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.2013.23.5.849

HSM을 이용한 효율적인 파일시스템 암호화 메커니즘*

강 철 오,[1†] 원 종 진,[1] 박 성 진,[1] 류 재 철[2‡]
[1]한국전자통신연구원 부설연구소, [2]충남대학교

Efficient File System Level Encryption Mechanism Using HSM*

Cheol-oh Kang,[1†] Jong-jin Won,[1] Sung-jin Park,[1] Jea-cheol Ryou[2‡]
[1]The Attatched Institute of ETRI, [2]Choongnam University

요 약

최근 모바일 컴퓨팅 환경 변화로 사용자 단말의 분실, 도난, 악성코드에 의한 위협이 증가하여 사용자 단말기에 저장된 중요 정보가 노출될 가능성이 증가하고 있으며, 대응책으로 사용자 단말의 인증, 저장 자료 암호화 솔루션이 적용되어야 한다. 암호화 솔루션은 암호학적 안전성과 암호 연산에 따른 성능 이슈가 trade-off 관계에 있다. 파일 암호화에 많이 사용되는 CBC 모드는 IV에 대한 재사용 불가, 예측불가능, 무결성 보장 요구사항을 갖고 있다. 본 논문에서는 HSM을 사용하는 환경에서 파일시스템 레벨의 암호화 솔루션 구현 시, IV 운용에 대한 요구사항을 충족하며, 효율적인 암호 세션 관리를 통해 성능을 향상할 수 있는 암호화 방식을 제안하고, 제안된 방식의 설계 및 구현을 통해 기존 방식에 비해 성능 측면에서도 우수함을 증명한다.

ABSTRACT

In today's mobile computing environment, there are many threats, such as device loss or theft, malware, to the sensitive information stored on end user device. To prevent disclosure of information, encryption and authentication method are properly adjusted to the device. In cryptographic file systems, CBC mode of operation has been commonly used. It requires an IV need not be secret, but must be unpredictable and protect integrity of the IV. In this paper, we propose file system-level encryption mechanism with HSM that satisfy the requirement of the IV and improve the performance. Moreover, Design and experimental results prove the efficiency of our proposed method.

Keywords: CBC Mode, IV, File System-Level Encryption, HSM

I. 서 론

최근 ICT(Information and Communication Technology)의 발전에 따라, 전산 업무 환경은 사무실내 유선 네트워크와 데스크탑 PC를 사용하는 고정된 형태에서 무선 인터넷과 소형 모바일 디바이스를 사용해 언제 어디서나 업무가 가능한 모바일 오피스 환경으로 변화하고 있다.

그러나, 이동성을 보장하는 모바일 단말의 사용은 분실/도난 위협의 증가, 악성코드에 의한 위협이 상존하여, 사용자 단말기의 중요 정보가 노출될 위험을 안고 있다.

이러한 위협에 대응책으로 사용자 단말의 접근을 제한하는 인증 메커니즘과 저장된 중요 정보를 암호화하는 방법이 강구되어야 함을 권고하고 있으며, 사용자 디바이스에 저장되는 데이터를 암호화하는 기술을 크게 FDE(Full Disk Encryption), Volume/Virtual Disk Encryption, File/Folder Encryption으로 구분하고 있다[1].

접수일(2013년 6월 17일), 게재확정일(2013년 7월 26일)
* 본 연구는 방위사업청과 국방과학연구소가 지원하는 국방위성항법연구센터 사업의 일환으로 수행하였습니다.
† 주저자, cyberkan@ensec.re.kr
‡ 교신저자, jcryou@home.cnu.ac.kr(Corresponding author)

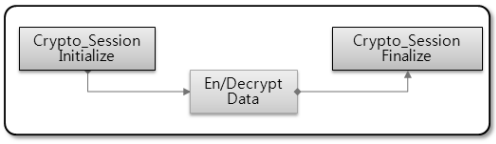
또한, 암호화를 수행하는 위치에 따라 유저레벨, 커널레벨, 디스크 레벨 암호화로 구분할 수 있다.

유저레벨 암호화는 사용자가 민감한 정보를 포함하고 있는 파일을 수동으로 암호화하는 방식으로 파일 암호화 응용 프로그램이 대표적이다.

커널레벨 암호화는 암호화를 수행하는 위치에 따라 다양한 기법이 존재한다. 파일시스템을 기준으로 파일시스템 상위 계층, 하위 계층에서 암호화를 수행하는 방법과 파일시스템 사이에서 암호화를 수행하는 방법으로 나뉜다.

디스크레벨 암호화는 저장 장치의 물리적 특성을 이용하여 암호화를 처리하는 방식으로 FDE가 대표적인 예이다.

암호 파일 시스템은 저장 데이터를 암호화하는 특성상 대칭키 기반의 블록 암호 알고리즘을 사용하고 있으며, 블록 암호 운영모드로 CBC(Cipher Block Chaining) 모드를 가장 많이 사용하고 있다[2]. CBC 모드를 포함해 블록 알고리즘 운용 시 데이터 암호화를 위해 각 암호화 단위로 [그림 1]과 같은 암호화 세션이 적용된다. 암호화 세션은 암호화 세션 초기화, 데이터 암호화, 암호화 세션 종료의 단계로 구성되며, 세션 초기화 작업에는 암호화에 사용될 암호 알고리즘, 운영모드, 암호화 키 그리고 IV(Initialization Vector)를 설정하고 세션 ID를 할당하는 작업 등이 수행된다.



[그림 1] 암호 세션 Lifetime

파일시스템 암호화 솔루션은 안전성, 성능과 사용 편의성이 가장 중요한 평가 요소이며, 안전성과 성능, 안전성과 사용 편의성은 항상 trade-off 관계를 안고 있다. 특히, 파일시스템 레벨 암호 솔루션은 암호 성능이 시스템의 전체 성능을 좌우하고, 심한 경우 시스템 crash를 발생할 수 있다.

파일시스템 암호화 시 암호화 단위는 구현 위치에서 관리하는 블록 크기 단위가 된다. 이에 따라 일반적인 구현 방법에서는 파일시스템이 처리하는 데이터 블록 단위로 암호화 세션을 적용하고 있다. 파일 크기에 비해 파일시스템 처리 블록은 고정된 크기의 작은 단위로 [그림 1]과 같은 암호화 세션 관리가 빈번하게 발생되어 전체적인 암호화 성능에 부하를 유발하는 이슈를 안고 있다.

S/W로 구현된 사용자 단말용 암호화 솔루션은 알고리즘, 암·복호용 키, 인증용 키 등 중요 정보들이 단말기의 저장 공간 및 실행 메모리상에 노출되는 위험성이 있다. 이러한 위험을 제거하기 위해 강력한 인증용 키를 저장하고 암호 연산을 자체적으로 수행하는 전용 H/W 모듈(HSM : Hardware Security Module)을 적용할 수 있다.

본 논문에서는 HSM을 적용한 파일시스템 레벨 의 암호화 솔루션 구현 시, CBC 모드의 안전한 IV 운용에 대한 요구사항을 만족시키면서 암호화 세션의 효율적 관리를 통해 잦은 암호 세션 초기화에 따른 I/O 빈도를 줄임으로써 암호 성능을 향상시킬 수 있는 암호화 운영방식을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 저장자료 암호화에 사용되는 암호화 기법들에 대해 살펴보고, 3장은 제안하는 암호화 방식을 설명하고 설계를 제안한다. 4장에서는 제안한 방식의 분석 및 구현 결과 실험 데이터를 통해 제안한 방식의 성능 개선 효과를 증명하고, 5장에서 결론을 맺는다.

II. 관련 연구

데이터 보호를 위한 다양한 필요성으로 인해 암호화하여 데이터를 저장하는 연구는 아주 오래전부터 진행되어 왔다. 본 장에서는 이와 관련된 주요 암호화 기법중에 저장자료암호화 기법을 암호화 수행 위치에 따라 분류한 방식을 살펴보고, 제안 방식과 관련된 SFS(Stackable File System), HSM에 대한 연구를 살펴본다.

2.1 저장자료 암호화 기법

저장되는 데이터를 암호화하는 기법은 암호화를 수행하는 위치에 따라 유저레벨, 커널레벨, 그리고 디스크레벨 암호화로 분류된다.

유저레벨 암호화 방식은 운영체제의 사용자 공간에서 암호화가 이루어지며, 파일 단위로 암호화를 수행하는 CryptoFS, EncFS, MetFS 등 파일 암호화 응용 프로그램이 대표적인 예이다. 또한, 워드 프로세서나 메일 클라이언트와 같이 응용 프로그램 자체에 암호화 기능을 제공하는 경우도 있다. 이 방식은 구현 및 적용이 쉬우나 사용자가 직접 파일을 읽거나 쓰기

전에 암·복호화를 수행해야 하는 불편함이 있으며, 사용자가 민감한 정보를 평문으로 보관할 수도 있는 문제점을 안고 있다.

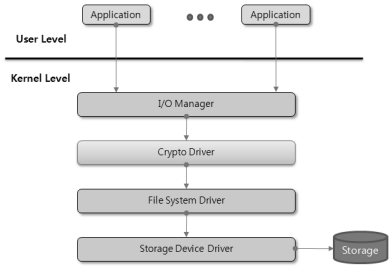
디스크 레벨 암호화는 파일 시스템보다 하위 레벨에서 섹터나 클러스터 등 하드디스크나 파티션의 데이터 처리 단위로 암호화하는 방법으로 리눅스를 대상으로 하는 dm-crypt[3]와 마이크로소프트의 BitLocker[4]가 대표적이다. FDE로 알려진 이 방법은 사용자의 상호작용 없이 디스크에 저장되는 모든 파일(swap 파일을 포함하여)을 자동으로 암호화 저장하여 사용자 편의성과 안전성을 제공하는 장점이 있으나, 외부로 전달되는 파일에 대해서는 자동 복호화되기 때문에 암호화를 보장하기 위해서는 별도의 암호화 방법이 제공되어야 한다. FDE 시스템의 대표적인 단점은 다음과 같다.

- ○ 단말기 동작 중 도난 : FDE는 단말이 켜져 있을때는 모든 데이터에 대한 접근이 가능(자동으로 복호화 됨)하여 단말기를 처음 구동할 때 인증 메커니즘만으로는 이와 같은 위협에 취약함
- ○ 실행 메모리상의 키 노출 : FDE는 정상적인 단말기 부팅 후 DRAM에 암·복호용 키를 가지고 있으며, cold_boot attack을 포함하여 DRAM backup 등에 의해 키 노출 위협 존재

커널 레벨 암호화 기법은 Microsoft의 EFS(Encrypting File System), SUN Microsystems의 ZFS 등과 같이 파일시스템 자체가 암호화를 지원하거나, 파일시스템 상·하 또는 중간 계층에 암호화 기능을 추가하는 형태로 구현된다. 암호화 기능이 추가된 파일시스템을 암호 파일 시스템 또는 파일시스템-레벨 암호화라고 한다. 암호 파일시스템은 커널 레벨에서 응용 프로그램과 독립적으로 동작되며, 사용자에게 암호화 판단에 대한 부담을 덜어 준다. 커널 레벨 암호화는 구현 방식에 따라 사용자에게는 자동 암·복호화에 따른 편의성을 제공할 수 있으며, 파일, 폴더, 볼륨, 디스크 단위의 암호화가 가능하여 암호 솔루션 운용의 유연성을 제공하는 장점이 있다.

2.2 Stackable File System

파일시스템 레벨 암호화 방식 중 파일시스템 상위 계층에서 암호화 기능을 수행하는 모듈을 삽입하여 구현하는 SFS[5]은 [그림 2]와 같이 파일시스템으로 전달되는 파일 I/O를 가로채 이를 암호화한 후에 실제 파일 시스템으로 전달하는 방식이다. SFS는 다음



[그림 2] Stackable File System 구조

과 같은 장점을 제공하고 있어 SFS을 기반으로 한 다양한 연구가 진행되었고, 많은 솔루션들이 SFS 방식을 사용하고 있다. Cryptfs[6], NCryptfs[7], eCryptfs 등이 대표적인 SFS이다.

- ○ 확장성 : SFS는 사용 중인 파일시스템 스택 사이에 필터 드라이버 형식으로 삽입되어 동작되어, 하부 파일시스템에 영향을 받지 않고 구현할 수 있다.
- ○ 개발 편의성 : 윈도우 운영체제의 파일시스템 필터 드라이버 같이 SFS 인터페이스를 운영체제 차원에서 개발 편의성을 고려해 제공하고 있다. 이를 통해 복잡한 파일 시스템과 시스템 캐쉬와의 복잡한 연관관계를 이해하지 않고도 암·복호 기능을 구현할 수 있다.
- ○ 사용자 편의성 : 유저레벨 암호화와 달리 디바이스를 사용자는 파일 읽기, 쓰기 중 암·복호화 수행 여부에 대해 인지하지 않고 일반적인 방법과 동일하게 사용할 수 있다.
- ○ 데이터 유출 가능성 최소화 : 암호화 된 파일을 읽을 때, 복호화된 임시파일이 생성되지 않기 때문에 높은 보안성을 제공하여 유출 가능성을 최소화 할 수 있다.

SFS 방식은 대칭키 기반의 블록 암호를 사용하고 다양한 블록 암호 운영모드로 구현될 수 있으며, CBC 운영 모드가 주로 사용되고 있다. Cryptfs, eCryptfs 등 SFS 방식도 Blowfish, AES 등의 블록 암호 알고리즘을 사용하고 있으며, 블록 암호 운용 모드로 CBC를 사용하고 있다.

2.3 HSM

중요 정보를 보호하기 위해 저장되는 자료를 암호화하는 경우, 암호 알고리즘, 암·복호용 키, 인증용 키 등의 안전한 저장과 관리가 보안 솔루션의 안전성을

판단하는 중요 요소가 된다. 특히 보호 대상 정보의 민감도에 따라 전용 알고리즘을 사용하여 키뿐만 아니라 암호 알고리즘도 비공개로 사용하는 경우가 있다.

S/W로 구현된 사용자 단말용 암호화 솔루션은 알고리즘, 암·복호용 키, 인증용 키 등 중요 정보들이 단말기의 저장 공간에 저장되며, TPM(Trusted Platform Module) 등의 H/W 토큰을 사용하더라도 실제 암호 연산은 단말의 실행 메모리상에서 수행된다. 실행 메모리상에 존재하는 키는 Cold Boot Attack[8] 등에 의해 키 값이 노출될 수 있는 위협이 상존하고 있다.

HSM은 암·복호화에 필요한 알고리즘과 키를 저장하고, 모든 암호화 연산은 HSM 내부에서만 수행하여 암호 연산 과정의 보안성을 보장하고, HSM의 분석을 위해 분해 시도 시 내부 중요 데이터가 자동으로 삭제되는 템퍼 방지 회로를 가지고 있기 때문에 높은 안전성을 보장하여, 보다 안전한 보안 솔루션을 위해 HSM을 적용한 암호 솔루션을 많이 사용하고 있다.

HSM은 다양한 인터페이스를 사용할 수 있다. 일반적인 PC에서는 PCI 버스, USB 인터페이스를 사용할 수 있으며, 스마트폰과 같은 모바일 디바이스는 MicroSD 슬롯, USIM 슬롯을 활용할 수 있다. 일반적인 PC 환경에서는 외장형 HSM으로 제공 가능한 범용 인터페이스인 USB 포트를 주로 활용할 수 있으며, 모바일 디바이스에서는 많은 디바이스들이 보유하고 있는 MicroSD 슬롯을 활용할 수 있다.

높은 보안성을 보장하기 위해 HSM을 사용할 때 발생하는 가장 큰 문제점은 암·복호 연산 수행을 위해 HSM으로 데이터, 명령어를 전송하고 처리된 결과를 받는데 따르는 I/O로 인한 성능 저하이다. 단말의 CPU를 사용한 S/W 방식은 디바이스 내부의 CPU와 메모리간의 데이터 전송이기 때문에 데이터 이동과 암호 처리 속도는 큰 문제가 되지 않는다. 하지만, HSM을 사용하는 경우 HSM과 연결된 인터페이스의 성능에 따라 I/O에 따른 성능 저하가 필연적으로 발생한다.

USB, PCI 등 다양한 데이터 버스는 기본 Loopback에 소요되는 I/O 지연을 갖고 있으며, 한 번에 전송 가능한 최대 데이터 블록 크기를 기준으로 성능을 제시하고 있다. 따라서, 실 데이터 크기와 관계없이 초당 데이터를 주고 받을 수 있는 최대 횟수에 제한을 갖고 있다. 예를 들어 USB 2.0 인터페이스 표준은 고속 모드에서 사용되는 Interrupt 또는 Isochronous Transfer 방식 적용 시 Data Payload 사이즈에 따라 12Mbps에서 480Mbps까지 성능을 제공할 수 있다. 고속 모드의 경우 125 usec 시간 단위로 microframe을 정의하고 있으며, 이는 초당 8,000 microframe이 발생하고 각 microframe당 7,500 Bytes의 데이터를 전달할 수 있다. 따라서, 각 microframe에 실어 보내는 데이터 크기에 따라 초당 전달 가능한 성능이 좌우된다. 즉, USB 2.0을 인터페이스로 사용하는 HSM의 경우 데이터 암호화를 위해 필요한 I/O 횟수를 최소화하는 것이 전체 성능을 향상할 수 있는 방법이 될 수 있다.

III. 제안 방안

3.1 CBC 모드의 IV 요구사항

CBC 모드는 각 평문 블록을 암호화하기 전에 이전 암호문 블록과 XOR하는 방식이다. 첫 번째 암호문 블록을 생성하기 위해서 첫 번째 평문 블록과 XOR될 IV(Initialization Vector)를 사용하고, 이때 사용된 IV는 암호문과 함께 저장되어 복호화에 사용된다. CBC 모드의 안전한 사용을 위해 국제표준인 ISO/IEC 10116과 미국의 NIST는 CBC의 IV와 관련된 다음과 같이 3가지 가이드라인을 제시하고 있다[9].

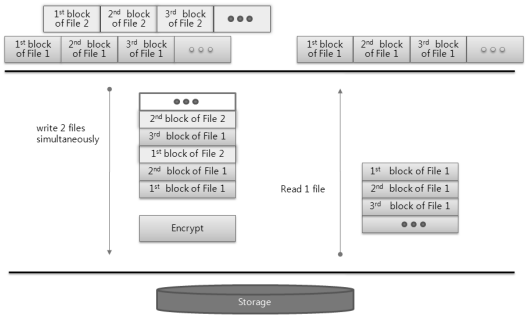
- ○ 재사용 불가 : 동일한 IV를 사용하는 환경에서 두 개의 메시지가 동일하다면 대응되는 암호문은 동일하다. 즉, 서로 다른 두 개의 메시지의 처음 m개의 블록이 같고, 서로 다른 IV를 사용하지 않는다면 대응하는 m개의 암호문 블록은 동일하다. 따라서 CBC 모드를 사용할 경우 매 암호화 세션마다 다른 IV를 사용해야만 한다.
- ○ 예측불가능성 : IV는 암호화 시점에 예측이 불가능해야 한다. 만약 암호화 시점에 IV가 예측 가능하다면 chosen-plaintext attack을 사용해 평문 정보를 얻어낼 수 있다[10]. 따라서 IV는 random, 또는 pseudo-random 하게 생성되어야 한다.
- ○ 무결성 보장 : 공격자가 IV의 특정 비트 값을 변조할 수 있으면 암호문의 복호화 결과 첫 블록의 해당 비트 값이 바뀌게 된다. 따라서 IV가 비밀일 필요는 없으나 무결성은 보장되어야 한다.

3.2 기존 파일시스템 레벨의 암호화 방식 분석

CBC 모드는 암호문 블록이 체이닝(chaining)되는 특성을 가지며 첫 블록을 위한 IV가 필요하다.

응용 프로그램에서 파일 암호화를 수행하는 경우, 파일 단위로 암호화 세션이 적용된다. 즉, 각각의 파일에 대해 〔그림 1〕과 같은 일련의 암호화 세션 관리 과정이 적용된다. 복호화 세션 역시 파일 단위로 적용된다.

파일시스템 레벨에서 암호화를 할 경우, 〔그림 3〕과 같이 여러 개의 파일이 각각 순차적으로 처리되지 못하고 파일 시스템에서 사용되는 데이터 블록(FS_DB) 단위로 동시에 랜덤하게 처리되기 때문에 파일 단위의 암호화 세션을 적용할 수 없다.



〔그림 3〕 File System-Level 암호화 시 파일 블록 처리 과정

이러한 동작 방식으로 인해 기존 파일시스템 암호 솔루션은 FS_DB 단위로 암호화 세션을 적용하고 있다. 즉, 기존 암호파일시스템은 FS_DB 단위로 IV를 생성하여 암호 연산을 초기화하고, 데이터 블록을 암호화한 후에 디스크에 저장한다. 그리고 응용 레벨에서 파일을 읽으면 디스크에 저장된 파일의 데이터 블록들을 각각 읽어 각각의 IV로 복호화 연산을 초기화하고, 데이터를 복호화한 후에 응용 프로그램으로 전달하는 방식을 사용하고 있다.

3.2.1 기존 암호 파일 시스템의 문제점

기존 암호파일시스템은 FS_DB 단위로 암호화 세션의 라이프타임을 가져감으로서, 데이터 암호화 연산 대비 많은 암호화 세션 관리 연산이 수행될 수 밖에 없다. 특히, 암호 세션 초기화는 IV를 생성하기 위한 연산이 수행되며, IV의 무결성을 보장하기 위해 추가의 인증 연산(MAC)이 수행되어야 한다. 또한 암호화된 데이터 블록을 복호화할 때는 IV의 MAC 체크가 수반되기 때문에, 데이터 블록의 암호화/복호화 이외에 추가적인 암호 연산으로 암호 파일시스템의 성능을 저하시키는 요인이 되고 있다.

Windows 운영체제의 경우 사용자에 의한 조작 외에도 운영체제 자체적으로 많은 파일 오퍼레이션을 수행한다. 한 예로 Windows explorer를 실행시키면, 폴더 구조와 선택된 폴더 내의 파일들에 대한 리스트를 표시하기 위해 Disk에서 폴더 내의 파일들의 일부 내용을 읽는 작업을 수행한다. 따라서 파일 시스템 레벨에서 암호화를 수행하기 위해서는 고속의 암호화 성능이 요구된다. 커널 레벨 암호화에 따라 암호화에 많은 시간이 소요되면 파일시스템이 느려지는 현상이 발생하고, 이러한 현상이 누적되면 파일 시스템 crash가 발생할 수 있다.

3.2.2 암호 파일 시스템 설계 시 고려사항

파일시스템 레벨에서의 암호화는 파일 단위의 순차적 처리가 불가능하며, FS_DB 단위로 처리할 수 있어야 한다. 또한, 불필요한 암호 연산을 제거해 연산 지연을 최소화해야 하며, 동시에 암호학적 안전성도 보장되어야 한다.

그러나 파일시스템의 특성상 암호 연산의 효율화와 안전성을 동시에 만족시키는 것은 쉽지 않다. FS_DB 단위의 암호화 세션을 초기화하기 위해 필요한 IV는 예측 불가능성을 제공하기 위해 통상 난수발생기(RNG : Random Number Generator)를 사용해 난수를 생성하여 IV로 활용한다. 이러한 난수를 FS_DB 단위로 생성하는 것은 전체 암호화 성능을 저하시키는 요소가 될 수 있다.

잦은 암호 초기화에 따른 성능 이슈와 더블어 CBC 모드를 사용하는 암호 파일시스템에 있어 가장 문제가 되는 부분은 IV의 무결성 보장을 위한 추가적인 연산 부하가 수반되는 것이다.

따라서, 본 논문에서는 기존 파일시스템 레벨 암호화 솔루션의 잦은 암호 세션 초기화를 회피하여 HSM과의 I/O 빈도를 줄이고, IV의 3가지 운용 요구조건을 충족하면서 암호화에 따른 부하를 최소화할 수 있는 할 수 있는 암호화 방안을 제안한다.

3.3 제안하는 암호화 방안

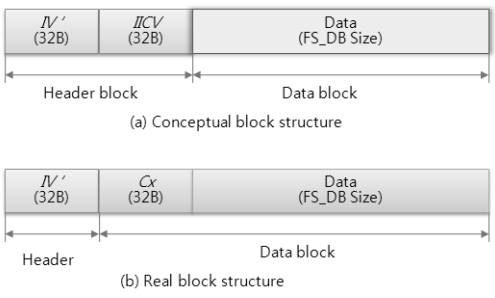
제안하는 방법은 SFS 방식을 적용하되, 암호 세션을 파일 시스템이 처리하는 FS_DB 단위가 아닌 파일 시스템 lifetime(시스템 부팅 후, 커널에 의해 파일시스템이 로딩된 시점부터 시스템이 종료되어 언로딩 시점까지) 단위로 적용한다. 즉, 파일 시스템이 커널에 의해 로딩되는 시점에 암호 세션 초기화가 수행되고 시스템이 종료될 때 암호 세션 종료를 수행하는 방식이다. 이에 따라, 시스템이 부팅되는 시점에 한번만 IV를 생성해 암호 세션 초기화를 수행하고, 이후 파일 시스템은 저장되는 데이터 블록에 대해 별도의 초기화 없이 계속 암호화하여 저장하는 방식이다. 결과적으로 파일 시스템 lifetime 동안 암호화되는 데이터 블록은 계속적으로 체이닝되어 저장된다. 파일시스템의 비정상적인 종료 및 재시작은 제안하는 암호세션의 종료 및 재시작을 의미하는 것으로 파일시스템의 오류에 의한 재기동시에 암호세션 관리를 동기화시킴으로서 파일시스템 오류에 의한 문제를 해결할 수 있다.

제안 방식에서도 암호화되는 대상 데이터는 FS_DB 단위의 블록들이며, 암호화시에는 모든 파일 시스템 블록이 체이닝되어 순차적으로 저장되어 암호화 처리에는 문제가 없다. 단, 복호화시에는 〔그림 3〕에서와 같이 암호화된 블록들이 순서대로 복호화가 이루어지지 않아 이전 블록에 대한 정보를 알 수 없어 문제가 발생할 수 있다. 따라서 데이터 블록 단위의 복호화를 위해서는 암호화된 블록에 추가적인 정보가 필요하게 된다.

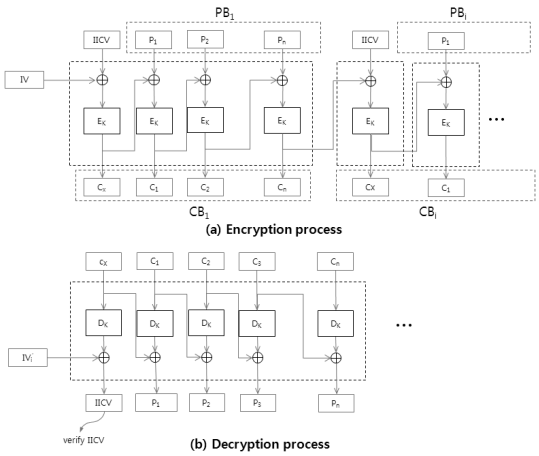
제안하는 방법은 암호 연산을 처리하는 Crypto Driver에서 FS_DB 단위로 데이터 블록을 구성하고, 암호화된 결과에 헤더를 추가하여 〔그림 4〕와 같이 암호화된 결과 블록을 구성한다. 헤더 블록은 데이터 블록 암호화에 사용된 IV'와 IV'의 무결성을 확인할 수 있는 $IICV$(IV Integrity Check Value)로 구성되며, 데이터 블록은 암호화된 데이터를 의미한다.

암호화 헤더의 IV' 필드에는 이전 암호화 블록의 마지막 32B 결과를 저장하여 복호화시 IV 값으로 사용한다. $IICV$는 IV'의 변조 여부를 확인할 수 있는 특정 값으로 구현 시 임의의 32B 값을 사용한다. 결과적으로 매번 RNG를 사용해여 IV를 생성해야 하는 오버헤드 없이 복호화시 필요한 IV를 제공할 수 있으며, 〔그림 3〕과 같이 파일 시스템 레벨의 블록 단위 처리(파일 시스템상에서 랜덤하게 처리되는 블록 특성)로 이전 블록에 대한 예측이 어려워 IV의 예측 불가능 조건을 만족하고 있다. 또한, 암호화 시 $IICV$ 블록을 추가해 암호화하고, 복호화시 $IICV$의 확인을 통해 IV'의 무결성을 보장하고, 이전 암호 블록의 마지막 32B를 IV'로 사용함에 따라 연속적으로 체이닝 처리되는 블록 암호 특성을 그대로 승계하고 있어 IV가 중복 사용되지 말아야 하는 조건도 충족하고 있다.

제안하는 방식은 AES-256 블록 알고리즘을 CBC 모드로 운용하며, 사용되는 기호와 용어는 다음과 같다. PB_i는 파일 시스템이 처리하는 i 번째 평문 데이터 블록, PB_i를 암호화한 결과는 $CB_i = E_K(IICV \parallel PB_i)$, P_i는 평문 블록의 i 번째 암호 단위 블록, C_i는 P_i를 암호화한 결과, IV_i'는 PB_i를 암호화하는데 사용된 IV, $PB_i(P_i)$는 PB_i 블록의 i 번째 암호 단위 블록, $CB_i(C_i)$는 $PB_i(P_i)$의 암호문을 의미한다.



〔그림 4〕 제안 방식의 암호 단위 블록 구조



〔그림 5〕 제안 방식의 암·복호화 처리

3.3.1 암호화

암호화 세션의 초기화는 암호화에 사용될 암호 알고리즘, 운영모드, 암호화 키 그리고 IV를 설정하는 작업이다. 제안하는 방식도 암호화 세션 초기화에 사용되는 IV는 매 세션 초기화마다 RNG를 이용하여 블록 알고리즘 크기의 nonce를 생성하여 적용한다.

암호화는 [그림 5]의 (a)와 같이 FS_DB 단위로 수행되며, IV로 사용된 값의 무결성 보장을 위해 32B의 $IICV$를 모든 FS_DB 블록의 앞에 추가하여 암호화를 수행한다. 따라서 암호화의 입력은 $IICV$와 평문 데이터 블록이 연접된 $PB_i' = IICV \| PB_i$로 표시한다. 암호화 연산 후, 복호화를 위해 PB_i'의 암호화에 사용된 IV_i'를 헤더에 추가하고, 암호화 결과 CB_i를 저장한다.

$$CB_i = E_K(PB_i') = E_K(IICV \| PB_i)$$
$$CB_i(C_X) = E_K(IICV \oplus IV_i') = E_K(IICV \oplus CB_{i-1}(Ch))$$
$$CB_i(C_i) = E_k(PB_i(P_i) \oplus CB_i(C_{i-1})),\quad CB_i(C_0) = CB_i(C_X)$$
$$IV_i' = CB_{i-1}(C_n),\quad IV_1' = IV$$

3.3.2 복호화

복호화는 암호화 저장된 블록 단위로 수행되며, [그림 5]의 (b)와 같이 각 블록의 헤더 정보를 이용하여 복호 세션 초기화 작업이 수행된다. 세션 초기화 시에 암호화 초기화와 달리 별도의 IV를 위한 연산 작업은 필요 없으며, 헤더에 포함된 IV_i'를 IV 값으로 초기화한다. 각 블록 단위는 독립적으로 복호화 연산이 가능하여 병렬 처리가 가능하다. 헤더에 포함된 IV_i'를 이용하여 CB_i의 복호화 결과로 $IICV$와 PB_i를 얻으며, 복호화된 $IICV$ 값을 약속된 $IICV$와 비교하여 일치 여부를 확인하고 IV_i'의 변조 여부를 판단할 수 있어 IV_i'의 무결성을 보장할 수 있다.

$$PB_i' = D_K(CB_i)$$
$$PB_i(IICV) = D_K(CB_i(C_X)) \oplus IV_i'$$
$$PB_i(P_i) = D_K(CB_i(C_i)) \oplus CB_i(C_{i-1}),\quad CB_i(C_0) = CB_i(C_X)$$
$$If\ (PB_i(IICV) \neq IICV)\ \ Then\ IV_i'\ is\ Modified$$

IV. 실험 결과

본 장에서는 USB 인터페이스를 사용하는 HSM을 적용하여 제안하는 방안의 구현 결과와 기존 방식의 구현 결과의 성능 비교를 통해 제안하는 방안의 성능 개선 효과를 확인한다.

4.1 비교 구현

커널 모드 파일 암호화를 SFS 방식을 적용하여 구현 시, 하나의 암호화 단위(일반적으로 4KB)에 대해 암·복호 연산 과정이 수행되는 연산은 PKCS#11[11]을 이용한 pseudo 코드로 작성 시 일반적으로 다음과 같다.

```
// ① IV 값 생성(RNG 사용)
C_GenerateRandom(iv);
// ② 암호 세션 초기화
C_SignInit(Key_sign);
C_EncryptInit(Key_enc, IV, CBC);
// ③ IV에 대한 HMAC 계산
C_SignUpdate(IV);
// ④ data에 대한 암호화와 HMAC 동시 수행
C_SignEncryptUpdate(data);
// ⑤ 암호 세션 종료
C_EncryptFinal();
C_SignFinal();
```

단위 데이터 블록에 대한 암호화 시 수행하는 명령어들은 암호세션 초기화, 데이터 암호 처리, 암호세션 종료의 과정으로 구성되며, 초기화 시 필요한 IV는 일반적으로 RNG를 이용하여 생성한다. HSM을 사용하는 환경에서는 RNG 또한 HSM을 이용하여 생성한다. 기본적인 데이터 암호 처리는 데이터의 기밀성 보장을 위한 암호화와 데이터와 IV의 무결성 보장을 위한 HMAC 계산이 수행되며, 암호화와 HMAC 연산을 동시에 계산하는 PKCS#11의 듀얼 함수를 사용한다.

위와 같이 일반적인 구현 방안은 단위 데이터 블록을 암호화하기 위해 최소한 7회 HSM과의 통신이 필요하게 된다.

제안하는 방법은 이전 데이터 블록의 마지막 암호 블록(32B)을 IV로 활용하여 매 단위 블록마다 IV생성에 필요한 연산을 제거하고, 암·복호 연산을 위한 암호 세션 초기화, 종료 함수(C_EncryptInit/Fi-

nal, C_DecryptInit/Final)를 시스템 부팅과 종료 시 한 번씩만 호출하면 되기 때문에 암·복호 데이터가 증가할수록, 기존 방식 대비 HSM과의 통신 횟수가 현저하게 줄어든다. 또한 추가적인 블록(*IICV*)을 IV 무결성 확인을 위해 활용하여 HSM에서의 IV에 대한 무결성 확인 연산(C_SignUpdate(IV))없이도 IV 무결성을 보장한다.

이를 PKCS#11을 이용한 pseudo 코드로 작성 시 다음과 같다.

```
// ① IV 값 생성(RNG 사용)
C_GenerateRandom(iv);
// ② 암호 세션 초기화(시스템 부팅 시)
C_EncryptInit(Key_enc, IV, CBC);
// 시스템 종료시까지 반복
until (system power off) for every Block
{
    C_SignInit(Key_sign);
    C_SignEncrypt Update(data);
    C_SignFinal();
}
// ③ 암호 세션 종료(시스템 종료 시)
C_EncryptFinal();
```

제안하는 방법에서 IV 무결성 관련 부분을 ($IV \| data$)에 대한 HAMC을 계산하는 방식으로 적용하면 pseudo code는 다음과 같다.

```
// ① IV 값 생성(RNG 사용)
C_GenerateRandom(iv);
// ② 암호 세션 초기화(시스템 부팅 시)
C_EncryptInit(Key_enc, IV, CBC);
// 시스템 종료시까지 반복
until (system power off) for every Block
{
    C_SignInit(Key_sign);
    C_EncryptUpdate(data);
    C_SignUpdate(IV || data);
    C_SignFinal();
}
// ③ 암호 세션 종료(시스템 종료 시)
C_EncryptFinal();
```

4.2 결과 분석

이와 같이 세 가지 접근 방안에 대해 동일한 HSM을 이용하여 파일시스템 상위의 파일 I/O를 후킹하여 암호화를 처리하는 필터 드라이버를 구현하여 4MB의 파일을 암호화하는데 소요되는 시간을 측정한 결과는 〔표 1〕과 같다.

〔표 1〕은 4MB의 데이터를 4KB 단위(1,000개 블록)로 암호화를 수행하는데 소요된 시간을 측정한 결과이며, 기존 방식 대비 제안한 방식이 약 2.7배의 성능 개선 효과를 나타냄을 알 수 있다. 이는 〔표 2〕에 나타난 각 함수의 소요 시간 측정 결과로 예측되는 성능 차이를 반영하고 있다.

〔표 1〕 4KB * 1,000 암호화 연산 시간 비교(sec)

횟수	Orginal	Mixed	Proposed
1	11.497	4.680	4.150
2	11.497	4.649	4.196
3	11.482	4.695	4.181
4	11.497	4.696	4.212
5	11.497	4.680	4.212
6	11.498	4.680	4.196
7	11.481	4.680	4.197
8	11.482	4.680	4.196
9	11.497	4.680	4.197
10	11.513	4.680	4.212
avg	11.4941	4.680	4.1949
평균성능 (Mbps)	2.919	7.170	7.999

각 함수별 소요 시간 중 암호 세션 초기화를 위한 연산에 많은 시간이 소요되고 있음을 알 수 있으며, 따라서, 초기화 빈도를 줄일 수 있는 제안한 방법이 기존 방식 대비 개선된 성능 결과를 나타내고 있다.

〔표 2〕 각 함수별 소요 시간

함수	시간(msec)
C_GenerateRandom()	1.899
C_EncryptInit()	4.642
C_SignInit()	2.583
C_SignEncryptUpdate()	0.748
C_EncryptUpdate()	0.730
C_EncryptFinal()	0.459
C_SignFinal()	0.616

〔표 3〕은 〔표 1〕에서의 구현 결과에서 암호화 단위를 32KB로 변경하여 130회(약 4MB 데이터) 반복

연산한 결과를 보이고 있다. 기존 방식에 비해 성능 개선 효과가 약 2.3배로 암호화 단위 블록이 커질수록 성능 개선 효과는 적어지기는 하지만 제안한 방식이 여전히 효율적임을 알 수 있다. 파일시스템 레벨 암호화 시 암호화 블록 단위를 크게 하면 저장 공간의 낭비를 초래하므로, 적절한 블록 크기를 선택하여 적용하여야 한다. 또한, IV 무결성 검증을 HMAC 연산에 포함하여 수행하는 Mixed 방식의 경우와 비교하여 *IICV* 개념을 적용한 제안 방식이 10 ~ 20% 정도의 성능 개선 효과가 있음을 알 수 있다.

〔표 3〕 32KB * 130 암호화 연산 시간 비교(sec)

	Orginal	Mixed	Proposed
1	1.685	0.858	0.702
2	1.654	0.874	0.702
3	1.685	0.873	0.717
4	1.653	0.874	0.718
5	1.669	0.858	0.702
6	1.654	0.874	0.702
7	1.630	0.873	0.702
8	1.638	0.874	0.702
9	1.654	0.858	0.702
10	1.612	0.873	0.717
avg.	1.6534	0.8689	0.7066
처리속도 (Mbps)	20.611	39.220	48.229

V. 결 론

사용자 단말에 저장되는 자료의 정보 유출 위협에 대응 방안으로 저장 자료를 암호화 저장하는 암호 솔루션이 적용되고 있다. 특히, 암호 연산 시 알고리즘, 키 등의 중요 정보를 보다 안전하게 관리하고, 암호 연산을 독립적으로 수행하는 HSM을 적용하는 보다 안전한 암호 솔루션이 요구되고 있다.

HSM 적용 시 단말기와 HSM간의 잦은 I/O는 암호 솔루션의 성능을 저하시키는 요인이 된다. 파일시스템 레벨 암호화 시 고정된 크기의 파일시스템 데이터 블록 단위로 암호화가 처리되며, 각 블록마다 암호화를 위한 암호 세션 초기화는 암호 솔루션의 성능에 부하를 주고 있으며, 암호 파일시스템에서 많이 적용하는 CBC 모드는 IV의 무결성 보장을 위한 추가적인 연산 부하가 수반되고 있다.

따라서, 본 논문에서는 HSM을 적용한 파일시스템 레벨의 암호화 솔루션 구현 시, CBC 모드 사용에 필요한 IV에 대한 안전한 운용 요구 조건을 만족시키면서 IV 무결성 확인을 위한 암호 연산을 최소화하고, 암호화 세션의 효율적 관리를 통해 잦은 암호 세션 초기화에 따른 I/O 빈도를 줄임으로써 암호 성능을 향상시킬 수 있는 암호화 운영방식을 제안하였다.

제안한 방식의 암·복호화 메커니즘을 설계하여 제시하고 구현한 결과를 통해, 기존 방식의 구현 결과와의 성능을 비교 분석함으로써 제안한 방식이 효율적이고 성능 개선 효과가 있음을 증명하였다. 비교 실험 결과 제안한 방식은 암호 단위 블록의 크기가 작을수록 기존 방식 대비 성능 개선 효과가 더 크게 나타남을 알 수 있다.

참고문헌

〔1〕 K. Scarfone, M. Souppaya, and M. Sexton, "Guide to Storage Encryption Technologies for End User Devices," SP 800-111, NIST, Nov. 2007.

〔2〕 S. Bruce, Applied Cryptography 2nd edition : protocols, algorithms, and source code in C, John Wiley & Sons, 2007.

〔3〕 Cryptsetup Project, 〔OnLine〕. Available: http://code.google.com/p/cryptsetup

〔4〕 N. Ferguson, "AES-CBC + Elephant diffuser A Disk Encryption Algorithm for Windows Vista," Microsoft White Paper, Aug. 2006.

〔5〕 J. S. Heidemann, G. J. Popek, "File system development with stackable layers," ACM Transactions on computer systems, vol 12, no. 1, pp. 58-89, Feb. 1994.

〔6〕 E. Zadok, I. Badulescu, and A. Shender, "Cryptfs: A stackable vnode level encryption file system," Technical Report CUCS-021-98, Computer Science Department, Columbia University, Jun. 1998.

〔7〕 C. P. Wright, M. C. Martino, and E. Zadok, "NCryptfs:A Secure and Convenient Cryptographic File System," Proceedings of the General Track: 2003 USENIX Annual Technical Conferance, pp. 197-210, Jun. 2003.

〔8〕 J. Alex Halderman, S.D. Scheon, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest We Remember: Cold Boot Attacks on Encryption Keys," Proc. 17th USENIX Security Symposium, Jul. 2008.
〔9〕 M. Dworkin, "Recommendation for Block Cipher Modes of Operation : Methods and Techniques," SP 800-38A, NIST, Dec. 2001.
〔10〕 G. V. Bard, "The Vulnerability of SSL to chosen-plaintext attack," Cryptology ePrint Archive, Report 2004/111, 2004.
〔11〕 RSA Laboratories, "PKCS#11:Cryptographic Token Interface Standard," 〔OnLine〕. Available: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf

〈저자소개〉

사 진

강 철 오 (Cheol-Oh Kang) 정회원
1993년 2월: 인하대학교 전자계산학과 졸업
1995년 2월: 인하대학교 컴퓨터공학과 석사 졸업
2002년 3월~현재: 충남대학교 컴퓨터공학과 박사과정
2000년 2월~현재: 한국전자통신연구원 부설연구소 책임연구원
〈관심분야〉 네트워크 보안, 스마트폰 보안

사 진

원 종 진 (Jong-Jin Won) 정회원
1998년 2월: 성균관대학교 정보공학과 졸업
2000년 2월: 성균관대학교 컴퓨터공학과 석사 졸업
2000년 2월~현재: 한국전자통신연구원 부설연구소 선임연구원
〈관심분야〉 네트워크 보안, 전산보안

사 진

박 성 진 (Sung Jin, Park) 정회원
2002년 2월: 인하대학교 컴퓨터공학과 졸업
2005년 2월: 포항공과대학교 컴퓨터공학과 석사 졸업
2008년 9월~현재: 한국과학기술원 전산학과 박사과정
2005년 5월~현재: 한국전자통신연구원 부설연구소 선임연구원
〈관심분야〉 시스템 보안

류 재 철 (Jea-Cheol Ryou) 종신회원
1985년 2월: 한양대학교 산업공학과 졸업
1988년 2월: Iowa State University 전산학과 석사 졸업
1990년 2월: Northwestern University 전산학과 박사 졸업
1991년~현재: 충남대학교 전기정보통신공학부 교수
〈관심분야〉 인터넷 보안