Bypassing DRM protection in e-book applications on Android

Sangmin Lee*, Kuyju Kim*, Jina Kang†, Hyoungshick Kim*
*Electrical and Computer Engineering
Sungkyunkwan University, Republic of Korea
{sangmin91, kuyjuKim, hyoung}@skku.edu
†National Security Research Institute, Republic of Korea
4558kang@nsr.re.kr

***Abstract*—Digital Rights Management (DRM) technologies allow content owners to protect their rights from illegal use of the digital contents (e.g., duplicating electrical resources). DRM technologies have commonly been used in the electronically published books (e-books) industry. In this paper, we introduce a general framework that can bypass the DRM protection deployed on e-book applications on Android. To demonstrate the feasibility of the proposed approach, we applied our approach to three popular e-book DRM applications (Kyobo Book Center, Ridibooks e-book, Interpark e-book) and demonstrated that it is possible to circumvent their DRM protection solutions in a semi-automated way. Also, we recommend several practical ways to develop secure DRM implementations for e-book applications.**

***Keywords*—DRM, Android, EPUB, e-book, reverse engineering.**

I. Introduction

With the development of networking technologies and e-commerce platforms, more and more digitized products and multimedia contents are easily sold and distributed over the Internet. In this environment, however, content owners are concerned about the potential loss of revenue from the piracy of their digital assets because digital contents can simply be reproduced and redistributed. In the entertainment industry (i.e., music, film and video game), therefore, many Digital Rights Management (DRM) technologies (e.g., [1], [2]) were introduced to protect content owners' rights from the illegal use of their contents over Internet. DRM technologies allow content providers and publishers to control the whole distribution chain and apply flexible usage rules [3]. Content providers distribute their digital contents online, but after its distribution they can still exercise control of those contents via DRM. One of the biggest markets for DRM technologies is the e-book distribution market [4]. In the e-book market, DRM generally acts either as a wrapper for the actual e-book file (e.g., PDF and EPUB), or as part of the package of the e-book file itself.

In mobile e-book applications, DRM is also popularly applied to permit users to only use e-book files under specific usage rules. However, the effectiveness of DRM technologies in practice is still questionable. For example, there exists an inevitable problem called the analog hole which is the duplication of DRM-protected contents by analog means (e.g., screen capturing each individual page and saving it from an e-book application).

In this paper, we particularly analyze the security of the DRM systems used in Android through case studies of three popular e-book applications (Kyobo Book Center, Ridibooks e-book, Interpark e-book) that are implemented to display e-book contents by using a particular package's implementation of `Webview`. In those applications, plaintext e-book (EPUB) files are securely encrypted with a cryptographic key, which can only be decrypted by an authorized DRM agent under specific usage rules and conditions. However, we found that DRM protection in those applications can be circumvented in a semi-automated way by extracting the plaintext e-book file before rendering it on the `Webview` component by a reverse engineering analysis. In Android applications using `Webview`, such timing can be obtained in a few locations. Moreover, this analysis can be used even when codes are obfuscated.

In the rest of this paper, we will present our analysis in detail. First, we will briefly present previous studies related to our work to provide a better understanding of our method for bypassing DRM protections and then present the design and implementation of our proof-of-concept system.

II. Related Work

In this section, we first provide some basic DRM techniques and then several DRM cracking attempts to circumvent such DRM solutions.

A. Hardware-based DRM

Hardware-based DRM is a popular strategy to make DRM-protected files work in a certain environment with a hardware device (e.g., dongle). In the past, dongles had static information which be confirm in the software. Modern dongles are complicated to be more secure such as co-processors program code. HDCP is a link protection scheme. This scheme is the decryption of media content outside the computer. Also, this scheme needs to be encrypted and decrypted for all content. Nowadays, many people use streaming services. Theses services permit playback without HDCP. This is because some systems(e.g., a virtual machines and net books) do not support HDCP. In general, encrypted content must be decrypted first, before being re-encrypted with HDCP. General media devices enable access to the unencrypted stream. Therefore, general media devices with HDCP cannot handle to protect encrypted content by MovieSteer.

B. Streaming DRM Platform

Many DRM solutions were introduced to protect multimedial contents (e.g., audio and video files). First, one of them is Microsoft PlayReady by using Netflix [5]. It is Microsoft's PlayReady DRM and is conducted in their Silverlight streaming platform. PlayReady supports individualization. It means that the content is encrypted with a key and then the key is different for every user. Second, another one is RTMPE. RTMPE is a lightweight link protection mechanism which is based on the Real Time Messing. This way is developed by Adobe. RTMPE generates a key using Diffie-Hellman key exchange. The generated key is used to encrypt the content stream using RC4 encryption algorithm. This way supports fast processing of the content streaming without dropping the connection.

C. Bypassing DRM

Many security experts agree that secure implementation of DRM technology is very tricky, and many commercial DRM systems have proven insecure (e.g., against memory analysis [6]). To make matters worse, even with a secure DRM implementation, there exists an inevitable problem called *analog hole* which is the duplication of DRM-protected contents by analog means [7] (e.g., screen capturing each individual page and saving it from an e-book application). It is very challenging to prevent analog hole. However, contents owners often are not concerned about this flaw because the conversion from analog content back to digital content typically results in a loss of quality.

A more serious issue is to extract the content encryption key from a DRM-protected file or forcibly modify DRM rules by reverse engineering the DRM implementation for an application. Jon first broke the Content Scramble System (CSS) algorithm used as part of the DRM system in DVD-decoders [8]. Those cracking results showed how difficult it is to securely implement a DRM solution in the real-world. Wang et al. [6] introduced a memory-based approach to circumvent the DRM protections for streaming services (Amazon Instant Video, Hulu, Spotify and Netflix). Choi et al. [9] demonstrated how to bypass the integrity checking of a DRM-protected file in the OMA (http://www.openmobilealliance.org/) DRM system by reverse engineering. We extend such existing techniques by focusing on DRM implementations for e-book applications. We suggest a generic technique to automatically recover the original EPUB file from its DRM protected file. Our technique particularly exploits the structural weakness in DRM implementations using `WebView` on Android.

III. APPROACH

A. EPUB structure

Fig. 1 is EPUB structure which is standard structure of e-book. EPUB is an electronic book disclosure standard proposed by the International Digital Publishing Forum (IDPF). EPUB is a compressed file which consists of OPF file, container.xml, minetype, etc. The OPF file defines the order and location of the contents(xhtml, css, javascript, image, etc) of the e-book. The table of contents is defined in the NCX file. The Container.xml file Defines the top-level root path. The Minetype file contains an "application/epub+zip", which confirms that the file is an EPUB file. EPUB consists of three definitions: Open Publication Structure (OPS), Open Packaging Format (OPF), and OEBPS Container Format (OCF) [10], [11].

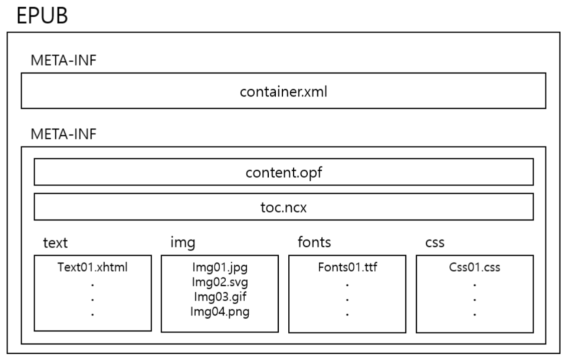


Fig. 1: Structure of EPUB file.

These commonly adopt EPUB, the de-facto standard file format for digital publications.

B. e-book viewer implementation on Android framework

The framework is a group of reusable codes. It consists of many classes and libraries. Framework provides a large number of standard code for application development. It benefits developers, keeping them from unnecessary re-writing of basic codes. On Android framework, a regular way to implement e-book viewer is using `Webview`, `WebviewClient`, `WebChromeClient`. The class `WebviewClient` and `WebChromeClient` are dependent on the class `Webview`, as depicted in the Fig. 2. `WebView` on Android framework uses the following three functions `loadUrl()`, `loadData()`, `loadDataWithBaseURL()` to display a website on the screen. `loadUrl()` is commonly used to load a web page from the internet and display it on `Webview`. `loadData()` and `loadDataWithBaseURL()` are used to display a web page located in local storage or dynamically created by the application. The former one is unable to set a custom root path, but the latter one is able to do so. What we have to concern is that if we only use `Webview` class to implement an e-books viewer, every new content page creates a new activity. This is because `Webview` class displays the web page using the system-default external browser application, unless the developer connect the two class `WebviewClient` and `WebChromeClient`. Therefore, we should connect the two classes to `Webview` class to avoid new screen for every new web page and let the application render a screen within its own view.

`WebviewClient` class renders a web page within the application. Basically, it processes events such as the initiation of a web page, loading the web page, completion of web page loading, error handling, intercept, or keystroke handling in the application. `webChromeClient` class is to implement browser event functions such as `alert`.

When an e-book stored in local storage of an Android device has the structure of EPUB, its contents are formatted in xhtml. Hence it may be printed out via `Webview`. We should

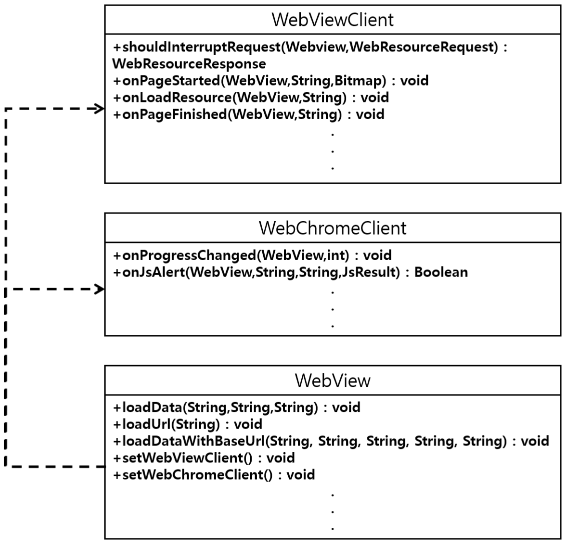

WebViewClient
+shouldInterruptRequest(Webview,WebResourceRequest) : WebResourceResponse
+onPageStarted(WebView,String,Bitmap) : void
+onLoadResource(WebView,String) : void
+onPageFinished(WebView,String) : void
WebChromeClient
+onProgressChanged(WebView,int) : void
+onJsAlert(WebView,String,String,JsResult) : Boolean
WebView
+loadData(String,String,String) : void
+loadUrl(String) : void
+loadDataWithBaseUrl(String, String, String, String, String) : void
+setWebViewClient() : void
+setWebChromeClient() : void

Fig. 2: Webview UML.

extract the EPUB-formatted e-book file with `unzip` library, and transfer the internal xhtml with `loadURL()`. In this case, we do not have to deal with it additionally, as the resources (e.g. images, fonts, css stylesheets, etc.) invoked by xhtml actually exist. The precedent three functions `loadUrl()`, `loadData()`, `loadDataWithBaseURL()` are used for every EPUB viewer implementation to display the contents. `Webview` itself is just a class to implement a web client, so it cannot provide a simple function to shift the e-book page back or forward. Hence, the developer should manually implement them, using `TouchListener` and the three functions above. When the internal resources for the xhtml do not exist as a complete source file but just in binary or other encrypted form, we should decrypt it before displaying it via `Webview`. Then we need the function `shouldinterceptrequest()` in the class `WebviewClient`.

C. Individual Vierer's DRM applying and generalization of the attack

The EPUB viewers analyzed in this paper apply DRM to themselves in three major ways.

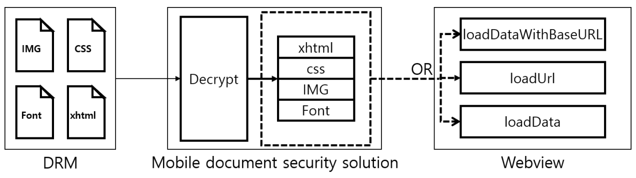


Fig. 3: All files encryption.

The first one, as depicted in the Fig. 3, only encrypts EPUB contents (e.g. xhtml, css, png, etc.) and creates a `*.drm` file, including opf, ncx, container.xml in plaintext. In this sort of DRM implementation, to display a content 'X' on `Webview`, it does not read the contents data from the actual contents path. Rather than that, it loads the contents data decrypted from the `drm` file on the memory. After that, contents handling method

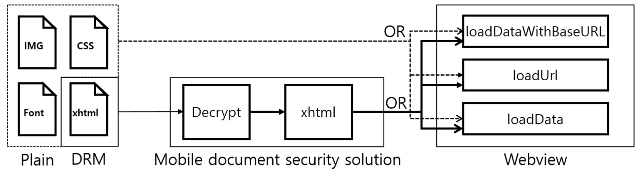


Fig. 4: Only text file encryption.

of `Webview` class transfers the data to display it. The second one stores all EPUB components into a file format optimized to `ops`. This, as depicted in the Fig. 4, encrypts xhtml only. xhtml includes the most significant part, i.e. text information in an e-book, and the viewer application only decrypts this xhtml to use it.

The last one is different; it implements the DRM with a unique data file format, not with a typical EPUB. This way does not need to use `Webview` for the own viewer application's development since it does not follow the EPUB structure. The first and second way implement the DRM and the viewer for the commercial EPUB to be read in own application. However, both ways use `Webview`, thus the attack proposed in this paper is practically menacing to these. Firstly, DRM implementation as depicted in Fig. 3 has no actually existing resource files for xhtml to load. Hence, `Webview` should override the method `shouldRequestIntercept()` from the class `Webviewclient` to handle the resource within the application layer. Additionally, when a xhtml calls the resource such as an image of css, it requires the root path for locating it. This means, only `loadDataWithBaseURL()` is available which may set the root path by itself, among other web page displayer function. `loadDataWithBaseURL()` function's parameter, the root path, it is including the xhtml filename to be displayed on `Webview`. It can refer content.opf, toc.ncx files stored in plaintext to verify the extracted contents whether they are correct contents file consisting the EPUB, or where they should be located. A complete EPUB file comprises the extracted content components, after compressing them into one file.

The DRM implementation as depicted in Fig. 4 does not require root path input since the resource files are actually stored in the path. Thus an application based on this second way is not guaranteed to use the function `loadDataWithBaseURL()`, but might use `loadData()`. Similarly, toc, ncx, content.opf, container.xml are in plaintext. Therefore, matching the extracted data and filenames, we may exactly extract the components of EPUB.

The last way is not dependent on `Webview`, but still there should be a moment when the encrypted contents get decrypted. We can catch the moment to extract the contents. However, unless we verify the exact structure of their unique file format, the full recovery of the original e-book from the extracted components is very difficult. except the last way, for viewer to display an e-book it should transfer the plaintext decrypted by DRM module to the viewer module. There is the attack vector. In other words, there is an e-book viewer application following a common method, and there is a generalized attack vector as well.

IV. IMPLEMENTATION

A. Method to extract original data from EPUB file

EPUB viewer application has a function `loadDataWithBaseURL()`, which actually displays the e-book content, requires the local e-book storage path as the argument. Therefore, we should find the location of `loadDataWithBaseURL()` since we cannot identify the argument input for the parameter of `loadDataWithBaseURL()` based on reverse engineering technologies. To identify the argument of `loadDataWithBaseURL()`, we have to re-write the `smali` codes and re-compile, then directly execute the application. In this paper, we use `APK Easy Tool` to decompile and re-compile the Android application [12]. With the tool, we may create a new project formatted in `smali` code form from the given APK file. First, We would find every point among `smali` codes where `loadDataWithBaseURL()` is invoked. Second, we would search the very location where e-book contents are printed out, based on our self-developed script code to modify the `smali` code which makes `loadDataWithBaseURL()` function's the first parameter `baseURL` be printed out into LogCat.

Algorithm 1 callExtractionFunciton

```
1: function CALLEXTRACTIONFUNCTION($file$)
2:     $lines = getlines(file)$          ▷ lines is string array
3:     for $i = 0$ to $length(lines)$ do
4:         $line = lines[i]$
5:         if "$loadDataWithBaseURL$" in line then
6:             $write(Log.i(baseURL))$
7:         end if
8:     end for
9: end function
```

The function described above in Algorithm 1 finds `loadDataWithBaseURL()` within `smali` file, and prints out the first parameter of it into LogCat via `Log.i` function. Every line of the input argument's file is stored line-by-line in the second-dimensional `lines` array. The above algorithm 1 finds the string *loadDataWithBaseURL* among the stored strings in `lines` array. When the targeted string is found, the algorithm attaches `Log.i(baseURL)` beneath the string and stores them in a `smali` file.

To manage and control the data flow in the e-book viewer application, we add some new functions in `smali` code such as the ones described above. But there are several characteristics of `smali` code we should watch out for. First, `smali` code only organizes a function with the limited register. When the newly added function in `smali` code is invoked, re-compile process may be faced with errors if the function calls the register with missing number of even the one already invoked by others. Hence, in this paper, we set the following standards of `smali` code extension to store the DRM-secured contents from the application into files.

1) Whenever a new code is attached to `smali` code, the line number should be added to the largest line number found in the previous `smali` code.
2) The newly attached `smali` code should be structured as an individual function, especially a method of the previous `smali` code's `class`.
3) The newly attached function should be after `baseURL` and data register set, but also must be injected before calling of `loadDataWithBaseURL()`.

In this paper, we add the function to extract DRM-secured contents, following the above standards. It is attached as a new method in the previous `smali` class which invokes `loadDataWithBaseURL()` to call e-book in the application.

Algorithm 2 addExtractionFunction

```
1: function ADDEXTRACTIONFUNCTION($originFile, extractionFunctionScript$)
2:     $lines = getLines(originFile)$  ▷ lines is string array
3:     Let $script = function\ script$
4:     Let $maxLineNumber = 0$    ▷ maxLineNumber for check last line number
5:     for $i = 0$ to $length(lines)$ do
6:         $line = lines[i]$
7:         if "$.line$" in line then
8:             $lineNumber = getLineNumber(line)$
9:             if $lineNumber > maxLineNumber$ then
10:                $maxLineNumber = lineNumber$
11:            end if
12:        end if
13:    end for
14:    for i=0 to length(extractionFunctionScript) do
15:        $line = extractionFunctionScript[i]$
16:        if "$.line$" in line then
17:            lineNumber=getLineNumber(line)
18:            write("$.line$"+$string(lineNumber + maxLineNumber)$)
19:        end if
20:    end for
21: end function
```

In Algorithm 2, `addExtractionFunction` function takes the following parameters: `smali` file in the decompiled project, and the `smali` code of contents-adder function. `addExtractionFunciton` finds the *lineNumber*, finding "*.line*" which clarifies the function location in *originFile*. *maxLineNumber* is the largest value among the found *lineNumber* values. We add *maxLineNumber* to "*.line*", when we attach the contents-adder function `smali` code to *originFile*.

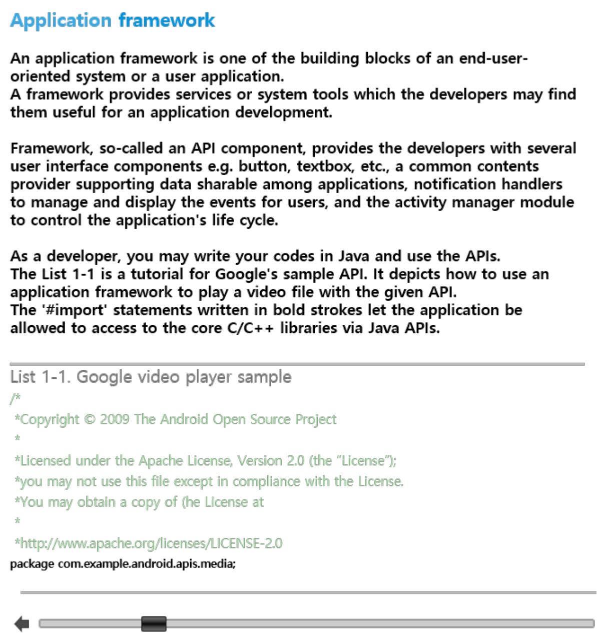
Now our re-compiled application stores the contents formatted in a file in the local device storage whenever `loadDataWithBaseURL()` is invoked in e-book `Webview`.

V. EXPERIMENT

The experiment has been targeted to Android 7.1.2, installed on `Nexus 5X` device made by LG. Specifically, we focused on three DRM-secured contents viewer application by Interpark e-book, Kyobo Book Center, and Ridibooks e-book. These are widely spread as representative DRM-secured contents viewer solution in Korea.

During the process, we found that each viewer application stores the DRM-applied e-books in various directories. Hence, we modified the configuration for the e-book storage Paths manually.

Because every e-book has its own DRM implementation, the amount of time required to execute a crack can vary greatly. In the case of an e-book with DRM encrypting only xhtml, each extraction process costs 0.8 seconds per page, i.e. it costs 400 seconds per an e-book with 500 pages approximately. In another case where the e-book's DRM encrypts all components except `container.xml`, `toc` file, and `opf` file, it costs much more time for extraction. Consequently, we can recover the original e-book files by the proposed technique (see Fig. 5).



Application framework

An application framework is one of the building blocks of an end-user-oriented system or a user application.
A framework provides services or system tools which the developers may find them useful for an application development.

Framework, so-called an API component, provides the developers with several user interface components e.g. button, textbox, etc., a common contents provider supporting data sharable among applications, notification handlers to manage and display the events for users, and the activity manager module to control the application's life cycle.

As a developer, you may write your codes in Java and use the APIs.
The List 1-1 is a tutorial for Google's sample API. It depicts how to use an application framework to play a video file with the given API.
The '#import' statements written in bold strokes let the application be allowed to access to the core C/C++ libraries via Java APIs.

List 1-1. Google video player sample

```
/*
 *Copyright © 2009 The Android Open Source Project
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of (he License at
 *
 *http://www.apache.org/licenses/LICENSE-2.0
package com.example.android.apis.media;
```

Fig. 5: Recovered the original e-book file.

VI. Discussion

On Android Framework, you may take several universalized methods to develop a viewer application to render commercial contents (e.g. charged media, document, etc.). This means there would be lots of potential attack vectors aimed at DRM-applied contents, not only such as e-book as we propose. For instance, let us suppose you use an audio player application to play DRM-applied audio files on Android. It would have probably been following the universalized development method. This indicates, with high possibility that there should be some method you may invoke to use the function `PLAY`. Hence, we may count on the fact that we can extract the DRM-secured content files with some modification on the parameters for media playing. Similarly, we suspect we would be able to extract data as well, during the contents streaming.

In this paper, we proposed a new attack targeted to DRM-secured contents. The attack requires, first of all, a unique viewer application for the targeted DRM-secured e-book. The victim must have the authority to activate the contents.

As previously explained, we could not generalize our attack method in the case when DRM is implemented in a proprietary file format rather than a normal EPUB file format. Amazon's case is one of the strongest counterexample to our attack. Amazon's DRM uses a proprietary e-book file format. Hence, even though we could spoof the contents, additional research is required to analyze the file structure and recover it.

VII. Conclusion

We proposed a new mechanism to circumvent DRM protection on mobile e-book applications by exploiting the structural weakness of the `Webview` component on Android.

In most e-book applications on Android, content decryption and viewer modules were implemented as independent components with some interfaces. When WebView is used for rendering e-book contents, such interfaces were limited by a few available options. Thus, we can extract the plaintext e-book content through such an interface through a reverse engineering analysis.

In future work, we plan to generalize our framework to circumvent DRM protection in e-book applications on Android by optimizing our implementation with a large sample of e-book applications.

Acknowledgment

We thank E. Kim, T. Kim and W. Aiken who provided valuable feedback on the earlier drafts of this paper. This research was supported by Basic Science Research Program through the NRF (2017R1D1A1B03030627), and the MSIT under the ITRC (IITP-2017-2015-0-00403) supervised by the IITP.

References

[1] Q. Liu, R. Safavi-Naini, and N. P. Sheppard, "Digital Rights Management for Content Distribution," in *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers*, 2003.

[2] A. Pretschner, M. Hilty, and D. Basin, "Distributed Usage Control," *Communications of the ACM*, vol. 49, no. 9, pp. 39–44, 2006.

[3] F. Hartung and F. Ramme, "Digital rights management and watermarking of multimedia content for m-commerce applications," *IEEE Communications Magazine*, vol. 38, no. 11, pp. 78–84, 2000.

[4] T. Hauser and C. Wenz, *DRM Under Attack: Weaknesses in Existing Systems*. Springer Berlin Heidelberg, 2003, pp. 206–223.

[5] Microsoft, "Microsoft PlayReady Content Protection Technology," online; accessed 2015.

[6] R. Wang, Y. Shoshitaishvili, C. Kruegel, and G. Vigna, "Steal This Movie: Automatically Bypassing DRM Protection in Streaming Media Services." in *USENIX Security Symposium*, 2013.

[7] M. Stamp, *Information Security: Principles and Practice*, 2nd ed. Wiley Publishing, 2011.

[8] V. Oksanen and M. Välimäki, "Transnational advocacy network opposing DRM – A technical and legal challenge to media companies," *International Journal on Media Management*, vol. 4, no. 3, pp. 156–164, 2002.

[9] J. Choi, W. Aiken, J. Ryoo, and H. Kim, "Bypassing the Integrity Checking of Rights Objects in OMA DRM: A Case Study with the MelOn Music Service," in *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication*, 2016.

[10] R. Iannella, "Digital rights management (DRM) architectures," *D-Lib Magazine*, vol. 7, no. 6, pp. 0–13, 2001.

[11] J. E. Cohen, "DRM and Privacy," *Commun. ACM*, vol. 46, no. 4, pp. 46–49, 2003.

[12] J. Xu, S. Li, and T. Zhang, "Security Analysis and Protection Based on Smali Injection for Android Applications," in *International Conference on Algorithms and Architectures for Parallel Processing*, 2014.