온라인 실행 코드 기술과 암호화 기반 안드로이드 앱 불법 복제 방지 시스템의 융합 설계

김희선[1], 김지현[2], 김성렬[3], 김진욱[4]

A Hybrid Design of Online Execution Class and Encryption-based Copyright Protection for Android Apps

Hee-Sun Kim[1], Ji-Hyun Kim[2], Sung-Ryul Kim[3], Jin-Wook Kim[4]

요 약

안드로이드는 기존의 모바일 운영체제와는 다르게 소스 코드를 모두 공개함으로써 누구라도 이를 이용하여 소프트웨어와 기기를 만들어 판매할 수 있도록 하였다. 단말기 제조사와 개발자가 안드로이드 내부 파일에 쉽게 접근할 수 있다. 안드로이드 애플리케이션의 불법 복제율이 97%[1]에 달할 정도이다. 안드로이드 애플리케이션의 지적 창작물을 보호하기 위해 현재 많은 연구가 이루어지고 있다. 본 논문에서는 안드로이드 애플리케이션 복제 방지 기술 중 온라인 실행 코드 기술과 암호화 기반 안드로이드 불법 방지 시스템을 융합하고, 융합 모델이 두 기술을 개별적으로 적용 할 때보다 향상된 보안 수준을 제공하는 것을 제시할 수 있다. 이 융합모델은 사용자가 참을 수 있는 인내 허용치(User Tolerance)[2] 안에서 보안성을 보장한다. 또한 융합 모델은 개발자의 애플리케이션의 특성에 맞게 성능과 보안 수준을 다양하게 설정할 수 있도록 한다.

핵심어 : 안드로이드, 달빅, 암호화 기반 안드로이드 불법 복제 방지 시스템, 온라인 실행 코드

Abstract

Google's Android platform is a widely used operating system for mobile phones and a fully commercial application (App) market has been established. However, because of the easy access to Android internal files, unlicensed Android Apps are widely being distributed. To protect the intellectual property of Android developers, various techniques are being proposed that can prevent the illegal access to Apps. This paper combines two of the proposed techniques, namely the Online Execution Class and Encryption-based Copyright Protection and will provide a smooth scaling between the two techniques. We will show that this smooth scaling provides better protection within user tolerance boundaries than any one of the techniques can provide by itself.

Keywords : Android, Encryption-bases Copyright Protection for Android Apps, Dalvik, Online Execution Class

접수일(2013년02월24일), 심사의뢰일(2013년02월25일), 심사완료일(1차:2013년03월04일, 2차:2013년03월18일)

게재일(2013년04월30일)

[1]143-701 서울시 광진구 화양동, 건국대학교 인터넷미디어공학부 석사.
email: aznturbo09@gmail.com

[2]143-701 서울시 광진구 화양동, 건국대학교 인터넷미디어공학부 학부.
email: milktea@konkuk.ac.kr

[3]143-701 서울시 광진구 화양동, 건국대학교 인터넷미디어공학부.
email: kimsr@konkuk.ac.kr

[4](교신저자) 110-744 서울시 종로구 연건동 23, 서울대학교병원 의료정보센터.
email: gnugi@snuh.org

* 본 논문은 문화체육관광부 및 한국저작권위원회의 2012년도 저작권 기술개발사업의 연구결과로 수행했습니다.

1. 서론

안드로이드 애플리케이션의 불법 복제 방지를 위해 현재까지 많은 연구가 진행되고 있다. 예를 들어, 워터마킹[3], 안드로이드 불법 앱 탐지 연구[4], 그리고 온라인 실행 코드 기술[5], 암호화 기반 안드로이드 애플리케이션 불법 복제 방지 시스템[6]이 있다. 이 논문에서는 온라인 실행 코드와 암호화 기반 불법 복제 방지 시스템을 융합하여 사용자가 참을 수 있는 오버헤드 안에서 보안 수준을 향상시킨다. 대부분의 개발자들이 자세한 보안 수준을 지정하는 파라미터를 이해하여 설정하는 것은 힘들다. 하지만 이전의 연구에서도 유사한 환경이 존재하며 이에 따라 모든 경우에는 전문가의 도움이 필요하다. 이 논문의 후분에서는 실제 생활에서의 예시를 통해, 보안 파라미터의 계산 예를 제공한다.

온라인 실행 코드 기술은 애플리케이션의 일부를 서버에 저장하고 나머지는 사용자 디바이스로 전송한다. 사용자가 애플리케이션은 실행 시 서버에 저장된 애플리케이션이 사용자 디바이스로 전송되어 인증된 사용자만이 정상적으로 애플리케이션을 실행할 수 있다. 암호화 기반 불법 복제 방지 시스템은 사용자에게 암호화된 애플리케이션을 전송한다. 정상 구매자가 암호화된 애플리케이션을 다운로드 받은 후 설치를 시작하면 디바이스 내의 복호화 모듈에 의해 애플리케이션이 복호화가 되고 설치가 끝난다. 애플리케이션 실행이 끝나면 기기 메모리에 적재된 복호화 된 애플리케이션 데이터는 삭제가 된다. 모든 안드로이드 불법 복제 방지 기술은 오버헤드가 발생할 수밖에 없다. 이 오버헤드는 주로 애플리케이션이 실행하는 데 발생하는 시간, 네트워크 비용 또는 월별로 발생하는 이용료, 모바일 기기의 배터리 소모량 등이 있다. 하지만 이렇게 발생된 오버헤드는 사용자가 참을 수 있는 범위이여야 한다.

본 논문은 제안한 융합 모델이 사용자가 참을 수 있는 오버헤드 범위 내에 있음을 보여주며 온라인 실행 코드 기술 또는 암호화 기반 불법 복제 방지 시스템을 개별로 적용 시보다 향상된 보안 수준을 제공함을 제시한다. 또한 융합 모델은 개발자가 자유롭게 개발자의 애플리케이션의 성격에 따라 보안 수준을 조절할 수 있다. 개발자가 보안 수준의 상세한 파라미터를 이해하지 않고도 간단히 계산 방법을 제시하여 이해를 도울 것이다.

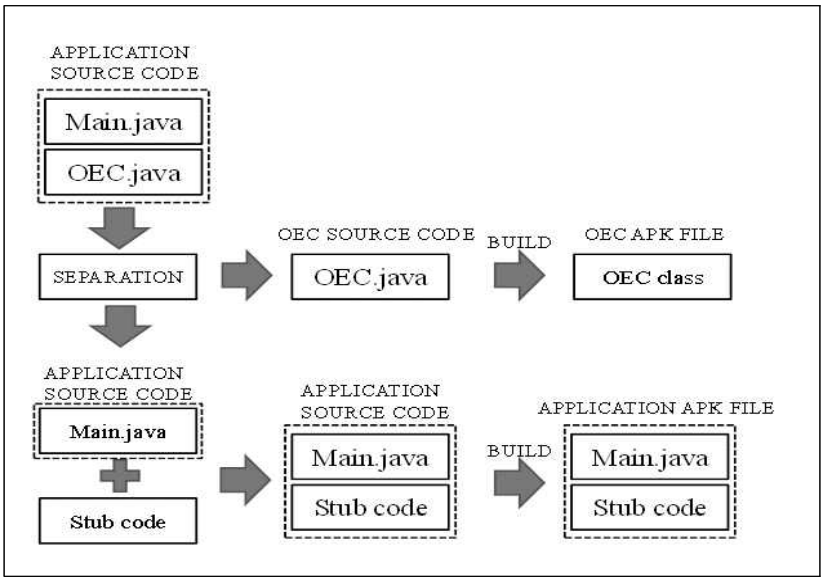
본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 두 기술을 융합해야 하는 정당성을 설명한다. 3장에서는 제안하는 융합 모델을 실제 적용한 예시를 제시한다. 마지막으로 4장에서는 본 논문의 결론을 맺는다.

2. 배경지식 및 관련연구

2.1 온라인 실행 코드 기술

온라인 실행 코드(OEC) 기술은 애플리케이션(앱) 실행 시 서버로부터 앱 전체 코드 중 클래스

일부분을 전송 받아 동적 적재하여 실행한다. 이를 위하여 앱을 개발할 때 전체 소스 코드 중 특정 클래스를 분리한다. 분리할 클래스는 JAVA 인터페이스이며 다른 사용자 정의 클래스의 상속을 받지 않아야 한다. 개발단계에서 배포용과 서버 보관용으로 2개의 APK 파일을 생성한다. 동적 적재를 위해 분리된 클래스의 메인 자바코드에 스턴(Stub)을 추가한다. 배포용 APK에 추가될 스턴은 사용자 인증 및 서버와의 통신을 수행하고 동적으로 클래스를 적재한다. 이 적재과정에서 APK에서 DEX를 추출하여 분리된 클래스를 동적으로 로드한다. 추가될 스턴은 서버와의 통신모듈과 암호화된 OEC를 받아서 복호화한 뒤에 안전한 저장소에 저장하고, 안정한 저장소에 저장된 OEC를 메모리에 동적 적재하는 모듈로 구성되어 있다.



[그림 1] 온라인 실행 코드 기술의 구조

[Fig. 1] Structure of Online Execution Class

2.2 암호화 기반 사용자 인증 및 안드로이드 앱 불법 복제 방지 시스템

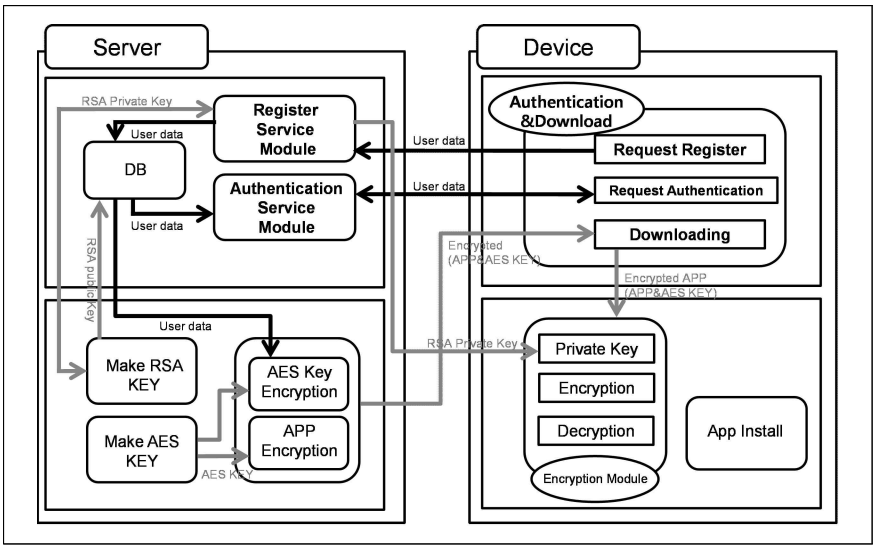
2.2.1 사용자 인증

사용자 등록 시, 사용자가 입력한 개인정보는 서버의 데이터베이스에 저장되며 사용자 인증을 할 때마다 사용된다. 이를 위하여 사용자 인증과정에서 RSA 암호화키를 생성한다. 인증과정을 마친 사용자의 개인정보는 서버에서 RSA 키 생성 모듈을 통하여 공개키와 개인키 쌍을 만든다. RSA 키 생성 모듈은 파일 형태로 공개키와 개인키를 생성한다. 키가 생성된 직후 개인키 사용자에게 전송하고 공개키는 해당 사용자의 데이터베이스에 저장한다. 전송과 전송 과정을 마친 후 서버는 생성되었던 RSA 개인 키 파일을 삭제한다.

2.2.2 모바일 디바이스

암호화된 앱을 설치하기 위하여 Dalvik VM 내에 암호화/복호화 모듈을 추가한다. 안드로이드 앱 설치과정의 일부분인 Optimization과 Verification 이전과 이후에 DEX 파일과 ODEX 파일의 checksum 검사와 offset 수정 과정에서의 에러 발생을 막기 위해서 암호화와 복호화를 수행해야 한다. 안드로이드 소스 코드 중 Dalvik 폴더에 /dexopt/Optmain.c에서 호출되는 dexZipExtractEntrytoFile 함수는 APK파일로부터 classes.dex파일을 얻는다. 이 함수는 dalvik/libdex/ZipArchive.c에 존재한다. Classes.dex 파일을 복호화하기 위해 복호화 모듈을 이곳에 삽입한다.

사용자가 앱을 실행하기 위해 앱을 클릭하게 되면 data/dalvik-cache 내에 있는 classes.dex 파일을 통해 앱이 실행된다. DEX 파일을 호출하는 부분은 dalvik/analysis/vm/dexprepare.c 내의 dvmOpenCachedDexFile 함수이며, ODEX 파일은 암호화된 상태로 저장되어 있기 때문에 이 부분에 추가한 복호화 모듈을 통해 복호화를 수행한 후 ODEX 파일을 리턴하여 앱을 실행한다.



[그림 2] 암호화 기반 안드로이드 앱 불법 복제 방지 시스템

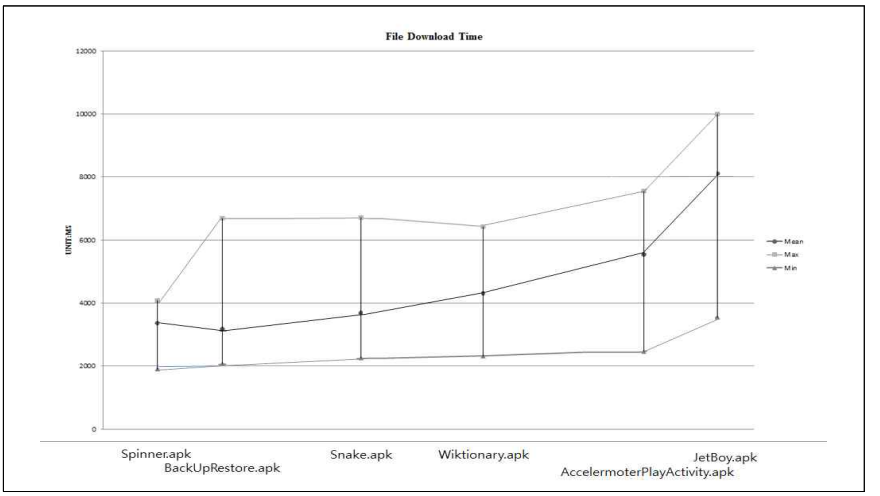
[Fig. 2] Structure of Encryption-based Copyright Protection

3. 융합 모델의 정당성

3.1 지연과 비용

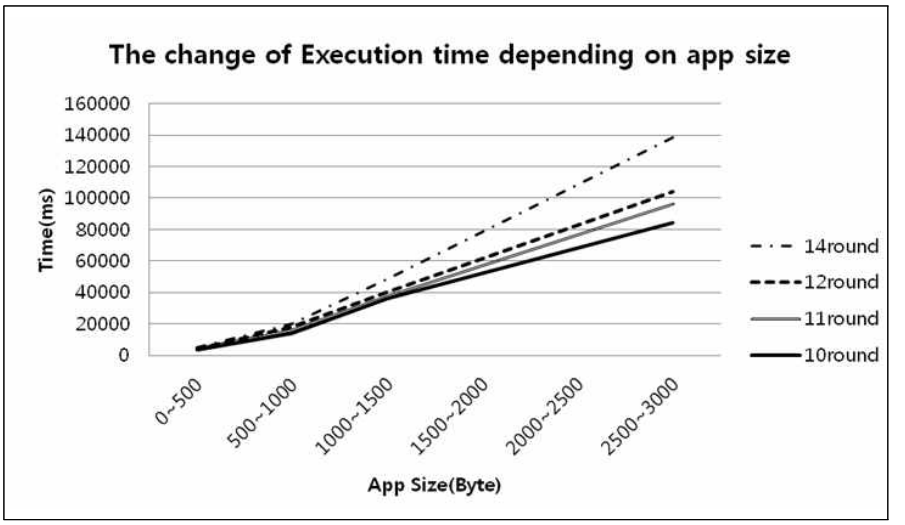
3G망을 사용하여 서버로부터 다양한 크기로 구성된 APK 파일을 다운로드 받았다. 다운로드 시

간은 APK 파일 크기가 증가함에 따라서 함께 증가하였다. 또 다른 실험은 AES 암호 알고리즘의 라운드 수를 다양하게 조절하여 복호화 시간을 측정하였다. 복호화 시간은 파일 크기에 비례하여 작은 크기의 파일에 10 라운드 (10라운드는 안전하지 못한 보안 수준임) 암호화를 적용하였지만 무시할 수 없는 지연시간을 보여줬다. 암호화 기반 불법 복제 방지 시스템 또한 온라인 실행 코드의 크기가 커지면 오버헤드가 증가하듯이, 암호화 기반 불법 복제 시스템도 또한 앱의 암호화 하는 크기가 커질수록 보안 수준은 향상되지만 이에 따른 오버헤드는 무시할 수 없을 만큼 커진다.



[그림 3] 다운로드 시간 실험 결과

[Fig. 3] Download time distribution



[그림 4] 앱 복호화 시간 실험 결과

[Fig. 4] Decryption time distribution

OEC 크기가 작을 경우, 스니핑 되었을 때 불법 복제를 시도하는 사람은 OEC를 복구하는데 필요한 시간과 노력이 적기 때문에 보안에 취약하다. 반면에 OEC가 커질수록 불법 복제를 시도하는 사람이 코드를 모두 모이기 어려워서 보안성이 향상되지만 다운로드 시간이 느려져서 부담이 된다. OEC 기술은 네트워크에 의존적이기 때문에 불안정한 모바일 무선 네트워크 환경에서는 OEC를 다운로드 받지 못해 앱이 정상적으로 실행되지 못하거나 앱의 실행이 지연되어 사용자에게 불편함을 줄 수 있다.

3.2 보안과 성능 이슈 및 융합 모델의 정당성

[2]에 의하면, 사용자가 오버헤드에 대해서 느끼는 불편함은 오버헤드가 특정한 값에 도달하면 크게 증가한다. 따라서 우리는 사용자가 참을 수 있는 범위 내의 오버헤드를 유지하면서 공격자로부터 안전하게 앱을 보호할 수 있어야 한다. 여기서 고려할 수 있는 오버헤드의 종류는 크게 세 가지로 분류할 수 있다. 첫 째, 온라인에서 코드를 다운로드 받는 금전적 비용, 둘 째, 코드를 다운로드 받고 암호화 된 코드를 복호화 하는데 생기는 실행 지연 시간, 마지막으로 두 비용을 모두 합친 전체 비용이다. 다운로드 금전 비용은 OEC의 크기에 비례하고 실행 지연 시간은 OEC 및 복호화 해야 하는 앱의 크기에 비례한다. 그리고 두 비용의 총합은 OEC와 복호화 크기에 비례한다. 각 비용을 고려하여 오버헤드를 계산하는 식은 다음과 같다.

다운로드 금전 비용 : C_f
실행 지연 시간 : C_t
비용 총합 : C_{tota}
복호화 크기 : S_{dec}
OEC 다운로드 크기 : S_{down}

$$C_f = S_{down} * K_1 \quad (1)$$

$$C_t = S_{down} * K_2 + S_{down} * K_3 \quad (2)$$

$$C_{total} = bC_r + dC_t = S_{down} * (bK_3 + dK_2) + S_{dec} + dK_3 \quad (3)$$

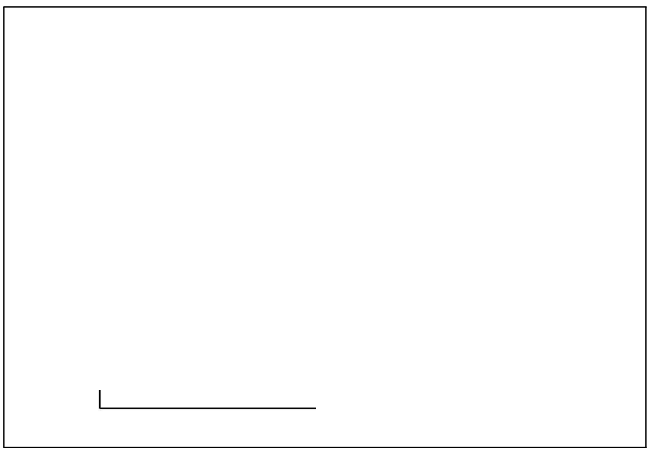
여기서, 앱을 불법 복제하려고 하는 공격자가 우리 시스템이 적용되어 보호된 파일의 일부를 얻기 위해 공격을 시도하여 한번에 L byte씩 랜덤하게 얻는데 이 때 공격자가 한 블록을 얻어낼 확률을 p라고 가정한다. L과 p의 값은 네트워크 상황이나 복호화 하는 기기 환경 등 여러 환경적 요인에 의해 결정되고 이러한 환경적 요인을 정확하게 측정 하는 것은 매우 어려운 일이다. 대입할

환경적 요인의 값이 설득력이 있다고 가정할 때 L과 p의 값은 비교적 작은 값일 것이다. 만약 L과 p의 값이 크다면 이는 곧 앱을 보호하기에 충분한 보안성이 제공되지 않는 환경이라는 것이다. 이 경우 L byte의 한 블록을 얻기 위해 필요한 공격자의 수는 1/p이 될 것이다. 여기서 적어도 0.99 이상의 확률로 X byte의 보호 기술이 적용 된 모든 코드를 얻어내기 위해 필요한 공격자의 수 N을 계산할 수 있다. 얻어낸 블록의 하나 이상이 포함되어 있는 보호된 코드일 평균 비율은 무작위로 선정된 byte가 적어도 하나 이상의 획득 블록이 포함되는 확률이다. 그리고 그것은 적어도 하나 이상의 블록이 L byte 구간의 왼쪽 끝에 포함될 확률이다. 1에서 구간의 어떤 왼쪽 끝도 위의 범위에 들어가지 않는 확률을 뺌으로써 확률을 계산 할 수 있다.

$$1 - ((pNL/X)^0 e^{-pNL/X})/0! = 1 - e^{-pNL/X} \quad (4)$$

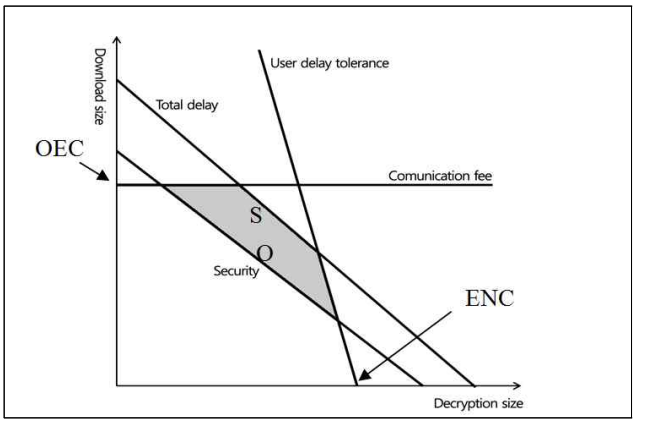
이 확률이 0.99가 되기 위해서는 pNL/X는 4.6이라는 값을 가져야 한다. 그러므로 L과 p가 임의의 값으로 고정되어 있다면 공격자의 수 N의 값은 N = 4.6X/pL이 되고 이 N은 코드 전체의 크기 X에 비례하는 값이다.

식 (1), (2), (3)에 의해 일정한 지연 시간 이하를 가지도록 하는 OEC 다운로드 크기 및 복호화 크기의 영역은 세 직선의 아래 영역이 교차하는 영역이다. 그리고 식 (4)의 N의 값을 충분히 크게 하여 보호 기술이 적용된 코드를 분석하여 얻어내기 위한 비용이 앱을 복제해서 얻어낼 수 있는 비용보다 더 커지게 만든다면 충분히 안전하다고 볼 수 있다. 식(4)에 의해 이 N의 값이 크기 X에 비례함을 알 수 있으므로 일정한 보안 수준 이상을 제공하는 복호화 크기 및 OEC 크기는 직선의 위쪽 영역이 될 것이다. 이 모든 그래프가 겹치는 영역이 바로 일정 수준 이상의 보안성을 보장하면서 사용자가 참을 수 있는 오버헤드 범위 내의 지연시간을 가지는 OEC 및 복호화 크기 영역이 된다.



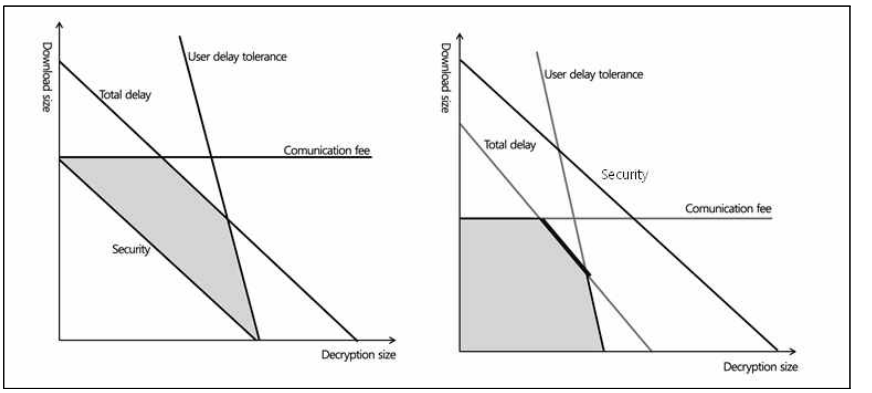
[그림 5] 보안 영역

[Fig. 5] The Area of Security



[그림 6] 사용자가 참을 수 있는 범위 내의 성능을 보장하며 일정한 보안 수준 이상을 제공하는 OEC 및 복호화 크기
[Fig. 6] The Area Guarantees the Security & Performance within User Delay Tolerance

그림 6의 색칠한 영역 내에 속하는 좌표 값으로 OEC 다운로드 크기 및 복호화 크기를 설정하면 일정 수준의 보안성과 성능을 유지할 수 있다. 각 그래프의 기울기는 비례 상수에 따라 달라질 수 있으며 각 비례 상수는 네트워크 환경 및 여러 가지 환경 요인에 의해 다르게 결정된다. 기울기가 어떻게 결정되느냐에 따라 두 기술을 모두 사용하는 것 보다 어느 한 쪽만 선택하여 사용하는 것이 나을 수도 있다. 하지만 일반적으로 그려지는 그래프를 보면 위와 같은 모양으로 영역이 생성되므로 OEC 기술과 암호화 기술을 함께 사용하는 것이 보안성을 보장할 수 있다.

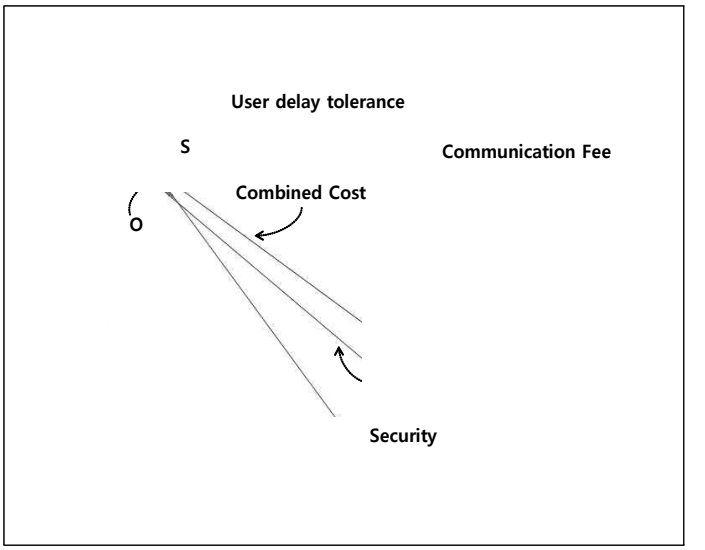


[그림 7] 다양한 케이스
[Fig. 7] Various other cases

4. 실제 적용 예제

우리는 식(1), (2), (3), (4)과 연구 및 실험을 통해 얻어 낸 상수 값들을 이용하여 실제 상황에서

의 다운로드 비용, 실행 지연 시간, 비용 총합을 계산할 수 있다. 우리는 사용자가 매월 유료 앱을 약 3,800원씩 결제하고 매월 55,000원 요금제를 이용하여 무제한 데이터 요금을 사용한다고 가정한다. 또한 사용자의 월 평균 데이터 사용량은 300MB 정도라고 가정한다. 그리고 [2]에 의해 사용자는 실행시간이 5초 이상이면 앱의 실행에 답답함을 느끼고 사용하지 않으려 한다고 가정한다. 이 값들과 식 (1), (2), (3), (4)를 이용하여 비용 총합을 계산해보면 Ctotal = 0.14b+5d = 4.945 * Sdec + 8Sdown이고 그 값은 5.14 이 된다.



[그림 8] 실제 생활에서의 OEC와 복호화 크기에 따른 다운로드 비용 및 실행 지연 시간, 비용 총합과 보안성 그래프

[Fig. 8] Download cost, execution delay time, cost expense and security depending on OEC and decryption size in real time graph

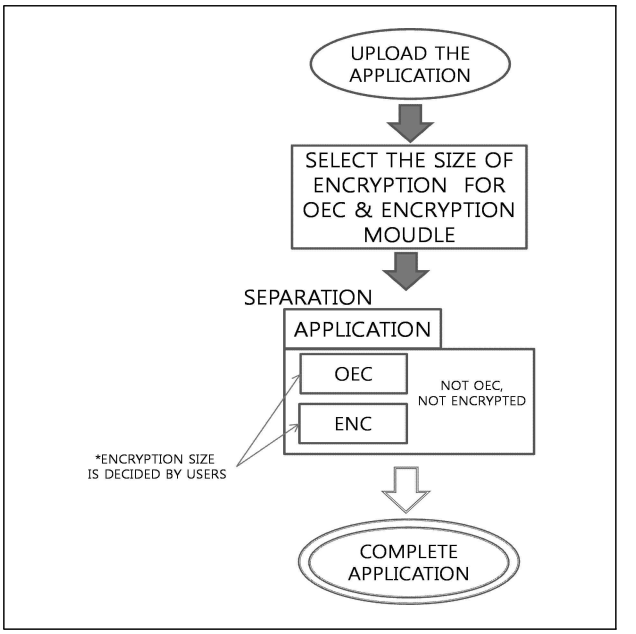
위 그래프에서 짙게 칠해진 영역이 일정 수준 이상의 보안성을 제공하면서 사용자가 참을 수 있는 범위 내의 오버헤드를 가지는 복호화 크기와 OEC 크기의 범위이다. 만약 개발자가 성능보다는 보안성을 중요히 여긴다면 그림 2의 S 주위의 좌표를 선택하여 OEC와 복호화 크기를 결정하는 것이 좋다. 반면에 보안성은 약간 약화되더라도 성능을 중요시 하는 프로그램의 경우 그림 2 내 O 주위의 좌표를 선택하게 되면 일정 수준 이상의 보안성을 유지하면서 오버헤드 또한 사용자가 참을 수 있는 범위 내로 유지할 수 있다. 위 그래프에서 만약 OEC의 크기를 0.8 MB, 복호화 크기를 0.1 MB로 하여 시스템에 적용한다면 이 좌표는 그래프의 색칠된 영역 내에 존재하는 좌표이고 이는 곧 보안성과 성능 모두를 보장할 수 있다는 것을 의미한다.

5. 통합 시스템의 구조

안드로이드는 리눅스(Linux) 기반으로 모바일용 운영체제이다. 다양한 라이브러리 세트, 멀티미디어 사용자 인터페이스, 폰 앱 등을 제공한다.

안드로이드 소스코드 중 Dalvik 폴더(/android/dalvik/libdex/Dexfile.h) 안에 위치한 DexOptHeader에 개발자가 설정한 보안옵션 값을 저장하는 플래그(flag)를 추가하였다. 기존에 존재하는 헤더에 4바이트를 추가하여 OEC와 ENC의 크기를 저장하도록 하였다

플래그는 OEC 크기와 DEX 파일을 암호화하는 DEX 크기를 결정한다. 위에서 언급했듯이, APK 파일을 두 개로 나누어 하나는 OEC로 또 다른 파일은 서버 보관용으로 저장한다. 개발 단계에서 플래그를 통해 보안 옵션을 쉽게 조절할 수 있다. 그림 9에서는 융합 모델의 전체적인 과정을 보여주고 있다.



[그림 9] 융합 모델 과정

[Fig. 9] Hybrid System Process

개발자가 APK 파일을 업로드 한다. 개발자가 설정한 보안 옵션에 따라 암호화크기를 결정한다. 암호화 크기에 따라 OEC 부분을 제외한 APK파일의 일부분을 암호화한다.

개발 단계에서 안드로이드 개발자는 소스코드를 두 가지 파일인 OEC 또는 OEC가 아닌 부분으로 나눈다. OEC는 다른 Java 클래스와 상속관계가 없어야 하며 인터페이스로 작성되어야 한다. Apache Ant 도구를 사용하여 OEC와 OEC가 아닌 파일의 디렉터리를 지정하였다.

Apache Ant는 자바 프로그래밍 언어로 사용되며 빌드 도구이다. 유닉스나 리눅스에서 사용되는 make와 비슷하지만 자바언어를 사용하여 자바 실행환경이 필요하다. Ant는 빌드를 위한 환경구성

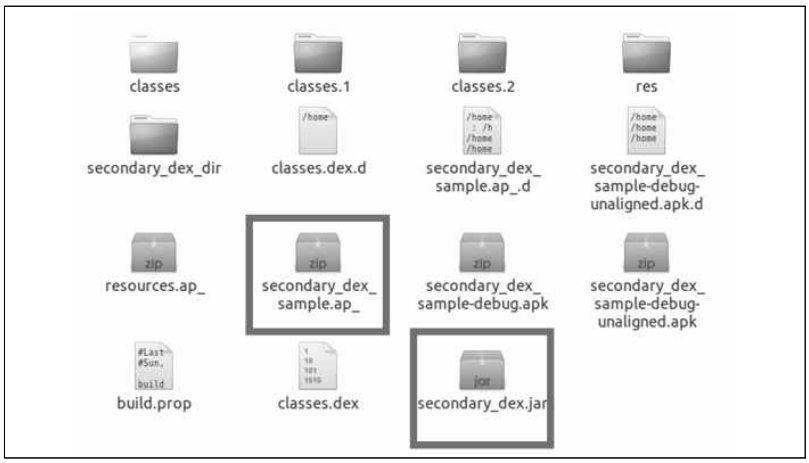
을 XML파일을 사용한다. 빌드 파일은 한 프로젝트에 하나만 넣어야 한다.

```
<target name="-dex" depends="-compile, -post-compile, -obfuscate"
        unless="do.not.compile">
    <if condition="${manifest.hasCode}">
        <then>
            <!-- Create staging directories to store .class files to be converted to the -->
            <!-- default dex and the secondary dex. -->
            <mkdir dir="${out.classes.absolute.dir}.1"/>
            <mkdir dir="${out.classes.absolute.dir}.2"/>

            <!-- Primary dex to include everything but the concrete library implementation. -->
            <copy todir="${out.classes.absolute.dir}.1" >
                <fileset dir="${out.classes.absolute.dir}" >
                    <exclude name="com/example/dex/lib/**" />
                </fileset>
            </copy>
            <!-- Secondary dex to include the concrete library implementation. -->
            <copy todir="${out.classes.absolute.dir}.2" >
                <fileset dir="${out.classes.absolute.dir}" >
                    <include name="com/example/dex/lib/**" />
                </fileset>
            </copy>

            <!-- Compile .class files from the two stage directories to the appropriate dex files. -->
            <dex-helper-mod input-dir="${out.classes.absolute.dir}.1"
                output-dex-file="${out.absolute.dir}/${dex.file.name}" />
            <mkdir dir="${out.absolute.dir}/secondary_dex_dir" />
            <dex-helper-mod input-dir="${out.classes.absolute.dir}.2"
                output-dex-file="${out.absolute.dir}/classes.dex" />
            <!-- Jar the secondary dex file so it can be consumed by the DexClassLoader. -->
            <!-- Package the output in the assets directory of the apk. -->
            <jar destfile="${asset.absolute.dir}/secondary_dex.jar"
                 basedir="${out.absolute.dir}/secondary_dex_dir" includes="classes.dex" />
        </then>
        <else>
            <echo>hasCode = false. Skipping...</echo>
        </else>
    </if>
</target>
```

[그림 10] build.xml

[Fig. 10] build.xml

그림10은 OEC 부분과 OEC가 아닌 부분을 보여주고 있다. secondary_dex.jar 는 OEC 부분이며 APK 파일은 OEC가 아닌 부분이다. APK 파일은 스턴 코드가 포함되어 있어 합법적인 사용자가 앱을 다운로드 받았을 때 실행이 가능하도록 해준다. 이렇게 두 가지 파일로 나누어 진 결과물을 개발자가 서버로 업로드 한다.



[그림 11] OEC 기술 적용 후 결과화면

[Fig. 11] FInal results after using OEC

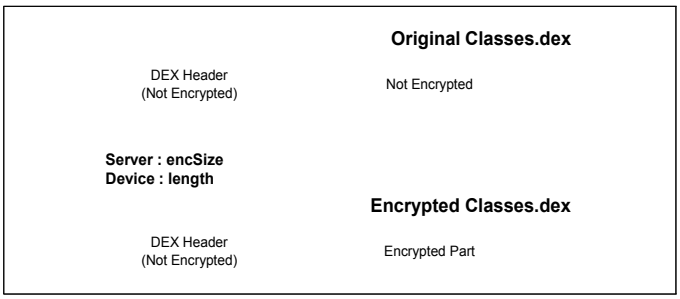
개발자가 파일을 업로드 후에는 서버는 동적 적재를 한다. 그림 12는 OEC 서버 코드의 일부분이다. 동적적재 과정에서, DEX파일에서 추출한 APK파일은 합법적인 사용자에게 전송되며 분리된 클래스를 로딩 한다. 분리된 클래스를 동적 로딩하기 위해 DexClassLoader 클래스가 필요하다. DexClassLoader 클래스를 사용하기 위해서는 안드로이드 SDK가 설치되어 한다.

```
DataOutputStream keyout = new DataOutputStream(new FileOutputStream(Key_File));
File infile_2 = new File(dec_apkName);
DataOutputStream foutput_2 = new DataOutputStream(new FileOutputStream(infile_2));
foutput_2.write(decrypted,0,decrypted.length);
keyout.write(S_Key);

if(din != null)     din.close();
if(foutput != null) foutput.close();
if(out != null) out.close();
if(in != null)  in.close();
if(sock != null)    sock.close();

Log.d("TimeCheck2","Receive_End and Loading_Start");
classloader = new DexClassLoader(dec_apkName, "/data/data/"+Mypackage+"/", null, origin.getClass().getClassLoader());
try {
    cls = classloader.loadClass(packageName+"."+className);
    cons = cls.getConstructor();
    m.add(cls.getMethod("getUserId"));
    m.add(cls.getMethod("setUserId", String.class));
    m.add(cls.getMethod("getCellPhone"));
    m.add(cls.getMethod("setCellPhone", String.class));
    m.add(cls.getMethod("getAge"));
    m.add(cls.getMethod("setAge", int.class));
    m.add(cls.getMethod("getSex"));
    m.add(cls.getMethod("setSex", boolean.class));
    m.add(cls.getMethod("getJob"));
    m.add(cls.getMethod("setJob", String.class));
    m.add(cls.getMethod("getPasswd"));
    m.add(cls.getMethod("setPasswd", String.class));

    Log.d("TimeCheck2","Loading_End");
} catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (SecurityException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (NoSuchMethodException e) {
```

[그림 12] OEC 받아오는 소스코드

[Fig. 12] Main code of receiving OEC

합법 사용자가 앱을 구매하면 OEC가 아닌 부분이 사용자 모바일로 전송된다. 전송 받은 암호화 앱을 암호화 기반 불법 복제 방지 시스템을 통해 복호화 한다. 암호화 앱이 복호화가 되면 앱은 정상적으로 설치가 된다. 사용자가 앱을 실행 시 서버로부터 OEC가 정상적으로 전송된다.



[그림 13] 융합 모델의 암호화 과정
[Fig. 13] Encryption Process in Hybrid System

aes_server_test.c

```
#include <stdio.h>
#include <string.h>

#define encSize 1024
#define AES_ENCRYPT    1
#define AES_DECRYPT    0

void aes_256_enc(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
void aes_256_dec(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
void aes_192_enc(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
void aes_192_dec(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
void aes_128_enc(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
void aes_128_dec(unsigned char *input, unsigned char *output, unsigned char *key_in, int length);
```

[그림 14] 암호화 모듈
[Fig. 14] Encryption Module

융합 모델의 암호화 과정(그림 12)에서 서버는 원본 classes.dex 파일을 복사하여 encSize 파라미터 크기만큼 제외하고 나머지를 부분을 암호화한다. Dexlength 파리미터는 안드로이드 소스 중 /dalvik/libdex/에 위치한 ZipArchive.c에 정의되어 있으며 Dexlength의 값은 encSize와 같다. 두 파라미터의 값을 동일하게 하여 복호화 과정이 정상적으로 실행되는지 확인하였다.

안드로이드 소스 중 /dalvik/libdex/에 위치한 ZipArchive.c에 dexZipExtractEntrytoFile 함수에 암호화/복호화 모듈을 삽입하였다. 안드로이드 소스 코드 중 dalvik/vm/dexprepare.c에 정의된 dexContinueoptimization 함수로 통해 classes.dex파일을 복호화하고 odex파일을 생성한다. 암호화 모듈은 Dalvik의 optimize과 verify 과정 후 삽입하였다. 암호화된 odex파일은 /data/dalvik-cache에 data@app@[filename]@classes.dex로 저장된다. 설치가 완료된 후에는 odex 파일은 다시 암호화가 된 상태로 저장한다.

개발자는 encSize 와 dexlength의 값을 수정하여 복호화 크기를 조절할 수 있다. 이 두 파라미터의 값이 클수록 복호화 크기 또한 증가한다. 그리고 AES 암호화 알고리즘을 사용하기 때문에 이 두 파라미터의 값은 16의 배수라야 한다.

6. 결론

본 논문에서는 기존의 안드로이드 불법 복제 방지 시스템에 온라인 실행 코드 기술을 적용하여 사용자 인내 허용 범위 안에서 보안 수준을 보장하고 개발자가 애플리케이션의 특성에 따라 보안 수준을 조정할 수 있도록 보안 옵션을 제시하였다. 그리고 안드로이드 불법 복제 방지 시스템과 온라인 실행 코드 기술을 융합한 모델이 각 기술을 따로 적용 시보다 향상된 보안성을 가짐을 확인할 수 있었을 뿐만 아니라 제안된 융합 모델의 정당성을 증명하였다. 향후 연구에서는 실제 상

황에 따라 보안성과 사용자의 인내 허용치 안에서 다양하게 보안 수준을 설정할 예정이다. 또한, 기존의 안드로이드 불법 복제 방지 시스템에서는 서버에서 암호화를 하기 때문에 사용자의 수가 급격하게 증가할 경우, 서버 부하가 발생할 수 있다. 그러므로 사용자의 수가 급격하게 증가할 경우를 대비한 암호화 방법에 대한 기술 연구가 필요하다.

참고문헌 [Reference]

[1] http://economy.hankooki.com/service/print/Print.php?po=economy.hankooki.com/lpage/industry/201008/e2010082317484970260.htm/, Oct 8 (2010).

[2] R.N.D. Silva, W. Cheng, W.T. Ooi, S. Zhao. Towards Understanding User Tolerance to Network Latency and Data Rate in Remote Viewing of Progressive Meshes. Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video, (2010) June 2-3; Vancouver, Canada

[3] J.Y Jang, C. Jeon, J.M Jung, B.J Kim, J.Y Park, Y.K Cho, Robust Static Software Watermarking Scheme for Copyright Protection of Mobile Software, Proceedings of the KIISE Dependable Computing 2012, (2012) February 6-8; Seoul, Korea

[4] S.W. Kim, E.H. Kim, J.Y. Choi, A study of detection method about Android app file on the network, Journal of Security Engineering. SERSC. (2012), Vol.9, No.1, pp.109-120.

[5] Y.S. Jeong, J.C. Moon, D.J. Kim, S.J. Cho, M.K. Park, Preventing Illegal Execution of Android App based on Encryption under Mobile Network Environment, Proceedings of the KIISE Dependable Computing 2012, (2012) February 6-8; Seoul, Korea

[6] S.R. Kim, J.H. Noh, Y.C. Moon, A.R. Kim, Design and Implementation of Cryptography-based Copy Protection System for Android Apps, Proceedings of the KIISE Dependable Computing 2012, (2012) February 6-8; Seoul, Korea

저자 소개

김희선 (Hee Sun Kim)

2013년 2월 : 건국대학교 인터넷미디어공학부 졸업
2013년 3월 ~현재: 건국대학교 인터넷미디어공학부 석사과정
관심분야: 정보보호, 컴퓨터보안, 데이터마이닝

김지현 (Jin Hyun Kim)

2007년 3월 ~현재: 건국대학교 인터넷미디어공학부 학부과정
관심분야: 정보보호, 컴퓨터보안

김성렬 (Sung Ryul Kim)

1993년 2월: 서울대학교 컴퓨터공학과 졸업
1995년 2월: 서울대학교 컴퓨터공학과 석사
2000년 2월: 서울대학교 컴퓨터공학과 박사
2002년 2월 ~현재: 건국대학교 인터넷미디어공학부 정교수
관심분야: 암호 알고리즘, 분산처리 알고리즘. 컴퓨터 보안, 클라우드 컴퓨팅, 데이터 마이닝

김진욱 (Jin Wook Kim)

1992년 2월: 서울대학교 수학과 졸업
2000년 2월: 서울대학교 컴퓨터공학과 석사
2006년 2월: 서울대학교 전기컴퓨터공학부 박사
2006년 9월~2009년 3월: ㈜에이치엠연구소 책임연구원
2009년 4월 ~2010년 2월: 인하대학교 컴퓨터정보공학부 연구교수
2010년 3월~현재: 서울대학교병원 의료정보센터 연구교수
관심분야: 컴퓨터이론, 알고리즘, 정보보호, 생물정보, 의료정보