YAFFS 기반의 암호화 플래시 파일 시스템의 설계 및 구현

김석현* · 조유근*

요 약

임베디드 기기에서 플래시 메모리의 사용량이 증가하고, 임베디드 기기가 여러 computing 환경에서 점점 중요한 위치를 점함에 따라 임베디드 파일 시스템의 보안이 중요한 문제가 된다. 또한 임베디드 기기의 경우 휴대성이 좋은 반면 분실의 위험도 크고, 분실한 경우 기기 내부의 플래시 메모리에 중요 정보가 있다면 사용자에게 큰 손실을 야기할 수 있다. 이처럼 다양한 상황에서 임베디드 기기 내부의 플래시 파일 시스템의 보안성을 향상시키기 위해 암호화 플래시 파일 시스템을 설계 및 구현 하였다. 이를 위해 현재 많이 사용되는 YAFFS 파일 시스템을 수정하였다. 수정된 YAFFS 암호화 파일 시스템을 통해 임베디드 기기의 보안성을 한 층 강화할 수 있다.

Design and Implementation of Flash Cryptographic File System Based on YAFFS

Seok Hyun Kim* · Yoo Kun Cho*

ABSTRACT

As the amount of flash memory being used in embedded device is increased and embedded devices become more important in many computing environments, embedded file system security becomes more important issue. Moreover embedded devices can be easily stolen or lost because of it's high portability. If the lost embedded device has very important information, there's no means to protect it except data encryption. For improving embedded devices' security this paper propose design and implementation of flash cryptographic file system. For this purpose YAFFS is used. By the modified YAFFS cryptographic file system, the security of embedded devices can be improved.

Key words : Flash File System, Cryptographic File System

* 서울대학교 컴퓨터공학과

1. 서 론

불법적인 해킹에 의한 컴퓨터 사용자의 피해는 나날이 증가하고 있다. 미국 전역에서 지난 2년간 해킹에 의한 피해는 총 70억 달러에 이른 것으로 집계 되었다[1]. 대부분의 PC에서 사용하는 윈도우 운영체제의 경우 수많은 보안상의 허점이 발견되어 왔고 지금도 발견되고 있다. 최근에는 OS 설계 시점부터 보안을 핵심적인 원칙으로 많이 고려하지만 과거에는 그렇지 못했다. 그래서 현재 사용하는 시스템들은 많은 보안상의 허점을 가지고 있는 상황이다.

컴퓨터 보안은 특정한 하나의 시스템으로 완성되지 않는다. 시스템 전체의 여러 취약점들에 맞게 각각의 부분을 보완해 나감으로써 전체적인 시스템의 보안성이 증대된다.

암호화 파일 시스템은 전체적인 시스템 보안을 향상시키는 중요한 기초 요소로서 활용될 수 있다. 예를 들어 암호화 파일 시스템과 DRM이 결합하면 클라이언트에서 보다 강력한 컨텐츠 보호를 제공할 수 있다. 또한 암호화 파일 시스템을 이용하여 특정 파일에 그 파일만을 위한 암호화 키를 설정하면, 설사 계정이 뚫린 경우에도 마지막으로 해당 파일에 대한 보호 메커니즘을 제공할 수 있다.

점점 플래시 메모리가 많이 사용되면서 이제 보안을 위해서 플래시 메모리를 위한 암호화 파일 시스템이 필요한 시점이다. 여러 임베디드 기기는 하드 디스크가 아닌 플래시 메모리가 주로 사용되고 있다. 점점 성능이 향상되는 임베디드 기기는 이미 네트워크에 접속하고 있으며, 앞으로는 임베디드 기기의 보안이 PC와 같이 큰 문제가 될 것이다.

또한 플래시 메모리가 주로 사용되는 기기나, 메모리 스틱의 경우 휴대가 간편하기 때문에 물리적인 분실의 위험성이 PC에 비해서 매우 높다. 이런 경우 암호화외에 중요 파일의 정보를 보호할 수 있는 별다른 방법은 없다고 보아야 한다. 따라서 플래시 파일 시스템을 위한 암호화 파일 시스템은 플래시 메모리 내부의 중요 정보를 물리적인 도난에서도 지켜주는 역할을 할 수 있다.

이러한 문제를 해결하기 위해 이 논문에서는 YAFFS(Yet Another Flash File System)를 기반으로 플래시 암호화 파일 시스템을 구축 하였다. 제 2장에서 YAFFS 파일 시스템에 대해서 살펴본다. 제 3장에서는 기존의 암호화 파일 시스템 관련 기술에 대해 기술한다. 제 4장에서는 암호화 파일 시스템의 설계 및 구현을 제시하고 제 5장에서 결론을 내린다.

2. YAFFS 파일 시스템

2.1 플래시의 특성과 YAFFS

YAFFS 파일 시스템은 NAND 플래시의 특성을 고려하여 설계된 플래시 전용 파일 시스템이다. YAFFS와 같은 플래시 전용 파일 시스템이 나오기 전에는 FTL(Flash Translation Layer)위에 기존의 파일 시스템을 구축 하였다. FTL이 기존의 플래시 위에 마치 기존의 디스크와 같은 page 변환 계층을 제공함으로써 FAT 등의 기존 파일 시스템을 플래시에 사용할 수 있게 한다. 그러나 FTL과 같은 방식은 플래시 메모리의 특성을 고려하지 않으므로 YAFFS와 같은 플래시 전용 파일 시스템에 비해서 좋지 않은 성능을 보인다.

플래시 메모리는 block과 page로 구성된다. 전체 플래시는 block으로 구분되고 block 안에 같은 수의 page들이 들어 있다. 읽기 작업의 경우 random access가 가능하다. 그러나 쓰기 작업의 경우 overwrite가 불가능하다. 이는 특정 block에서 쓰기 연산을 수행하면 이미 쓴 내용은 바꿀 수 없고 block을 모두 채울 때 까지 연속적인 쓰기만 가능함을 의미한다. 이미 쓴 내용을 고치기 위해서는 지우기 연산을 수행한 후 다시 쓰기 연산을 수행해야 한다. 또한 읽기, 쓰기는 page 단위로 가능하지만 지우기는 block 단위로만 가능하다. YAFFS는 NAND의 이런 특성을 고려하여 좋은 성능을

보여준다.

2.2 YAFFS 파일 시스템 구조

NAND의 각 페이지에는 파일 시스템의 data 또는 metadata가 들어있다. 처음 파일 시스템이 mount될 때 YAFFS는 NAND 상의 모든 metadata를 읽어 들여서 메모리 상에 파일 시스템의 디렉토리 구조를 만들어 놓는다. 그리고 이 구조를 통하여 NAND를 조작함으로써 파일 시스템 성능을 향상시킨다.

NAND 플래시에서 특정 파일에 대한 수정 작업이 일어나면 수정된 data가 플래시에 쓰여진다. 이전 data가 들어있는 페이지는 이후 garbage collection이 수행될 때 수거된다. 파일이 수정 되었으므로 파일에 대한 metadata 역시 수정된다. 플래시는 overwrite가 불가능하기 때문에 metadata에서 바뀐 부분만 덮어 쓰는 것은 불가능하다. 따라서 새로운 페이지에 바뀐 부분을 수정하여 전체 metadata를 다시 써넣는다. 각 metadata는 일종의 일련번호를 가지고 있다. 그리고 새롭게 metadata가 쓰여질 때 이 번호를 하나 증가시키게 된다. 따라서 특정 data에 대한 여러 metadata중 이 일련번호를 통해 최근의 metadata를 알 수 있게 된다[2].

새로운 페이지가 할당될 때에는 data, metadata 구분 없이 플래시의 앞쪽에서 뒤쪽으로 가면서 새 공간이 할당되게 된다. 새로운 공간이 부족하면 가장 invalid page가 적은 block을 찾아서 garbage collection을 수행한다. valid page가 많으면 이를 다른 block으로 복사해야 하므로 overhead가 커지기 때문이다.

3. 암호화 파일 시스템 관련 기술

3.1 암호화 파일 시스템

파일 시스템이 암호화 및 키 관리 기법을 제공하는 암호화 파일 시스템에 대해 여러 연구가 있어 왔다. Cryptfs[3]와 Ncryptfs[4]는 stackable file system으로써 VFS와 사용자 응용 사이에 스택처럼 존재하는 암호화 파일 시스템이다. Cryptfs는 stackable file system의 개념을 중심으로 구현되었고 Ncryptfs는 이를 발전시켜 본격적인 암호화 파일 시스템으로 구현 되었다. Ncryptfs는 다양한 암호화 알고리즘을 제공하며 여러 사용자를 동시에 인증하는 등의 기능성과 편의성을 보여준다. 또한 CFS[5], TCFS[6] 같은 사용자 계층 암호화 파일 시스템에 비해 커널 영역에서 동작하므로 많은 성능 향상을 보인다.

CFS는 사용자 계층에서 동작한다. 따라서 잦은 데이터의 복사가 발생하여 성능이 좋지 않다. 작동 방식은 NFS 서버처럼 작동하면서 암호화 대상 파일 및 디렉토리의 내용에 접근할 때 암복호화를 수행하여 사용자 프로그램에 넘겨준다. TCFS는 CFS의 네트워크 확장 버전으로 비슷한 작동 방식을 가진다.

EFS[7]는 마이크로 소프트 윈도우즈에서 제공하는 암호화 파일 시스템이다. CFS와 TCFS는 사용자 계층에서 NFS 서버처럼 동작하는 암호화 파일 시스템이다. EFS는 윈도우즈 NT 서버에 기반하며 커널 내부에 존재하며 NTFS에 암호화 기능을 추가한다. 그러나 암호화 및 사용자 인증 기능 자체는 사용자 영역에 있는 DLL에 구현해 놓았다.

3.2 장치 드라이버 암호화

파일 시스템 하위 계층의 장치 드라이버에서 장치로 오고가는 모든 데이터를 암호화 및 복호화 하는 방법이다. 상위 파일 시스템에 대해 전혀 알 필요가 없기 때문에 손쉽게 특정 장치를 암호화 할 수 있는 장점이 있지만 중요 파일에 대해서만 선택적으로 암호화 하는 것은 불가능 하다.

리눅스의 Cryptoloop[8], NetBSD를 위한 CryptoGraphic Disk Driver[9], BestCrypt[10]가 이러한 방식을 사용하고 있다.

3.3 저장 장치 암호화

저장 장치 암호화는 디바이스가 자체적으로 암복호화 하드웨어를 장착하고 자신에게 들어오고 나가는 데이터에 대해서 자동적으로 암복호화를 수행하는 방식을 말한다. 이 기법은 OS와 아무런 관련이 없이 하드웨어적으로 수행되므로 저장 장치 암호화를 위해 OS에 추가적인 어떤 조작도 할 필요가 없는 장점을 가진다. 또한 하드웨어적으로 암복호화가 수행되므로 속도도 빠르다. 단점은 디바이스 전체를 단위로 암호화하는 것만 가능하다는 것이다. 디바이스는 파일 시스템을 알지 못하므로 전체를 통체로 암호화 할 수 밖에 없다.

이런 방식을 사용하는 제품으로 DataTravler Secure가[11] 있다. 이는 USB 메모리에 데이터 보호 기능을 추가하였다.

4. 암호화 파일 시스템 설계 및 구현

4.1 YAFFS 파일 시스템 모듈 구성

4.1.1 yaffs_guts

YAFFS는 파일 시스템의 핵심적인 알고리즘을 yaffs_guts.h/.c에 모두 구현해 놓았다. 나머지부분들은 커널에서 제공하는 MTD(Memory Technology Device) API에 대한 wrapper 함수, ECC(Error Correction Code) 계산 함수, 여러 보조 함수 등이다.

yaffs_guts에는 파일 관련 연산, 디렉토리 관련 연산 등의 파일 시스템을 직접 다룰 수 있는 함수들이 제공된다.

4.1.2 VFS layer

YAFFS 파일 시스템을 VFS(Virtual File System) interface를 통해 Linux에서 사용할 수 있도록 해주는 계층이다. VFS interface에 정의된 방식으로 super block의 정보 설정, 여러 파일 시스템, 디렉토리, 파일 관련 함수 포인터의 설정 작업 등을 한다.

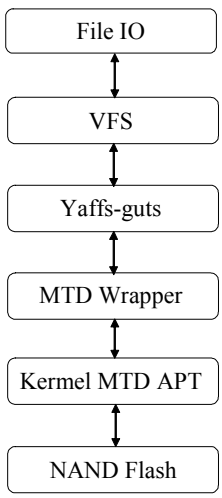
VFS layer에서 이상과 같은 작업을 하는데 있어서 4.1.1에서 설명한 yaffs_guts를 사용한다. 즉, VFS layer는 Linux와 yaffs_guts를 interfacing 해주는 역할을 하고 있다.

4.1.3 MTD interface

MTD는 플래시와 같은 solid state 메모리 디바이스를 위해 Linux에서 제공하는 interface이다. 커널에 struct mtd_info가 제공되어 있고 이 구조체 안에 실제 플래시 메모리에 대해 읽기, 쓰기 연산을 수행할 수 있는 함수 포인터가 들어있다. YAFFS는 이러한 MTD에 대한 wrapper 함수를 내부적으로 가지고 있고, 이를 통해 실제 플래시에 대한 읽기 및 쓰기를 수행한다. 4.1.2의 yaffs_guts는 이 MTD interface를 이용하여 플래시에 접근한다.

4.1.4 전체 구조

이상에서 기술한 YAFFS 파일 시스템의 전체 구조도는 (그림 1)과 같다. 이를 통해 VFS, yaffs_guts, MTD interface의 상호 관계를 볼 수 있다.



(그림 1) YAFFS 구조

4.2 암호화 파일 시스템 요구 사항

4.2.1 특정 파일, 디렉토리 단위 암호화

암호화를 파일 시스템에서 제공하면 장치 드라이버 암호화나 저장 장치 암호화에서의 지원과 달리 특정 파일이나 디렉토리에 대한 선별적 암호화를 수행할 수 있다. 이것이 장치 드라이버 및 저장 장치 암호화에 파일 시스템이 갖는 장점이다. 따라서 이와 같은 선별적 암호화를 지원할 수 있는 구조를 갖추어야 한다.

4.2.2 파일에 따른 다른 암호화 키 사용

만약 모든 파일에 대해 같은 암호화 키를 사용하면 하나의 파일에 대해 암호화 키를 알아내면 전체 파일 시스템의 보안이 뚫리게 된다. 따라서 모든 파일은 각기 다른 파일 암호화 키를 사용해야 한다.

그러나 이를 사용자에게 요구하면 사용자는 수많은 파일 및 디렉토리의 암호화 키를 모두 관리해야 한다. 이는 현실적으로 매우 어려운 일이다.

따라서 하나의 master key를 통해 특정 알고리즘으로 각각의 파일에 대한 파일 암호화 키를 생성하여 이를 통해 각 파일을 암호화 하는 방식을 사용한다. 이렇게 하면 사용자는 하나의 키만 관리하면서도 각 파일은 서로 다른 파일 암호화 키로 암호화 된다.

4.2.3 커널 수정 없는 암호화 제공

Linux 커널은 계속해서 변화되고 있다. 그리고 Linux 커널은 핵심 기능 외의 추가적인 기능은 모듈을 통해 지원함으로써 커널 기능의 동적인 추가 확장을 가능하게 하고 있다. 따라서 YAFFS 상에서 구현하는 암호화 파일 시스템도 이러한 리눅스 커널의 모듈 기능을 활용하여 커널 수정 없이 암호화 기능을 제공해야 한다.

4.3 암호화 파일 시스템 설계 및 구현

4.3.1 키 관리

YAFFS는 내부적으로 yaffs_Object, yaffs_ObjectHeader라는 핵심 자료구조를 가지고 있다. 이 둘은 각각 메모리 및 NAND 상에서의 파일 시스템 구조를 담는 자료구조이다. 이 구조체에 수정을 가하여 파일 암호화 키를 저장한다.

4.2.2에서 말한 파일에 따른 다른 파일 암호화 키를 생성하기 위해 사용자가 제공하는 키와 파일의 inode 번호를 이용하여 HMAC을 계산한 다음 그 결과를 파일 암호화 키로 사용한다. 이렇게 하면 사용자가 제공한 하나의 키를 통해 파일마다 다른 파일 암호화 키를 만들 수 있다. 또한 파일 암호화 키를 가지고 원래 사용자가 제공하는 키를 유추할 수 없다.

이렇게 만든 파일 암호화 키를 메모리 및 NAND에 저장할 때 plain text로 저장할 수 없다. 만약 그렇게 한다면 메모리 스왑 파일을 조사하거나 NAND 메모리의 page를 읽어서 조사하는 것만으로도 쉽게 파일 암호화 키가 공개된다. 따라서 사용자 제공키로 위와 같은 방식으로 만들어낸 파일 암호화 키를 암호화 하여 저장한다.

사용자 제공 키는 사용자의 필요에 따라 하나를 사용할 수도 있고 여러 가지를 사용할 수도 있다. 사용자가 정책적으로 중요 파일에 대해서는 다른 키를 사용함으로써 해당 파일의 보안성을 좀 더 높이는 효과를 낼 수도 있다.

4.3.2 Cryptography API 사용

커널이 제공하는 Cryptography API를 사용하기 위해 yaffs_Device 구조체에 struct crypto_tfm 구조체의 포인터를 추가한다. crypto_tfm은 커널이 제공하는 암복호화 기능을 이용할 수 있게 해주는 interface이다. yaffs_guts의 초기화 및 종료 함수에 Cryptography API 초기화 및 종료 함수를 호출해 준다.

4.3.3 기타 구현 관련 사항

4.2.1에서 언급한 것과 같이 암호화 파일 시스템은 사용자가 선택한 파일 및 디렉토리에 대해서만 암복호화를 제공한다. 특정 파일, 디렉토리의 메타데이터가 NAND의 page에 저장될 때 spare 영역에 YAFFS는 ECC 등의 정보를 저장한다. 이 곳에 남는 공간이 있는데 여기에 해당 파일 및 디렉토리가 암호화 대상인지의 여부를 저장한다.

암호화된 파일에 대해서 암복호화가 수행되는 곳은 (그림 1)의 MTD wrapper 함수이다. 커널에서 제공하는 MTD API를 이용하여 플래시에서 읽기 및 쓰기 작업을 수행할 때 암호화 파일이라면 복호화 및 암호화를 수행한다.

5. 결 론

본 논문은 YAFFS(Yet Another Flash File System)을 기반으로 플래시를 위한 암호화 파일 시스템의 디자인 및 구현을 제시하였다.

이를 위해 플래시 암호화 파일 시스템의 요구사항을 정의하였다. 그리고 여러 측면에서 YAFFS 기반의 플래시 암호화 파일 시스템의 구조를 제시하였다.

먼저 키 관리 기법에 대해서 정의하였다. 제안된 방법을 통해 모든 파일은 각각 서로 다른 파일 키로 암호화 된다. 그러나 사용자는 자신이 원하는 수의 암호화 키를 제공하면 된다. 사용자의 필요에 따라 모든 파일에 대해 같은 키로 접근할 수도 있고 일부 파일에 대해 다른 키로 접근함으로써 보안성을 향상 시킬 수도 있다.

암호화에 있어서는 Linux 커널이 제공하는 표준 암호화 API를 사용하였으며 암호화 모듈이 YAFFS 파일 시스템 모듈 내부의 MTD wrapper 함수에 존재하여 모듈화된 설계를 지향하였다.

제안된 YAFFS 기반의 보안 파일 시스템을 통하여 YAFFS를 사용하는 임베디드 기기는 손쉽게 암호화 파일 시스템을 사용할 수 있게 된다. 제안된 시스템은 기존의 YAFFS와 완전히 호환되면서 추가적으로 사용자가 원하는 파일 및 디렉토리에 대해서만 암호화를 제공한다.

향후 연구 과제는 우선 암복호화 루틴의 효율성을 개선하는 것이다. 현재의 구현은 같은 파일을 계속 읽을 경우에도 같은 복호화 연산을 계속 수행하므로 개선의 여지가 있다. 또한 보안 파일 시스템에서 이슈가 되고 있는 안전 삭제와 같은 추가 기능의 구현이 앞으로 필요하다.

참 고 문 헌

[1] http://www.etnews.co.kr/news/detail.html?id = 200708080107.

[2] http://www.yaffs.net/.

[3] E. Zadok, I. Badulescu, and A. Shender, “Cryptfs : A stackable vnode level encryption file system”, Computer Science Department, Columbia University, Tech. Rep. CUCS-021-98, 1998.

[4] C. Wright, M. Martino, and E. Zadok, “Ncryptfs : A secureand convenient cryptographic file system”, in Proc. of the Annual US ENIX Technical Conference, pp. 197-210, 2003.

[5] M. Blaze, “A cryptographic file system for unix”, in CCS ’93 : proc. of the 1st ACM conference on Computer and communications security, New York, NY, USA : ACM Press, pp. 9-16, 1993.

[6] G. Cattaneo, L. Catuogno, A. D. Sorbo, and P. Persiano, “The design and implementationof a transparent cryptographic file system for unix”, in Proc. of the FREENIX Track ; 2001 USENIX Annual Technical Confrence, Berkeley, CA, USA : USENIX Assocation, pp. 199-212, 2001.

[7] Microsoft Corporation, “Encrypting file system for windows 2000”, Tech. Rep., July.

[8] GNU License, “The GNU/Linux CryptoAPI”, August 2003. [Online]. Available : http://www.kernel.org.
[9] R. Dowdeswell and J. Ioannidis, “The cryptographic disk drive”, in Proc. of the Annual USENIX Technical Conference, FREENIX Track, June 2003.
[10] Jetico Inc., “Bestcrypt corporate edition”, 2001. [Online]. Available : http://www.jetico.com
[11] Kinston Technology company, “Datatraveler Secure”, 2007. [Online]. Avaiable : http://www.kingston.com/flash/DataTravelers_enterprise.asp.

김 석 현

2001년 서울대학교 재료공학부 (공학사)
2001년～2004년 엔틱스소프트 (현 레드덕)
2004년～2005년 이네트
2006년～현재 서울대학교 컴퓨터 공학부 석사과정

조 유 근

1971년 B.E. at Seoul National University
1973년 M.S. at Seoul National University
1978년 Ph.D. in Computer Science, at University of Minnesota
1979년～현재 Professor, Seoul National University
1985년～현재 Visiting Assistant Professor, University of Minnesota
1993년～1995년 Director of ERCC, Seoul National University
1995년～1996년 Vice President of Korea Information Science Society
1999년～2001년 Associate Dean of College of Engineering
2001년～2002년 President of Korea Information Science Society
2003년～현재 Member, National Academy of Engineering of Korea