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.2013.23.5.831

해쉬 체인 기반의 안전한 하둡 분산 파일 시스템 인증 프로토콜*

정 소 원,† 김 기 성, 정 익 래‡
고려대학교 정보보호대학원

Secure Authentication Protocol in Hadoop Distributed File System based on Hash Chain*

So Won Jeong,† Kee Sung Kim, Ik Rae Jeong‡
Graduate School of Information Security, Korea University

요 약

모바일 대중화에 따른 소셜 미디어의 확산과 함께 다양한 형태의 데이터가 대량으로 생산되고 있다. 이에 따라 대규모 데이터 분석을 통해 가치 있는 비즈니스 정보를 얻고자 하는 기업들의 빅데이터 기술 도입 및 활용 또한 날로 증가하는 추세이다. 특히, 하둡은 테라바이트 단위의 파일 저장 능력과 저렴한 구축비용, 빠른 데이터 처리 속도로 가장 대표적인 빅데이터 기술로 손꼽힌다. 하지만 현재 하둡 분산 파일 시스템의 사용자 인증을 위한 인증 토큰 시스템은 토큰 재전송 공격, 데이터노드 해킹 공격에 취약하다. 이는 하둡 분산 파일 시스템 상에 저장된 기업 기밀 데이터 및 고객 개인 정보 등의 안전을 위협할 수 있다. 본 논문에서는 토큰 및 데이터노드가 공격자에게 노출되었을 때 발생 가능한 하둡 분산 파일 시스템의 보안 취약성을 분석하고, 해쉬 체인을 사용한 보다 안전한 하둡 분산 파일 시스템 인증 프로토콜을 제안한다.

ABSTRACT

The various types of data are being created in large quantities resulting from the spread of social media and the mobile popularization. Many companies want to obtain valuable business information through the analysis of these large data. As a result, it is a trend to integrate the big data technologies into the company work. Especially, Hadoop is regarded as the most representative big data technology due to its terabytes of storage capacity, inexpensive construction cost, and fast data processing speed. However, the authentication token system of Hadoop Distributed File System(HDFS) for the user authentication is currently vulnerable to the replay attack and the datanode hacking attack. This can cause that the company secrets or the personal information of customers on HDFS are exposed. In this paper, we analyze the possible security threats to HDFS when tokens or datanodes are exposed to the attackers. Finally, we propose the secure authentication protocol in HDFS based on hash chain.

Keywords: Big Data, Hadoop, HDFS, Block Access Token, Hash Chain

접수일(2013년 5월 30일), 수정일(2013년 7월 19일), 게재확정일(2013년 9월 12일)

* 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, jsw0429@korea.ac.kr
‡ 교신저자, irjeong@korea.ac.kr(Corresponding author)

I. 서 론

최근 스마트 기기의 확산과 무선 네트워크의 발달에 따른 페이스북, 트위터 등의 다양한 소셜 미디어의 성장이 가히 폭발적인 수준이다. 소셜 미디어 사용자들은 스마트 기기를 이용하여 언제 어디서나 자유롭게 원하는 콘텐츠를 수집 · 활용하고 게시할 수 있으며, 이 과정에서 텍스트, 그림, 비디오 등 다양한 형태의 비정형 데이터들이 실시간으로 생산되고 있다. 한편 이렇게 생성된 대규모 데이터를 분석, 가공하여 사용자 정보, 소비자 기호와 시장 상황, 소비자 행태 등의 가치 있는 정보를 추출하여 정보 우위를 차지하려는 국가나 기업들의 정보 수집 행위도 함께 증가하고 있다. 이러한 지속적인 대규모 데이터 생산과 데이터 수집 행위는 빅데이터 시대의 도화선이 되었다.

빅데이터는 2001년 Meta Group(현 Gartner)의 Doug Laney의 연구에서 처음 사용된 용어로, 일반적인 데이터베이스가 저장, 관리, 분석할 수 있는 범위를 벗어나는 대규모의 데이터를 의미한다. 빅데이터는 정형 데이터와 비정형 데이터를 모두 포괄하는데 최근에는 비정형 데이터가 대부분을 차지하고 있다. 하지만 기존의 데이터 처리에 사용되던 관계형 데이터베이스 관리 시스템(RDBMS, Relational Data Base Management System)은 정제되고 체계화된 정형 데이터 처리에 적합하도록 구현된 것으로 대규모의 비정형 데이터 처리에는 알맞지 않는다. 특히 기존의 방식은 고가의 장비로 이루어져 있어 데이터 규모가 커질수록 시스템 구축에 고비용이 발생한다. 이에 따라 저렴한 비용으로 대규모 데이터를 처리할 수 있는 빅데이터 기술에 대한 기업들의 요구가 증가하게 되었다.

대규모 기업 데이터 및 고객 정보를 다루는 많은 기업들에서 사용되는 대표적인 빅데이터 기술로는 하둡(Hadoop)[1]을 꼽을 수 있다. 하둡은 2005년 Doug Cutting에 의해 개발된 대용량 분산 처리를 위한 자바 기반의 오픈소스 프레임워크이다. 이는 대규모 데이터를 분산 저장, 관리하는 하둡 분산 파일 시스템(HDFS, Hadoop Distributed File System)과 대규모 데이터를 분석, 처리하는 맵리듀스(MapReduce)를 포함한다.

하둡은 저렴한 구축 비용과 빠른 데이터 처리 속도를 장점으로 다양한 기업들에 빠르게 확산되었지만 기업 데이터와 고객 개인 정보 등 민감한 데이터를 관리하기 위해 마련된 보안 장치가 따로 존재하지 않았다. 이에 따라 하둡은 보안 향상을 위하여 커버로스를 통한 초기 인증과 토큰 시스템을 통한 사후 인증을 채택하게 되었다. 그러나 하둡 분산 파일 시스템의 토큰 인증 방식은 토큰 갈취 후 재전송 공격에 취약하며, 데이터노드 해킹 시 임의로 토큰을 생성하여 데이터에 접근하는 데이터노드 해킹 공격이 가능하다. 따라서 본 논문에서는 이를 해결하기 위한 해쉬 체인 기반의 안전한 하둡 분산 파일 시스템 인증 프로토콜을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 하둡 분산 파일 시스템의 구성 및 보안에 대해 기술한다. 3장, 4장에서는 기존의 하둡 분산 파일 시스템 상에 존재하는 보안 취약점을 분석하고 이를 해결한 새로운 기법을 제안한다. 5장에서는 그에 따른 안전성 및 성능을 분석하며, 6장에서 결론을 맺는다.

II. 하둡 분산 파일 시스템

하둡(Hadoop)[1]은 대표적인 빅데이터 기술로써 대용량 데이터 분석 처리를 위한 오픈소스 프레임워크이다. 이는 2003년, 2004년에 발표되었던 구글의 구글 분산 파일 시스템(GFS, Google File System)[4]과 맵리듀스(MapReduce)[5]를 구현한 것으로, 그 설계와 아이디어가 많은 부분에서 구글의 것과 유사하다. 하둡은 대용량 데이터을 분산 저장하고 관리하는 하둡 분산 파일 시스템(HDFS, Hadoop Distributed File System)[2]과 대용량 데이터의 분석을 수행하는 하둡 맵리듀스(Hadoop MapReduce)[3]로 구성된다.

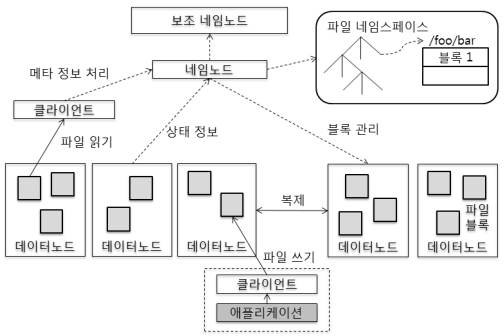
하둡 분산 파일 시스템은 다수의 리눅스 서버에 설치되어 운영되며, 뛰어난 확장성으로 페타바이트 이상의 대용량 데이터 저장 공간을 확보할 수 있다. 또한 여러 대의 서버가 동시에 데이터를 분산 처리하여 대규모 데이터 처리에 있어 빠른 속도를 보장한다. 특히, 리눅스 장비를 사용함으로써 고가의 장비를 사용하는 RDBMS에 비해 시스템 구축비용이 저렴하다. 저가의 장비 사용으로 인한 데이터 유실 및 저장 실패 등의 문제는 데이터 복제본을 분산 저장하여 해결한다.

초기 하둡 분산 파일 시스템은 데이터 무결성을 보장하기 위하여 한번 저장한 데이터는 더는 수정할 수 없고 이동, 삭제, 복사, 읽기만 가능하였으나, 추후에는 저장된 파일에 붙여넣기(append)할 수 있는 기능이 추가되었다.

2.1 하둡 분산 파일 시스템 구성

하둡 분산 파일 시스템[2]은 대규모 데이터를 분산 저장 및 관리하기 위한 분산 파일 시스템으로서, 저장하고자 하는 파일을 블록 단위로 나누어 분산된 서버에 저장한다. 블록 사이즈는 기본적으로 64MB로 설정되어 있으며, 데이터가 64MB로 나누어떨어지지 않는 경우에 블록을 나누고 남은 부분은 그 크기 그대로의 블록으로 저장된다. 나누어진 블록은 장애 발생 시 데이터 손실을 방지하기 위해 기본 3개씩 복제되어 분산 저장된다. 블록 사이즈와 블록 복제본의 수는 하둡 환경 설정 파일에서 수정할 수 있다.

하둡 분산 파일 시스템은 [그림 1]과 같이 하나의 네임노드(NameNode), 보조 네임노드(Secondary NameNode)와 다수의 데이터노드(DataNode)로 구성된다[6]. 이 때, 클라이언트는 하둡 분산 파일 시스템에 파일을 저장하거나 저장된 파일을 읽기 위해서 사용자에게 애플리케이션 형태로 제공되는 프로그램이다.



[그림 1] 하둡 분산 파일 시스템 구성[6]

네임노드는 하둡 분산 파일 시스템의 모든 메타데이터(디렉토리명, 파일명, 파일 블록 등에 대한 트리 형태의 네임 스페이스)를 관리하며, 클라이언트가 하둡 분산 파일 시스템 상에서 파일 읽기 및 저장을 요청할 때 메타데이터를 기반으로 데이터노드에 저장된 블록 위치를 조회하거나 파일의 복제본이 저장될 데이터노드를 결정한다.

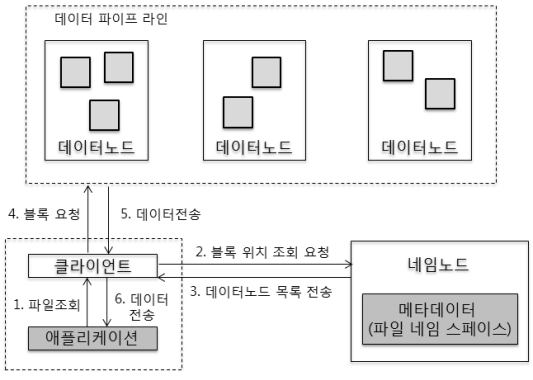
데이터노드는 블록 단위로 나눠진 데이터를 저장하는 데이터 서버로서, 네임노드와 클라이언트의 데이터 입출력 요청을 관리한다. 데이터노드는 주기적으로 자신의 상태 정보를 포함한 하트비트(heartbeat) 메시지와 블록의 목록을 담은 블록 리포트(block report)를 네임노드에게 전송한다. 네임노드는 하트비트와 블록 리포트를 통하여 데이터노드의 정상 작동 여부와 데이터노드 내의 모든 블록 목록을 확인하고, 네임노드와 클라이언트의 파일 읽기 및 저장 요청 시 활용한다.

하둡 분산 파일 시스템은 전체 파일 시스템의 네임스페이스를 관리하는 네임노드가 하나이기 때문에 네임노드 손상 시 시스템 전체가 마비될 수 있다. 보조 네임노드는 네임노드의 장애 발생 시 시스템 마비를 방지하기 위해서 기본 1시간 단위로 네임노드의 네임스페이스를 파일 시스템 이미지 파일로 백업해두는 역할을 한다. 네임노드 손상 시 보조 네임노드에 백업된 파일 시스템 이미지로 하둡 분산 파일 시스템의 파일 네임스페이스를 재구성 할 수 있다. 하지만 보조 네임노드에 백업된 파일 네임스페이스 정보는 네임노드의 것보다 최신이 아니기 때문에 어느 정도의 데이터 손실이 발생할 수 있다[9]. 파일 시스템 이미지 파일 백업 간격은 하둡 환경 설정 파일에서 수정할 수 있다.

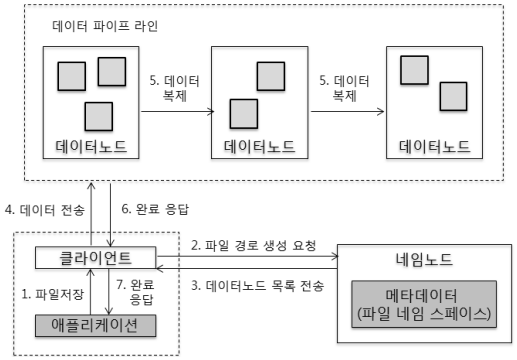
2.2 하둡 분산 파일 시스템 동작 방식

사용자가 클라이언트를 통하여 하둡 분산 파일 시스템에 파일 저장 및 읽기를 요청할 때 시스템 동작 방식은 [그림 2], [그림 3]과 같다[7]. 모든 서버는 TCP(Transmission Control Protocol) 기반 프로토콜을 이용하여 통신한다.

사용자가 클라이언트를 통하여 네임노드에 파일 저장을 요청하는 경우, 네임노드는 데이터노드들로부터 받은 하트비트 메시지와 블록 리포트를 참고하여 파일의 복제본이 저장될 데이터노드 목록을 반환한다. 클라이언트는 목록 내에 데이터노드에 접근하여 블록을 저장한다.



[그림 2] 하둡 분산 파일 시스템 파일 읽기 동작 방식[7]



〔그림 3〕 하둡 분산 파일 시스템 파일 저장 동작 방식〔7〕

사용자가 클라이언트를 통하여 네임노드에게 파일 읽기를 요청하는 경우, 네임노드는 파일 네임 스페이스를 통해 요청받은 파일의 복제 블록들이 저장된 데이터노드의 목록을 반환한다. 클라이언트는 목록 내의 데이터노드들에 블록을 요청하고, 요청을 받은 데이터노드는 해당 블록을 사용자에게 전달한다.

2.3 하둡 분산 파일 시스템 보안

초기 하둡은 시스템을 사용하는 사용자들은 서로 신뢰 관계에 있고, 안전한 환경에서 시스템이 운영된다는 가정 하에 설계되었다. 따라서 데이터 유실을 막는 것에만 초점을 두었을 뿐 시스템 보안에 대한 고려는 거의 없었다. 하지만 하둡을 사용하는 기업들이 증가하면서, 하둡은 기업 기밀 데이터 및 고객 개인 정보와 같은 민감한 데이터들을 대량으로 다루기 시작하였고, 이에 따라 하둡 보안의 필요성이 거론되었다.

초기에는 하둡에 대한 사용자 접근 제어를 위한 메커니즘으로 SSL(Secure Socket Layer)이 제안되었다. 그러나 SSL은 공개키 방식을 사용하여 처리 속도가 느리고 계산량이 많다는 단점을 가진다. 따라서 현재는 SSL 방식은 거의 사용하지 않고, 대칭키 방식의 커버로스(Kerberos) 인증을 사용하고 있다.

하둡의 커버로스 인증 도입은 2009년 야후에 의해 제안되었으며〔8〕, 제안된 내용에는 자체적인 인증 토큰 시스템도 포함되어 있다. 커버로스 및 인증 토큰 시스템은 하둡 0.20 시리즈들이 발표된 이후부터 적용되었다. 인증 토큰 시스템은 클라이언트와 네임노드 사이의 빈번한 호출로 발생할 수 있는 커버로스의 과부하 현상을 막기 위해 제안된 것으로 자세한 내용은 2.5장에서 살펴본다.

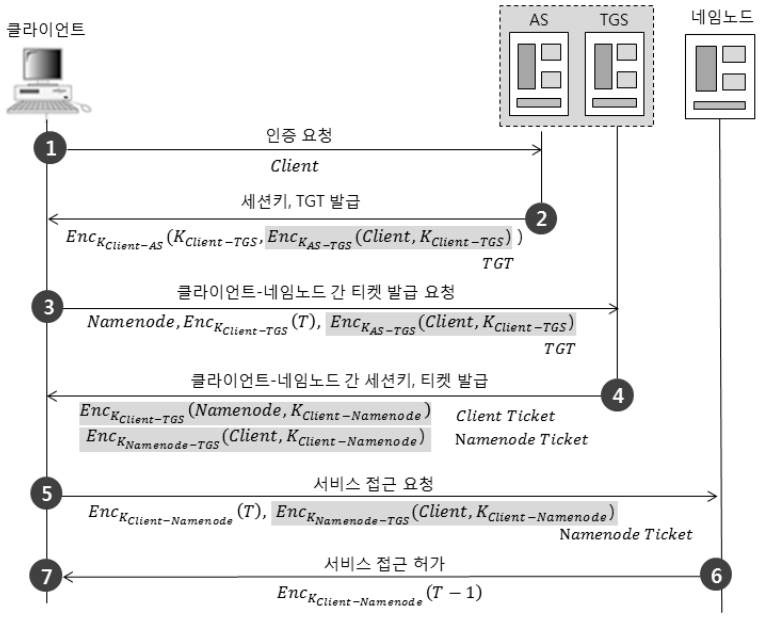
2.4 커버로스

커버로스〔11〕는 티켓을 통하여 클라이언트와 서버 간의 안전한 상호 인증을 제공하는 네트워크 인증 프로토콜이며, 그와 동시에 인증에 사용되는 대칭키를 분배하는 키 분배 센터(KDC, Key Distribution Center)이기도 하다. 커버로스 인증 프로토콜은 대칭키 기반으로 이루어지며, KDC는 신뢰할 수 있는 제 3자이다.

커버로스는 인증 서버(AS, Authentication Server), 티켓 발급 서버(TGS, Ticket Granting Server), 서비스 서버(Service Server)로 구성된다. 인증 서버는 사용자를 인증하며, 사용자에게 사용자와 티켓 발급 서버 사이에 사용될 세션키를 발급하여 사용자와 티켓 발급 서버 간의 암호화 통신을 돕는다. 티켓 발급 서버(TGS, Ticket Granting Server)는 서비스 서버에 대한 사용자의 인증값인 티켓을 사용자에게 발급해주는 역할을 한다. 서비스 서버는 사용자에게 서비스를 제공하는 데이터 서버이다.

하둡 분산 파일 시스템에서는 클라이언트가 사용자에 해당되며, 네임노드가 서비스 서버이다. 각 클라이언트는 커버로스를 통한 인증을 제공받기 위해 사전에 클라이언트 아이디와 패스워드를 인증 서버에 등록하고, 인증 서버는 클라이언트 아이디와 대응되는 패스워드를 보관한다. 하둡 분산 파일 시스템 상의 자세한 커버로스 인증 과정은 〔그림 4〕와 같으며〔13〕, 이 때 K_{a-b}는 a, b 가 공유한 대칭키이고 T는 타임스탬프이며, $Enc_K(M)$은 비밀키 K로 평문 M을 암호화하는 대칭키 암호화 함수이다. $K_{Client-AS}$와 K_{AS-TGS}는 사전에 공유되어 있다.

① 클라이언트는 인증 서버에 등록된 아이디와 패스워드를 통하여 AS에 평문으로 인증을 요청한다.
② AS는 데이터베이스 상의 클라이언트 아이디와 패스워드를 확인한 후, 클라이언트에게 $K_{Client-TGS}$와 TGT(Ticket Granting Ticket)을 $K_{Client-AS}$로 암호화하여 전송한다. 이 때, AS는 TGS에게도 동일한 키 $K_{Client-TGS}$를 전달한다. TGT는 클라이언트가 TGS에 접속할 때 사용하기 위한 티켓이다.
③ 클라이언트는 전달받은 값을 $K_{Client-AS}$로 복호화하여 $K_{Client-TGS}$와 TGT를 얻는다. 그 후, 네임노드와의 상호 인증을 원할 때 네임노드 아



〔그림 4〕 하둡 분산 파일 시스템 상의 커버로스 인증 과정〔13〕

이디와 $K_{Client-TGS}$로 암호화한 타임스탬프 T와 TGT를 TGS에게 전송한다.

④ TGS는 클라이언트가 전송한 TGT를 K_{AS-TGS}로 복호화하여 클라이언트를 인증하고, $K_{Client-TGS}$를 얻는다. 그리고 $K_{Client-Namenode}$가 담긴 두 개의 티켓을 클라이언트에게 발급한다. 각 티켓은 $K_{Client-TGS}$와 $K_{Namenode-TGS}$로 암호화된 것으로 클라이언트와 네임노드에게 각각 $K_{Client-Namenode}$을 안전하게 전하기 위한 것이다.

⑤ 클라이언트는 전달받은 두 개의 티켓 중 하나를 $K_{Client-TGS}$로 복호화하여 $K_{Client-Namenode}$를 얻는다. 클라이언트는 나머지 티켓과 $K_{Namenode-TGS}$로 암호화한 타임스탬프 T를 네임노드에게 전송함으로써 네임노드에 인증을 요청한다.

⑥ 네임노드는 전달받은 티켓을 $K_{Namenode-TGS}$로 복호화하여 클라이언트를 인증하고, $K_{Client-Namenode}$를 얻는다. 네임노드는 $(T-1)$을 $K_{Client-Namenode}$로 암호화하여 전송함으로써 클라이언트에 인증을 요청한다.

⑦ 클라이언트는 수신한 타임스탬프에 1을 더하여 확인한 후 네임노드를 인증한다.

인증 완료 후 $K_{Client-Namenode}$는 제거된다.

2.5 인증 토큰 시스템

클라이언트와 네임노드, 데이터노드 사이의 호출이 빈번한 하둡 분산 파일 시스템에서의 커버로스 인증 방식은 KDC에 높은 부하를 발생시킬 수 있다. 야후는 커버로스 인증 후 KDC에 재접근할 필요 없이 클라이언트 인증이 이루어질 수 있도록 하기 위하여 인증 토큰 시스템을 제안하였다〔8〕. 제안된 방식에서 클라이언트는 초기 인증 시 커버로스를 통하여 인증하고, 이 후 네임노드로부터 인증 토큰을 발행받아 일정 시간동안 인증한다.

하둡 분산 파일 시스템에서의 인증 토큰은 위임 토큰(Delegation Token)과 블록 접근 토큰(Block Access Token)으로 구성된다. 위임 토큰은 네임노드에 대한 클라이언트 인증을 제공하며, 블록 접근 토큰은 데이터노드에 대한 클라이언트 인증을 제공한다. 토큰을 통한 모든 인증은 HMAC- SHA1에 기반을 둔다.

2.5.1 위임 토큰

위임 토큰은 클라이언트와 네임노드가 인증을 위하여 공유하는 비밀값이다. 클라이언트는 초기 커버로스 인증이 완료된 후 네임노드로부터 〔표 1〕과 같은 구조의 위임 토큰을 발행받고, 그 후의 네임노드에 대한

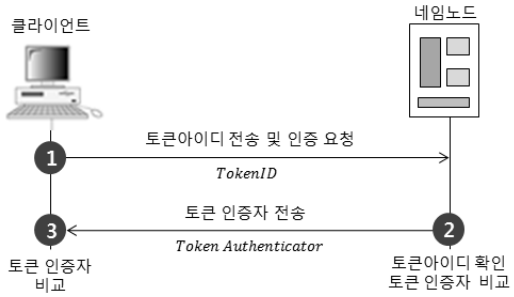
인증에 이를 사용한다. 네임노드는 발행한 위임 토큰들이 만료될 때까지 메모리에 보관한다. 위임 토큰의 유효 시간은 기본 10시간이며, 시간이 만료되기 전에 갱신하여 사용할 수 있다. 유효 시간 내에 위임 토큰을 갱신하지 않은 클라이언트는 다시 커버로스 인증을 받고, 위임 토큰을 재발급 받아야 한다. 클라이언트는 위임 토큰을 발행받음으로써, 일정 시간 동안 비교적 간단하게 네임노드에 자신을 인증하고, 블록 접근을 요청할 수 있다.

〔표 1〕 위임 토큰의 구조[8]

TokenID	*ownerID*, *renewerID*, *issueDate*, *maxDate*, *sequenceNumber*
Token Authenticator	*HMAC*− *SHA*1 (*masterKey*, *TokenID*)
Delegation Token	*TokenID*, *Token Authenticator*

위임 토큰은 토큰 아이디(TokenID)와 토큰 인증자(Token Authenticator)로 구성된다. 토큰 아이디는 클라이언트 고유 식별자인 소유자 아이디(ownerID), 지정된 갱신자의 고유 식별자인 갱신 아이디(renewerID), 위임 토큰의 발행일(issueDate)과 위임 토큰의 만료일(maxDate), 위임 토큰 생성 카운터인 시퀀스넘버(sequenceNumber)로 구성된다. 토큰 인증자는 마스터키(masterKey)로 토큰 아이디의 HMAC-SHA1 값을 구한 것으로 네임노드에 대한 클라이언트 인증 시 직접적으로 사용되는 값이다. 네임노드는 기본 10시간 주기로 랜덤한 마스터키를 새롭게 생성하여 위임 토큰의 토큰 인증자 연산에 사용한다.

위임 토큰을 통한 클라이언트 인증 과정은 〔그림 5〕와 같다. 클라이언트는 커버로스 인증을 거쳐 네임노드로부터 위임 토큰을 발행받은 상태이며, 네임노드는 발행한 위임 토큰들을 메모리에 보관 중이다.



〔그림 5〕 위임 토큰을 통한 인증 과정

① 클라이언트는 토큰 아이디를 전송하여 네임노드에 인증을 요청한다.
② 토큰 아이디를 전달받은 네임노드는 보관 중인 위임 토큰 중 동일한 토큰 아이디가 존재하는지 확인하고, 전달받은 토큰 아이디로 연산한 HMAC-SHA1 값이 저장된 토큰 인증자와 일치하는지 확인함으로써 클라이언트를 인증한다. 인증이 완료되면 클라이언트에게 해당 토큰 인증자를 전송한다.
③ 클라이언트는 전달받은 값이 자신이 가진 위임 토큰 상의 토큰 인증자와 동일한지 확인함으로써 네임노드를 인증한다.

2.5.2 블록 접근 토큰

블록 접근 토큰은 데이터노드에 저장된 데이터에 대한 접근 권한을 얻기 위해 클라이언트가 네임노드로부터 얻는 비밀값이다. 네임노드는 위임 토큰을 통하여 인증을 거친 클라이언트에게 요청하는 블록에 대한 블록 접근 토큰을 발행한다. 클라이언트는 블록 접근 토큰을 발행받음으로써, 데이터노드 접근 시마다 네임노드에 인증할 필요 없이 일정 시간 동안 비교적 간단하게 데이터노드에 접근할 수 있다.

네임노드는 위임 토큰에서 사용하는 마스터키와 별개로 블록 접근 토큰 생성에 사용될 또 다른 랜덤키를 주기적으로 생성하며, 랜덤키는 기본 10시간을 주기로 생성된다. 네임노드는 하트비트 메시지와 함께 랜덤키를 데이터노드들에게 전달한다.

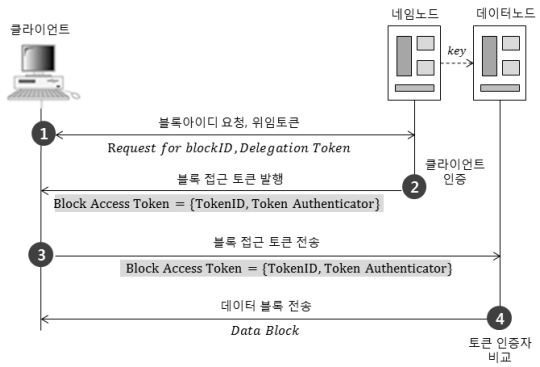
블록 접근 토큰은 토큰 아이디와 토큰 인증자로 구성된다. 구조는 〔표 2〕와 같다[8]. 토큰 아이디는 토큰의 만료일(expirationDate), 블록 접근 토큰 생성 시 사용되는 랜덤키의 고유 식별자인 키 아이디(keyID), 클라이언트의 고유 식별자인 소유자 아이

〔표 2〕 블록 접근 토큰의 구조[8]

TokenID	*expirationDate*, *keyID*, *ownerID*, *blockID*, *accessModes*
Token Authenticator	*HMAC*− *SHA*1 (*key*, *TokenID*)
Block Access Token	*TokenID*, *Token Authenticator*

디(ownerID), 접근 요청한 블록의 위치를 나타내는 블록 아이디(blockID), 접근 모드(accessModes)로 이루어져 있다. 접근 모드는 접근 요청한 블록에 대한 읽기, 쓰기, 복사, 재배치 권한를 명시하는 값이다. 토큰 인증자는 네임노드와 데이터노드가 공유한 키로 구한 토큰 아이디의 HMAC-SHA1 값을 의미한다.

블록 접근 토큰을 통한 클라이언트와 데이터노드 간의 인증과 파일 요청 과정은 〔그림 6〕과 같다.



〔그림 6〕 블록 접근 토큰을 통한 인증 및 파일 요청 과정

① 클라이언트는 네임노드로부터 발행받은 위임 토큰과 함께 접근하고자 하는 파일의 블록 아이디에 대한 요청을 네임노드에 전달한다.
② 네임노드는 전달받은 위임 토큰을 통하여 클라이언트를 인증하고, 정당한 클라이언트인 경우 요청받은 파일에 해당하는 블록들의 블록 접근 토큰을 클라이언트에게 발행한다.
③ 클라이언트는 해당 데이터노드에 블록 접근 토큰을 전달한다.
④ 데이터노드는 하트비트 메시지와 함께 네임노드로부터 전달받은 랜덤키를 사용하여 블록 접근 토큰 내의 블록 아이디에 대한 HMAC-SHA1 값을 연산한다. 그 값이 토큰 아이디 내의 토큰 인증자와 일치하면 클라이언트를 인증하고 해당 데이터 블록에 대한 클라이언트의 접근을 허가한다.

설명한 위임 토큰과 블록 접근 토큰 생성 시 사용되는 랜덤키는 서로 다르며, 생성 후 일정 시간이 되면 갱신된다. 이전 키는 현재 키와 함께 그 키로 발행된 토큰이 만료될 때까지 메모리 상에 보관되며, 대응되는 토큰들이 모두 만료되면 그와 함께 삭제된다. 즉, 네임노드는 아직 만료되지 않은 위임 토큰들에 대한 랜덤키 목록을 유지하며, 데이터노드는 네임노드가 블록 접근 토큰을 생성 후 하트비트 메시지를 통해 전달한 만료되지 않은 블록 접근 토큰들에 대한 랜덤키 목록을 유지한다. 토큰 생성 시에 사용되는 키는 현재 키이며, 이전 키들은 토큰 검증을 위해서만 사용된다. 이전 키들에 대한 식별은 키 아이디를 통하여 이루어진다.

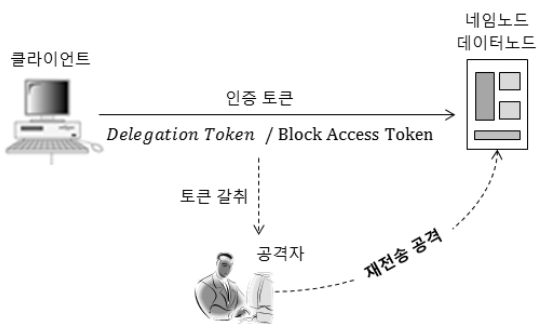
III. 하둡 분산 파일 시스템의 보안 위협

하둡 분산 파일 시스템은 데이터노드에 접근하여 파일을 저장하거나 읽고자 하는 클라이언트에 대한 인증을 위하여 커버로스와 인증 토큰 시스템을 도입하였다[8]. 그러나 인증 토큰 시스템은 재전송 공격, 데이터노드 해킹 공격 등에 있어 매우 취약하며, 이는 하둡 분산 파일 시스템 상에 저장된 민감한 데이터의 유출을 야기할 수 있다.

3.1 재전송 공격

하둡 분산 파일 시스템은 클라이언트 인증을 위하여 초기 커버로스 인증 후 위임 토큰과 블록 접근 토큰을 사용한다. 위임 토큰과 블록 접근 토큰의 토큰 아이디 내에는 소유자 아이디(ownerID) 값이 존재한다. 이 값은 네임노드로부터 해당 토큰을 발행받은 클라이언트의 고유 식별자이다. 위임 토큰과 블록 접근 토큰을 통한 클라이언트 인증 과정에는 소유자 아이디에 대한 인증이 따로 존재하지 않기 때문에 공격자가 클라이언트를 가장하여 토큰 유효 시간 동안 〔그림 7〕과 같은 재전송 공격을 시도할 수 있다.

위임 토큰을 가로챈 공격자는 위임 토큰에 대한 재



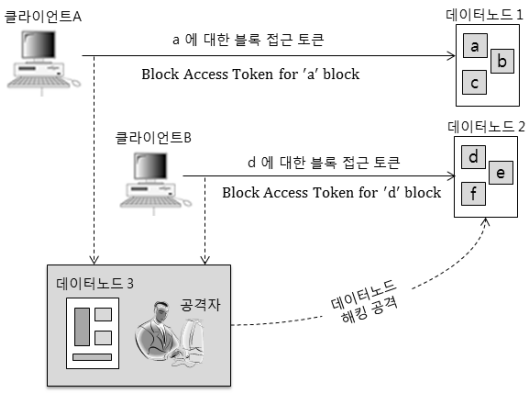
〔그림 7〕 재전송 공격

전송 공격을 통하여 네임노드에 본인을 클라이언트로 가장하여 인증하고, 블록 접근 토큰을 발행받을 수 있다. 이로써 공격자는 정당한 블록 접근 토큰을 통하여 데이터노드에 접근하여 데이터 블록을 얻을 수 있다. 블록 접근 토큰을 가로챈 공격자는 위임 토큰을 통한 인증 과정 없이 블록 접근 토큰에 대한 재전송 공격으로 데이터 노드에 곧바로 접근하여 데이터 블록을 얻을 수 있다.

3.2 데이터노드 해킹 공격

클라이언트는 네임노드에 재접근하여 위임 토큰 인증을 받을 필요 없이 데이터노드에 간편하게 접근하여 블록을 요청하기 위해 네임노드로부터 블록 접근 토큰을 발행받는다. 네임노드는 블록 접근 토큰을 발행할 때 사용하는 랜덤키를 하트비트 메시지와 함께 모든 데이터노드에게 전달한다. 이로써 모든 데이터노드는 만료되지 않은 모든 블록 접근 토큰들에 대한 동일한 랜덤키 목록을 가지게 된다. 따라서 만약 다수의 데이터노드 중 임의의 데이터노드가 해킹 당한다면 랜덤키 목록이 노출되게 된다.

데이터노드를 해킹한 공격자는 추출해낸 랜덤키 목록을 이용해서 다른 데이터노드 내의 블록에 대한 블록 접근 토큰을 생성할 수 있다. 이러한 데이터노드 해킹 공격 과정은 〔그림 8〕과 같다. 〔그림 8〕 내의 클라이언트A와 클라이언트B는 각각 데이터노드1의 블록 a와 데이터노드2의 블록 d에 대한 블록 접근 토큰을 사용하여 정상적으로 파일 읽기를 요청한다. 공격자는 이 때 사용되는 토큰들을 갈취하여 블록 아이디를 습득한다. 그 후 해킹한 데이터노드3에서 얻어낸 랜덤키 목록 내의 만료되지 않은 랜덤키 중 하나를 이



〔그림 8〕 데이터노드 해킹 공격

용하여 원하는 시점에 언제든지 갈취한 블록 아이디에 대한 블록 접근 토큰을 생성하여 데이터노드1이나 데이터노드2에 접근할 수 있다. 이러한 취약점의 근본적인 원인은 네임노드가 블록 접근 토큰을 생성할 때 사용하는 랜덤키 즉, 생성 권한을 데이터노드 역시 동일하게 소유하고 있다는 점이다.

3.3 암호화 통신 문제

하둡 분산 파일 시스템의 네임노드와 데이터노드는 SSH(Secure Shell)을 통하여 데이터를 주고 받는다. SSH란, 안전하지 못한 네트워크에서 원격 시스템을 통하여 다른 컴퓨터에 로그인하여 명령을 실행하고, 파일을 다른 시스템으로 복사할 수 있도록 해 주는 프로토콜이다. 따라서 네임노드와 데이터노드 간의 통신은 암호화되어 있으며 공격자에게 데이터가 노출될 우려가 없다.

반면 현재까지 네임노드-클라이언트, 데이터노드-클라이언트의 암호화 통신은 필수 사항이 아니다. 암호화 통신을 위해서 하둡 분산 파일 시스템이 제공하는 공개키 기반의 키 교환 시스템이 하둡의 성능 저하를 불러올 수 있기 때문이다.

공개키 기반의 키 교환 시스템을 사용하여 데이터노드와 클라이언트 간의 암호화 통신을 수행하는 경우, 클라이언트는 새로운 데이터노드에 접근할 때마다 높은 연산량을 요구하는 공개키 기반의 키 교환 프로토콜을 수행해야 한다. 또 응답시간을 최소화하기 위해 사전에 미리 세션키를 공유하는 경우 클라이언트는 데이터노드 수만큼의 키를, 데이터노드는 클라이언트 수만큼의 키를 서로 생성, 저장, 갱신해야 한다. 클라이언트는 데이터노드 뿐만 아니라 네임노드와의 암호화 통신을 위한 키 교환 또한 수행해야 하기 때문에 이는 매우 비효율적이다. 빠른 연산속도를 최우선으로 하는 하둡 분산 파일 시스템에서 이러한 결과는 치명적인 단점이다.

따라서 네임노드-클라이언트, 데이터노드-클라이언트 간의 암호화 통신을 제공하기 위해서는 기존 하둡 분산 파일 시스템과 비슷한 수준의 통신량, 연산량, 공간 효율성을 유지하면서 동시에 비효율적인 공개키 기반의 키 교환 프로토콜을 사용하지 않는 키 공유 방안이 제시되어야만 한다.

IV. 제안하는 기법

본 장에서는 3장에서 설명한 하둡 분산 파일 시스템의 보안 취약점들을 해결하고자 해쉬 체인 기반의 안전한 하둡 분산 파일 시스템 인증 프로토콜을 제안한다. 제안하는 기법은 블록 접근 토큰을 통한 데이터 노드의 클라이언트 인증 기능을 유지하면서, 재전송 공격, 데이터노드 해킹 공격을 막고, 암호화 통신 문제를 해결한다. 제안하는 기법은 다음을 가정한다.

- 하둡 분산 파일 시스템을 구성하는 클라이언트, 네임노드, 데이터노드의 시간은 서로 동기화 되어 있으며, 동일한 대칭키 암호화 함수 E_K와 해쉬 함수 H를 공유하고 있다.
- 하둡 분산 파일 시스템 내의 네임노드와 데이터노드는 기존의 방식과 마찬가지로 SSH(Secure Shell)을 통하여 암호화 통신을 한다.
- 공격자가 재전송 공격을 수행하기 위해서는 최소 t_{replay} 시간이 필요하다.

4.1 용어 설명

본 논문에서 제안하는 기법을 설명하기 위한 값들을 다음 〔표 3〕과 같이 표기한다. 재전송 공격 제한 시간 t_{replay}는 공격자가 토큰을 갈취하여 재전송 공격을 수행할 때 필요한 최소한의 시간을 의미한다. 즉, t_{replay} 시간 이내에는 공격자가 재전송 공격을 수행할 수 없음을 말한다. 제안하는 기법은 기존의 네임노드가 클라이언트에게 블록 접근 토큰을 직접 생성해서 전달하는 방식이 아니라, 클라이언트가 해당 토큰을 직접 생성할 수 있도록 권한을 주는 방식이다. 이를 위해, 토큰 생성 권한 부여 시점 $t_{delegation}$과 실제 토큰을 생성하여 사용하는 시점 t을 각각 정의한다. 블록 접근 토큰 사용 시점 t는 블록 접근 토큰 생성 권한 부여 시점 $t_{delegation}$으로부터 블록 접근 토큰 유효 시간 t_{tmax} 내에 존재하여야 하므로 $0 \le t - t_{delegation} \le t_{tmax}$을 만족해야 한다.

〔표 3〕 제안하는 기법에서 사용하는 표기법

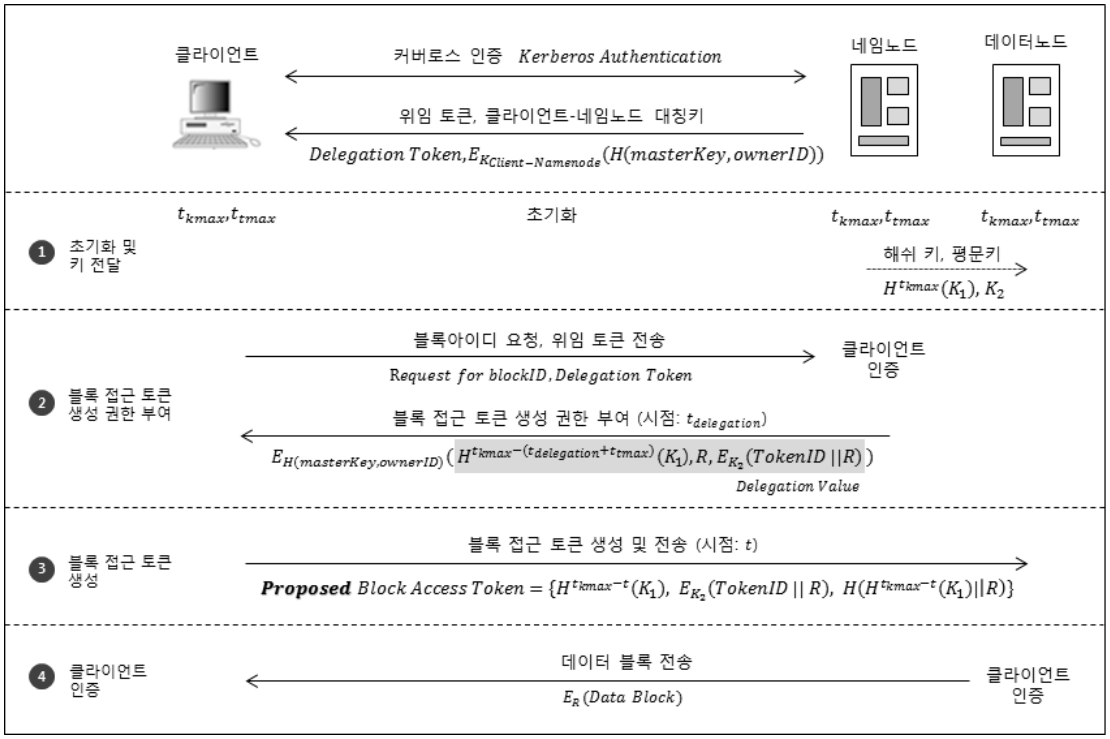
용어	의미
t_{kmax}	랜덤키 유효 시간
t_{tmax}	블록 접근 토큰 유효 시간
$t_{\text{delegation}}$	블록 접근 토큰 생성 권한 부여 시점
t	블록 접근 토큰 사용 시점
t_{replay}	재전송 공격의 제한 시간
K_1, K_2	네임노드가 생성한 임의의 랜덤키
R	네임노드가 생성한 임의의 난수
$E_K(M)$	비밀키 K로 평문 M을 암호화하는 대칭키 암호화 함수
H	일방향성 및 충돌저항성을 만족하는 해쉬 함수
$H^n(M)$	해쉬 함수 H로 평문 M를 n번 해쉬한 값(단, $H^0(M) = M$이다.)

네임노드가 생성한 난수 R을 클라이언트와 데이터노드 간의 암호화 통신을 위한 대칭키로 사용하고자 하는 경우, 네임노드는 난수 R을 대칭키 암호화 함수 E_K에 알맞은 키 길이로 생성하도록 한다.

4.2 제안하는 기법

네임노드와 클라이언트는 초기에 커버로스를 통하여 상호 인증을 한다. 커버로스 인증이 완료되면 클라이언트와 네임노드 간에 공유되었던 세션키 $K_{Client-Namenode}$는 삭제되며, 네임노드는 자신이 생성한 랜덤한 마스터키를 이용하여 사후 인증값인 위임 토큰을 클라이언트에게 발행하였다. 제안하는 기법에서는 기존 하둡 분산 파일 시스템에서는 보장되지 않는 클라이언트-네임노드 간의 암호화 통신을 제공하기 위하여 위임 토큰 발행 과정에 클라이언트-네임노드 간의 대칭키로 $H(masterKey, ownerID)$을 공유하는 과정을 추가하였으며, 위임 토큰 전송과 클라이언트-네임노드 간의 대칭키 공유가 완료되기 전까지 세션키 $K_{Client-Namenode}$를 삭제하지 않는다. 클라이언트와 네임노드의 커버로스 인증이 완료되고 나면, 네임노드는 랜덤한 마스터키를 생성하여 위임 토큰을 만들고, $H(masterKey, ownerID)$을 연산한다. 그 후 위임 토큰과 함께 $K_{Client-Namenode}$로 암호화한 $H(masterKey, ownerID)$을 클라이언트에게 전송한다. 이 때, 위임 토큰의 구조는 기존 위임 토큰의 구조 〔표 1〕과 동일하다. 클라이언트는 보유한 세션키 $K_{Client-Namenode}$로 전달받은 값을 복호화하여 클라이언트-네임노드 간의 대칭키 $H(masterKey, ownerID)$을 얻는다. 본 과정이 완료되면 세션키 $K_{Client-Namenode}$는 삭제된다. 마스터키를 보유한 네임노드는 $H(masterKey, ownerID)$값을 언제나 연산할 수 있으므로 과정이 완료되면 해당 값을 삭제하고 필요시에 재생성하여 사용한다.

제안하는 기법은 〔그림 9〕와 같다. 클라이언트가



〔그림 9〕 제안하는 기법

네임노드로부터 발행받은 블록 접근 토큰으로 데이터노드에 인증하던 기존의 방식과 다르게, 네임노드로부터 블록 접근 토큰을 생성할 수 있는 권한을 부여받은 클라이언트가 블록 접근 토큰을 생성하여 데이터노드에 인증하는 방식을 사용한다. 제안하는 기법은 초기화 및 키 전달 단계, 블록 접근 토큰 생성 권한 부여 단계, 블록 접근 토큰 생성 단계, 클라이언트 인증 단계로 나뉜다.

4.2.1 초기화 및 키 전달 단계

시스템이 시작되면 시간을 의미하는 t_{kmax}, t_{tmax} 값은 재전송 공격의 제한 시간 t_{replay} 단위로 초기화된다. 예를 들어, 재전송 공격 제한 시간이 $t_{replay}=1$(분)이라고 가정하자. 랜덤키 유효 시간이 $t_{kmax}=10$(시간)이고 블록 접근 토큰 유효 시간이 $t_{tmax}=3$(시간)인 경우, t_{replay}를 기본 단위로 하여 t_{kmax}, t_{tmax}은 $t_{kmax}=600$(분), $t_{tmax}=180$(분)으로 각각 초기화된다.

네임노드는 블록 접근 토큰 생성에 사용될 랜덤한 두 개의 키 K_1, K_2을 동시에 생성하며, $H^{t_{kmax}}(K_1)$와 K_2를 하트비트 메시지와 함께 모든 데이터노드에게 전달한다. 랜덤키 유효 시간 t_{kmax}가 지나면 새로운 랜덤키 쌍을 생성하고 이전의 랜덤키 쌍은 모두 삭제한다.

블록 접근 토큰 생성 권한 부여 단계 및 블록 접근 토큰 생성 단계에서 사용되는 $t_{delegation}$, t은 현재 블록 접근 토큰 생성에 사용되는 랜덤키 생성 시간을 기점(=0)으로 하여 재전송 공격 제한 시간 t_{replay} 단위로 초기화된다. 예를 들어, 새로운 랜덤키 생성 시간을 기점으로 하여 블록 접근 토큰 생성 권한 부여 시점이 $t_{delegation}=2$(시간) 후, 블록 접근 토큰 사용 시점이 $t=3$(시간) 후인 경우, $t_{delegation}$, t는 $t_{delegation}=120$(분), $t=180$(분)로 초기화되어 사용된다.

이해를 돕기 위해 〔표 4〕에서 기술하는 값들을 실제 프로토콜에 적용하면 다음과 같다. 오전 1시에 네임노드는 K_1, K_2를 갱신한다고 가정하자. 해당 키는 $t_{kmax}=600$ 동안 유효하기 때문에 오전 11시까지 사용될 것이다. 클라이언트는 $t_{delegation}=100$이 지난 시점 즉, 오전 2시 40분에 블록 접근 토큰 생성 권한을 요청해 네임노드로부터 권한을 부여받는다. 해당 권한

은 $t_{tmax}=300$만큼 유효하기 때문에 클라이언트는 오전 7시 40분까지 블록 접근 토큰을 생성할 수 있게 된다. 마지막으로 실제 클라이언트는 블록 접근 토큰을 $t=200$에 의해 오전 4시 20분에 생성 및 사용한다.

〔표 4〕 제안하는 기법 설명에 적용되는 값

용어	값
t_{kmax}	600(분)
t_{tmax}	300(분)
$t_{\text{delegation}}$	100(분)
t	200(분)
t_{replay}	1(분)

4.2.2 블록 접근 토큰 생성 권한 부여 단계

시스템 시작 후 초기화 및 키 전달이 완료되면 클라이언트는 네임노드에게 위임 토큰을 전송하여 자신을 인증한다. 인증이 완료되면 네임노드는 블록 접근 토큰 생성에 필요한 권한값(Delegation Value) $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$, R, $E_{K_2}(TokenID\|R)$을 생성한다. 그 후 커버로스 인증 완료 후 네임노드로부터 받은 $H(masterKey, ownerID)$로 권한값을 암호화하여 클라이언트에게 전달함으로써 클라이언트에게 블록 접근 토큰 생성의 권한에 대한 권한을 부여한다.

기본적인 아이디어는 $H^{i}(K_1)$에서 $H^{i+1}(K_1)$을 계산하기는 쉽지만 $H^{i-1}(K_1)$을 계산하는 것은 어려운 해쉬 체인의 일방향성(preimage resistance) 성질을 이용하여, 클라이언트에게 $H^{600}(K_1)$의 낮은 차수의 해쉬 체인 값인 $H^{200}(K_1)$을 전달함으로써 시간의 흐름에 따라 스스로 높은 차수의 해쉬 체인 값을 계산하여 토큰을 생성하게 하는 방식이다.

제안하는 기법에서 사용되는 해쉬 체인 값 $H^{t_{kmax}-t}(K_1)$은 블록 접근 토큰의 사용 시점을 확인하기 위한 값으로, t 시점에 클라이언트가 생성하는 블록 접근 토큰 내에는 $H^{t_{kmax}-t}(K_1)$값이 포함되어 있어야 한다. 예를 들어, 클라이언트가 현재 랜덤키 생성을 기점으로 $t=100$ 또는 $t=200$ 후에 블록 접근 토큰을 생성한다고 가정하면 각 블록 접근 토큰 내에는 $H^{t_{kmax}-t}(K_1)=H^{600-100}(K_1)=H^{500}(K_1)$과 $H^{t_{kmax}-t}(K_1)=H^{600-200}(K_1)=H^{400}(K_1)$의 해쉬 체인 값이 포함되어 있어야 한다. 데이터노드는 이 값을 통하여 블록 접근 토큰의 생성 시점을 확인한다.

$(t_{delegation}+t_{tmax})=400$은 클라이언트가 권한값으로 블록 접근 토큰을 생성할 수 있는 마지막 시점을 의미한다. 즉, 권한값 내의 $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$은 클라이언트가 권한 만료 시점에 생성하는 블록 접근 토큰에 해당하는 해쉬 체인 값이다. 클라이언트는 블록 접근 토큰 생성 단계에서 $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$값에 추가적으로 해쉬하여 블록 접근 토큰 사용 시점에 알맞은 해쉬 체인 값들을 연산할 수 있다.

블록 접근 토큰 생성 권한을 부여받는 시점 $t_{delegation}$로부터 현재 랜덤키 K_1, K_2의 만료 시점까지의 시간 $t_{kmax}-t_{delegation}$이 블록 접근 토큰 유효 시간 t_{tmax}보다 짧은 경우, 즉, $t_{kmax}-t_{delegation}<t_{tmax}$인 경우에는 남아있는 유효 시간 $t_{kmax}-t_{delegation}$에 대한 권한값과 초과하는 시간 $t_{tmax}-(t_{kmax}-t_{delegation})$에 대한 권한값을 함께 부여한다. 남아있는 유효 시간에 대한 권한값은 현재 랜덤키 K_1, K_2를 이용하여 동일한 방식으로 부여하고, 초과하는 시간에 대한 권한값은 다음 랜덤키 K_1', K_2'를 이용하여 부여한다. 네임노드는 초과하는 시간만큼의 권한값을 부여하기 위하여 다음 랜덤키 K_1', K_2'을 미리 생성하여 하트비트 메시지를 통해 $H^{t_{kmax}}(K_1')$과 K_2'를 모든 데이터노드에게 전달해둔다. 클라이언트는 현재 랜덤키로 부여받은 권한값으로 블록 접근 토큰을 생성하여 사용하다가 현재 랜덤키가 만료되면 해당 권한값을 버리고 다음 랜덤키 K_1', K_2'로 부여된 권한값을 사용하여 블록 접근 토큰을 생성하여 사용한다.

4.2.3 블록 접근 토큰 생성 단계

블록 접근 토큰은 클라이언트가 부여받은 권한값 $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$, R, $E_{K_2}(TokenID\|R)$을 이용하여 연산한 토큰 인증자(Token Authenticator)이며 구조는 〔표 5〕과 같다. 토큰 아이디 내의 값들의 의미는 기존의 블록 접근 토큰의

〔표 5〕 제안하는 블록 접근 토큰의 구조

TokenID	$expirationDate$, $ownerID$, $blockID$, $accessModes$
Token Authenticator	$H^{t_{kmax}-t}(K_1)$, $E_{K_2}(TokenID\|R)$, $H(H^{t_{kmax}-t}(K_1)\|R)$
Block Access Token	$Token\,Authenticator$

것들과 동일하다. 생성된 블록 접근 토큰 사용 시점에 사용하는 랜덤키는 현재 랜덤키인 K_1, K_2 뿐이므로 기존의 토큰 아이디 내에 존재했던 키 아이디는 따로 필요하지 않다.

토큰 인증자는 $H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$, $E_{K_2}(TokenID\|R)$, $H(H^{t_{tmax}-t}(K_1)\|R)=H(H^{400}(K_1)\|R)$로 구성된다. 클라이언트는 권한값으로 부여받은 $H^{t_{tmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$에 $(t_{delegation}+t_{tmax})-t=200$번 추가 해쉬하여 $H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$을 연산한다. $E_{K_2}(TokenID\|R)$는 권한값으로 전달받은 값을 그대로 사용하며, 연산한 $H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$과 권한값으로 부여받은 난수 R로 $H(H^{t_{tmax}-t}(K_1)\|R)=H(H^{400}(K_1)\|R)$을 연산한다.

$H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$ 값은 해당 블록 접근 토큰이 $t=200$ 시점에 생성되었음을 증명하는 값이다. $E_{K_2}(TokenID\|R)$는 클라이언트의 토큰 아이디 조작을 막기 위한 장치인 동시에 데이터노드에게 난수 R를 안전하게 전달하는 역할을 한다. 클라이언트는 랜덤키 K_2를 모르기 때문에 $E_{K_2}(TokenID\|R)$ 내의 토큰 아이디와 난수 R을 조작할 수 없다. $H(H^{t_{tmax}-t}(K_1)\|R)$은 데이터노드에게 클라이언트를 인증하는 실질적인 인증값이다.

4.2.4 클라이언트 인증 단계

$A=H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$, $B=E_{K_2}(TokenID\|R)$, $C=H(H^{t_{tmax}-t}(K_1)\|R)=H(H^{400}(K_1)\|R)$가 클라이언트가 생성한 블록 접근 토큰 내의 토큰 인증자 값이라고 하자. 데이터노드는 블록 접근 토큰 내의 토큰 인증자 값 A, B, C를 평문으로 전달받는다. 데이터노드가 전달받은 블록 접근 토큰으로 클라이언트를 인증하는 과정은 다음과 같다.

① 클라이언트는 생성한 블록 접근 토큰을 데이터노드에게 전달한다.

② 모든 네임노드와 데이터노드의 시간은 동기화되어 있으므로 데이터노드는 블록 접근 토큰의 사용 시점 $t=200$을 알고 있다. 데이터노드는 블록 접근 토큰의 A 값에 블록 접근 토큰 사용 시점 $t=200$만큼 추가적으로 해쉬한 $H^t(A)=H^{200}(A)$이 네임노드로부터 전달받은 $H^{t_{tmax}}(K_1)=H^{600}(K_1)$과 일치하는지 확인한다. 일치하면 클라이언트가 블록 접근 토큰 사용 시점 $t=200$에 해당하는 해쉬 체인 값을 전달했음을 확인할 수 있다. 또한 랜덤키 K_1을 모르는 클라이언트는 권한값의 $H^{t_{tmax}-(t_{delegation}+t_{tmax})}(K_1)=H^{200}(K_1)$ 없이는 $A=H^{t_{tmax}-t}(K_1)=H^{400}(K_1)$을 생성할 수 없기 때문에 데이터노드는 해당 클라이언트가 네임노드로부터 블록 접근 토큰 생성 권한을 획득하였다는 것을 확인할 수 있다.

③ 데이터노드는 네임노드로부터 전달받은 랜덤키 K_2로 블록 접근 토큰 내의 $B=E_{K_2}(TokenID\|R)$를 복호화하여 토큰 아이디와 난수 R을 얻는다.

④ ③에서 얻어낸 난수 R와 클라이언트로부터 전달받은 A 값으로 $H(A\|R)$을 연산하여 블록 접근 토큰 내의 $C=H(H^{t_{tmax}-t}(K_1)\|R)=H(H^{400}(K_1)\|R)$ 값과 일치하는지 확인한다. 일치하면 토큰 아이디의 무결성이 확인되며, 클라이언트 인증이 완료된다.

⑤ 데이터노드는 인증이 완료된 클라이언트에게 토큰 아이디에 명시된 데이터 블록을 전송한다. 이때, 클라이언트와 데이터노드 사이의 암호화 통신을 위하여 난수 R를 대칭키로 데이터 블록을 암호화하여 전송할 수 있다.

V. 분석

본 장에서는 제안하는 기법이 실제로 재전송 공격, 데이터노드 해킹 공격에 있어 안전한지에 대해 분석하며, 제안하는 기법을 통한 기존의 비효율적인 암호화 통신 문제에 대한 해결 방안을 제시한다. 그리고 제안하는 기법이 하둡 분산 파일 시스템의 성능 저하의 영향을 주지 않으며 빠르게 연산이 이루어짐을 증명한다.

5.1 안전성 분석

5.1.1위임 토큰에 대한 재전송 공격

제안하는 기법에서는 〔표 1〕과 같은 기존과 동일한 구조의 위임 토큰을 사용한다. 기존의 위임 토큰과 블록 접근 토큰은 소유자 아이디(ownerID)에 대한 미인증으로 토큰 재전송 공격에 취약하였다. 그 중 위임

토큰에 대한 재전송 공격의 경우, 가로챈 위임 토큰을 이용하여 네임노드로부터 거짓으로 인증을 받은 공격자가 클라이언트를 가장하여 네임노드로부터 블록 접근 토큰을 발행받을 수 있었다. 공격자는 발행받은 블록 접근 토큰으로 데이터노드에 접근하여 데이터 블록을 얻을 수 있었다.

제안하는 기법에서는 위임 토큰에 대한 재전송 공격에 의해 발생할 수 있는 공격자의 블록 접근 토큰 습득을 막기 위하여, 위임 토큰으로 본인을 인증한 클라이언트에게 블록 접근 토큰 자체를 발행하는 것이 아니라, 블록 접근 토큰을 생성할 수 있는 권한값을 커버로스 인증 시 위임 토큰과 함께 클라이언트에게 전달된 $H(masterKey,ownerID)$로 암호화하여 전송하는 방식을 제시하였다. 공격자는 정당한 소유자 아이디를 가지고 있지 않기 때문에 커버로스 인증을 통하여 위임 토큰과 $H(masterKey,ownerID)$을 전달받지 못하였으므로, 갈취한 위임 토큰만으로는 $H(masterKey,ownerID)$로 암호화된 권한값을 알 수 없다. 따라서 기존의 방식과 달리 공격자는 위임 토큰에 대한 재전송 공격을 통하여 블록 접근 토큰을 얻을 수 없으며, 데이터노드에 접근할 수도 없다.

5.1.2 블록 접근 토큰에 대한 재전송 공격

제안하는 기법에서 클라이언트는 네임노드로부터 블록 접근 토큰 생성 권한을 부여받는 시점 $t_{delegation}$에 권한값으로 $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)$을 부여받아 블록 접근 토큰 사용 시점 t에 $H^{t_{kmax}-(t_{delegation}+t_{tmax})}(K_1)$ 값에 $(t_{delegation}+t_{tmax})-t$번 추가 해쉬하여 t 시점에 해당하는 해쉬 체인 값 $H^{t_{kmax}-t}(K_1)$을 생성한다.

만약 공격자가 t 시점에 토큰 갈취하여 재전송 공격 제한 시간 t_{replay}를 초과한 $t_{replay}+\alpha(\alpha \geq 0)$ 시간 후에 재전송 공격을 시도한다고 가정하자. 공격자가 해당 시점에 재전송 공격을 통하여 데이터노드 접근에 성공하려면 전송하는 블록 접근 토큰 내에 공격 시점에 해당하는 해쉬 체인 값 $H^{(t_{kmax}-t)-(t_{replay}+\alpha)}(K_1)$이 포함되어 있어야만 한다.

하지만 갈취한 블록 접근 토큰 내에는 t 시점에 해당하는 해쉬 체인 값 $H^{t_{kmax}-t}(K_1)$이 존재하기 때문에 재전송 공격 제한 시간 t_{replay} 이후의 재전송 공격은 불가능하다. 정당한 소유자 아이디(ownerID)를 가진 클라이언트만이 초기 커버로스 인증 후 네임노드로부터 위임 토큰과 $H(masterKey,ownerID)$을 전달 받아 $H(masterKey,ownerID)$로 암호화된 블록 접근 토큰 생성 권한값을 부여받을 수 있으며, 권한값을 가지고 있는 클라이언트만이 공격 시점 $t_{replay}+\alpha$ 에 해당하는 블록 접근 토큰 내 해쉬 체인 값 $H^{(t_{kmax}-t)-(t_{replay}+\alpha)}(K_1)$을 생성할 수 있기 때문이다. 다시 말해서, 정당한 소유자 아이디를 가진 클라이언트만이 특정 시점에 해당하는 블록 접근 토큰을 생성할 수 있는 권한을 얻을 수 있으므로, 제안하는 기법은 소유자 아이디를 갖지 못한 공격자의 블록 접근 토큰에 대한 재전송 공격에 있어 안전하다.

5.1.3 데이터노드 해킹 공격

각 데이터노드는 네임노드가 새로운 랜덤키 쌍을 생성할 때마다 하트비트 메시지를 통하여 하나의 해쉬된 키 $H^{t_{kmax}}(K_1)$와 하나의 평문키 K_2을 전달받는다. 공격자가 데이터노드 해킹 공격을 통하여 다른 데이터노드에 접근하려면 공격 시점 t에 해당하는 해쉬 체인 값 $H^{t_{kmax}-t}(K_1)$을 생성할 수 있어야 한다. 하지만 데이터노드 해킹을 통하여 공격자가 얻을 수 있는 정보는 $H^{t_{kmax}}(K_1)$, K_2 뿐이다. 따라서 공격자가 토큰 아이디와 난수 R를 임의로 생성하여 토큰 인증자의 구성 요소 중 $E_{K_2}(TokenID \| R)$을 연산해낸다 해도 나머지 토큰 인증자 값인 $H^{t_{kmax}-t}(K_1)$과 $H(H^{t_{kmax}-t}(K_1) \| R)$을 연산해내는 것은 불가능하다. 해쉬 함수의 일방향성에 의해 $H^{t_{kmax}}(K_1)$값으로 이보다 낮은 차수의 해쉬 체인 값 $H^{t_{kmax}-t}(K_1)$을 연산할 수 없기 때문이다.

또한 하나의 데이터노드를 해킹한 공격자가 하나의 블록 접근 토큰을 갈취하여 토큰 인증자 $H^{t_{kmax}-t}(K_1)$, $E_{K_2}(TokenID \| R)$, $H(H^{t_{kmax}-t}(K_1) \| R)$을 부가적으로 얻었다고 가정하더라도 공격자의 데이터노드 접근은 불가능하다. 토큰을 갈취하여 $t_{replay}+\alpha$ 후에 공격한다고 하였을 때, 해쉬 함수의 일방향성에 의해 공격 시점에 해당하는 낮은 차수의 해쉬 체인 값 $H^{(t_{replay}-t)-(t_{replay}+\alpha)}(K_1)$을 생성할 수 없기 때문이다. 즉, 제안하는 기법은 과거의 블록 접근 토큰 내에 존재하는 높은 차수의 해쉬 체인 값으로 미래의 블록 접근 토큰에 알맞은 낮은 차수의 해쉬 체인 값을 연산할 수 없는 해쉬 체인의 일방향성 성질을 이용하여 데이터노드 해킹 공격을 막는다.

5.1.4 암호화 통신 문제

현재 하둡 분산 파일 시스템의 네임노드-클라이언트, 데이터노드-클라이언트의 암호화 통신은 필수 사항이 아니다. 암호화 통신을 위하여 하둡 분산 파일 시스템이 제공하는 공개키 기반의 키 교환 시스템은 높은 연산량으로 인하여 하둡의 성능 저하를 야기할 수 있기 때문이다. 본 논문에서는 빠른 속도를 최우선으로 하는 하둡 분산 파일 시스템에 적합하지 않은 공개키 기반의 키 교환 시스템을 사용하는 대신 제안하는 기법 내에서 공유되는 비밀값을 활용한 암호화 통신 방안을 제안한다.

제안하는 기법에서는 네임노드-클라이언트의 암호화 통신을 위하여 커버로스 인증 완료 후 네임노드가 클라이언트에게 $H(masterKey,ownerID)$ 값을 위임 토큰과 함께 전달한다. $H(masterKey,ownerID)$는 블록 접근 토큰 생성 권한 부여 단계에서 네임노드가 클라이언트에게 전달하는 권한값을 암호화하는 대칭키로 사용된다.

데이터노드-클라이언트의 암호화 통신 문제는 블록 접근 토큰 생성 권한 부여 단계에서 네임노드가 클라이언트에게 제공하는 난수 R을 이용하여 해결할 수 있다. 클라이언트는 블록 접근 토큰 생성 권한 부여 단계에서 네임노드로부터 $H(masterKey,ownerID)$로 암호화된 권한값을 전달받아 이를 복호화하여 평문 형태의 난수 R을 얻는다. 데이터노드는 클라이언트가 전달한 블록 접근 토큰 내의 $E_{K_2}(TokenID \| R)$ 값을 K_2로 복호화하여 난수 R을 얻는다. 이렇게 데이터노드와 클라이언트 사이에 공유된 난수 R을 데이터노드-클라이언트의 암호화 통신을 위한 대칭키로 사용할 수 있다.

제안하는 암호화 통신 방안은 암호화 통신을 위한 별도의 키 교환 과정을 수행하지 않고, 위임 토큰의 발행과 블록 접근 토큰 생성 권한 부여 및 블록 접근 토큰 전송 과정에서 네임노드-클라이언트, 데이터노드-클라이언트 간 암호화 통신 시 필요한 대칭키를 교환하여 암호화 통신에 사용한다. 이는 네임노드-클라이언트, 데이터노드-클라이언트 간의 안전한 통신을 보장함과 동시에 기존의 키 교환에 소요되었던 시간을 단축시킴으로써 하둡의 빠른 처리 속도를 유지할 수 있도록 한다.

5.1.5 메시지 위·변조 공격

제안하는 기법은 각 초기화 및 키 전달 단계, 블록 접근 토큰 생성 권한 부여 단계, 블록 접근 토큰 생성 단계, 클라이언트 인증 단계에서의 클라이언트, 네임노드, 데이터노드 간의 통신 상의 메시지 위·변조 공격에 대하여 안전하다.

우선, 초기화 및 키 전달 단계에서 네임노드로부터 데이터노드에 전송되는 해쉬된 키 $H^{t_{known}}(K_1)$와 평문키 K_2는 기존의 하둡 분산 파일 시스템에서와 마찬가지로 SSH(Secure Shell)을 통하여 암호화되어 전송되기 때문에 암호화된 메시지 속의 해쉬된 키 $H^{t_{known}}(K_1)$와 평문키 K_2을 공격자의 의도대로 위·변조할 수 없다.

블록 접근 토큰 생성 권한 부여 단계는 두 개의 과정으로 나뉠 수 있다. 첫째는 클라이언트가 위임 토큰을 이용하여 네임노드에게 블록 아이디를 요청하는 과정이고, 둘째는 위임 토큰을 통하여 클라이언트를 인증한 네임노드가 클라이언트에게 블록 접근 토큰 생성에 대한 권한 값을 부여하는 과정이다.

첫 번째 경우에 공격자가 위·변조를 시도할 수 있는 요소는 위임 토큰과 블록 아이디 요청 메시지이다. 공격자가 위임 토큰 위·변조 공격에 성공하여 네임노드로부터 $H(masterKey,ownerID)$로 암호화된 권한값을 전달받는다 할지라도 공격자는 $H(masterKey,ownerID)$로 암호화된 권한값을 복호화할 수 없으므로 블록 접근 토큰을 생성할 수 없으며, 데이터노드에 접근할 수 없다. 블록 아이디 요청 메시지에 대한 위·변조 공격은 공격자가 클라이언트의 블록 아이디 요청을 갈취하여 다른 블록에 대한 요청으로 맞바꾸는 것인데, 이는 클라이언트가 요청한 데이터 블록이 아닌 다른 데이터 블록을 데이터노드가 클라이언트에게 반환해주는 결과를 불러온다. 하지만 잘못된 블록을 전달받은 클라이언트는 데이터노드에 원래의 블록을 언제든지 재요청할 수 있으며, 공격자가 블록 아이디 요청 메시지를 위·변조하여도 블록 접근 토큰 생성에 대한 권한값이 없는 공격자에게 데이터 블록이 노출될 가능성은 없다.

두 번째 경우에는 네임노드가 클라이언트에게 부여하는 권한값이 $H(masterKey,ownerID)$로 암호화 되어 있기 때문에 암호문 내의 권한값을 공격자의 의도대로 위·변조하기 힘들다.

블록 접근 토큰 생성 단계에서 또한 메시지 위·변조 공격은 불가능하다. 공격자는 $H(masterKey,ownerID)$ 없이 $H(masterKey,ownerID)$로 암호화된 권한값을 얻을 수 없으므로 블록 접근 토큰을 임의로 위조할 수

없다. 그렇다면 공격자가 클라이언트로부터 데이터노드에게 전송되는 블록 접근 토큰을 가로챈 후 이를 변조하여 데이터노드에 접근을 시도한다고 가정해보자. 블록 접근 토큰 생성 및 전송 시점인 t 시간 이후의 공격 시점에 알맞게 해당 블록 접근 토큰을 변조하여 데이터노드에 접근하려면 $H^{t_{kmax}-t}(K_1)$보다 낮은 차수의 해쉬 체인 값을 계산할 수 있는 능력을 가지고 있어야 한다. 하지만 권한값을 가지지 못한 공격자는 해쉬 함수의 일방향성에 의해 $H^{t_{kmax}-t}(K_1)$보다 낮은 차수의 해쉬 체인 값을 계산할 수 없으며, $H(H^{t_{kmax}-t}(K_1)\|R)$ 값도 동일하게 해쉬 함수의 일방향성에 따라 변경 불가능하다. 그리고 K_2를 알 수 없는 공격자는 $E_{K_2}(TokenID\|R)$ 내의 $TokenID$, R 값을 의도대로 위·변조할 수도 없다. 따라서 블록 접근 토큰 생성 단계에서 위·변조 공격을 통하여 공격자가 데이터노드에 접근하여 데이터 블록을 얻을 수 있는 방법은 없다.

마지막으로 클라이언트 인증 단계에서는 데이터노드가 클라이언트로부터 전달 받은 블록 접근 토큰을 통하여 클라이언트를 인증하고, 인증이 완료되면 요청 받은 데이터 블록을 난수 R로 암호화 하여 전송한다. 따라서 난수 R을 알지 못하는 공격자는 난수 R로 암호화된 데이터 블록을 자신의 의도대로 위·변조 할 수 없다.

5.2 성능 분석

다음 〔표 6〕은 하나의 클라이언트가 하나의 데이터 블록에 대한 접근을 요청하는 과정에서 제안하는 기법과 기존의 하둡 분산 파일 시스템 상의 인증 프로토콜의 성능 및 안전성을 비교한 것이다. 제안하는 기법은 하둡 분산 파일 시스템의 성능을 유지함과 동시에 재전송 공격, 데이터노드 해킹 공격에 있어 안전성을 제공하고, 효율적인 암호화 통신 방안을 제시하였다.

기존의 방식과 비교하였을 때 제안하는 기법에서 네임노드, 데이터노드, 클라이언트가 각각 저장해야 하는 키의 개수와 연산량은 다소 증가하였다. 하지만 하둡 분산 파일 시스템 전체로 보았을 때, 키 저장에 사용되는 공간은 매우 미미하다. 또한 기존의 방식에서는 랜덤키가 갱신될 때마다 데이터노드에 이를 누적하여 저장해야 했던 것에 비해, 제안하는 방식은 현재 랜덤키 쌍을 랜덤키 유효 시간 t_{kmax}동안만 데이터노드에 저장해두었다가 시간이 만료되면 새로운 랜덤키 쌍으로 교체하기 때문에 장기적으로 보았을 때 데이터노드의 키 저장 공간은 오히려 감소한다고 볼 수 있다. 그리고 제안하는 기법의 각 단계에서 사용되는 대칭키 암호화 함수와 해쉬 함수는 빠른 계산을 특징으로 하여 모든 과정에서 비교적 빠르게 연산이 이루어지도록 돕는다. 이는 빠른 데이터 처리를 강조하는 하둡 분산 파일 시스템에 매우 적합하며, 하둡 분산 파일 시스템에 큰 연산 부담을 주지 않는다.

한편 제안하는 기법은 기존의 높은 연산량과 부가적인 키 교환 시간을 요구하였던 공개키 기반의 키 교환 시스템 대신 기법 내에 공유되는 $H(masterKey,ownerID)$과 난수 R을 각각 네임노드-클라이언트, 데이터노드-클라이언트 간의 암호화 통신을 위한 대칭키로 활용함으로써 안전한 상호간 통신을 보장함과 동시에 암호화 통신 제공에 따라 발생할 수 있는 부가적인 시간을 제거하였다. 특히, 네임노드의 경우 네임노드-클라이언트의 암호화 통신에 사용되는

〔표 6〕 기존 HDFS 인증 프로토콜과 제안하는 기법의 성능 및 안전성 비교

		HDFS	Ours
총 라운드 수		6	6
저장하는 키 개수	네임노드	1	2
	데이터노드	1	2
	클라이언트	0	2
추가되는 연산량	커버로스 인증 및 위임토큰 발행	-	1 Enc, 1 $Hash$
	초기화 및 키 전달 단계		t_{kmax} $Hash$
	블록 접근 토큰 생성 권한 부여 단계		$t_{kmax}-(t_{delegation}+t_{tmax})$ $Hash$
	블록 접근 토큰 생성 단계		$(t_{kmax}-t+1)$ $Hash$
	클라이언트 인증 단계		1 Enc
안전성	재전송 공격 내성	X	O
	데이터노드 해킹 공격 내성	X	O
	데이터 암호화 여부	X	O

$H(masterKey, ownerID)$을 따로 저장할 필요 없이 암호화 통신이 필요할 때 즉각적으로 생성하여 사용할 수 있기 때문에, 네임노드-클라이언트의 암호화 통신을 위한 네임노드의 키 저장 공간을 절약할 수 있다.

그리고 기존의 인증 토큰 시스템의 경우 네임노드로부터 발행된 블록 접근 토큰을 클라이언트가 메모리 상에 저장해 두었다가 사용했기 때문에 제안하는 기법에서 네임노드가 부여하는 권한값에 대한 클라이언트의 메모리 사용은 기존에 비해 크게 증가한다고 볼 수 없다.

VI. 결론

본 논문에서는 해쉬 체인 기반의 안전한 하둡 분산 파일 시스템 인증 프로토콜을 제안하였다. 제안하는 기법은 기존 하둡 분산 파일 시스템의 인증 프로토콜의 변경을 최소화하면서 기존의 하둡 분산 파일 시스템이 갖고 있는 재전송 공격 및 데이터노드 해킹 공격에 대한 안전성을 제공하였으며, 그와 동시에 네임노드-클라이언트, 데이터노드-클라이언트 간의 암호화 통신을 제공한다. 또한 추가된 해쉬 체인은 빠른 연산 속도로 시스템 성능에 거의 영향을 주지 않으며, 기존의 인증 프로토콜과 비교하였을 때 네임노드 및 클라이언트의 부가적인 저장 공간이 요구되지 않고 데이터노드의 경우 오히려 적은 키 저장 공간을 사용하기 때문에 공간 효율성 측면에서 유리하다.

참고문헌

〔1〕 Apache Hadoop. http://hadoop.apache.org/.

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

〔3〕 Apache Hadoop MapReduce Tutorial. http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html/.

〔4〕 S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," Proceedings of the nineteenth ACM symposium on Operating systems principles, vol. 37, no. 5, pp. 29-43, Oct. 2003.

〔5〕 J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Sixth Symposium on Operating System Design and Implementation(OSDI04), pp. 137-150, Dec. 2004.

〔6〕 김형준, 조준호, 안성화, 김병준, 클라우드 컴퓨팅, 에이콘, 2010년 12월.

〔7〕 정재화, 시작하세요 하둡 프로그래밍: 기초부터 실무까지 하둡의 모든 것, 위키북스, 2012년 10월.

〔8〕 O. O'Malley, K. Zhang, S. Radia, R. Marti, and Christopher, Hadoop security design, Yahoo!, Oct. 2009.

〔9〕 T. White, Hadoop: The Definitive Guide, 2nd Ed., O'Reilly Media, Yahoo! press, May 2011.

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

〔11〕 B.C.Beuman and T.Tso, "Kerberos: An authentication service for computer network," IEEE Communications, vol. 32, no.9, pp. 33-38, Sep. 1994.

〔12〕 L. Lamport, "Password authentication with insecure communication," Communications of the ACM, vol. 24, no. 11, pp. 770-772, Nov. 1981.

〔13〕 Behrouz A. Forouzan, Cryptography And Network Security, McGraw-Hill, Feb. 2007.

〈저자소개〉

정 소 원 (So Won Jeong) 학생회원
2012년 2월: 서울시립대학교 수학과 졸업
2012년 3월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 프라이버시향상기술(PET), 클라우드 컴퓨팅

김 기 성 (Kee Sung Kim) 학생회원
2008년 2월: 서울시립대학교 수학과 졸업
2011년 2월: 고려대학교 정보보호대학원 석사 졸업
2012년 3월~현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안, 암호 이론

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~2011년 8월: 고려대학교 정보경영공학전문대학원 조교수
2011년 9월~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 프라이버시향상기술(PET), 데이터베이스 보안, 암호 이론