안드로이드 이중 코드서명 체계 연구

박경용*, 위성근*, 서승현**, 조태남*
*우석대학교 정보보안학과
**한국인터넷진흥원 인터넷침해대응센터
e-mail:1aze@naver.com, destinywave@naver.com, seosh@kisa.or.kr,
tncho@ws.ac.kr

A Study on Android Double Code-Signing

GyeongYong Park*, SungGeun Wi*, SeungHyun Seo**, Taenam Cho*
*Dept of Information Security, Woosuk University
**Internet Incidents Response Division, KISA

요 약

전 세계적으로 스마트폰의 대중화와 보급률이 증가함에 따라 모바일 악성코드 확산으로 인한 피해가 늘어나고 있다. 많은 플랫폼들 중 최근 사용량이 증가하고 있는 안드로이드 마켓에는 악성코드 검증 절차가 없기 때문에 악성코드 배포에 용이한 환경을 가지고 있다. 또한 개발자가 생성한 자가 서명 인증서를 사용하기 때문에 개발자의 신원을 확인하기 어렵고, 유통 중에 발생할 수 있는 어플리케이션의 변조 유무를 확인하기 어렵다는 등의 취약점이 존재한다. 이러한 취약점들을 고려한 이중 코드서명 기법이 제안되었으나 기존 환경을 유지하려는 제약사항 때문에 취약점들을 보완하는 것에 한계가 있었다. 본 연구에는 안드로이드가 기반하고 있는 자바 코드서명 방식을 개선함으로써 기존연구가 해결하지 못한 취약점을 해결하였다.

1. 서론

스마트폰은 컴퓨터와 관련된 동일한 문제에 취약점을 갖고 있는 휴대용 컴퓨터이다. 또한 스마트폰은 24시간 켜져 있고 무선통신망을 사용한다는 특성 때문에 항상 위험에 노출되어있다. 여러 스마트폰 플랫폼 중 최근 사용량이 급증하고 있는 안드로이드 플랫폼은 개방형 플랫폼이고, 오픈소스이기 때문에 악의적인 공격에 쉽게 노출될 수 있다. 최근 조사에 따르면 안드로이드용 악성 어플리케이션의 누적 수가 2만 5천종 이상 있는 것으로 조사되었다[1].

안드로이드 어플리케이션의 배포와 설치를 위해서는 어플리케이션에 대한 개발자의 서명이 되어있어야 한다. 누구나 개발자의 키를 생성하여 서명을 할 수 있는 개방성 때문에 편리하기도 하지만, 유통 중에 발생할 수 있는 어플리케이션의 변조를 확인하기 어렵다는 취약점이 존재한다. 공격자가 유명한 어플리케이션을 다운로드하여 악성코드를 삽입하고 공격자 자신의 키로 서명을 하여 다시 배포하는 리패키징(repackaging) 하는 방식이 많이 사용되고 있다[2]. 사용자들은 변조된 어플리케이션과 원래의 어플리케이션을 구분하기 어렵기 때문에 의심 없이 사용하게 된다. 안드로이드 마켓에서 자체적으로 어플리케이션의 악성코드 삽입 여부를 검증하는 절차도 없고, 서명에 필요한 인증서도 공신력 있는 기관에서 발급받은 인증서가 아니므로 어플리케이션의 변조 유무를 확인할 수 없다.

위의 취약점에 대한 보안책으로 제시된 이중서명기법이 존재하지만[3], 활용도를 높이기 위해 기존환경을 유지해야 한다는 제약조건 때문에 취약점을 완전히 해결하지 못하였다. 본 논문에서는 기존 이중서명 기법이 해결하지 못한 취약점을 분석하고, 자바 서명 및 검증방식을 개선한 이중서명 기법을 제안한다.

2. 안드로이드 코드서명 체계

안드로이드 시스템은 어플리케이션에 대한 개발자의 서명을 요구하며, 서명이 없이는 설치가 불가능하다. 안드로이드에서 서명은 어플리케이션의 저자 식별, 모듈화, 업그레이드, 코드 및 데이터 공유를 목적으로 하며 어플리케이션의 통제목적으로 사용되지는 않는다[4]. 목적 자체가 어플리케이션에 대한 통제와 제어에 있지 않기 때문에 서명은 보안측면에서의 역할이 거의 없는 상태이다.

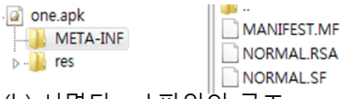
서명에는 디버그 모드와 릴리즈 모드가 있는데 디버그 모드는 개발, 테스트용으로 사용되며 마켓에 배포할 때는 릴리즈 모드를 사용한다. 어플리케이션에 서명을 하기 위해서는 개발자의 공개키와 개인키가 필요하며 JDK(Java Development Kit)에 포함된 Keytool을 이용하여 공개키와 개인키를 생성할 수 있다[5]. JDK 에 포함된 Keytool을 이용하여 생성한 개인키로 대상 어플리케이션에 서명을 할 수 있다. 서명파일은 실행파일인 .apk 파일의 META-INF 디렉터리 안에 생성된다. 서명파일의 기본 파일명은 사용된 키의 alias로 지정되며 옵션을 주어 다른 파일명으로 변경이 가능하다. [그림 1]의 (a)는 alias가 normal인 키를 사용하여 Jarsigner로 signTest.apk 파일에 서명하는 과정이다. 추가된 목록을 보면 alias명으로 서명파일이 생성되는 것을 볼 수 있다. (b)는 서명을 하고 난 후 어플리

케이션의 디렉터리 구조이다.

Jarsigner는 다중서명을 허용한다. 서명 파일명이 중복되면 이전 서명파일을 덮어씌우지만 -sigfile옵션으로 서명파일명을 지정해주거나 기존 서명에 쓰인 키가 아닌 다른 키로 서명을 하면 기존 서명이 지워지지 않고 새로운 서명이 생긴다[6].

```
D:₩jarsigner>jarsigner -keystore signKey.keystore -verbose
-sigalg MD5withRSA -signedjar one.apk signTest.apk normal
Enter Passphrase for keystore:
   adding: META-INF/MANIFEST.MF
   adding: META-INF/NORMAL.SF
   adding: META-INF/NORMAL.RSA
  signing: res/layout/activity_main.xml
  signing: res/menu/activity_main.xml
  signing: AndroidManifest.xml
  signing: resources.arsc
  signing: res/drawable-hdpi/ic_action_search.png
  signing: res/drawable-hdpi/ic_launcher.png
  signing: res/drawable-ldpi/ic_launcher.png
  signing: res/drawable-mdpi/ic_action_search.png
  signing: res/drawable-mdpi/ic_launcher.png
  signing: res/drawable-xhdpi/ic_action_search.png
  signing: res/drawable-xhdpi/ic_launcher.png
  signing: classes.dex
```

(a) alias가 normal인 키로 서명

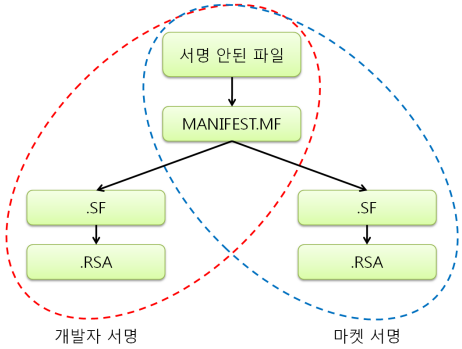


(b) 서명된 apk파일의 구조

(그림 1) alias에 따른 서명파일명과 디렉터리 구조

3. 마켓을 통한 어플리케이션의 배포

마켓은 개발자가 업로드한 어플리케이션의 정책 준수 여부만 검사하고 공개한다[7]. 이러한 기본 안드로이드 서명 체계를 따르고 있는 국내 마켓은 T스토어, OZ스토어가 있다. 그러나 올레마켓은 다른 마켓들과 달리 개발자의 서명을 제거하고, 개발자별로 공개키와 개인키를 생성해서 서명한다[8]. 이 경우 만약 개발과정에서 악성코드가 삽입된다면 개발자의 서명이 없기 때문에 개발자는 어플리케이션이 원본이 아니라고 주장할 수 있게 된다. 이러한 불분명한 책임의 소재를 명확하게 하기 위해서는 개발자의 서명과 마켓의 서명이 둘 다 필요하다. 개발자는 자신의 키로 서명함으로써 개발 단계에 대한 책임을 지고, 마켓은 마켓의 키로 서명함으로써 배포 단계에 대한 책임을 지는 것이다. 현 Jarsigner로 개발자와 마켓의 서명을 추가하면 [그림 2]와 같이 된다.



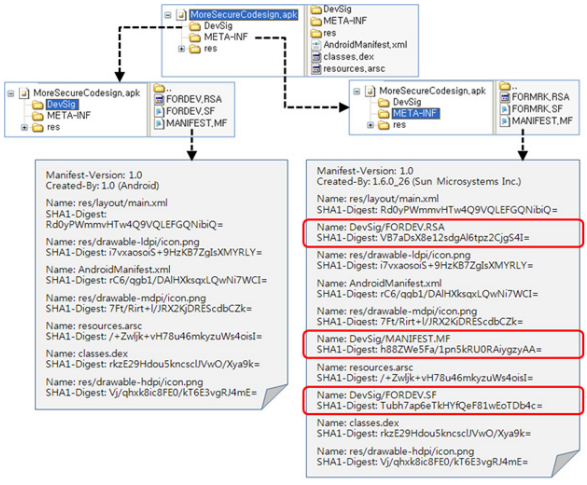
(그림 2) 개발자 서명과 마켓 서명 관계

개발자의 서명과 마켓의 서명은 서로 연관성이 없이 독립적이다. 즉, 2차서명인 마켓서명은 1차 서명인 개발자 서명을 서명대상에 포함시키지 않는다. 마켓의 서명 후에도 개발자서명이 변조되거나 삭제될 수 있기 때문에, 개발 및 유통 과정에 대한 무결성을 제공하지 못한다.

4. 관련 연구

[3]에서는 이미 기존 서명 체계에서 드러난 문제점을 파악하고, 코드 서명의 취약점을 해결하기 위한 이중서명 기법을 제안하였다. 사용자가 개발자와 응용 프로그램의 유통 마켓을 모두 확인할 수 있도록 지원하기 위해 개발자와 마켓의 서명이 모두 포함되도록 하였다. 또한 배포 과정에서의 무결성 보장을 확인하고 프로그램 개발 및 위·변조에 대한 책임 소재를 명확히 하기위해 개발자와 마켓이 각자 자신의 키를 관리하고 서명하도록 하였다[3]. 제안된 이중서명의 첫 번째 서명은 기존 체계와 마찬가지로 개발자의 서명이 들어가지만 Keytool로 생성한 자가 서명 인증서가 아닌 공신력 있는 기관에서 발급받은 개발자의 코드서명용 공인인증서로 서명을 한다. 개발자의 명의도용 문제를 해결하고 신원을 확보하기 위함이다. 개발자가 마켓에 업로드하면, 개발자의 서명까지 서명 대상으로 포함하여 마켓의 서명을 추가한다. 먼저 마켓에서는 개발자의 서명을 META-INF 디렉터리에서 DevSig 디렉터리로 이동시킨다. 그 다음 apk 전체에 대하여 마켓의 개인키로 서명을 생성한다. META-INF 디렉터리의 개발자 서명들은 이미 DevSig 디렉터리로 이동되었기 때문에 마켓의 서명 파일들은 META-INF 디렉터리에 생성된다. 마켓의 서명단계에서 개발자의 서명파일들이 MANIFEST 목록에 포함되므로(즉, 개발자 서명파일이 마켓 서명 대상에 포함됨) 마켓의 서명 후 개발자의 서명이 변경될 수 없다. 즉 불법적 리패키징이 불가능하다[그림 3].

제안된 기법은 기존의 개방적인 안드로이드 코드 서명 정책, 응용 개발 환경 및 코드 서명 체계의 개발 툴들을 수정하지 않도록 진행되었으며 기존 안드로이드 기기에서도 실행되도록 호환성을 고려하였다.



(그림 3) 이중 서명 파일 구조

5. 취약점 분석 및 보안 요구사항

5.1 취약점 분석

다중서명에서 1차 서명 이후의 서명이 이전 서명을 포함하지 않는다는 사실에 근거하여 이중 서명의 취약점 분석을 위해 다음과 같이 이중서명에 대한 3가지 실험을 수행하였다.

1) apk → 정상적인 키로 서명→ 만료된 키로 서명
2) apk → 정상적인 키로 서명→ 다른 어플리케이션의 서명파일 삽입
3) apk → 정상적인 키로 서명 → 이미지 파일의 확장자를 .RSA, .SF로 변경하여 META-INF에 삽입

실험에 사용된 키는 정상적으로 생성한 “normal” 키와 기간이 만료된 “expired” 키다. 다른 어플리케이션에서 가져온 서명파일은 INVALID.SF와 INVALID.RSA이며 이미지 파일의 확장자를 변경한 서명파일은 FAKE.SF와 FAKE.RSA이다.

```
D:Wjarsigner>jarsigner -verify -verbose expired.apk

        1053 Sat Aug 25 20:14:24 KST 2012 META-INF/MANIFEST.MF
        1174 Sun Aug 26 11:27:44 KST 2012 META-INF/NORMAL.SF
         916 Sun Aug 26 11:27:44 KST 2012 META-INF/NORMAL.RSA
        1174 Sat Aug 25 20:14:24 KST 2012 META-INF/EXPIRED.SF
         943 Sat Aug 25 20:14:24 KST 2012 META-INF/EXPIRED.RSA
sm       640 Wed Aug 22 21:18:28 KST 2012 res/layout/activity_main.xml
sm       420 Wed Aug 22 21:18:28 KST 2012 res/menu/activity_main.xml
sm      1540 Wed Aug 22 21:18:28 KST 2012 AndroidManifest.xml
sm      2348 Wed Aug 22 21:18:28 KST 2012 resources.arsc
sm       409 Wed Aug 22 21:15:50 KST 2012 res/drawable-hdpi/ic_action_s
sm      4129 Wed Aug 22 21:15:50 KST 2012 res/drawable-hdpi/ic_launcher
sm      1756 Wed Aug 22 21:15:50 KST 2012 res/drawable-ldpi/ic_launcher
sm       311 Wed Aug 22 21:15:50 KST 2012 res/drawable-mdpi/ic_action_s
sm      2654 Wed Aug 22 21:15:50 KST 2012 res/drawable-mdpi/ic_launcher
sm       491 Wed Aug 22 21:15:50 KST 2012 res/drawable-xhdpi/ic_action_
sm      5456 Wed Aug 22 21:15:50 KST 2012 res/drawable-xhdpi/ic_launche
sm    327008 Wed Aug 22 21:18:28 KST 2012 classes.dex

  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope

jar verified.

Warning:
This jar contains entries whose signer certificate has expired.

Re-run with the -verbose and -certs options for more details.
```

(그림 4) 정상서명+만료서명 검증결과

```
D:Wjarsigner>jarsigner -verify -verbose invalid.apk
jarsigner: java.lang.SecurityException: invalid SHA1 signature file digest for res/drawable-xhdpi/ic_launcher.
png
```

(그림 5) 정상서명+잘못된 서명 검증결과

```
D:Wjarsigner>jarsigner -verify -verbose fake.apk

        1053 Thu Aug 23 20:47:18 KST 2012 META-INF/MANIFEST.MF
        1174 Sun Aug 26 11:28:10 KST 2012 META-INF/NORMAL.SF
         916 Sun Aug 26 11:28:10 KST 2012 META-INF/NORMAL.RSA
           3 Sat Aug 25 20:16:48 KST 2012 META-INF/FAKE.SF
           3 Sat Aug 25 20:16:48 KST 2012 META-INF/FAKE.RSA
sm       640 Wed Aug 22 21:18:28 KST 2012 res/layout/activity_main.xml
sm       420 Wed Aug 22 21:18:28 KST 2012 res/menu/activity_main.xml
sm      1540 Wed Aug 22 21:18:28 KST 2012 AndroidManifest.xml
sm      2348 Wed Aug 22 21:18:28 KST 2012 resources.arsc
sm       409 Wed Aug 22 21:15:50 KST 2012 res/drawable-hdpi/ic_action_
sm      4129 Wed Aug 22 21:15:50 KST 2012 res/drawable-hdpi/ic_launche
sm      1756 Wed Aug 22 21:15:50 KST 2012 res/drawable-ldpi/ic_launche
sm       311 Wed Aug 22 21:15:50 KST 2012 res/drawable-mdpi/ic_action_
sm      2654 Wed Aug 22 21:15:50 KST 2012 res/drawable-mdpi/ic_launche
sm       491 Wed Aug 22 21:15:50 KST 2012 res/drawable-xhdpi/ic_action
sm      5456 Wed Aug 22 21:15:50 KST 2012 res/drawable-xhdpi/ic_launch
sm    327008 Wed Aug 22 21:18:28 KST 2012 classes.dex

  s = signature was verified
  m = entry is listed in manifest
  k = at least one certificate was found in keystore
  i = at least one certificate was found in identity scope

jar verified.
```

(그림 6) 정상서명+가짜서명 검증결과

<표 1> 서명방법과 Jarsigner 검증 및 App설치 결과

이중서명방법	Jarsigner 검증	App 설치
정상적인 서명 + 만료된 서명	검증은 되나 만료된 서명이 포함되어 있다는 경고가 나온다[그림 4].	성공
정상적인 서명 + 잘못된 서명	SecurityException이 발생한다[그림 5].	성공
정상적인 서명 + 가짜 서명	아무런 경고 없이 검증이 성공한다[그림 6].	성공

실험 결과를 보면 만료된 서명과 가짜서명을 포함한 어플리케이션은 검증이나 설치가 되지 않아야 하는 경우임에도 불구하고 검증이 성공하였다. 또한 모든 어플리케이션의 설치가 성공적으로 완료되었다. 따라서 가짜 서명이나 만료된 서명, 잘못된 서명이 포함되어 있어도 정상적인 서명이 하나라도 포함되어 있다면 그 어플리케이션은 설치가 된다는 것을 알아낼 수 있었다. 가짜서명이 포함된 경우는 Jarsigner 검증에서 경고조차 나오지 않는 사각지대를 보여준 실험 결과였다. 이 경우, 악성코드가 서명파일로 위장되어 삽입될 수 있다는 가능성을 보여준다. [3]에서 제안한 이중서명 방식은 가짜서명파일 삽입에 대한 해결책을 제시하지 않았다.

5.2 보안 요구사항

보완되어야할 보안 요구사항을 요약하면 다음과 같다.

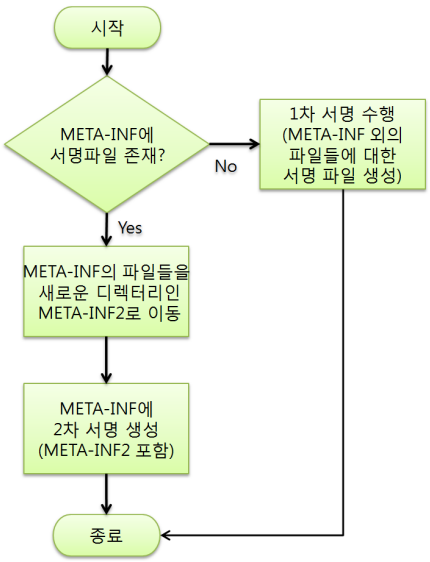
(1) 어플리케이션에 대한 개발자 서명이 있어야 한다.
(2) 개발자 서명까지 포함한 어플리케이션에 대한 마켓의 서명이 있어야 한다.
(3) 최종 서명에 포함되지 않은 어떠한 file도 존재해서는 안 된다.
(4) 검증할 수 없는 서명파일이 포함되어서는 안 된다.

6. 이중 코드서명 제안

5장에서 살펴본 보안 요구사항을 만족하기 위하여 Jarsigner의 서명 및 검증 방법을 수정하였다. 본 연구에서는 개발자, 마켓의 서명을 생성・검증하기 위한 것으로 2차 서명까지만 허용했으며 필요시 3차 이상으로 확장 가능하다.

6.1 서명 과정

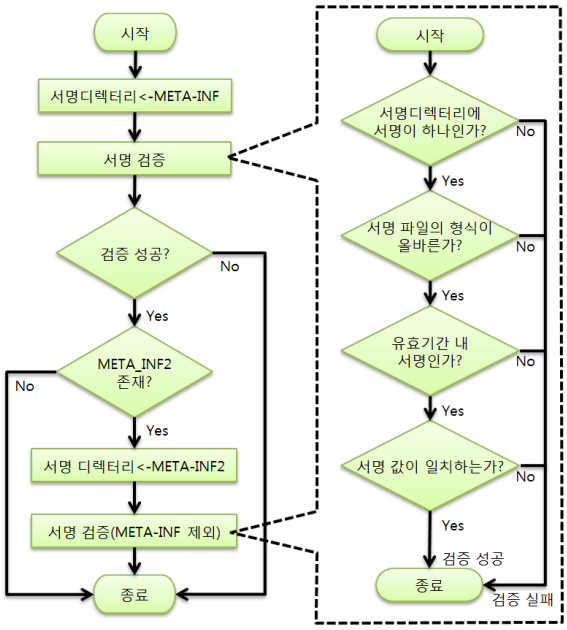
서명은 1차와 2차를 구분하여 진행하고 각각의 디렉터리를 생성하여 서명 파일들을 분리한다. 2차 서명 때는 1차 서명파일까지 대상에 포함하여 서명한다. 서명 프로세스의 전체적인 흐름은 [그림 7]과 같다.



(그림 7) 서명 흐름도

6.2 검증 과정

먼저 META-INF에 있는 서명을 검증한다. 이 검증은 2차 서명이 되어 있는 경우라면 1차 서명까지 포함한 서명에 대한 검증으로서, 어플리케이션 전체에 대한 서명 검증이다. 검증에 성공하면, 이중 서명되어 있는지 확인하고, 이중 서명이 되어 있다면, 2차 서명인 META-INF 폴더를 서명대상에서 제외한 META-INF2의 서명을 검증한다. 서명 검증은 한 서명 폴더에 한 개의 서명 파일만이 존재하여야 하고, 유효기간 내의 서명이어야 하며 파일이 서명 파일 형식에 맞는 경우에만 서명값을 검증한다. 그 외의 경우에는 모두 검증 실패로 종료한다. 검증 프로세스의 전체적인 흐름은 [그림 8]과 같다.



(그림 8) 검증 흐름도

7. 결론 및 향후 계획

현 안드로이드 서명체계는 자가 서명을 이용하기 때문에 다른 개발자의 명의를 도용할 수 있으며 개발자 신원 확인에 어려움이 있다. 또한 유통 중의 변조를 확인할 수 없다는 취약점이 존재한다. 이러한 취약점을 분석하고 보완한 이중서명 기법이 제안되었지만 기존 안드로이드 환경 유지를 제한하고 있어 해결책에 한계가 있었다. 수동으로 진행된 디렉터리 구조 변경과 서명파일들의 이동이 이뤄졌고, 정상적으로 서명된 파일에 가짜서명 파일을 삽입해도 검증이 성공한다는 문제점을 해결하지 못했다.

본 논문에서는 JDK의 서명 툴인 Jarsigner를 수정하여 이중 서명과 이중 서명의 검증을 자동화시킴으로써, 개발자와 마켓의 서명 생성과 검증이 동일한 Jarsigner를 통해 일관되게 수행될 수 있도록 하였다. 또한 악성코드가 가짜 서명으로 위장하여 삽입 되는 것을 방지하기 위해 검증 시 개발자와 마켓의 서명파일들을 확인하여 한 디렉터리 내에 서명 파일이 여러 개 존재하거나 만료된 서명, 가짜 서명이 발견되면 검증에 실패하도록 하였다. 이미 설치된 어플리케이션의 서명 검증에 활용되도록 하기 위해서는 검증 결과를 세분화하여 기간 만료된 서명에 대해서는 검증 실패가 아닌 경고로 처리하도록 할 수 있을 것이다.

Jarsigner의 수정은 어플리케이션 개발 환경인 eclipse에 반영하여 실행하였으나 실제 스마트폰에는 install 할 수 없었다. 추후 에뮬레이터를 통한 실험을 수행할 계획이며, 서명을 위장한 가짜 서명파일을 이용하여 악성코드를 삽입하는 기법을 연구할 것이다.

참고문헌

[1] Trend Micro, "巨大ビジネスになる サイバー犯罪、攻撃はより 'パーソナル'に," July 2012.

[2] http://www.ilovepc.co.kr/bbs/board.php?bo_table=hardware&wr_id=1284.

[3] Taenam Cho, Seung-Hyun Seo and Nammee Moon, "Double Code-Signing for Enhanced Android Application Security," Information Vol. 15 No. 5, pp.1913-1926, May 2012.

[4] http://developer.android.com/tools/publishing/app-signing.html.

[5] Oracle, JDK Tools and Utilities, http://docs.oracle.com/javase/7/docs/technotes/tools/.

[6] jarsigner - JAR Signing and Verification Tool, http://docs.oracle.com/javase/6/docs/technotes/tools/windows/jarsigner.html.

[7] Publishing Checklist for Google Play, http://developer.android.com/distribute/googleplay/publish/preparing.html.

[8] KISA, "국내외 스마트폰 애플리케이션 블랙마켓 현황 조사 및 국내 스마트폰 애플리케이션 마켓에 적합한 코드 서명 기술 연구", 2011.11.