다중 레이어 구조로 된 보안 파일 포맷 및API 설계

박종문* · 윤정호* · 조현태* · 김기창*

*인하대학교

File Formats with a Multi-Layer Structure and API Design

Jong-Moon Park* · Jeong-Ho Yoon* · Hyeon-Tae Jo* · Ki-Chang Kim*

*Inha University

E-mail : pinul1@gmail.com, yjh0801@hotmail.com, holsn@naver.com, kchang@inha.ac.kr

요 약

컴퓨터와 인터넷의 보급과 더불어 스마트폰의 확산으로 인하여 하루에도 수많은 데이터가 생산되고 수정되고 있다. 이렇게 데이터양의 증가로 인하여 이를 안전하게 저장하는 방법이 새로운 문제로 떠오르고 있다. 이에 본 논문에서는, 계층 데이터 구조로 이루어진 다중 레이어 방식으로 빅 데이터를 저장하며 레이어별 암호화를 적용하는 새로운 보안 파일Format과 이를 이용할 수 있는 API를 소개하고자 한다. 본 논문에서 소개하는 새로운 보안 파일 Format은 앞으로 많은 분야에 적용되어 사용되기를 기대한다.

ABSTRACT

Since the propagation of computers and Internet along with proliferation of smartphones rise, a large amount of data is being produced and modified daily. As the usage of data soars, a way of securely storing data emerged as a new problem. In this paper, saving big-data by using hierarchical data structure with multi-layer form, to come up with new security file format and API by applying encryption on each layers, is introduced. Moreover, we expect to see shown file format in this paper to be used in various fields.

키워드

Multi Layer, Multi User, Security, Big Data

Ⅰ. 서 론

정보화 시대가 가속화되면서 데이터의 종류와 양이 기하급수적으로 증가하였고, 이로 인해 데이터의 처리와 이를 안전하게 보관하는 방법이 새로운 화두가 되고 있다. 이런 다양한 데이터를 효율적으로 관리하기 위한 파일 형식 중 하나로HDF (Hierarchical Data Format)가 있다. 이 파일 포맷은 계층적 파일 구조를 가지며 단일 파일에 여러 유형의 데이터 저장이 가능하여 크기가 큰 다양한 범주의 데이터를 처리하는데 유용하다[1]. 하지만, HDF는 암호화 기능이 없어 보안의 위협으로부터 노출되어 있다. 이를 보완하기 위해서는 암호화 시스템을 이용하여야하는 번거로움이 있다. 이런 암호화 시스템 중엔 대표적으로eCryptFS가 있으나, 이 파일 시스템은 Linux 환경에서 단일 파일에 대한 암호화만 가능하다[2]. 이는 HDF의 계층적 구조 특성을 살리기에 어려움이 있다.

본 논문에서는 계층 데이터 구조를 가지며 레이어 단위의 데이터 암·복호화가 가능하도록 하는 새로운 보안 파일 포맷과 이를 이용할 수 있는 API를 제안할 것이다. 이 보안 파일은 데이터를 레이어 단위로 암호화하여 권한이 없는 경우 데이터를 확인할 수 없는 이점을 이용하여 여러 사람이 공동으로 진행하는 프로젝트나 여러 기업 간 공동 작업을 진행할 때와 같이 다양한 분야에서 활용이 가능할 것이다.

본 논문의 구성은 다음과 같다. Ⅱ장에선 논문을 작성하기 위해 참고하였던 기술인 eCryptFS, HDF, inode에 대해서 알아보고, Ⅲ장에서는 본 논문이 제시하는 보안 파일의 포맷과 계층적 파일 구조, Data 암호화 방법에 대해서 살펴볼 것이다. 또한 Ⅳ장에서는 파일 포맷과 API를 소개하고, 마지막 Ⅴ장에서 결론과 향후 과제에 대하여 논한다.

Ⅱ. 참고 기술

본 장에서는 본 논문에서 참고한 기술들에 대해서 알아본다.

Ⅱ－Ⅰ HDF(Hierarchy Data Format)

HDF 파일의 논리적인 구조는UNIX 파일 시스템의 계층적 구조와 비슷하며 UNIX의 directory와 file에 대한 개념을 Group과 Dataset으로 적용한다. HDF는 테이블 형태나 래스터 영상과 같은 여러 유형의 데이터 객체들을 하나의 파일에 저장하여 관리하기 용이하다. 현재는 HDF5까지 개발되었다[1]. 본 논문에서는 HDF의 이런 계층적인 파일 구조에서 착안된 계층적인 다중 레이어 구조를 설계한다.

Ⅱ－Ⅱ eCryptFS

eCryptFS는 리눅스 환경에서 구현된 Stackable한 암호화 파일 시스템으로써 단일 파일에 대한 암·복호화만을 지원한다. eCryptFS는 CryptFS를 확장한 파일 시스템으로 passphrase를 이용한 대칭키 방식과 OpenSSL의 RSA방식을 이용하는 비대칭키 방식을 지원한다[2]. 본 논문에서는 eCryptFS의 암호화 부분을 착안하여 Data와 Session Key의 암호화를 진행하였다.

Ⅱ－Ⅲ inode(index node)

inode는 UNIX의 파일 시스템에서 사용하는 자료구조로써 directory, file 등 파일 시스템에 관한 정보를 가지고 있다. 파일들은 각각 1개의 inode를 가지고 있으며, 이 구조체는 해당 파일에 관한 여러 정보를 담고 있다. 모든 파일들은 고유한 inode 번호를 통해 식별이 가능하며, 파일의 실제 데이터 블록들의 위치를 저장하는 포인터에 대한 정보도 가지고 있다[3]. 본 논문에서는 inode로 부터 lnode의 개념을 새롭게 만들어 사용한다.

Ⅲ. 제안하는 보안 파일 포맷

본 장에서는 본 논문이 제시하는 보안 파일의 포맷, 레이어 구조와 사용하는 암호화 방법을 설명한다.

Ⅲ－Ⅰ File Format

그림 1은 본 논문에서 제시하는 보안 파일의 전체적인 포맷이다. 이 파일은 6가지의 영역으로 구분되며, 첫 번째 영역은 Super Block으로 파일의 크기, 파일 마커, 버전 등의 전체적인 정보를 담

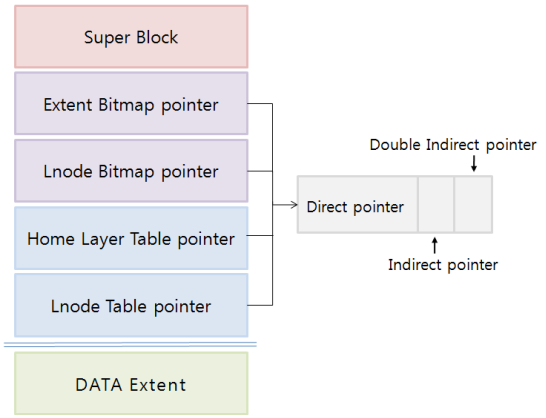

그림 1. File format

고 있는 header부분이다. 두 번째 영역은 Extent Bitmap Pointer이다. 여기서 Extent란 4k byte 크기의 블록으로서 데이터를 저장하기 위한 하나의 단위로 사용된다. Extent Bitmap은 비트의 위치에 따라 그 값이 1일 경우 해당 위치에 따른 Extent가 사용되고 있음을 나타내며, 이와 반대로 0일 경우 비사용의 상태임을 나타낸다. 1개의 Extent Bitmap은 4k*8bit=32k bit를 가지며, 이는 파일의 최대 크기가 32k*4k = 128M byte라는 제한을 두게 된다. 본 논문은 대용량 데이터의 저장 및 관리가 목적이기 때문에 Extent Bitmap Pointer의 구조를 갖추어 파일의 크기 제한 범위를 늘렸다. 8개의 Direct Pointer와 1개의 Indirect Pointer, 1개의 Double Indirect Pointer의 총 10개의 pointer로 구성된다. Direct Pointer는 Extent Bitmap 위치정보를 가리키고, Indirect Pointer는 1024개의 Direct Pointer가 저장된 Extent를 가리킨다. 마지막으로 Double Indirect Pointer가 가리키는 곳은 1024개의 Indirect Pointer가 저장된 Extent의 위치이다. 위의 구조는 총 8+1024+1024 *1024개의 Extent Bitmap을 만들 수 있고, 이는 파일의 제한 크기를 128Tera byte까지 확장한다. 이로써 대용량 데이터의 저장과 보관이 가능하다. 세 번째 영역인 Lnode Bitmap Pointer 또한 Extent Bitmap Pointer와 같은 구조로써, Lnode Bitmap은 현재 사용 중인 Layer의 번호를 저장한다. 네 번째와 다섯 번째 영역인 Home Layer Table Pointer와 Lnode Table Pointer는 레이어의 정보를 저장하는 Home Layer Table과 Lnode Table의 Pointer 집합이며, 그 구조는 위의 두 Pointer와 같다. Home Layer Table과 Lnode Table은 다음 절에서 자세히 논의한다. 마지막 여섯 번째 영역인 Data Extent는 파일의 실제적인 데이터들을 저장하는 영역으로, 새로운 정보가 추가될 때마다 Extent 단위로 크기가 늘어난다.

Ⅲ－Ⅱ Layer Structure

레이어의 구조는 그림2 와 같다. 레이어는 Home

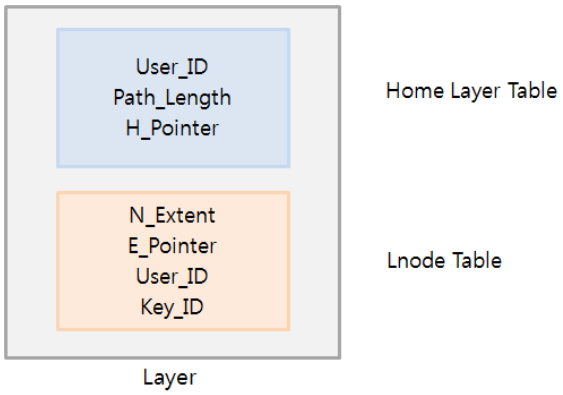

그림 2. Layer structure

Layer정보와 Lnode정보로 구성된다. Home Layer의 Path_Length는 레이어의 계층적 레벨을 저장한다. User_ID는 레이어 소유자의 ID 정보를 저장한다. H_Pointer는 레이어의 절대적 위치를 나타내는 Home_Path와 접근 가능한 모든 레이어들의 Session Key를 저장하고 있는 Extent들의 위치정보를 저장한다. H_Pointer[0]는 Home_Path의 정보를 저장하고 있는 Extent의 번호를 가리킨다. Home_Path는 현재 레이어의 절대적인 경로를 나타내는 정보이며, 접근 가능한 모든 레이어들을 나타낸다. H_Pointer[1], H_Pointer[2], ... H_Pointer[10]은 Direct pointer이며 접근 가능한 레이어들의 Session_key 정보를 저장한 Extent의 위치정보를 저장한다. H_Pointer[11]는 Indirect Pointer이며 Direct Pointer Extent의 위치정보를 저장한다.

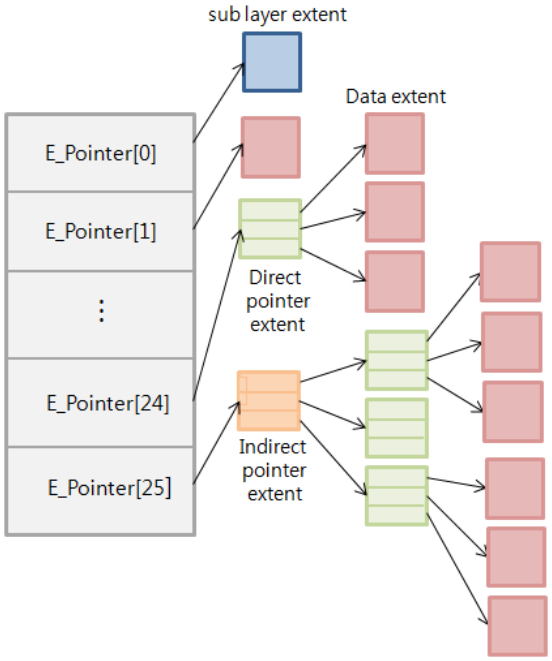

그림 3. Layer extent pointer

H_Pointer[12]는 Double Indirect Pointer로써 Indirect Pointer Extent를 가리킨다. 저장된 Session_key는 128byte의 사이즈를 갖으며 128개의 임의의 문자(0~256)의 값을 갖는다. 이 정보는 해당 레이어 사용자의 Private key를 사용하여 암호화 한 후 저장된다. Lnode의 N_Extent는 레이어가 사용하고 있는 Extent의 총 개수를 저장하여 데이터를 읽을 때 오류를 확인할 수 있도록 한다. Key_ID는 사용자의 Private key에서 추출한 정보를 가공하여 Key의 유효성을 판단하기 위한 정보이다. E_Pointer는 레이어 데이터들의 위치정보를 저장하며 그림3 은 이를 도식화한 것이다. E_Pointer는 26개로 구성되며, E_Pointer[0]는 레이어의 Sub Layer Extent를 가리킨다. Sub Layer Extent는 Unix 시스템의 계층 구조를 이용하기 위한 개념으로 자신의 노드 번호, 상위 레이어의 노드 번호, 하위 레이어의 이름과 노드 번호의 세 가지 정보를 저장한다. E_Pointer[1], E_Pointer[2], ..., E_Pointer[23]은 Direct pointer이며 Data Extent의 위치정보를 저장한다. E_Pointer[24]는 Indirect Pointer이며 Direct Pointer Extent의 위치정보를 저장한다. E_Pointer[25]는 Double Indirect Pointer로써 Indirect Pointer Extent를 가리킨다. 이는 하나의 레이어가 최대 4G byte의 정보를 저장할 수 있도록 한다. 레이어의 Home Layer 정보와 Lnode 정보는 각각의 특색에 따라 Home Layer Table과 Lnode Table에 나뉘어 저장된다.

Ⅲ-Ⅲ Hierarchy Structure

본 논문의 보안 파일은 계층적 구조를 가지고 있으며, 이는 HDF의 계층 구조를 착안하여 설계하였다. 계층 구조는 Layer의 path에 따라서 하위 Layer의 사용자는 상위 Layer의 권한까지 갖는 구조로 세부적으로 내려갈수록 더 높은 권한을

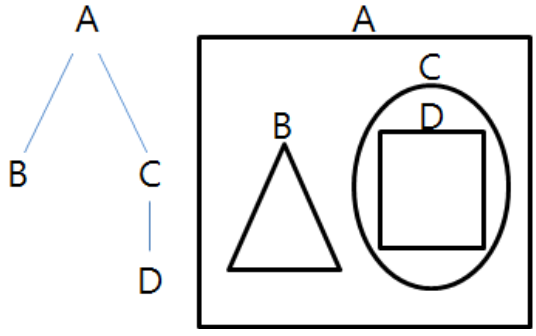

그림 4. 간단한 예

갖는 구조이다. 5명의 사용자로 이루어진 간단한 예인 그림4를 살펴보면, A가 최상위 Layer이며 그 밑으로 B, C가 위치하고 C의 하위로는 D Layer가 구조하고 있다. 즉, 그림4의 예에서 A는 A의 Layer만 접근할 수 있고, B는 A, B Layer, C는 A, C Layer, D는 A, C, D의 Layer에 접근이 가능하다.

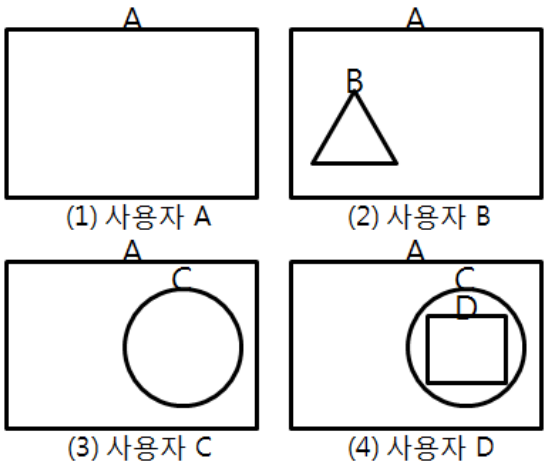

그림 5. 사용자의 관점의 Layer

그림4를 각 사용자의 측면에서 보게 되면 그림5와 같다. 사용자 A는 최상위 Layer로서 자기 자신의 Layer만 접근이 가능하다. 이를 나타낸 그림이 그림5-(1)이다. 사용자 B는 자기 자신과 그 위의 A Layer 접근이 가능하다 이를 나타낸 것이 그림5-(2)이다. 그림5-(3)은 사용자 C가 접근할 수 있는 A, C를 나타내고 있으며, 그림5-(4)에서는 사용자 D가 접근 가능한 A, C, D에 대하여 나타내고 있다.

Ⅲ-Ⅳ Cryptography

본 절에서는 본 논문이 제안하는 보안 파일에서 사용되는 암호화 부분을 설명한다.

- Session Key : Layer의 Session Key 암·복호화 과정은 OpenSSL에서 제공하는 RSA방식을 이용하며, 그 과정은 아래와 같은 절차로 진행된다.

·Encryption

1. 해당 User의 Lnode에서 사용자의 Key ID를 확인하고 해당 ID의 Public Key를 읽어온다.
2. 암호화할 Layer의 Session Key를 사용자의 Public Key로 암호화한다.

·Decryption

1. 암호화된 Layer의 User Lnode에서 사용자의 Key ID를 확인하고 해당 ID의 Private Key를 읽어온다.
2. 암호화된 Layer의 Session Key를 사용자의 Private Key로 복호화한다.

- Data : 실질적인 Layer Data의 암·복호화 과정은 OpenSSL에서 제공하는 AES-128(ecb)방식을 이용하며, 그 과정은 아래와 같은 절차로 진행된다.

·Encryption

1. Layer의 Session Key(128bytes)를 MD5 (Message-Digest algorithm 5)방식으로 해싱하여 128bits의 키를 생성한다.
2. 생성된 128bits의 키를 AES-128방식의 대칭키로 사용하여 Layer Data의 암호화를 수행한다.

·Decryption

1. Encryption에서와 같이 Session Key를 MD5 해싱을 통하여 128bit의 Key를 생성한다.
2. 생성된 128bits의 Key를 AES-128방식의 대칭키로 사용하여 Layer Data의 복호화를 수행한다.

Ⅳ. 제공되는 API

본 논문에서 제시한 보안 파일 포맷을 사용하기 위하여 다음의 API를 제공하며 그 내용은 표 1과 같다.

표 1. 제공되는 API와 간단한 설명

함 수	내 용
MLE_fcreate	파일 생성
MLE_fopen	파일 열기
MLE_fclose	파일 닫기
MLE_fsave	파일 저장
MLE_lcreate	레이어 생성
MLE_lopen	레이어 열기
MLE_ldelete	레이어 삭제
MLE_lsave	레이어 저장
MLE_lclose	레이어 닫기
MLE_lwrite_data	레이어에 데이터 저장
MLE_lread_data	레이어 데이터 읽기
MLE_display_meta	메타 데이터 출력
MLE_display_file_view	데이터 구조 출력

Ⅴ. 결론

IT 기술의 눈부신 발전으로 인하여 데이터양의 증가와 함께 그에 따른 대용량 데이터의 저장 보안 기술은 매우 중요해졌다. 이에 본 논문에서는 이런 문제의 해결을 돕고자 기존의 방식에서 발전된 새로운 보안 파일 포맷과 API를 제시하였다. 본 논문에서 제시하는 보안 파일 포맷은 대용량 데이터 저장 문제를 해결하며 다양한 분야에서 활용이 가능할 것으로 기대한다.

참고문헌

[1] Introduction to HDF5, http://www.hdfgroup.org/HDF5/doc/H5.intro.html

[2] Michael Austin Halcrow, "eCryptfs: An Enterprise-class Cryptographic Filesystem for Linux", International Business Machines, Inc. Proceedings of the 2005 Linux Symposium, 2005

[3] "Inode Definition", The Linux Information Project, http://www.linfo.org/inode.html, September, 2006