노드와 클라이언트 사이에 m번, 데이터노드와 클라이언트에 $n\times m$번의 추가적인 키교환이 필요한 반면에 제안하는 프로토콜은 기존 6라운드의 인증과정에서 키교환을 함께 수행하므로 따로 키 교환이 필요하지 않다. 따라서 기존 시스템에 비해 비용적인 면과 계산량에서 매우 향상되었다.

VI. 결 론

본 논문에서는 하둡 분산 파일 시스템에 도입된 토큰시스템이 재전송 공격이나 가장 공격과 같은 공격에 취약하다는 것을 알 수 있었다. 기존 프로토콜을 강화

하여 토큰 시스템 사용 시에도 네임노드와 클라이언트, 데이터노드와 클라이언트 간에 즉각적인 상호인증이 가능한 새로운 프로토콜을 제안하였다. 또한 이 키를 데이터노드와 클라이언트가 암호화 통신에 사용하여 추가적인 키 교환을 필요로 하지 않도록 하였다. 이 때 공개키 시스템이 아닌 대칭키 시스템으로만 구성하여 하둡의 성능이나 비용적인 특징을 유지하였다.

참고문헌

〔1〕 K. Shvachko, H. Huang, S. Radia, and R. Chansler, "The hadoop distributed file system," Proceedings of the 2010 IEEE 26th Symposium on Massive Storage Systems and Technologies (MSST), pp. 1-10, May 2010.

〔2〕 A. Becherer, "Hadoop security design just add kerberos? really?," iSEC PARTNER, 2010.

〔3〕 C. Neuman, T. Yu, S. Hartman, and K. Raeburn, "The kerberos network authentication service (V5)," RFC 4120, Jul. 2005.

〔4〕 O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," http://bit.ly/75011o, Oct. 2009.

〔5〕 Apache Hadoop, http://hadoop.apache.org/

〔6〕 S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," Proceedings of ACM Symposium on Operating Systems Principles, pp. 29 - 43, Oct. 2003.

〔7〕 T. White, "Hadoop: the definitive guide," O'Reilly Media, Yahoo! Press, Jun. 2009.

〔8〕 E. Sammer, "Hadoop operations," O'R-eilly Media, Oct. 2012.

〔9〕 D. Borthakur, "The hadoop distributed file system: architecture and design," http://hadoop.apache.org/docs/r1.0.4/hdfs_design.pdf

〔10〕 J. Gantz and D. Reinsel, "Extracting value from chaos," IDC, Jun. 2011.

〔11〕 A. Melnikov and K. Zeilenga, "Simple authentication and security layer (SASL)," RFC 4422, Jun. 2006.

〔12〕 J. Dean and S. Ghemawat, "Map- Reduce: simplified data processing on large clusters," Communications of the ACM, pp. 107 - 113, 2008.

〔13〕 김형준, 조준호, 안성화, 김병준, 클라우드 컴퓨팅 구현 기술, 에이콘 출판사, 2012년 5월.

〔14〕 O. O'Malley, "Integrating kerberos into apache hadoop," Kerberos Conference 2010, Oct. 2010.

〈저 자 소 개〉

박 소 현 (So Hyeon Park) 학생회원
2012년 2월: 서울시립대학교 수학과 학사
2012년 3월~현재: 고려대학교 정보보호대학원 석사
〈관심분야〉 프라이버시향상기술(PET), 빅데이터 보안, 클라우드 컴퓨팅

정 익 래 (Ik Rae Jeong) 정회원
1998년 2월: 고려대학교 전산학과 학사 졸업
2000년 2월: 고려대학교 전산학과 석사 졸업
2004년 8월: 고려대학교 정보보호대학원 박사 졸업
2006년 6월~2008년 2월: 한국전자통신연구원 암호기술연구팀 선임연구원
2008년 3월~현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안, 암호 이론