www.calsec.or.kr
http://dx.doi.org/10.7838/jsebs.2013.18.4.045

안전한 협업 디자인 작업을 위한 다중 레이어 파일 포맷

Multiple Layer File Format for Safe Collaborative Design

김기창(Kichang Kim)*, 유상봉(Sang Bong Yoo)**

초록

타겟 오브젝트의 복잡도가 높아질수록 디자인 파일의 사이즈도 커질 것이다. 이러한 큰 파일은 클라우드 시스템 같은 거대한 병렬 저장 시스템에 저장되고 수많은 디자이너들이 동시에 이 파일에 접속하여 디자인 작업을 수행하게 될 것이다. 이러한 경우 필연적으로 각 사용자가 정해진 파일 내 위치에서만 작업하도록 하는 제어 장치가 필요하게 될 것이다. 이 제어 장치에는 두 가지 접근 방법이 가능하다. 하나는 Role-Based Access Control(RBAC) 등의 전통적인 접근 제어 방법이고 다른 하나는 암호화 방법이다. 본 논문은 두 번째 방법을 택하되 파일이 각 레이어에 따라 다르게 암호화 되어 각 사용자가 자신의 레이어에서만 작업을 할 수 있도록 하는 기법을 제안하고 있다. 파일의 각 레이어의 키 역시 그 레이어를 접근할 수 있는 사용자만 복호화할 수 있는 형태로 암호화되어 저장된다. 본 논문은 이러한 목표를 이루기 위한 파일 포맷을 제시하며 그 파일을 다룰 수 있는 API 함수를 제시하고 설명한다.

ABSTRACT

A design file can get larger in size as the complexity of the target object increases. A large design file may reside in a large parallel computing system, such as cloud computing systems, and many designers may work concurrently on the same design file. In such a case, it is obvious that we need some kind of protection mechanism so that each user can access only the area of the file he or she is entitled to. Two approaches can be taken for this problem: one is the traditional access control mechanisms and the other encryption techniques. We take the latter approach to ensure the safety of the file even in public domain such as clouding systems, and in this paper, we suggest an encryption scheme for a file where the file is encrypted in multi-layer so that each user is allowed to access the file only at the layer for which the user has the proper access right. Each layer of the file is encrypted with different keys and these keys are exposed only to those who have the right access permit. The paper explains the necessary file format to achieve this goal and discusses the file manipulation functions to handle this new file format.

키워드 : 다중 레이어 암호화, 협업 디자인, 접근 제어, 객체 트리, 파일 포맷
Multiple Layer Encryption, Collaborative Design, Access Control, Object Tree, File Format

---

본 논문은 인하대학교의 지원 및 2012년도 교육과학기술부의 재원으로 한국 연구재단의 지원을 받아 수행된 연구임(과제번호: 2012R1A1A2006653).

* School of Information and Communication Engineering, Inha University(kchang@inha.ac.kr)
** Corresponding Author, School of Computer and Information Engineering, Inha University(syoo@inha.ac.kr)

2013년 02월 14일 접수, 2013년 10월 01일 심사완료 후 2013년 10월 11일 게재확정.

1. 서 론

현대 디자인 작업에서 협업 디자인을 해야 하는 상황이 빈번히 발생한다[19]. 협업 디자인에서는 수많은 디자이너들이 동일한 파일에서 작업을 해야 하는 경우가 발생할 수 있고 이 경우 각 디자이너가 원활하게 자신의 영역 안에서 안전하게 작업할 수 있도록 하는 접근제어 장치가 필요하다. 이 문제에 대한 전통적인 해결 방안은 Role-Based Access Control(RBAC)[1]이다. RBAC에서는 각 디자인 객체에 롤(role)을 부여하고 각 사용자에게 하나 혹은 복수개의 롤을 역시 부여한다. 각 사용자는 자신의 롤이 허용하는 객체만 접근할 수 있다. RBAC은 그 동안 여러 협업 디자인 문제에 성공적으로 적용되었으며 광범위하게 연구되어 왔다[19]. 하지만 디자인 스페이스의 크기와 복잡도가 증가하면서 “롤 폭발”이라고 불리우는 확장성 문제를 노출하게 되었다[11]. 특히 클라우드 시스템 같은 데이터 공유 환경에서 암호화 없는 접근제어는 언제나 정보노출의 문제점을 지니고 있다는 것이 지적되고 있다[2, 3, 10, 20].

위의 이유로 파일 접근 제어 목적으로 파일 암호화 해결방안이 제안되어 왔다. Wang et al.[15]은 사용자의 속성에 기반한 키 조합으로 파일을 암호화하고 키 조합에 참여한 사용자에게만 파일 접근을 허용하는 기법을 제안하였다. eCryptfs[8]은 각 파일을 임의의 키로 암호화하고 이 키를 파일 헤더에 다시 한번 암호화된 형태로 기록하는 파일 시스템을 제안하고 있다. 이 경우 이 파일을 접근할 수 있는 사용자의 공개키로 이 키를 암호화하므로(공개키 시스템을 사용하는 경우), 이 임의의 키를 복호화할 수 있는 사용자에게만 파일 접근을 허용하는 효과를 거둘 수 있다. 암호화된 키를 포함하여 파일을 접근하는데 필요한 모든 정보가 파일 헤더에 포함되어 있으므로 더 이상의 중앙 관리자의 필요 없이 파일을 자유롭게 그리고 안전하게 주고받을 수 있다. 하지만 여전히 여러 사용자가 각각 파일의 해당 부분만을 접근하도록 하는 다중 사용자의 동시 파일 접근을 허용하지는 못한다. 즉 eCryptfs는 합법적인 사용자만이 동일 파일을 전체적으로 접근할 수 있도록 하지만, 각 사용자가 자신에게 허용된 파일의 서로 다른 부분만을 접근하도록 하는 기능은 제공하지 못하고 있다.

본 논문은 다중 레벨 암호화에 의해 여러 사용자가 각각 자신에게 허용된 파일의 일부 영역 안에서만 작업하도록 하는 파일 접근제어 기법을 제안한다. 제안된 기법은 우선 파일을 계층적으로 구성된 디자인 객체들의 집합으로 보고 이것들을 객체 트리로 표현한다. 이 객체 트리의 노드는 각 객체가 되며 에지는 객체간의 부모 자식 관계를 표현한다. 제안된 기법은 각 객체를 서로 다른 임의의 암호화 키(세션 키라고 불리운다)로 암호화 한다. 파일 헤더에는 각 객체의 물리적 위치와 해당 세션 키를 암호화된 형태로 가지고 있는 Layer Node Table이 존재한다. 암호화된 세션 키 자체의 길이가 길기 때문에(현재 구현된 시스템에서는 256바이트) 모든 암호화된 세션 키는 별도의 테이블인 ESKN Table (Encrypted Session Key Node Table)에 저장하고 세션 키를 저장해야 하는 위치에는 이 테이블의 해당 키 인덱스만을 저장한다. 따라서 Layer Node Table에도 각 객체마다 해당 세션 키

가 실제 저장되어 있는 대신 ESKN Table에서의 위치가 저장되어 있다.

이 파일을 접근할 수 있는 사용자들에 대한 정보도 파일 헤더에 포함되어 있다. 이 정보는 User Node Table에 기록되어 있으며 각 합법적 사용자 마다 그 사용자의 홈 레이어 위치, 그 사용자가 접근할 수 있는 레이어들에 대한 암호화된 세션 키 위치(ESKN Table에서)가 저장되어 있다. 시스템은 이 파일에 접근하는 사용자 마다 User Node Table을 참조하여 이 사용자가 파일의 어느 레이어에 접근 가능한 지를 판단하고 Layer Node Table을 참조하여 해당 레이어들의 물리적 위치를 알아낼 수 있다. 위 방법을 사용하여 본 논문이 제안하는 MLE 파일 포맷은 ECryptfs처럼 파일 접근 제어에 대한 모든 정보를 파일 헤더에 담고 있으므로 각 파일이 독립적으로 중앙 관리자의 도움없이 처리될 수 있도록 하고있다. 즉 MLE 파일 포맷을 이해하는 MLE 클라이언트 프로그램은 합법적 개인키를 제공하는 사용자에게 그 사용자에게 허용된 파일 부분을 복호화해서 보여주고 편집할 수 있도록 서비스를 제공할 수 있다. 본 논문의 이후 진행은 제 2장에서 관련 연구를 소개하고, 제 3장에서 제안된 MLE 파일 포맷을 상세히 설명하며, 제 4장에서는 MLE 파일 API 함수들을 소개하고, 제 5장에서는 MLE 서버와 클라이언트의 상호 작용에 대해 설명하며 마지막으로 제 6장에서 결론을 짓는다.

2. 관련 연구

수많은 연구자들이 협업 디자인 관점에서 정보 보호 문제에 대해 연구를 수행하였다[5, 6, 16]. 협업 디자인은 필연적으로 네트워크 환경을 사용하게 되는데 [9]는 이러한 환경에서 웹 기반 CAD를 사용하여 안전하게 협업하는 문제를 논의하고 있다. 이 논문에서는 웹 기반 CAD에서 안전한 협업을 위해 3가지 카테고리를 정의하고 있다 : 클라이언트 보안, 전송 보안, 그리고 서버 보안이 그것이다. 클라이언트와 전송 보안 문제는 다른 분야에서의 문제와 유사하므로 기존 기술을 사용할 수 있다. 하지만 서버 보안 문제는 협업 디자인에 고유한 문제로서 이 논문이 제안하는 기본 해결책은 다른 디자이너에게 노출되는 정보를 최소화하는 것이다. 코멘트 제거, Netlist 평면화, 함수 제거 등등의 방법이 제안되고 있다. 하지만 해결책의 대부분이 사용자의 액션에 의존한다는 문제점이 있다.

Cera et al.[4]은 3D 모델을 파티션하고 각 파티션을 다중 해상도매시 계층구조로 표현한 후 중앙 접근 제어 장치가 각 사용자의 뷰 및 편집 권한을 계산하여 해당 부분만 접근하도록 하는 기법을 제안하고 있다. 중앙 접근 제어 장치가 모든 사용자의 접근을 제어해야 하므로 디자인 복잡도가 높아지면 관리 오버헤드가 크게 증가하는 문제점이 있다.클라우드 환경이 대두됨에 따라 데이타 공유 환경에서 암호화를 통해 파일 접근제어를 하는 방법들에 대해서도 연구가 진행되고 있다. Wang et al.[16]은 데이터 셀을 전송할 때 그 서브 데이터 셀들을 서로 다른 키로 암호화하여 전송하는 다중 레벨 암호화 기법을 제안하고 있다. 이와 같은 다중 레벨 암호화에 의해 클라이언트들은 같은 데이터를 전송 받아도 서로 다른 데이터를 보게된다. 이 기법

은 하지만 저자들이 정의한 XML 포맷에 의해 데이터를 안전하게 전송할 때 사용되는 암호화 기법으로서 우리가 다루고 있는 파일의 다중 레벨 암호화 문제와는 차이가 있다[15]는 디자인 데이타를 그 데이타를 접근할 수 있는 합법적 롤의 속성을 키로 하여 암호화하는 방식을 제안하고 있다. 롤이나 속성에 의해 접근을 제어하는 방법은 많은 경우에 효과적이기는 하나 타겟 디자인의 복잡도가 증가하면 중앙 제어장치에서 접근 제어 비용이 급증하는 "롤 폭발" [11]의 문제에 직면하게 된다. 한편, 접근제어를 위한 기존의 암호화 기법은 대부분 파일 전체를 동일 키로 암호화하는 방식으로서 다중 사용자가 각각 허용된 영역에서 안전하게 동시 작업을 하는 파일의 다중 레이어 암호화 문제에는 그대로 적용하기가 어렵다.

본 논문은 디자인이 복잡해져도 접근 제어의 비용을 증가시키지 않으면서 동시에 동일 파일 내의 다중 레이어에 대한 안전한 동시 접속을 가능하게 하는 기법을 제안한다. 본 논문의 기법은 eCryptfs[8], HDF[7], ROI 암호화[17] 등의 기술을 기반으로 하고 있다. eCryptfs는 파일을 암호화하고 그 키를 합법적 사용자의 공개키로 다시 암호화하여 파일 헤더에 저장하는 기법이다(공개키 방법을 쓰는 경우). 다중 사용자가 이 파일을 접근해야 한다면 키를 사용자 수 만큼 복사하여 각 사용자의 공개키로 암호화하여 저장한다. 본 논문은 파일 헤더에 접근 제어 정보를 직접 저장하는 기법을 eCryptfs로부터 빌려왔다. 하지만 eCryptfs는 여러 사용자가 같은 파일을 안전하게 접근할 때 사용되는 파일 시스템으로 여러 사용자가 파일의 서로 다른 부분을 안전하게 접근하도록 해주는 MLE 시스템과 근본적인 차이점이 있다. HDF는 데이타셀을 계층적 구조로 파일에 저장하는 기술이다. 데이타셀은 유닉스/리눅스에서의 디렉토리 구조와 유사한 형태의 계층적 트리로 구성된다. 본 논문은 디자인 객체를 계층적 트리로 구성하는 기법을 HDF로부터 빌려왔다. 하지만 HDF 역시 각 디자인 객체에 대해 암호화 기능을 제공하는 MLE 파일 시스템과는 차이가 있다.

마지막으로 ROI 암호화는 입력 이미지 파일에 ROIs(Regions Of Interest)를 설정하고 각 ROI들을 서로 다른 키로 암호화하는 기법이다. 예를 들어 현 이미지 파일에 ROI1, ROI2, ROI3이 level-of-detail순으로 존재한다고 할 때(ROI3이 가장 상세한 레벨), 이 기법은 각 ROI를 위 순서대로 키(k1), (k1, k2), (k1, k2, k3)으로 각각 암호화한다. (k1, k2)는 k1으로 먼저 암호화하고 그것을 다시 k2로 암호화하는 것을 의미한다. 키 k1을 가진 사용자는 ROI1만 복호해서 이미지를 볼 수 있다. k1, k2, k3를 모두 가진 사용자는 모든 ROI를 볼 수 있다. 각 ROI를 서로 다른 키 조합으로 암호화함으로써 이 기법은 같은 파일이 여러 사용자에게 여러 레벨에서 보여지도록 하고 있다. 유사한 다중 레벨 암호화는 비디오 이미지에 대해서도 적용되고 있다[18]. 비디오 이미지는 중첩된 비트 플레인(bit-plane)으로 저장된다. [18]에서 제안하는 기법은 서로 다른 비트 플레인을 서로 다른 키로 암호화함으로써 모든 키를 다 가지고 있는 사용자는 최대 해상도에서 화면을 볼 수 있고 가장 기초적인 레벨의 키만 가지고 있는 사용자는 최소 해상도에서 화면을 보는 등 사용자 마다

서로 다른 해상도 레벨에서 비디오 파일을 접근할 수 있도록 제어하고 있다. 본 논문은 서로 다른 객체를 서로 다른 키로 암호화하는 기법을 ROI로부터 빌려왔다. ROI 기법은 본 논문이 제안한 MLE 기법의 기본 알고리즘을 구성하는데 기여하고 있다. 하지만 ROI 기법은 디자인 객체들을 트리 형태로 저장하고 이것들을 암호화한 키 값에 대한 정보를 조직적으로 파일 헤더에 저장하는 MLE 파일 시스템과는 차이가 있다.

결국 본 논문이 제안하는 기법은 디자인 파일의 객체를 HDF에서 처럼 트리 형태의 계층적 구조로 조직하고, 각 객체를 ROI 암호화에서 처럼 서로 다른 키로 암호화하며, 이 키에 대한 정보, 합법적 사용자들에 대한 정보, 디자인 객체들의 위치 정보 등을 eCryptfs에서 처럼 파일 헤더에 저장하도록 함으로써 여러 사용자가 동시에 동일 파일을 각자 자신에게 허용된 파일 영역(혹은 파일 레이어)에서만 안전하게 작업할 수 있도록 하고 있다.

3. MLE(Multiple-level Encryption)파일 포맷

MLE 파일 포맷은 안전한 협업 디자인을 위한 파일 포맷이다. 단일 디자인 파일에 수많은 디자이너가 동시에 작업을 해야 하는 상황을 가정한다면 각 디자이너가 자신의 디자인 기술이 누출되는 것을 막으면서 효율적으로 다른 디자이너와 협업을 할 수 있는 파일 형식이 필요하다는 것은 자명하다. 여러 사용자가 단일 디자인 파일에 대해 안전하면서도 효율적인 협업을 할 수 있기 위해서는

- 디자인 객체들의 레이어 트리 정보
- 이 파일을 접근할 수 있는 합법적 사자들에 대한 정보
- 디자인 객체들을 암호화한 키에 대한 정보

등 세 가지 정보가 관리되어야 한다. MLE 파일시스템은 레이어노드 테이블에 레이어 트리 정보를 기록하고 합법적 사용자들에 대한 정보를 사용자 노드 테이블에 저장하고 마지막으로 암호화 키에 대한 정보를 세션키노드 테이블에 저장한다.

3.1 레이어노드 테이블(LNTAB)

레이어노드 테이블은 레이어노드(LNODEs)의 테이블로서 각 디자인 객체(본 논문에서는 레이어라고 불리운다)의 위치 정보와 해당 세션키를 저장하고 있다. <Figure 1>은 디자인 프로젝트의 예를 보여주고 있으며 <Figure 2>는 해당 디자인 객체 트리를 보여주고 있다. 그림은 5개의 디자인 객체를 보이고 있으며 따라서 레이어노드 테이블은 5개의 LNODE를 가지게 된다. 각 LNODE는 64바이트 크기로서 다음의 구조를 지닌다.

```
classLayerNode{
// on-disk data
int num_ext;
int loc[MEXT];
int loc_sub;
int esk;
unsigned char reserv[12];
// on-memory data follows
.....
};
```

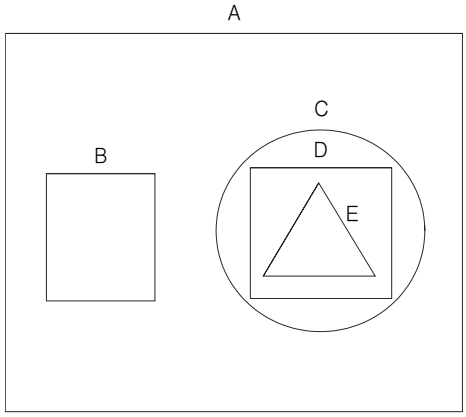
*num_ext*는 이 레이어가 소유한 데이터 익

스텐트(혹은 블록)의 갯수를 나타낸다. 각 레이어는 적어도 두 개의 익스텐트를 가진다. 하나는 이 레이어의 데이터를 포함하고 있는 데이터 익스텐트이고 다른 하나는 이 레이어에 소속된 하위 레이어에 대한 정보를 담고 있는 디렉토리 익스텐트이다. loc[ ]배열은 각 데이터 익스텐트의 위치를 담고 있다. MEXT는 10으로 정의되어 있다. loc[ ]배열의 첫 8개의 방에는 위치 정보에 대한 직접 포인터가 있으며 그 다음 방에는 간접 포인터, 마지막 방에는 더블 간접 포인터가 들어 있다. 직접 포인터는 실제 해당 익스텐트의 위치를 직접 표시하는 포인터이다. 각 익스텐트의 크기가 4096바이트이므로 주어진 MLE 파일의 첫 4096 × 8바이트는 직접 포인터를 사용해서 위치를 찾을 수 있다. 간접 포인터는 1024개의 직접 포인터가 저장되어 있는 익스텐트를 가리키는 포인터이다. 더블 간접 포인터는 1024개의 간접 포인터가 저장되어 있는 익스텐트를 가리키는 포인터이다. 간접 포인터와 더블 간접 포인터까지 사용하면 최대 4096×8 + 1024 × 4096 + 1024 × 1024 × 4096바이트(4G바이트보다 조금 큰 크기) 크기의 파일에 대해 위치 정보를 기록할 수 있다.

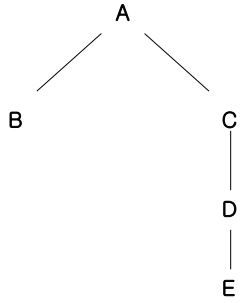
*loc_sub*는 하위 레이어에 대한 정보를 담고 있는 익스텐트이다. 만일 현 레이어가 객체 트리의 말단이라면 loc_sub는 무의미하다. 하지만 그렇지 않다면 이 포인터는 아래의 dirent 구조체(32바이트 크기)의 리스트가 들어있는 익스텐트를 가리킨다.

```
structdirent{
  char   lname[MAX_NAME_LEN];
  intlnode_idx;
  char   reserv[12];
};
```

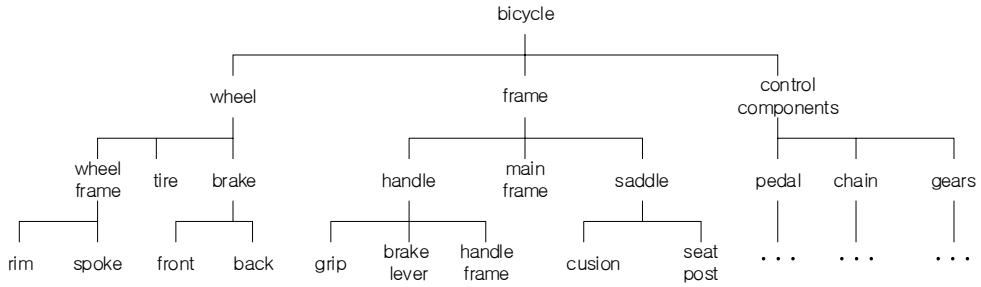
위 구조체에서, *lname*은 해당 하위 레이어의 이름을 말하며 *lnode_idx*는 그 레이어가 Layer Node Table에 기록되어 있는 인덱스를 말한다. *esk*는 Encrypted Session Key Node Table(ESKNTAB)에서 이 레이어의 세션 키 스케쥴이 암호화되어 저장되어 있는 위치를 가리킨다. 새로운 레이어가 생성될 때 시스템은 그 레이어를 위한 8바이트 DES 키



〈Figure 1〉 Example of Simple Design Project



〈Figure 2〉 Object Tree for〈Figure 1〉



〈Figure 3〉 Object Tree for Bicycles

를 생성하고 그것으로부터 8×16바이트 키 스케쥴을 계산한다. 이 키 스케쥴이 MLE 서버의 마스터 공개키로 암호화되어 ESKNTAB에 저장된다. 이 저장된 위치를 가리키는 것이 *esk*이다.

<Figure 4>는 <Figure 2>의 객체 트리에 대한 Layer Node Table을 보여주고 있다. LNODE 0는 레이어 A를 나타내고, LNODO 1은 레이어 A/B, LNODE 2는 레이어 A/C, LNODO 3는 레이어 A/C/D, LNODE 4는 레이어 A/C/D/E를 각각 나타낸다. 이 트리에 대해 각 레이어의 데이터 익스텐트가 모두 하나뿐이라고 가정하면 그림에서처럼 모든 레이어의 *num_ext*는 1이 되고 *loc[0]*는 각 레이어의 데이터 익스텐트 위치를 나타내게 된다. 각 레이어의 디렉토리 익스텐트 위치는 그림의 *loc_sub* 컬럼에 나와 있다. 레이어 B와 E는 말단 레이어이므로 디렉토리 익스텐트에 아무 데이터가 없지만 레이어 A, C, D는 서브 레이어를 가지고 있으며 <Figure 5>는 이들 레이어의 디렉토리 익스텐트의 내용

	num_ext	loc[0]	loc[1]…loc[9]	loc_sub	esk
0	1	37		38	0
1	1	39		40	1
2	1	41		42	2
3	1	43		44	3
4	1	45		46	4
	.....	.....	.....	.....	.....

〈Figure 4〉 Layer Node Table for Object Tree in〈Figure 2〉

ext 38

lname	Inode_idx
B	1
C	2

ext 42

lname	Inode_idx
D	3

ext 42

lname	Inode_idx
E	4

〈Figure 5〉 Directory Extent for Layers A, C, and D

	ekdy	lnode_idx
0	23 24 3a 4b	0
1	d 2c 44 52	1
2	21 110 3b	2
3	a2 67 21 33	3
4	de 34 55 4b	4

〈Figure 6〉 Encrypted Session Key Node Table

을 보여주고 있다. 레이어 A는 두 개의 서브 레이어인 B와 C를 가지고 있다. 따라서 그림에는 익스텐트 38에 레이어 B와 C의 LNODE 인덱스 값이 나와 있다. 레이어 C와 D는 각각 한 개의 서브 레어를 가지고 있으며 역시 해당 서브 레이어의 LNODE 인덱스 값이 그림에 나와 있다. <Figure 4>에는 또한 각 레이어에 대해 암호화된 세션 키(실제로는 세션 키 스케쥴)가 ESKNTAB의 어디에 있는지가 *esk* 컬럼에 나와 있다. 예를 들어 레이어 A (LNODE 0)는 *esk* 값이 0이므로 그림 6의 ESKNTAB의 인덱스 0에 있는 23 24 3a 4b … 바이트 나열이 레이어 A의 암호화된 세션 키 스케쥴이다.

3.2 사용자 노드 테이블(UNTAB)

사용자 노드 테이블(User Node Table)은 User Node(UNODE)의 테이블이다.

UNODE는 해당 MLE 파일에 대한 접근 권한이 있는 등록 사용자들에 대한 정보를 표현하는 구조체로서 64바이트의 크기를 가지며 아래 구조를 가진다.

```
class UserNode{
  // on-disk data
  char  uname[MAX_NAME_LEN];
  int   home;
  int   esk[MAX_ESK];
  unsigned char reserv[12];
  // on-memory data follows
  .....
};
```

*uname*은 사용자의 이름을 나타낸다. *home*은 이 사용자의 홈 레이어의 LNODE 인덱스이다. 각 사용자는 홈 레이어가 정의되어 있어서 이 홈 레이어와 그 밑의 하위 객체 트리에 있는 모든 레이어에 대한 접근 권한을 가진다. 예를 들어 사용자 UA가 <Figure 2>의 레이어 A에 홈 레이어를 갖는다면 이 사용자의 *home* 값은 0이고 이 사용자는 <Figure 2>의 모든 레이어에 대해 접근 권한을 갖는다. *esk[ ]*에는 현 사용자가 접근할 수 있는 레이어들에 대한 세션 키들이 이 사용자의 공개키로 암호화되어 저장된 ESKNODE들의 인덱스 번호가 들어있다. 사용자 X가 레이어 Y를 홈레이어로 등록된다면 레이어 Y 및 그 지점에서 시작하는 객체 트리의 모든 하위 레이어가 접근 가능 레이어가 되므로 이들 레이어에 대한 세션 키가 ESKNTAB에 암호화되어 저장되고(이 사용자의 공개키로 암호화되어) 그 위치가 *esk*[ ]에 저장되는 것이다.

<Figure 7>은 세 명의 사용자가 <Figure 2>의 객체 트리에 등록되었을 때의 User Node Table을 보여준다. 이 사용자들의 이름은 UA, UC, UE이며 각각 레이어 A(LNODE 0), 레이어 A/C(LNODE 2), 레이어 A/C/D/E (LNODE 4)에 홈 레이어를 두고 있다. UA의

홈레이어(A 혹은 LNODE 0)는 디자인 객체 트리의 루트이다. 따라서 UA는 트리의 모든 레이어를 접근할 수 있어야 하며 이를 위해 UA의 공개키로 모든 레이어의 세션 키를 암호화하여 저장하고 있다. <Figure 8>의 ESKNTAB 테이블은 인덱스 5, 6, 7, 8, 9에 UA가 접근할 수 있는 모든 레이어의 암호화된 세션 키가 저장되어 있는 모습을 보여주고 있다.각 세션 키 옆에 *lnode_idx*가 나타나 있으며 이 컬럼이 타겟 LNODE 인덱스를 나타낸다. 각 세션 키의 인덱스 값은 UA가 저장되어 있는 UNODE 0의 *esk*[ ]배열에 저장되어 있다(<Figure 7> 참조). 사용자 UC와 UE도 유사하게 등록된다. UC는 홈 레이어 A/C를 가지고 있으며 레이어 A/C(자기 자신)와 레이어 A/C/D, 그리고 레이어 A/C/D/E(하위 레이어)등을 접근 할 수 있다. 이 레이어들의 세션 키 역시 UC의 공개키로 암호화되어 ESKNTAB 테이블 인덱스 10, 11, 12에 저장된다. 그리고이 인덱스들은 UC의 UNODE인 UNODE 1에 저장된다. 마지막으로 UE는 홈 레이어 A/C/D/E에 있으며 하위 레이어가 없으므로 홈 레이어의 세션 키만 암호화되어 ESKNTAB 테이블의 인덱스 13에 저장된다. UNODE 2(UC의 UNODE)는 이 인덱스 값을 *esk*[0]에 기억하고 있다.

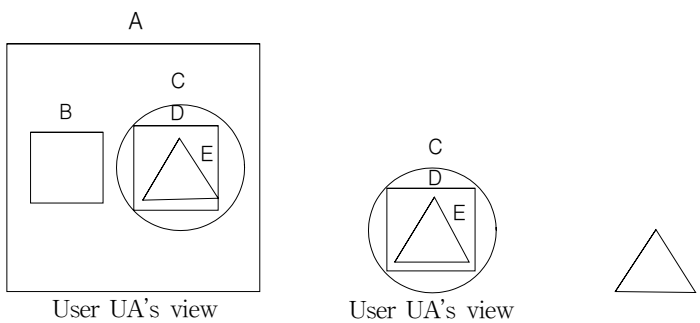
각 사용자는 자신의 홈 레이어가 있는 위치에서 시작하는 객체 트리에 대해서만 뷰나 편집을 할 수 있다. <Figure 9>는 각 사용자에게 보여지는 객체 트리를 보여주고 있다.

	uname	home	esk[0]	esk[1]	esk[2]	esk[3]	esk[4]	·····
0	UA	0	5	6	7	8	9	
1	UC	2	10	11	12			
2	UE	4	13					

〈Figure 7〉 User Node Table

	ekdy	lnode_idx
	·····	···
5	1d 22 24 52 ·····	0
6	11 22 33 44 ·····	1
7	12 aa bb 22 ·····	2
8	d1 24 54 4b ·····	3
9	1a 2a 3a 22 ·····	4
10	13 14 15 16 ·····	2
11	21 31 41 51 ·····	3
12	1e 2e 3e 4e ·····	4
13	22 33 41 53 ·····	4

〈Figure 8〉 Encrypted Session Key Node Table After 3 Users Registrate



〈Figure 9〉 Objects Tree Viewed by Different Users

3.3 암호화 세션 키 노드 테이블 (ESKNTAB)

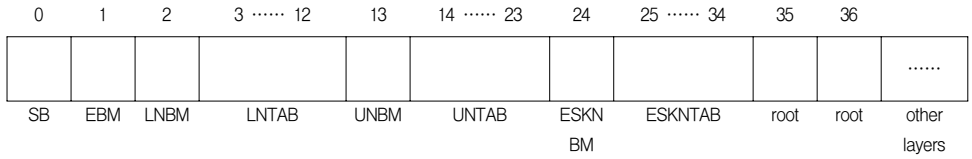
ESKNTAB 테이블은 암호화된 세션 키 노드(ESKNODE)들을 저장하고 있는데 각 ESK NODE는 암호화된 세션 키와 그 키가 사용되는 타겟 LNODE의 인덱스 값을 가지고 있다. ESKNODE는 260바이트 크기이며 아래의 구조를 가지고 있다.

```
classESKNode{
  // on-disk data
  unsigned char ekey[EKEY_LEN];
  intlnode_idx;
  // on-memory data follows
  .....
};
```

위에서 *ekey*가 암호화된 세션 키이고 *lnode_idx*가 타겟 LNODE의 Layer Node Table 인덱스이다. <Figure 8>에 ESKNTAB 테이블의 예가 나와있다. 14개의 ESKNODE가 저장되어 있다. ESKNODE 0에서 4는 LNODE 0에서 4의 세션 키를 MLE 서버의 마스터 공개키로 암호화한 값이 들어있다. ESKNODE 5에서 9 역시 LNODE 0에서 4까지에 대한 암호화된 세션 키이다. 하지만 이번에는 UA의 공개키로 암호화한 값들이다. UA는 자신의 개인키로 이 값들을 복호화하여 어떤 레이어든지 접근할 수 있다. ESKNODE 10, 11, 12는 LNODE 2, 3, 4의 키가 UC의 공개키로 암호화한 값을 가지고 있다. 사용자 UC는 이 키들을 복호화하여 레이어 A/C 및 그 밑의 하위 레이어들을 접근할 수 있다. 마지막으로 ESKNODE 13은 LNODE 4의 세션 키를 UE의 공개키로 암호화하여 가지고 있으며 UE가 레이어 A/C/D/E를 접근하는데 사용된다.

3.4 MLE 파일의 전체구조

<Figure 10>에 MLE 파일의 전체 구조가 나와있다. 그림에 앞에서 설명한 주요 3테이블의 위치가 나와있다. Layer Node Table(LNTAB)은 익스텐트 3에서 12까지이고, User Node Table(UNTAB)은 익스텐트 14에서 23, Encrypted Session Key Node Table(ESKNTAB)은 25에서 34까지의 익스텐트를 차지하고 있다. 각 테이블 마다 그 테이블의 어떤 엔트리가 비어있고 어떤 엔트리가 사용되고 있는지 정보를 보여주는 비트맵이 있다. LNBM은 LNTAB의 비트맵으로서 익스텐트 2에 있다. UNTAB의 비트맵인 UNBM은 익스텐트 13에 있다. ESKNTAB의 비트맵인 ESKNBM은 익스텐트 24에 있다. 익스텐트 1에도 EBM이라고 불리우는 비트맵이 있다. 이 비트맵은 현 MLE 파일의 각 익스텐트의 사용/비사용 여부를 보여준다. 마지막으로 익스텐트 0는 각 테이블이나 비트맵의 위치를 저장한 수퍼블록이다. MLE 파일이 처음 생성되면 위에서 설명한 메타 데이터들이 먼저 생성된다. 그리고 루트 레이어를 위해 2개의 익스텐트가 추가로 할당된다. <Figure 10>에 보면 익스텐트 35가 루트 데이터 익스텐트로 할당되고, 익스텐트 36이 루트 디렉토리익 스텐트로 할당되어 있는 것을 볼 수 있다. 이후 생성되는 다른 레이어에 대해서도 시스템은 초기 상태로 두 개의 익스텐트를 할당하여 하나는 데이터를 담도록 하고 다른 하나는 이 레이어 밑으로 생성될 하위 레이어들의 이름과



〈Figure 10〉 Overall Structure of an MLE File

LNODE번호를 담도록 하고 있다. 데이터 익스텐트는 데이터가 증가함에 따라 추가로 할당된다. 수퍼블록의 구조는 아래와 같다.

```
class Superblock{
  // on-disk data
  intfsize;
  char  marker[8];
  char  version;
  char  reserve[3];
  unsigned long  loc_ebm;
  unsigned long  loc_lnbm;
  unsigned long  loc_lntab;
  unsigned long  loc_unbm;
  unsigned long  loc_untab;
  unsigned long  loc_esknbm;
  unsigned long  loc_eskntab;
  // on-memory data follows
  ..........
}
```

4. MLE 파일 API 함수

MLE 파일 시스템은 사용자 프로그램이 MLE 파일을 열고 읽거나 쓸 수 있도록 라이브러리 함수들을 정의한다. 관리자 프로그램을 위해서는 MLE 파일을 생성하거나 그 안에 레이어를 생성하고 사용자를 등록하는 함수를 정의한다. 이를 위해 7가지 주요 함수를 정의하는데 이들은 MLE_fcreate, MLE_lcreate, MLE_regiter_user, MLE_fopen, MLE_read, MLE_write, MLE_close 등이다. 앞의 세 함수 (MLE_fcreate, MLE_lcreate, MLE_regiter_user)는 관리자 프로그램만 사용할 수 있는 함수이다. MLE_fcreate()이 MLE 파일을 생성하는 함수이다. 이 함수는 오퍼레이팅 시스템에게 요청하여 파일을 만들고 그 안에 MLE 파일 헤더와 루트 레이어를 초기화한다. 수퍼블록은 각 비트맵과 테이블의 위치값을 갖게 되고 EBM은 비트 0에서 36까지 1로 세팅된다(익스텐트 0에서 34까지는 메타 데이터이고 익스텐트 35, 36은 루트 익스텐트이다). LNTAB은 루트 레이어에 해당하는 LNODE 0 하나만을 가진 상태로 초기화된다. LNBM은 비트 0에 1로 세팅되어 이 상태를 반영한다. LNODE 0에는 루트 레이어의 익스텐트 위치 정보가 들어가게 된다. UNTAB 역시 UNODE 0 하나만을 가지게 되며 사용자 “루트”에 대한 정보로 초기화된다. 사용자 “루트”는 특수 사용자로서 객체 트리를 초기화하는 데에만 사용된다. MLE_fcreate은 MLE_register_user(“/”, “root”, root_pub_key)를 호출함으로서 “루트” 사용자를 등록한다. UNBM의 비트 0도 이를 반영

하여 세팅된다. 마지막으로 ESKNTAB은 두 개의 엔트리를 갖게 된다. ESKNODE 0는 루트 레이어의 세션 키를 MLE 서버의 마스터 공개키로 암호화한 값을 가지고 있다. ESKNODE 1은 같은 키를 루트의 공개키로 암호화한 값을 가지고 있다. 현 구현에서는 루트의 공개키는 서버의 마스터 공개키와 같은 값이다. ESKNBM은 비트 0와 1이 세팅되어 두 개의 엔트리가 사용되고 있음을 반영한다. LNODE 0의 *esk* 값은 0으로세팅되어 이 레이어의 세션 키가 ESKNODE 0에 들어있음을 나타내게 된다. UNODE 0는 esk[0] = 1로 세팅되어 이 사용자가 접근할 수 있는 첫 레이어의 세션 키를 ESKNODE 1에서 찾을 수 있음을 나타내게 된다.

MLE_lcreate()은 레이어를 추가하는 함수이다. 이 함수는 아래 순서로 진행된다.

1) 두 개의 익스텐트 할당
2) 임의의 세션 키 생성, 마스터 공개키로 암호화, ESKNTAB에 저장
3) 이 세션 키에 대해 레이어 트리(객체 트리)상의 모든 상위 레이어 중 그 레이어를 홈 레이어로 가지고 있는 UNODE에게 통지
4) LNODE를 할당하고 위에서 할당한 두 익스텐트의 위치와 세션 키가 ESKNTAB에 저장된 위치를 기록
5) 이 레이어를 부모 레이어의 디렉토리익스텐트에 자식 레이어로 기록

위에서 스텝 3)은 홈 레이어를 할당받은 사용자가 홈 레이어와 그 밑에 생성되는 모든 하위 레이어에 대해 접근 권한을 가질 수 있도록 하기 위해서이다. 이를 위해 사용자는 모든 접근 가능한 레이어의 세션 키를 알고 있어야 하고 따라서 레이어가 생성될 때마다 상위 레이어에 있는 사용자들의 UNODE에 세션 키 정보를 통보하는 것이다.

MLE_register_user()는 사용자를 MLE 파일에 등록하기 위해 사용된다. 이 함수는 세 개의 입력 값을 받는다. 예를 들어 MLE_register_user("/x/y", "user_x", pubkey_x). 첫 인자는 이 사용자의 홈 레이어 위치를 나타내고 두 번째 인자는 이 사용자의 이름, 마지막 인자는 이 사용자의 공개키이다. 관리자가 어떤 사용자를 등록하기 위해서는 사전에 그 사용자의 공개키를 사용자로부터 받아야 한다. 이 함수는 아래와 같은 순서로 진행된다.

1) UNODE 할당
2) 사용자 이름과 홈 레이어 위치 기록
3) 홈 레이어 및 그 아래 모든 하위 레이어의 세션 키를 이 사용자의 공개키로 암호화하고 그것들을 ESKNTAB에 저장한 후 그 위치를 esk[] 배열에 기록

MLE_fopen()은 기존의 MLE 파일을 오픈한다. 이 함수는 지정된 MLE 파일을 오픈하고 모든 메타데이터를 읽어서 아래의 MLE_fs 구조체를 생성한다.

```
classMLE_fs{
    Superblock *sb;
    EBM *ebm;
    Lnbm *lnbm;
    LayerNodeTable *lntab;
    Unbm *unbm;
```

```
    UserNodeTable *untab;
    Esknbm *esknbm;
    ESKNodeTable *eskntab;
    Extents *extents;
    .....
};
```

MLE_fs는 해당 MLE 파일의 모든 메타 정보와 부모-자식 레이어 관계, 각 사용자의 데이터 익스텐트의 물리적 위치 등 실행시간에 필요한 추가 정보를 가지고 있다. MLE 파일을 접근하기 위해서는 MLE_fcreate나 MLE_fopen()을 호출하여 MLE_fs 구조체를 먼저 생성해야 한다. MLE_write()은 데이터를 MLE 파일에 기록하는 함수이다. 현 사용자가 데이터를 쓰려는 익스텐트는 이 사용자가 접근 권한이 있는 레이어이어야 한다. 데이터는 해당 레이어의 세션 키로 암호화되어 기록된다. MLE_read()은 데이터를 읽기 위한 함수이다. 마찬가지로 해당 레이어에 대해 현 사용자가 접근 권한이 있는 경우에만 허용된다. MLE 서버는 사용자가 요청한 레이어의 데이터를 암호화된 상태로 해당 레이어의 세션 키(이 사용자의 공개키로 암호화된)와 함께 사용자에게 보낸다. 사용자 프로그램은 이 세션 키를 복호화하여 그 키로 데이터를 다시 복호화 할 수 있다.

5. 활용 예 및 성능 평가

MLE 파일 포맷은 공간비용 면에서 레이어 트리구조와 사용자 정보 그리고 키 정보를 관리하기 위한 헤더 부분을 추가적으로 요구한다. 시간비용 면에서는 세션 키의 암호화/복호화, 데이터의 암호화/복호화시간을 추가적으로 요구한다. 따라서 MLE 파일 포맷은 같은 데이터를 일반 파일에 저장했을 경우보다 공간 사용량이 늘어나고 파일 처리 시간이 증가한다. 하지만 여러 사용자가 동시에 안전하게 협업할 수 있는 MLE 파일 시스템의 이점에 비해서 결코 지나친 증가량이 아니라고 본 연구팀은 판단한다. 본 장은 실제 MLE 파일 시스템을 구현하고 이 시스템에 대한 분석 및 실험을 통해 이러한 본 연구팀의 판단에 대한 근거를 제시한다.

5.1 MLE 서버 및 MLE 클라이언트

우선 MLE 파일 포맷을 사용하여 여러 사용자가 같은 파일에서 동시 작업을 하는 상황을 시뮬레이트 하기 위해 단일 MLE 서버와 다중 MLE 클라이언트로 이루어지는 시스템을 구현하였다. MLE 서버와 클라이언트는 SSL(Secure Socket Layer)프로토콜을 통해 암호화된 통신을 한다. MLE 서버는 수많은 클라이언트를 동시에 서비스하기 위해 select() 시스템 콜을 사용한다. Fork()를 사용하는 것도 가능하지만 fork()는 시스템 자원을 많이 사용하므로 클라이언트의 수가 증가함에 따라 속도 저하 문제가 있고 또한 공유메모리 사용이 자유롭지 못하므로 클라이언트 간에 MLE 파일의 변경을 서로 신속히 주고받는 데 시간 지연을 초래할 수 있다. MLE 서버의 주 알고리즘은 다음과 같다.

```
open an SSL port;
for(;;){
```

```
    wait for client packets on select();
    process client packets;
}
```

SSL 포트는 TCP port이다. 포트를 오픈한 후 서버는 select()를 호출해서 복수개의 클라이언트로부터 패킷을 기다린다.

select(……, andrset, …);

위에서 *rset*는 모니터할 소켓 번호에 1이 세팅된 비트 벡터이다. *rset*는 초기에는 SSL 소켓 번호만 세팅되어 있지만 클라이언트로부터 연결 요청이 들어올 때마다 해당 클라이언트와의 통신을 위한 소켓이 새로 생성되고 이 통신 소켓들이 rset에 계속 세팅되므로 여러 클라이언트가 작업을 하고 있는 상황에서는 SSL 소켓과 현재 접속한 클라이언트 수만큼의 소켓번호에 대해 1로 세팅되어 있게 된다. 클라이언트로부터 패킷이 들어오면 서버는 *process_client_packet()*을 호출한다. 클라이언트 패킷은 3가지 중에 하나이다. 즉 SSL 연결 요청이든지, SSL 프로토콜에 따른 클라이언트 HELLO이든지, 아니면 SSL 연결이 된 후에 클라이언트가 보낸 MLE 파일 명령어이다.

```
process_client_packet(){
    if SSL connection request packet
        call accept() to establish a connection;
    else if client HELLO
        callS SL_accept() to process SSL
        handshake protocol;
    else if an MLE client command
        process MLE command;
}
```

만일 클라이언트 패킷이 SSL 연결 요청이면, 새로운 클라이언트가 이 서버에 접속했다는 것을 의미한다. 서버는 이 클라이언트와의 통신을 위한 소켓을 따라 만들고 이 소켓 번호를 *rset* 비트 벡터에 표시한다. 다음 라운드에서의 *select()*는 이렇게 새로 생성된 소켓들까지 포함하여 모니터링을 하게 될 것이다. 클라이언트 패킷이 SSL 클라이언트 HELLO이면 이 패킷은 해당 클라이언트와의 통신을 위해 할당된 소켓을 통해 들어온 패킷일 것이며 클라이언트가 SSL 프로토콜을 시작했다는 의미이다. 서버는 *SSL_accept()*를 호출하여 클라이언트와 SSL 프로토콜을 진행한다. SSL 프로토콜이 성공적으로 진행되었으면 이제 서버와 클라이언트는 해당 통신 소켓을 통해 암호화된 통신(SSL 채널)을 할 수 있게 된다. 이 채널을 통해 클라이언트가 MLE 파일 명령어를 보내는 경우가 마지막 경우이다. 클라이언트가 보내는 MLE 파일 명령어는 MLEFS_FCREATE, MLEFS_LCREATE, MLEFS_REGISTER_USER, MLEFS _FOPEN, MLEFS_READ, MLEFS_WRITE, MLEFS_ CLOSE 등 7가지이다. 서버는 각각의 명령어에 대해 제 4장에 설명한 해당 MLE API 함수를 호출한다.

```
process_mle_command(){
    if MLEFS_FCREATE command
        callprocess_mlefs_fcreate() to create
        an empty mle file;
    else if MLEFS_LCREATE command
        callprocess_mlefs_lcreate() to add a
        layer to current mle file;
    else if MLEFS_REGISTER_USER
```

```
    command
      call process_mlefs_register_user() to
      add a user to current mle file;
    else if MLEFS_FOPEN command
      call process_mlefs_fopen() to open
      an mle file;
    else if MLEFS_READ command
      call process_mlefs_read() to read
      data from current mle file;
    else if MLEFS_WRITE command
      call process_mlefs_write() to write
      data into current mle file;
    else if MLEFS_CLOSE command
      call process_mlefs_close() to close
      current mle file;
  }
```

클라이언트 쪽 코드는 서버에 비해 훨씬 간단하다. 클라이언트 역시 서버가 오픈한 SSL 포트와 같은 포토를 열고 *connect()*를 호출하여 서버에 연결 요청을 보낸다. 서버에 접속되면 *SSL_connect()*,를 호출하여 클라이언트 HELLO 패킷을 보냄으로써 SSL 프로토콜을 진행한다. SSL 프로토콜이 성공적으로 진행되면 확립된 SSL 채널을 통해 MLE 명령어 패킷을 보내기 시작한다. MLE 클라이언트의 주 알고리즘은 아래와 같다.

```
  open an SSL port;
  connect it to the MLE server;
  callSSL_connect() to establish an SSL
  channel with the server;
  for(;;){
    readclient command;
    process client command;
  }
```

*process_client_command()*는 서버의 *process_mle_command()*와 유사하게 7개의 명령어에 대해 해당 함수를 호출한다.

```
  process_cli_command(){
    if MLEFS_FCREATE command
      call process_fcreate_command() to
      send MLEFS_FCREATE command;
    else if MLEFS_LCREATE command
      call process_lcreate_cmd() to send
      MLEFS_LCREATE command;
    else if MLEFS_REGISTER_USER
    command
      call process_register_user_cmd() to
      send MLEFS_REGISTER_USER
      command;
    else if MLEFS_FOPEN command
      call process_fopen_cmd() to send
      MLEFS_FOPEN command;
    else if MLEFS_READ command
      call process_read_cmd() to send
      MLEFS_READ command and
      receive data;
    else if MLEFS_WRITE command
      call process_write_cmd() to send
      data to the server with
      MLEFS_WRITE command;
    else if MLEFS_CLOSE command
      call process_close_cmd() to send
      MLEFS_CLOSE command;
  }
```

각 함수는 해당 명령어를 담은 커맨드 패킷

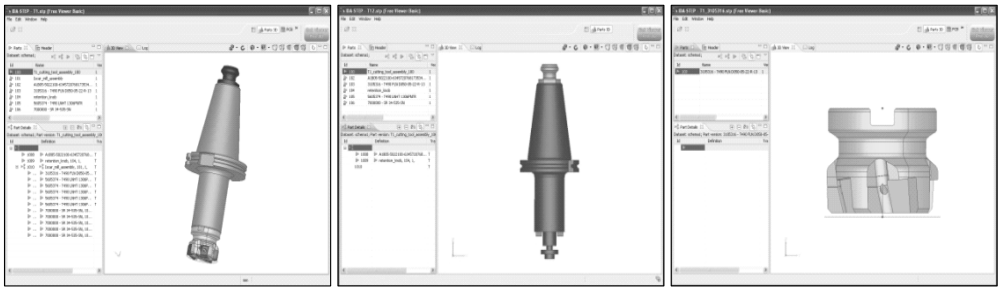
을 만들어 서버에게 전송한다. 하지만 *processread_cmd()*나 *process_write_cmd()*의 경우는 전자는 서버로부터 암호화된 데이터를 받아 복호화하는 과정을 포함한다. 복호화하는데 필요한 키는 역시 같이 전송된 암호화된 세션 키를 자신의 개인키로 복호화하여구한다. 후자의 경우는 사용자가 지정한 파일의 데이타를 서버에게 전송한다.

5.2 MLE 파일의 활용 예

본 논문에서 제안한 MLE 파일 포맷의 활용 예를 STEP 파일을 이용하여 설명한다. STEP은 제품 데이터 즉 제품의 생산에 관계되는 데이터를 컴퓨터가 인식 가능한 형태로 완벽하게 표현하고 교환하기 위한 국제 표준이며, 그 목적은 제품의 전 생명 주기(Life Cycle)에 걸쳐서 제품 데이터를 표현할 수 있는 중립적인 메커니즘을 제공하는 것이다[13]. STEP은 표준 자체의 구조를 반영하여 주는 파트들의 여러 그룹으로 나뉘어져 있다. 이러한 그룹에는 응용 프로토콜(application protocols, AP), 통합자원(integrated resources), 서술 방법(description methods), 구현 방법(implementation methods), 적합성 시험 방법 및 골격(conformance testing methodology and framework), 그리고 가상 시험 세트(abstract test suites)가 있다.

<Figure 11(a)>는 밀링 툴 어셈블리의 설계도면으로서 STEP 응용 프로토콜 AP203 파일을[14] 뷰어[12]를 이용하여 도시한 것이다. AP203은 구성 관리, 제품 구조, 규격, 기하학적 형상 등으로 구분 할 수 있다. 구성 관리는 제품 데이터 승인, 제품 개발을 진행하기 위한 작업 지침, 제품 모델의 구성 부품의 계획된 사용에 관한 정보, 부품이나 부품의 설계를 생성하는 조직에 관한 정보 등을 포함한다. 제품 구조는 BOM 데이터, 조립품, 부품 등에 관한 정보 등을 나타내며, 규격은 부품이나 제품의 품질과 관련된 정의, 공정 및 규칙 등을 포함하는 문서이다. 기하학적 형상은 고등경계 표현, 각진 경계 표현, 다양체 곡면, 위상 정보가 있는 와이어 프레임, 그리고 위상 정보가 없는 곡면 및 와이어프레임 등 6개의 형상을 포함한다.

MLE 파일을 이용한 협업설계의 예로서, A 회사의 개발실에서 진행하는 밀링 툴 어셈블리 <Figure 11(a)>의 설계 과정을 설명한



(a) (b) (c)

〈Figure11〉 (a)Milling Tool Assembly(b) Holder(c) Tool

다. 밀링 툴 어셈블리는 홀더 <Figure 11(b)>와 툴 <Figure 11(c)>로 구성되며 각각 협력 업체 B와 C의 부품을 이용한다. 협력 업체 B와 C는 납품을 위해 A회사에 설계 데이터를 제공하지만, 설계의 보안을 위해 다른 협력 업체에 공개되는 것을 원하지 않는다. MLE 파일을 이용한 협업 설계과정은 다음과 같다.

1) A회사에서 새로운 MLE 파일 d.mle를 생성하고 루트 디렉토리(/)의 마스터 공개키/개인키를 설정한다.
2) A회사에서는 d.mle에 새로운 레이어/b와/c를 생성하여 각각 협력 업체 B와 C의 공개키를 이용하여 이들에게 사용 권한을 준다.
3) 협력 업체 B와 C는 d.mle의 레이어/b와/c에 홀더와툴의 설계 데이터를 입력한다.
4) A회사에서는 마스터 키를 이용하여 입력된 홀더와 툴의 정합성을 공차 등을 고려하여 확인하고, 필요하면 설계의 수정을 협력 업체에 요청한다.

이와 같이 MLE 파일은 계층적으로 이루어진 제품 구조상에서 접근 권한을 서로 다른 사용자에게 제공하여 제품 정보의 보안을 유지하면서 협업 설계를 가능하게 한다. 레이어 별로 다른 접근 권한을 부여함으로써 협업 설계에 사용되는 설계 데이터의 적절한 보안을 가능하게 한다. 이러한 기능은 협업 설계에 포함되는 사용자가 많아지고 제품의 구조가 복잡할 경우 더욱 유용하게 활용될 수 있다.

5.3 공간 및 시간비용에 대한 평가

앞에서도 언급했듯이 MLE 파일이 같은 데이터를 포함하는 일반 파일에 비해 공간 면에서 추가로 요구하는 것은 주로 MLE 파일 헤더 부분이다. 그리고 각 레이어마다 하부 레이어노드 번호들이 기록된 블록(익스텐트)이 추가로 요구된다. MLE 파일 헤더는 파일 사이즈가 커지더라도 혹은 레이어개수가 늘어나도 고정된 비용이고 하부 레이어 정보 저장 블록은 레이어개수에 비례하여 증가하는 비용이다. 레이어개수가 n이라면

$$\text{추가 공간 비용} = n \times 4k + 148K$$

로 표시된다. 블록 하나의 크기가 4k이며 MLE 파일 헤더에 37개의 블록이 필요하므로 (<Figure 10> 참조) MLE 파일 헤더에 37 × 4k = 148k바이트가 필요하고 각 레이어마다 4k 블록이 추가적으로 필요하다. 디자인 파일은 보통 사이즈가 매우 크기 때문에 고정값인 148k는 큰 오버헤드라고 볼 수는 없다. 레이어 개수가 늘어나면서 증가하는 n × 4k 부분도 선형적으로 증가하므로 MLE 파일이 제공하는 서비스를 얻기 위해 충분히 감당할 수 있는 비용이라고 판단된다.

시간비용 면에서 MLE 파일이 일반 파일에 비해 추가로 요구하는 것은 세션 키와 데이터에 대한 암호화/복호화 비용이다. 세션 키에 대한 암호화/복호화 비용은 사용자의 홈레이어의 위치(홈레이어에 대해서는 3.2 참조)와 관계가 있고 데이터에 대한 암호화/복호화 비용은 사용자가 읽거나 쓰려는 레이어의 데이터 크기와 관계가 있다. 주어진 사용

자의 홈레이어의 위치가 그 밑으로 n개의 하위 레이어를 가지고 있고 이 사용자가 크기가 m × 256바이트인 레이어에 대해 데이터를 쓰거나 읽는 작업을 하는 경우 MLE 파일이 일반 파일에 대해 추가적으로 발생시키는 시간비용은

$$\text{추가 시간 비용} = n \times T_{skey} + m \times T_{data} \times x$$

로 표현된다. 여기에서 T_{skey}는 세션키 하나를 암호화/복호화 하는데 걸리는 시간이고 T_{data}는 256바이트를 암호화/복호화 하는데 걸리는 시간이며 x는 이 사용자가 이 레이어에 대해 읽거나 쓰기를 반복한 횟수이다. 본 연구에서 서버와 클라이언트는 모두 2.4GHz 인텔 CPU이고 세션 키는 2048비트 RSA 공개키 알고리즘으로 암호화/복호화하였으며 데이터는 DES 알고리즘으로 암호화/복호화하였다. 세션 키의 사이즈는 8바이트이며 데이터는 256바이트 단위로 잘라서 암호화/복호화하였다. 본 연구에서 구현한 MLE 서버/클라이언트 시스템에서 측정한 값은

$$T_{skey} = 0.014327$$
$$T_{data} = 0.000011$$

이다. 세션 키의 암호화/복호화가 공개키 알고리즘을 사용하므로 데이터 암호화/복호화에 비해 약 1000배 정도 시간을 더 소요한다는 것을 알 수 있다. 하지만 추가 시간 비용에서 세션 키의 암호화/복호화가 차지하는 부분($n \times T_{skey}$)은 사용자가 처음 이 MLE 파일에 등록될 때 발생되는 비용으로서 한번만 발생되는 고정 비용이다. 일단 등록이 된 후는 뒤 부분의 비용($m \times T_{data} \times x$)만 일반 파일에 비해 MLE 파일이 추가로 요구하는 시간 비용이다. 이 비용은 데이터의 크기에 비례해서 선형적으로 증가하기 때문에 그리고 T_{data}의 값이 매우 적기 때문에 MLE 파일이 제공하는 서비스의 이점을 상쇄할 만큼 크지는 않다고 판단된다.

6. 결 론

수많은 디자이너가 단일 디자인 프로젝트에서 서로 협력하며 디자인 작업을 수행하는 협업 디자인의 경우 주요 문제 중의 하나는 정보 보호이다. 즉 각 디자이너가 작업을 수행하기 위한 최소한의 범위 안에서만 다른 디자이너의 정보를 접근할 수 있어야 한다는 것이다. 디자인 작업이 단일 디자인 파일에 대해 여러 디자이너가 작업을 수행하는 경우라면 이 파일을 개별 디자인 객체의 경계를 따라 여러 파티션으로 나누고 각 디자이너가 자신에게 허용된 파티션 안에서만 작업하도록 하는 조치가 필요할 것이다. 본 논문은 이 객체간의 경계를 디자인 레이어트리로 표현하고 각 레이어를 서로 다른 키로 암호화하며 이러한 레이어들의 위치 정보와 레이어를 암호화하는 데 사용된 키에 관한 정보를 파일 헤더에 저장하는 MLE 파일 포맷을 제안한다. 본 논문이 제안하는 파일 포맷은 각 디자이너가 자신에게 허용된 레이어들에 대해서만 작업을 할 수 있도록 강제함으로써 다중 사용자가 안전하게 같은 디자인 파일에 대해 협업할 수 있도록 하는 환경을 제공할 것이다.

References

[1] Ausanka-Cures, R., "Methods for Access Control : Advances and Limitations," http://www.cs.hmc.edu/~mike/public_html/courses/security/s06/projects/ryan.pdf

[2] Baek, S. I., Shin, J. Y., Kim, J. W., "Exploring the Korean Government Policies for Cloud Computing Service," The Journal of Society for e-Business Studies, Vol. 18, No. 3, pp. 1-15, 2013.

[3] Bertram, S., Boniface, M. Surridge, M., Briscombe, N., Hall-May, M., "On-Demand Dynamic Security for Risk-Based secure Collaboration in Clouds," In the proceedings of Could Computing, 2010 IEEE 3rd International Conference, 2010.

[4] Cera, C. D., Kim, T., Han, J. H., Regli, W. C., "Role-based viewing envelopes for information protection in collaborative modeling," Computer-Aided Design Vol. 36, pp. 873-996, 2004.

[5] Chadwick, D. W., Otenko, O. A., "The PERMIS X. 509 role based privilege management infrastructure," Future Gener. Compt. Syst. Vol. 19, No. 2, pp. 277-248, 2003.

[6] Chen, T. Y., Chen, Y. M., Chu, H. C. "Developing a trust evaluation method between co-workders in virtual project team for enabling resource sharing and collaboration," Computers in Industry, Vol. 59, No. 6, pp. 565-579, 2008.

[7] Folk, M., Heber, G., Koziol, Q., Pourmal, E., Robinson, D., "An overview of the HDF5 technology suite and its applications," Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases, pp. 36-47, 2011.

[8] Halcrow, M. A., "eCryptfs : An Enterprise-class Cryptographic Filesystem for Linux," http://ecryptfs.sourceforge.net/ecryptfs.pdf.

[9] Hauck, S., Knol, S., "Data security for Web-based CAD," Proceedings of the 35th annual Design Automation Conference, pp. 788-793, 1998.

[10] Kim, D. H., Lee, J. H., Park, Y., P., "A Study of Factors Affecting the Adoption of Cloud Computing," The Journal of Society for e-Business Studies, Vol. 17, No. 1, pp. 111-136, February 2012.

[11] Kuhn, D. R., Coyne, E. J., Weil, T. R., "Adding Attributes to Role-Based Access Control," Computer Vol. 43, No. 6, pp. 79-81, June 2010.

[12] LK Soft Ware GmbH, www.lksoft.com

[13] SCRA, STEP Application Handbook ISO 10303, North Charleston, SC, 30 2006, available at http://www.uspro.org/documents/STEP_application_hdbk_63006_BF.pdf.

[14] STEP Tools, Inc., http://www.steptools.com/demos/.

[15] Wang, G., Liu, Q., Wu, M., Guo, M., "Hierarchical attribute-based encryption and scalable user revocation for sharing

data in cloud servers," Compters and Security Vol. 30, pp. 320-331, 2011.

[16] Wang, Y., Ajoku, P. N., Brustoloni, J. C., Nnaji, B. O., "Intellectual Property Protection in Collaborative Design through Learn Information Modeling and Sharing," Journal of Computing and Information Science in Engineering, Vol. 6, pp. 149-159, 2006.

[17] Wong, A., Bishop, W., "intelligent multi-level regions-of-interest(ROI) document image encryption using an online learning model," proceedings of the Fourth conference on IASTED International conference, 2007.

[18] Yuan, Chun., Zhu, Bin. B., Su, Ming., Wang, Xiaoming., Li, Shipeng., Zhong, Yuzhuo. "Layered Access Control for MPEG-4FGS Video," Proceedings of Image Processing, 2003.

[19] Zeng, Y., Wang, L., Deng, X., Cao, X., Khundker, N., "Secure collaboration in global design and supply chain environment : Problem analysis and literature review," Computers in Industry, Vol. 63, No. 6, Auguset pp. 545-556, 2012.

[20] http://www.nsa.gov/ia/_files/support/Cloud_Computing_Guidance.pdf.

저 자 소 개

김기창 (E-mail : kchang@inha.ac.kr)
1986년 California State Polytechniqe University, B.S. in Computer Science
1988년 University of California at Irvine, M.S. in Computer Science
1992년 University of Californai at Irvine, Ph.D. in Computer Science
1994년~현재 인하대학교 정보통신공학부 교수
관심분야 정보보안, 운영체제, 컴파일러

유상봉 (E-mail : syoo@inha.ac.kr)
1982년 서울대학교제어계측공학과 (학사)
1986년 University of Arizona, M.S. in Electrical and Computer Engineering
1990년 Purdue University, Ph.D. in Electrical and Computer Engineering
1992년~현재 인하대학교 컴퓨터정보공학부 교수
관심분야 데이터베이스, 시스템통합, 정보보안