情報保護學會論文誌
第14券 第2號, 2004. 4

신뢰할 수 없는 DRM 클라이언트 시스템 하에서 키 보호를 위한 Secure Storage Device의 연구

이 기 정[a)†], 권 태 경[b)‡], 황 성 운[c)], 윤 기 송[c)]
장미디어[a)], 세종대학교[b)], 한국전자통신연구원[c)]

A Study on the Secure Storage Device for Protecting Cryptographic Keys in Untrusted DRM Client Systems

Kijung Lee[a)†], Taekyoung Kwon[b)‡], Seong-woon Hwang[c)], IKi-song Yoon[c)]
Jang Media[a)], Sejong University[b)], ETRI[c)]

요 약

DRM (Digital Rights Management)[1][2]은 디지털 컨텐츠의 불법 사용방지를 위한 기술로서 인터넷과 같은 전송 매체를 통해 디지털 컨텐츠가 배포될 때 DRM의 복제 방지 기능을 사용해서 디지털 컨텐츠의 안전한 배포 및 판매를 촉진시키기 위한 기술이다. 이러한 DRM 환경 하에서 디지털 컨텐츠 저작권자의 가장 큰 요구 사항인 저작권 보호와 컨텐츠 보호를 위한 핵심 기능을 수행하는 DRM Client 시스템은 신뢰할 수 없는 사용자 환경에 설치되어 향후 배포되는 디지털 컨텐츠들에 대한 확인 및 플레이(실행)를 지속적으로 처리하게 되며, 이러한 처리 과정 중에 DRM Client 시스템은 사용자에 대한 인증이나 암호화된 디지털 컨텐츠 데이터의 복호화 및 디지털 컨텐츠 사용규칙을 명시한 라이센스에 관한 데이터를 처리하며 이러한 데이터는 사용자가 접근 못하도록 안전하게 보호되어야 한다. 본 논문은 이러한 사용자 인증키, 암호키, 라이센스 데이터를 DRM Client 시스템 하에서 안전하게 보호할 수 있는 Secure Storage Device (SSD)를 구현하여 소개하는데 그 목적이 있다.

ABSTRACT

DRM is the ability to brand digital contents with features that ensure copy protection and affect the way in which digital contents are played back. DRM is a technology that enables the secure distribution, promotion, and sale of digital contents on the Internet. The DRM Client System that operates on the untrusted user environments has to meet the requirements of the contents owner, including copyright and contents protection. After the DRM Client System is installed on the untrusted user environments, it verifies and plays digital contents. With these procedures it carries out user authentication, contents decryption, and license management. During these procedures, the sensitive data, including authentication information, decryption data and license data, must be secured against any illegal access from users. The goal of this thesis is to introduce the implementation of Secure Storage Device which can protect user's authentication key, cryptographic key, and license data in safe where the DRM Client System is running.

Keywords: *DRM, Secure Storage, Software Security*

접수일: 2003년 10월 31일; 채택일: 2004년 4월 8일
† 주저자, fumaro@hotmail.com
‡ 교신저자, tkwon@sejong.ac.kr

Ⅰ. 서 론

DRM은 디지털 컨텐츠의 불법 사용방지 및 저작권 보호를 위한 기술로서 암호기술, 트랜잭션 처리기술, 분산 처리 기법 등 다양한 요소들이 복합되어 운영되는 기술이며, 디지털 컨텐츠는 저적권자가 정한 비즈니스 룰(사용규칙)과 함께 암호화되어 패키징되며, 이후 컨텐츠를 이용하고자 하는 사용자는 컨텐츠와 함께 패키징된 비즈니스 룰을 만족시키는 경우 DRM Client 시스템(DRM-enable 컨텐츠 플레이어)을 통해 서비스를 제공 받을 수 있게 된다. 하지만 DRM Client 시스템이 설치되는 신뢰할 수 없는 사용자 환경 하에서는 소프트웨어적인 모든 요소가 사용자의 제어 아래 놓이게 되며, 특히 DRM Client 시스템은 디지털 컨텐츠 확인, 즉 컨텐츠 사용을 위한 인증과 저작권 보호에 관한 기능을 위해서 민감한(sensitive) 정보, 예를 들면 사용자 인증키 암호키, 라이센스 정보 등을 포함해야 하므로, 이에 대한 정적 분석 및 동적 분석에 의해서 중요한 정보의 노출 우려가 크다. 또한 향후 배포되는 DRM-enable된 컨텐츠 역시 직접적인 공격의 대상이 될 수 있다. 이와 같이 숨겨진 정보가 노출되거나 위조될 경우 기대했던 DRM 기능을 수행할 수 없게 된다.

따라서 DRM 기반하의 안전한 디지털 컨텐츠 유통 시스템에서 수용할 수 있는 수준의 기술 제시가 이루어져야 하며, 특히 DRM Client 시스템에서 컨텐츠 사용을 위한 인증과 저작권 보호를 위해 사용되는 사용자 인증키, 암호키, 라이센스 데이터 등과 같은 민감한 정보는 본 논문에서 소개하는 Secure Storage Device (SSD)와 같은 안전한 데이터 보관소에 저장이 되어야 한다.

SSD는 윈도우 운영체제의 커널이 제공하는 System Services[3]에 대한 API 후킹[4]을 통해 윈도우 운영체제의 행동 패턴을 변화 시키는 기법을 사용해서 구현하였으며, DRM Client 시스템과 함께 설치되는 SSD는 DRM Client 시스템의 사용자가 인증키, 암호키, 라이센스 데이터 등과 같은 민감한 정보를 접근하지 못하도록 보호를 한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서 DRM 시스템의 전반적인 개요에 대해 기술하였으며, Ⅲ장에서 SSD의 설계 및 구현에 대해 기술하였다. Ⅳ장에서는 SSD의 평가 및 분석 내용을 기술하였으며, Ⅴ장에서는 본 논문의 결론을 기술하였다. 추가로 본 논문의 Appendix에는 SSD의 구현 원리를 좀더 쉽게 이해할 수 있도록 System Services에 대한 내용을 기술하였다.

Ⅱ. DRM 시스템의 개요

DRM은 암호화 기술을 사용하여 허가되지 않은 사용자로부터 디지털 컨텐츠를 안전하게 보호함으로써 컨텐츠 저작권자의 권리 및 이익을 지속적으로 보호 및 관리하는 시스템으로 정의 할 수 있다. 즉, 전자상거래를 통해 디지털 컨텐츠가 저작권자 및 유통업자의 의도에 따라 안전하고 편리하게 유통될 수 있도록 제공되는 모든 기술과 서비스 절차 등을 포함하는 개념이다.

2.1 DRM 시스템의 전체 구성 및 컨텐츠 유통 절차

DRM 시스템은 Contents Owner, Client, Shopping Mall, CH/PMS, CH/LIMS, P/G 등과 같은 노드들로 구성이 되며, 그림 1은 DRM 시스템 하에서 컨텐츠의 유통 과정을 나타낸 것이다.

DRM 환경 하에서 컨텐츠 저작권자는 판매할 디지털 컨텐츠와 컨텐츠에 적용할 비즈니스 룰을 함께 암호화 하여 Secure Container로 패키징한 후 인터넷을 통해 사용자에게 전달하거나 혹은 Shopping Mall에 등록을 하여 컨텐츠 사용자가 Shopping Mall에서 원하는 디지털 컨텐츠 상품을 검색할 수 있도록 한다. 이후 패키징된 디지털 컨텐츠의 사용자는 자신이 구매한 디지털 컨텐츠를 주위 동료들에게 e-mail, 디스켓, CD_ROM등의 매체를 통

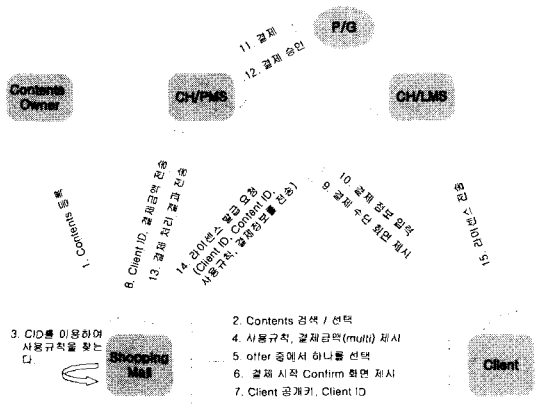


그림 1. DRM 시스템 하에서 컨텐츠 유통과정

해 자유롭게 재배포를 할 수 있다. 하지만 라이센스의 구매 없이 복사된 디지털 컨텐츠의 획득만으로는 컨텐츠의 사용이 불가능하다. DRM의 이러한 특징은 디지털 컨텐츠의 불법 사용을 방지하면서도 인터넷을 통해 급속히 확산할 수 있는 장점을 제공한다.

2.2 DRM Client 시스템

DRM Client 시스템은 최종 사용자에게 Secure Container에서 컨텐츠를 추출하여 라이센스에 정의된 사용규칙에 따라 사용자에게 디지털 컨텐츠를 재생해주는 프로그램이다. 이러한 DRM Client 시스템이 설치되는 사용자 환경은 디지털 컨텐츠의 저작권자나 유통업자 입장에서는 절대 신뢰할 수 없는 사용자 환경이며, DRM 유통 시스템의 전체 영역 중에서도 각종 부적절한 행위에 노출이 가장 많은 부분으로서 프로그램으로서 완벽한 보안성을 제공해야 한다.

그림 2는 DRM Client 시스템을 구성하는 전체 모듈 구성을 나타낸 것이며, DRM Client 시스템은 암/복호화 기능을 지원함으로써 디지털 컨텐츠에 대한 보안 서비스 강화 및 정당한 절차를 거쳐 라이센스를 구매한 사용자에 대한 인증 기능을 수행한다. 또한, 디지털 컨텐츠의 라이센스에 따라서 복제 방지, 사용 횟수 제한, 사용 기간 제한 등과 같이 다양한 사용 규칙을 적용해서 디지털 컨텐츠의 유료 사용을 위한 과금관리 기능의 지원 및 영화, 음악, 만화, 문서, 게임 등 다양한 디지털 컨텐츠의 특성에 따라서 기존의 플레이어에 플러그인을 추가하거나 특정 컨텐츠만을 재생시킬 수 있는 자체 플레이어를 통해서 사용자에게 이미 전달된 디지털 컨텐츠의 불법 접근을 통한 해킹이나 디지털 컨텐츠 플레이어에서 복호화된 데이터의 복사, 저장, 인쇄, 화면 캡쳐 등을 방지하는 기능을 제공한다.

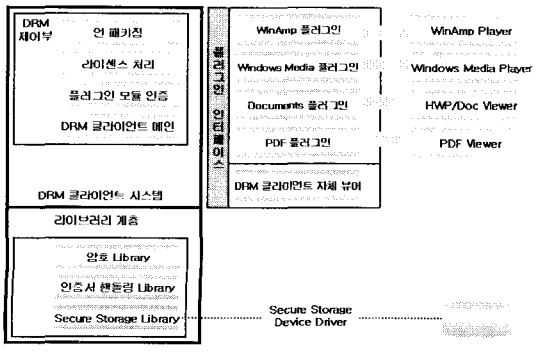


그림 2. DRM Client 시스템의 세부 모듈

특히, 그림 2의 SSD Driver는 라이센스 데이터, 컨텐츠 복호화 키 및 Secure Container에 들어있는 비즈니스 룰 등의 데이터를 사용자가 임의로 조작하지 못하도록 하여 신뢰할 수 없는 DRM Client 시스템 하에서 디지털 컨텐츠를 안전하게 보호함으로써 보다 안정적으로 디지털 컨텐츠가 유통될 수 있는 DRM 환경을 제공하게 된다.

Ⅲ. Secure Storage Device (SSD) 설계 및 구현

3.1 Secure Storage Device (SSD)의 구조

DRM Client 시스템의 한 구성 요소인 SSD는 컨텐츠에 대한 라이센스 정보, 컨텐츠 복호화 키 정보, 모듈간 인증 정보 등의 관리 및 보호를 목적으로 하며, 디지털 컨텐츠의 불법적인 사용을 방지하기위한 DRM Client 시스템의 핵심 모듈이다.

SSD가 데이터 암호화를 위해 사용하는 키를 마스터 키라 부르며, 이러한 마스터 키는 윈도우 파일 시스템 및 레지스트리에 저장이 된다. 하지만 이러한 마스터 키를 사용자 모르게 윈도우 파일 시스템 및 레지스트리에 숨긴다는 것은 근본적으로 불가능한 일이다. 이러한 문제의 해결책으로 본 논문에서 소개하는 SSD는 윈도우 운영체제의 operation을 변형시키는 기법을 제안하였다.

SSD는 신뢰할 수 없는 DRM 클라이언트 환경에서 사용자가 마스터 키에 접근을 하지 못하도록 하기 위해 파일 시스템 및 레지스트리를 핸들링하는 System Services API[3] 함수들에 대한 후킹 기법을 사용해서 마스터 키를 보호하고 있으며, Win9x 계열에서는 파일 시스템 API 및 레지스트리 핸들링 API 함수를 후킹하는 가상장치 드라이버 (VxD)[5] 형태로 구현을 하였으며, WinNT 계열에서는 윈도우 커널의 핵심 요소인 NTOSKRNL[6] 모듈에서 제공하는 System Services API 함수들 중 파일 및 레지스트리를 핸들링하는 API 함수들을 후킹하는 디바이스 드라이버[7][8][9]의 형태로 구현을 하였다. Win9x 계열과 WinNT 계열에서의 후킹 기법은 거의 비슷하므로 본 논문에서는 시스템의 구조가 좀더 복잡한 WinNT 계열의 SSD의 구현 기법에 대해 기술한다.

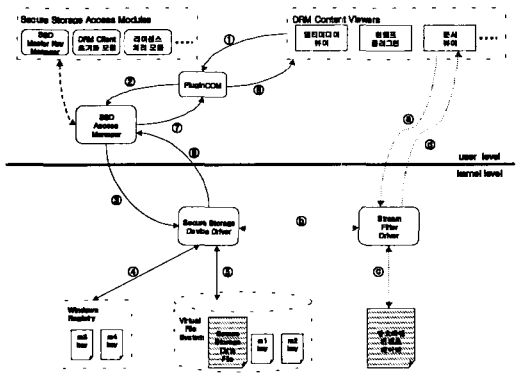


그림 3. Secure Storage Device (SSD)의 구조

그림 3은 DRM Client 시스템을 구성하는 각 모듈과의 상호 작용 및 DRM Client 시스템 내부의 SSD의 구조를 나타낸 것이다. 그림 3에서 SSD 모듈은 크게 3부분으로 나누어지며, 각각의 모듈은 SSD Access Manager, SSD Driver, Virtual File System으로 구성이 된다. 이 밖에 DRM Client 시스템의 구성요소로서 SSD Master Key Manager, DRM Client 초기화 모듈, 라이센스 처리 모듈, PlugInCOM 모듈 등이 있으며, 이들 모듈은 SSD Access Manager를 통해 SSD가 관리하는 데이터 파일에 접근하게 된다. 즉, SSD Access Manager는 User Mode 영역에서 동작하는 DRM Client 시스템의 각 모듈들과 Kernel Level 영역에서 동작하는 SSD Driver와의 인터페이스를 담당하는 모듈이다.

DRM Client 시스템에서 컨텐츠 재생을 담당하는 멀티미디어 뷰어, 윈앰프 플러그인, 문서 뷰어 등의 모듈들을 DRM Content Viewers라고 부르며, 이들 모듈이 암호화된 컨텐츠를 재생할 때는 PlugInCOM 모듈에게 컨텐츠 복호화 서비스 요청을 통해 컨텐츠 재생 서비스를 수행하게 된다. 또한, Kernel Level 영역에서 동작하는 Stream Filter Driver는 메모리 스트림 인터페이스를 제공하지 않는 재생 애플리케이션들을 위해 파일 시스템 드라이버 수준에서 복호화 서비스를 제공하는 모듈로서 어떠한 재생 애플리케이션에 대해서도 복호화 스트림을 제공하기 위한 모듈이다.

다음은 멀티미디어 뷰어가 PlugInCOM 인터페이스를 통해 암호화된 컨텐츠 데이터를 복호화해서 재생 서비스를 수행하는 세부 절차를 나타낸 것이다.

① 멀티미디어 뷰어는 PlugInCOM 모듈에게 복호화된 컨텐츠 데이터를 요청한다.

② PlugInCOM 모듈은 SSD Access Manager 모듈에게 컨텐츠 복호화 키를 요청한다.

③ SSD Access Manager 모듈은 SSD Driver에게 컨텐츠 복호화 키를 요청한다.

④ Storage Device Driver는 SSD Access Manager 모듈에 대한 인증 작업을 수행한 이후, 윈도우 레즈스트리에 저장되어 있는 마스터키 m3, m4를 읽어온다.

⑤ Storage Device Driver는 Virtual File System에 저장되어 있는 마스터키 m1, m2 및 하드웨어 고유정보(하드디스크 고유 넘버, CPU 일련번호, 랜카드 일련번호 등)를 읽어온다. 이후 마스터키 m1, m2, m3, m4 및 하드웨어 고유정보를 조합해서 Secure Storage Data File을 복호화 할 수 있는 복호화 키를 생성한 후, Secure Storage Data File을 복호화 한다.

⑥ 복호화된 Secure Storage Data File에서 컨텐츠 복호화 키를 추출한 후 SSD Access Manager 모듈에게 전달한다.

⑦ SSD Access Manager 모듈은 컨텐츠 복호화 키를 PlugInCOM 모듈에게 전달한다.

⑧ PlugInCOM 모듈은 Secure Container 내부에 암호화된 컨텐츠 데이터를 컨텐츠 복호화 키를 사용해서 복호화 한 이후 복호화된 컨텐츠 데이터에 대한 스트림 서비스를 멀티미디어 뷰어에게 제공한다.

특히, SSD Driver는 일반 윈도우 시스템의 Device Driver와 마찬가지로 IOCTL 코드를 통해 User LeveL 영역의 애플리케이션과 통신을 수행하며, 본 논문에서는 SSD Access Manager 모듈과 통신을 하게 된다. 그러나 SSD Driver와 SSD Access Manager 모듈 사이에 주고받는 IOCTL 코드를 공격자가 알고 있거나 혹은 추론을 통해 공격을 시도할 수 있다는 문제점이 있다. 이를 해결하기 위해 본 논문에서는 SSD Driver에게 서비스를 요청하는 애플리케이션에 대한 인증 메커니즘을 도입하였으며, 인증 메커니즘은 다음과 같다.

① 특정 애플리케이션이 SSD Driver에게 서비스를 요청한다.

② SSD Driver는 현재 실행되고 있는 애플리케이션의 프로세스의 ID 및 실행 이미지 데이터 정보를 구한다.

③ SSD Driver는 현재 수행되고 있는 애플리케이

션의 실행 이미지 파일에 대한 MD5 해쉬 값을 계산한 후 SSD Driver가 초기 설치될 때 함께 저장된 SSD Access Manager 모듈의 실행 이미지 데이터의 해쉬 값과 비교를 한다.

④ 두 해쉬 값이 동일할 경우에 SSD Driver는 서비스를 제공하며, 그렇지 않을 경우에는 서비스 거부 메시지를 전송한다.

3.2 Secure Storage Device (SSD)의 구현

SSD는 System Services API Hooking 기법을 사용해서 구현한 디바이스 드라이버이며, 본 절에서는 Hooking의 개념 및 System Service API Hooking 기법을 통한 SSD의 구현 원리에 대해서 기술한다. 추가로 Appendix에는 Windows NT 계열의 시스템 구조 및 System Services의 개념에 대해 보다 상세히 기술하였다.

3.2.1 Hooking의 개념

후킹이란 특정 프로그램 혹은 OS 자체의 실행 코드 영역을 가로채어 이 실행 코드 영역에서 발생하는 행동 혹은 이벤트를 감시하거나 변경하는 것을 말한다. 이러한 후킹에는 User-Level Hooking 및 Kernel-Level Hooking과 같은 2가지 타입이 있다.

User-Level Hooking의 경우에는 다시 윈도우 메시지 후킹[4], DLL Injection[4], API 후킹[4] 기법 등으로 세분화 되어진다. 메시지 후킹을 사용하게 될 경우에는 특정 윈도우의 내용을 얻는 것은 물론 이러한 윈도우의 컨트롤까지도 가능하게 된다. DLL Injectin 기법의 경우는 동일한 프로세스 영역이 아닌 다른 프로세스 영역에서 발생하는 이벤트 메시지를 가로채고자 할 경우에 사용이 되며, 이러한 DLL Injection 기법의 원리는 하나의 DLL을 만들어서 이 DLL을 후킹 하고자 하는 프로세스 영역으로 삽입한 다음 DLL의 후킹 결과를 후킹을 시도한 메인 프로세스에게 전달하는 방식이다. 이러한 메시지 후킹이나 DLL Injection 기법을 사용해서 윈도우 프로그램이 항상 최상단에 위치하게 한다던지 특정 애플리케이션에서 발생하는 이벤트를 가로채어 새로운 이벤트 메시지로 교체를 하는 것이 가능하게 된다. 즉, 윈도우 애플리케이션에서 발생하는 모든 이벤트 메시지를 가로챌 수 있으며, 결국은 모든 애플리케이션의 이벤트 메시지에 대한 컨트롤이 가능하게 된다. API 후킹의 경우는 다양한 API 함수의 호출을 가로채어 이 함수들의 행동 패턴을 변경 시킬 수 있는 기법으로 메시지 후킹이나 DLL Injection 기법보다 더욱 강력한 컨트롤 기능을 제공한다.

Kernel-Level Hooking의 경우는 VxD나 device driver 프로그래밍을 통해 윈도우 Kernel이 제공하는 Primitive Function들을 후킹하는 기법을 말하며, User-Level의 API 후킹과 동일한 개념이다. User-Level API 후킹과 Kernel-Level API 후킹의 차이는 후킹의 영역이 User-Level API 후킹의 경우 User Level에 제한되어 있다는 것이며, Kernel-Level API 후킹의 경우는 윈도우 시스템 전체 영역에 대한 후킹이 가능하다는 것이다. 또한 Kernel Level API 후킹의 경우는 Primitive Function들에 대한 내용이 undocumented 되어있기 때문에 이들 함수들에 대한 parameter들의 해독이 먼저 수행되어야 한다는 어려움이 있다.

3.2.2 System Service API Hooking

User Mode 영역의 NTDLL 모듈은 Kernel Mode 영역의 NTOSKRNL 모듈이 제공하는 System Services에 대한 인터페이스를 제공하며, Win32 API 함수들은 이러한 NTDLL 모듈의 인터페이스를 통해 NTOSKRNL 모듈이 제공하는 System Services를 요청하게 된다. 즉, KERNEL32이나 ADVAPI32 모듈 등이 제공하는 Win32 API 함수들은 NTDLL 모듈이 제공하는 함수를 통해 System Services를 요청하며, NTDLL 모듈은 Win32 API 함수들로부터 전달받은 System Services 요청 메시지를 NTOSKRNL 모듈에게 전달하게 된다.

그림 4는 System Service Descriptor Table (SSDT)과 System Service Parameter Table (SSPT)의 구조 및 User Mode의 애플리케이션에서 Win32 API를 통해 요청한 System Services가 어떻게 Kernel Mode로 전달되는지를 나타낸 것이다.

다음은 NTDLL 모듈이 Win32 API 함수들로부터 요청받은 System Services를 NTOSKRNL 모듈에게 전달하는 과정은 다음과 같다.

① User Mode 영역의 애플리케이션이 Win32 API 호출

② 애플리케이션이 호출한 Win32 API는

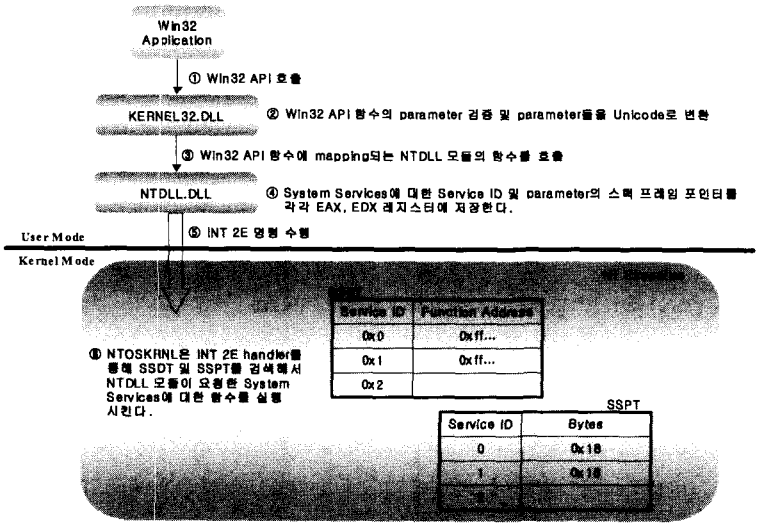


그림 4. System Services의 요청 과정 및 SSDT 및 SSPT의 구조

KERNEL32이나 ADVAPI32 모듈 등과 같은 Subsystem DLL들이 제공을 하며, 이러한 Subsystem DLL들은 먼저 Win32 API 함수의 파라미터를 검증한 이후 파라미터들을 모두 Unicode 포맷으로 변환을 시킨다. 이후 Win32 API 함수에 mapping 되는 NTDLL 모듈의 함수를 호출한다.

③ NTDLL 모듈은 요청받은 Stystem Services에 대한 Service ID를 EAX 레지스터에 저장하고 Win32 API 함수들에 대한 parameter 들의 스택 프레임에 대한 포인터를 EDX 레지스터에 저장한 이후 INT 2E 명령을 수행해서 processor가 Kernel Mode에서 수행되도록 한다.

④ processor는 IDT (Interrupt Descriptor Table)로부터 INT 2E 명령에 대한 handler를 수행한다.

⑤ INT 2E handler는 User Mode의 스택에 저장된 parameter 들의 데이터를 Kernel Mode의 스택으로 복사를 하며, 이들 스택 프레임에 대한 base pointer는 EDX 레지스터에 저장된 값에 의해 식별이 된다.

⑥ NTOSKRNL 모듈 내부에는 System Service Descriptor Table (SSDT)이 있으며 이 테이블에는 System Services에 대한 함수들의 포인터와 이들 함수 포인터들을 식별하기 위한 Service ID의 값이 함께 저장되어 있다. INT 2E handler는 EAX 레지스터에 의해 전달된 Service ID 값을 통해 System Service Descriptor Table (SSDT)에서 Service ID에 mapping 되는 함수의 포인터를 찾아 System Services에 대한 요청을 수행하게 된다. 또한 NTOSKRNL 모듈 내부에는 System Service Parameter Table (SSPT)이 있으며 이 테이블은 INT 2E handler에게 각 Service 함수들의 parameter들에 대한 바이트 수를 제공한다.

그림 4에서 Executive 컴포넌트의 하나인 NTOSKRNL 모듈은 ServiceDescriptorTable이라는 구조체 타입의 변수를 export 시키며, SSD Driver는 이 export된 변수를 통해 System Service Descriptor Table에 접근한 후 서비스 함수의 포인터 변경을 통해 후킹을 시도하는데 이러한 기법을 System Services API Hooking 이라고 한다.

3.2.3 Secure Storage Device (SSD) Driver

Kernel Executives 컴포넌트에 해당되는 NTOSKRNL 모듈은 다양한 System Services 함수를 제공하며, 이러한 Services 함수들은 User Mode 영역에 있는 Win32 Subsystem DLL 모듈들이나 Kernel 컴포넌트들에게 다양한 서비스를 제공한다.

SSD Driver의 구현 원리는 NTOSKRNL 모듈

이 제공하는 System Services API의 후킹을 통해 Windows OS의 행동 패턴을 변화 시켜서 파일 및 레지스트리 키의 형태로 저장된 마스터 키의 접근을 원천 봉쇄하는 것이다. 예를 들어 User Mode 영역에 있는 임의의 애플리케이션에서 파일을 핸들링하기 위해 파일을 열게 될 경우 애플리케이션은 Win32 Subsystem DLL들에서 제공하는 fopen() 함수를 사용하게 되며, 이 파일 오픈 요청에 대한 메시지는 NTOSKRNL에서 제공하는 ZwCreateFile() 함수나 ZwOpenFile() 함수를 호출하게 되는 것이다. 즉, User Mode 영역의 애플리케이션에서 사용하는 대부분의 API 함수에 대한 서비스 요청은 Kernel Mode 영역에 존재하는 NTOSKRNL 모듈로 전달되어서 User Mode API 함수에 mapping되는 System Service API 함수를 다시 호출하게 되는 것이다.

SSD Driver는 마스터 키를 사용해서 라이센스 데이터, 컨텐츠 복호화 키 등과 같은 DRM Client 시스템 사용자에게 노출되어서는 안되는 중요한 데이터를 암호화 시켜서 저장하며, 이러한 마스터 키는 각각 파일과 레지스트리 키의 형태로 저장이 되어있다. SSD Driver는 DRM Client 시스템의 사용자가 마스터 키의 데이터에 접근을 못하도록 하기위해 System Services API 함수들을 후킹해서 DRM Client 시스템 사용자가 마스터 키에 접근하게 되더라도 OS 자체가 마스터 키에 대한 접근을 거부하도록 구현하였다.

NTOSKRNL 모듈 내부에는 System Service Descriptor Table (SSDT) 이라는 어레이 형태의 테이블이 있으며, 이 테이블 내부에는 NTOSKRNL 모듈이 제공하는 수많은 System Service API 함수들에 대한 포인터 주소가 저장되어 있다. SSD

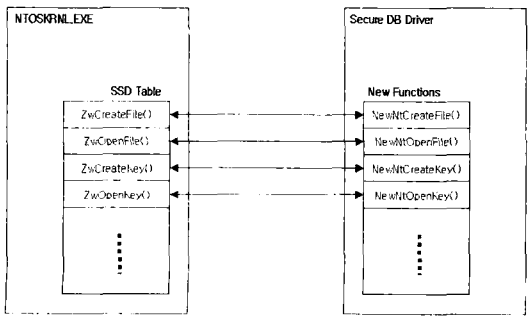


그림 5. Secure Storage Device (SSD) Driver의 System Service API 후킹 방법

Driver는 이러한 System Services API 함수들 중 ZwCreateFile, ZwOpenFile, ZwQueryKey, ZwCreateKey, ZwOpenKey, ZwSetValueKey, ZwDeleteValueKey와 같은 함수들의 후킹을 통해 파일 및 레지스트리 키의 형태로 저장된 마스터 키에 대해 DRM Client 시스템 이외의 어떠한 모듈도 접근을 못하도록 하였다.

그림 5는 SSD Driver가 NTOSKRNL 모듈이 제공하는 System Service API 함수를 후킹하는 방법을 나타낸 것이다.

여기서 SSD Driver는 NewNtCreateFile(), NewNtOpenFile(), NewNtCreateKey(), NewNtOpenKey() 등과 같은 새로운 서비스 함수들을 구현한 이후 이들 새로운 함수들에 대한 포인터 주소를 System Service Descriptor Table (SSDT) 내부의 함수들의 주소와 교체를 시키게 된다. 함수들의 주소가 교체된 이후에는 SSD Driver가 제공하는 새로운 시스템 서비스 함수가 호출되며, SSD Driver는 이들 새로운 서비스 함수 내부에서 마스터 키에 대한 접근 요청을 검사한 후 마스터 키에 대한 서비스를 거부하게 된다.

SSD Driver는 윈도우 시스템이 초기 부팅될때 함께 실행되고 이후 윈도우 시스템이 종료되면서 함께 종료가 된다. 따라서 마스터 키에 접근하는 시스템 서비스 API 함수들에 대한 후킹은 항상 실행되고 있는 상태이므로 DRM Client 시스템의 사용자는 마스터 키의 정보를 획득할 수 없게 되며, 만일 사용자가 SCM (Service Control Manager)을 사용해서 SSD Driver를 강제로 언로드시킬 경우에는 윈도우 시스템이 자동으로 다운되도록 구현되어 있다.

SSD Driver와 DRM Client 시스템 사이에는 IOCTL 코드를 사용해서 서로 통신을 하게 되는데, 이 경우 동일한 IOCTL 코드를 사용하는 외부의 애플리케이션이 SSD Driver에게 서비스를 요청할 경우 문제가 될 수 있다. 이를 해결하기 위해 SSD Driver는 서비스를 제공하기 전에 먼저 서비스를 요청한 애플리케이션에 대한 프로세스 인증 작업을 수행한다.

Ⅳ. 평가 및 분석

4.1 Secure Storage Device (SSD) Driver의 실행 결과

SSD Driver는 윈도우 시스템이 부팅 됨과 동시에 함께 로드되며, 이후 마스터 키에 대한 접근 감시를 통해 DRM Client 시스템 모듈을 제외한 외부의 어떠한 애플리케이션이 또는 디바이스 드라이버가 마스터 키에 접근하는 것을 봉쇄한다.

그림 6은 SSD Driver가 설치되지 않은 상황에서 윈도우 레지스트리에 저장된 마스터 키의 정보를 나타낸 것이며, 그림 7은 SSD Driver가 설치된 이후의 DRM Client 시스템 사용자가 마스터 키에 접근을 시도하게 될 경우 마스터 키의 접근이 거부되는 장면을 나타낸 것이다. 두 그림에서 보이는 바와 같이 마스터 키는 W2K_SDBDrv라는 레지스트리 키의 하위에 저장되어 있으며, SSD Driver가 설치된 상태에서는 그림 7과 같이 에러 메시지 다이얼로그 박스가 나타나 마스터 키에 대한 접근을 할 수 없게 되며, W2K_SDBDrv라는 레지스트리 키의 하위에 저장된 서브 키들도 보이지 않게 된다.

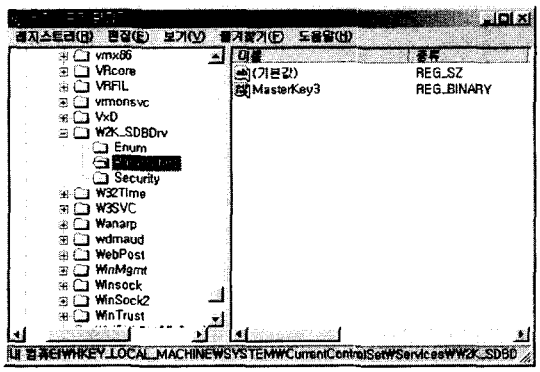


그림 6. Secure Storage Device (SSD) Driver가 설치되지 않은 상태에서 마스터 키에 접근하는 경우

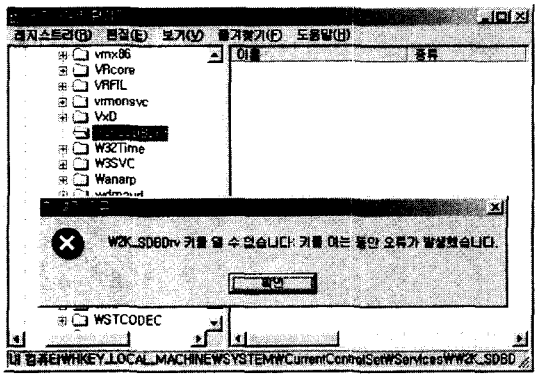


그림 7. Secure Storage Device (SSD) Driver가 설치된 상태에서 마스터 키에 접근하는 경우

4.2 Secure Storage Device (SSD)와 다른 Storage 장치와의 비교

DRM Client 시스템은 Secure Container에 포함되어 있는 비즈니스 룰 및 사용자가 구매한 라이센스 정보에 따라서 암호화된 디지털 컨텐츠의 재생 서비스를 제공하며, 이러한 서비스를 제공하는 과정에서 DRM Client 시스템은 컨텐츠 복호화 키, 라이센스 데이터, 모듈간 인증 정보 등과 같은 민감한 정보에 대한 보호를 필요로 하게 된다. 본 논문의 DRM Client 시스템은 이러한 민감한 정보를 암호화해서 저장을 하며, 민감한 정보의 암호화에 사용되는 키를 마스터 키라 부른다. 하지만 이러한 마스터 키 또한 신뢰할 수 없는 사용자 환경에 저장되어야 하기 때문에 사용자에게 쉽게 공격의 대상이 될 수 있다. 이러한 공격의 대응 방안으로 HSM (Hardware Security Module)과 같은 Tamper-Resistant한 Storage에 저장하는 방안이 있다. 하지만 이러한 하드웨어로 구성된 장비는 상당히 고가이기 때문에 현실적으로 DRM Client 시스템에 적용하기는 아직 어려우며, 특히 HSM 장비의 경우 PKI 기반 시스템에서 루트 인증서를 보호하기 위해 CA (Certificate Authority)와 같은 큰 규모의 기관에서 사용하기에 적합한 장비이며, DRM Client 시스템에 적용하기에는 부적합하다. 하지만 본 논문의 SSD는 Tamper-Resistant한 Storage를 소프트웨어적으로 구현하여 DRM Client 시스템에 쉽게 적용이 가능하다는 장점을 제공한다.

4.3 Secure Stroage Device에 대한 공격

SSD는 DRM Client 시스템이 설치될 때 함께 설치되며, 설치와 동시에 SSD는 실행이 된다. 따라서 SSD가 보호하는 마스터 키, 라이센스 정보, 컨텐츠 복호화 키 정보, 모듈 인증 정보 등의 데이터 접근이 원천 봉쇄된다. 하지만 공격자가 새로운 후킹 드라이버 모듈을 제작해서 SSD 드라이버가 로드되기 전에 먼저 마스터 키에 접근할 경우 마스터 키는 노출이 되게 된다. 이러한 공격 시도를 막기 위해 SSD는 Virtual File System을 함께 구현해야 하며, 이 Virtual File System 내부에 마스터 키 및 암호화된 데이터를 저장해야 한다.

Virtual File System에 데이터가 저장될 경우 공격자는 Virtual File System의 로직을 알아야

만 공격이 가능하게 되며, 이를 위해 공격자는 SSD에 대한 reverse engineering[10] 공격을 시도해야 한다. 하지만 reverse engineering 이라는 기술 자체가 상당히 고난이도의 스킬을 요구하기 때문에 좀처럼 시도하기 어려운 기법이다. 또한, 본 논문에서는 이러한 reverse engineering 공격에 대응하기 위해 SSD에 Code Obfuscation[11] 기법을 적용해 공격자가 쉽게 프로그램 로직을 분석할 수 없도록 하는 한편 SSD Driver가 메모리에 로드 된 이후 자신의 실행이미지 파일의 데이터를 메모리로 읽어 들인 후 하드 디스크에서 삭제한 후 윈도우 시스템이 종료될 때 다시 하드 디스크에 기록하는 기법 및 SSD Driver가 설치된 하드 디스크가 다른 윈도우 시스템에 옮겨져서 실행 될 경우를 대비해 SSD Driver는 초기 로딩 시 자신이 실행되고 있는 시스템의 하드 디스크 고유 정보나 CPU 고유 정보, MAC 어드레스등과 하드웨어 고유 정보를 분석해서 SSD Driver가 초기 설치 시 저장된 정보와 일치하지 않을 경우 실행을 멈추고 자신의 실행 이미지 데이터를 하드 디스크 상에서 완전히 제거하는 기법을 고안하였다. 또한 공격자가 윈도우 시스템을 안전모드로 부팅을 하더라도 SSD Driver가 자동으로 로드 되도록 구현 하였다.

V. 결 론

컴퓨터 기술과 정보통신 기술의 발전은 지식과 정보가 부가가치를 창출하는 정보화 사회로 이끄는 원동력이 되고 비디오, 오디오, 텍스트, 이미지 등의 디지털 컨텐츠를 처리하는 멀티미디어 기술의 발전 또한 네트워크의 초고속화와 더불어 정보화 시대를 가속화하고 있으며, 이에 발맞추어 디지털 컨텐츠의 불법 사용방지 및 저작권 보호 기술인 DRM에 대한 연구도 국내외적으로 활발히 진행되고 있는 추세이다.

디지털 컨텐츠 유통 환경에서 컨텐츠의 안전한 보호를 위해서는 신뢰할 수 없는 사용자 환경에 설치되는 DRM Client 시스템의 안전성 여부가 무엇보다도 중요하며, 이를 위해 본 논문에서는 DRM Client 시스템의 Security Architecture에 대한 면밀한 분석을 행하였으며, 또한 디지털 컨텐츠의 보호를 위해 DRM Client 시스템에서 사용되는 다양한 암호 키들의 종류, 용도 및 이들의 연관관계 정의 및 안전한 키 관리 메커니즘에 대한 요구 사항을 정의해 DRM Client 시스템에 적용하였다.

DRM Client 시스템은 디지털 컨텐츠 저작권자의 가장 큰 요구 사항인 저작권 보호 및 디지털 컨텐츠 보호를 위한 핵심 기능을 수행하며, 이러한 기능을 수행하기 위해서는 컨텐츠 보호를 위해 요구되는 암호키나 라이센스 정보와 같은 중요한 데이터는 SSD와 같은 안전한 보관소에 저장이 되어야 하며, 이렇게 저장된 데이터는 DRM Client 시스템의 사용자 조차도 접근을 허락해서는 안 된다. 이러한 기능을 제공하기 위해서는 정보 은닉 기법, 모듈 인증 메카니즘, reverse engineering 방지를 위한 obfuscation 기법 등과 같은 다양한 기술이 요구되는데, 본 연구에서는 이러한 다양한 기법들에 대한 연구를 진행하였으며, 특히 정보 은닉 기법의 일환으로 System Services API Hooking 기법을 적용한 SSD를 개발하였다. 향후 연구에서는 obfuscation 기법을 SSD에 적용하는 방안 및 메모리 덤핑을 통해 마스터 키와 같은 중요한 데이터의 누출을 막기 위한 기법을 개발해야 할 것이다.

참 고 문 헌

[1] Joshua Duhl, Susan Kevorkian, "Understanding DRM Systems" *An IDC* White Paper, 2001

[2] *Digital Rights Management for Ebooks: Publisher Requirements version 1.0*, AAP, 2000

[3] SVEN B. SCHREIBER, *UNDOCUMENTED WINDOWS 2000 SECRET*, ADDISON- WESLEY, pp. 265-289, 1999

[4] Jeffrey Richter, *Programming Application for Microsoft Windows*, Microsoft, pp. 751-850, 1999

[5] Karen Hazzah, *Writing Windows VxDs and Drvice Drivers*, CMP Books, 1997

[6] David Solomon, Mark Russionovich, *Inside Microsoft Windows 2000*, Microsoft

[7] Chris Cant, *Writing Windows WDM Device Drivers*, R&D, 1999

[8] Edward N. Dekker, Joseph M. Newcomer, *Developing Windows NT*

Device Drivers: A Programmer's Handbook, ADDISON-WESLEY, 1999

[9] Walter Oney, *Programming the Microsoft Windows Driver Model*, Microsoft

[10] Fravia, Fravia's Pages of Reverse Engineering, see http://www.woodmann.com/fravia/

[11] C. Collberg, C. Thomborson and D. Low, "A Taxonomy of Obfuscating Transforms", Technical Report Number 148, Dept. of Computer Science, University of Auckland, New Zealand, July 1997.

APPENDIX

A. Windows NT 계열의 시스템 구조 및 System Services의 개념

System Services란 Windows OS에 의해 제공되는 함수들의 집합을 말하며, 이러한 함수들을 System Services 혹은 Primitive Functions이라고 한다.

Windows 2000과 같은 WinNT 계열의 OS는 시스템이 크게 User Mode 영역과 Kernel Mode 영역으로 나누어지며 User Mode 영역에는 Win32 Subsystem, POSIX Subsystem, OS/2 Subsystem과 같은 다양한 컴포넌트들이 있으며, 이중 Win32 Subsystem의 경우 윈도우 애플리케이션들에게 C Runtime Library나 MFC Library와 같은 다양한 종류의 API (Application Programming Interface)를 제공한다. Kernel Mode 영역에 존재하는 시스템 컴포넌트로는 Executive 컴포넌트, IO Manager, Memory Manager, Device Driver, HAL (Hardware Abstraction Layer) 등과 같은 다양한 컴포넌트들이 있다.

그림 8은 Windows NT계열의 System에 대한 전반적인 구성 요소를 나타낸 것이며, User Mode 영역의 모든 Subsystem들에서 제공하는 API는 Kernel Mode 영역의 Executive 컴포넌트가 제공하는 System Services를 사용해서 대부분의 핵심적인 기능이 구현되어 있다. 특히, Executive 컴포넌트에 속하는 NTOSKRNL 모듈은 윈도우 커널을 구성하는 핵심 모듈의 하나로서 Windows System 컴포넌트에게 다양한 System Service를 제공한다.

Windows 애플리케이션 개발자들은 Win32 Subsystem에 포함되어있는 Kernel32, User32, Gdi32, Advapi32와 같은 다양한 DLL들을 링크시켜서 이들 DLL이 제공하는 API 함수를 호출하게 되는데 이 과정에서 DLL API 함수들은 내부적으로 System Services를 호출하게 된다. POSIX API 함수들의 경우도 마찬가지로 내부적으로는 System Services를 호출하게 된다.

그림 9는 User Mode 영역의 Win32 애플리케이션이 CreateProcess() 함수를 호출해서 새로운 프로세스를 생성하는 경우와 POSIX 애플리케이션이 fork() 함수를 호출해서 새로운 프로세스를 생성할 때 Kernel Mode 영역의 Executives 컴포넌트가 제공하는 System Services에 어떻게 mapping 되는지를 나타낸 그림이다. 그림 9에서는 Win32 애플리케이션과 POSIX 애플리케이션이 새

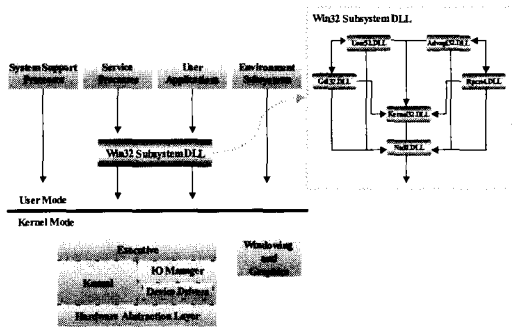


그림 8. Windows NT 계열의 System Architecture

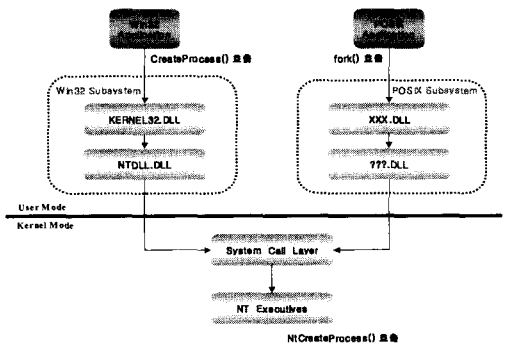


그림 9. Win32와 POSIX 애플리케이션이 System Services를 호출하는 과정

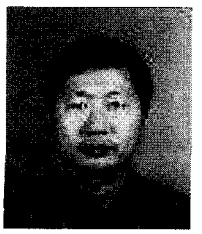
로운 프로세스를 생성하기 위해 User Mode 영역에서 서로 상이한 함수를 호출하지만 Kernel Mode 영역의 Executives 컴포넌트는 NtCreateProcess() 함수를 동일하게 호출하게 된다. 따라서 System Services란 Unix 환경에서의 System Call과 유사한 개념이다.

〈著者紹介〉

이 기 정(Kijung Lee) 학생회원
2002년: 세종대학교 컴퓨터공학과 졸업
2004년: 세종대학교 소프트웨어공학과 석사
2004년~현재: 장미디어 연구원
〈관심분야〉 DRM, 소프트웨어 보안

권 태 경(Taekyoung Kwon) 종신회원
1992년: 연세대학교 컴퓨터과학과 졸업
1995년: 연세대학교 컴퓨터과학과 석사
1999년: 연세대학교 컴퓨터과학과 박사
1999년~2000년: U.C. Berkeley Post-Doc. (과학재단 지원)
2001년~현재: 세종대학교 컴퓨터공학부 컴퓨터소프트웨어학과 조교수, 정보보호학회 편집위원, TTA 암호분과 특별위원
〈관심분야〉 정보보호, 암호프로토콜, 네트워크 보안, 무선 네트워크 등

황 성 운(Seong-woon Hwang) 회원
1994년: 서울대학교 수학과 졸업
1998년: 포항공대 정보통신학 석사
2001년~현재: KAIST 전산학 박사과정
1994년~1996년: LG-CNS 소프트웨어 엔지니어
1998년~현재: 한국전자통신연구원 디지털콘텐츠연구단 선임연구원
〈관심분야〉 전자현금 프로토콜, DRM, 암호시스템, PKI 기반 응용, 형식 기법 등



윤 기 송(Ki-song Yoon) 회원
1984년: 부산대학교 조선공학과 졸업
1988년: The City University of New York 전산학 석사
1993년: The City University of New York 전산학 박사
1993년~현재: 한국전자통신연구원 디지털콘텐츠연구단 책임연구원
〈관심분야〉 디지털 컨텐츠, DRM 등