DroidMark: A Lightweight Android Text and Space Watermark Scheme Based on Semantics of XML and DEX

Lingling Zeng[1], Wei Ren[1](✉), Min Lei[2], and Yu Yang[2]

[1] School of Computer Science, China University of Geosciences, Wuhan, China
weirencs@cug.edu.cn

[2] Information Security Center, Beijing University of Post and Telecommunications, Wuhan, China
leimin@bupt.edu.cn

Abstract. Android platform induces an open application development framework to attract more developers and promote larger market occupations at the same time. However, the open architecture also makes it easier to reverse engineering and application piracy. These result in the property loss for developers and companies, and increase the risks of mobile malicious code. Copyright protection for android application is thus of significant importance. Currently, many solutions for application copyright protection apply overload methods, assuming the availability of source code, which could be impractical for a large scale application protection. In this paper, we propose a lightweight copyright protection method for android application called DroidMark. The copyright is protected by text and space watermark based on semantics of xml and dex. Functional files are chosen as watermark carriers to increase watermark semi-fragileness and concealment. And the DroidMark can be accomplished without secret keys. Models and algorithms are proposed and analyzed all sidedly. The experiment results and analysis justified that DroidMark is secure and efficient.

1 Introduction

In recent years, copyright consciousness has been highly valued unprecedentedly. The copyright in Apps shows its value in both economy and society.

For example, "Repackage", one of the android potential safety hazard [1] for application copyright, is a technique to produce fake Android applications on the basis of the legit App. Fake Apps cause damage to the legit achievements and benefits of the developer, while reduce the user experience for customers at the same time. Moreover, fake Apps may be embedded malicious codes, which may threat to user privacy and property security. Therefore, as the most widely use mobile platform worldwide, copyright protection for large-scale Android applications attracts more and more attentions in research communities.

© Springer International Publishing AG 2018
L. Barolli et al. (eds.), *Advances in Internetworking, Data & Web Technologies*,
Lecture Notes on Data Engineering and Communications Technologies 6,
DOI 10.1007/978-3-319-59463-7_75

Till now, many schemes have been proposed in app copyright protection. However, those methods induce a large amount of computation overhead such as reverse engineering detection and code manipulation, which may not be suitable for the app protection in a large scale. Therefore, a lightweight and covert method in terms of computation cost will be more applicable. Watermarking is a technique to provide data integrity and authenticity in public or in private. For this perfect attribute, watermarking can be applied in application copyright protection.

Current methods for watermarking-based application protection can be roughly divided into three folders. First of all, put copyright information directly into the APK provides chances to adversaries to extract the watermark by keyword search, and modify and forge the information without detection. Secondly, encrypts the information with asymmetric key brings high challenge for key storage and distribution in an open environment. Lastly, the method embeds the watermark in an additional carrier file will be noticed and separated easily, resulting in the loss of the watermark. Although several watermarking-based schemes have been proposed, most of them are not suitable enough for application copyright protection.

Through the analysis above, we can conclude that the watermarking-based copyright protection should accurately tackle three aspects: good choice of carriers, message pretreatment for hiding, and easiness of embedding and extracting algorithms for requirement of lightweight and large-scale. For these regards, we proposed a novel android watermark method for the protection of Android application copyright, called DroidMark. The method is designed to be consist of DroidMark-XML and DroidMark-DEX, distinguished on the basis of the choice of the carrier files.

The contribution of this paper can be summarized as follows:

1. Secret key and encryption is avoided. The embedding and extracting process only need to scan the carrier file for characteristic strings and fields, and embed or extract watermarking information on the basis of the features and semantics of the carrier file.
2. The modification of the application is able to tracked. The watermark is semi-fragile, therefore, DroidMark can detect the modification of Apps and maintain the copyright information in Apps. Any modification of Apps, repackage, re-optimization, will be noticed.

The rest of the paper is organized as follows. Section 2 gives an overview on relevant prior work. And Sect. 3 provides the detailed description of our proposed methods and algorithms. Analyzation and evaluation for both security and performance of the scheme are provided in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Related Work

Some android copyright protection schemes are proposed recently. Sanghoon Choi et al. [2] proposed a copyright protection technology based on forensic mark, which is marked in the classes.dex file using the IMSI of the buyer to identify illegal Apps. This method aimed at personal software validation, but not simplified for the software copyright protection. Sung Ryul Kim et al. [3] proposed a hybrid copyright protection design on android applications that combines two proposed techniques: online execution class and encryption-based copyright protection. This method needs the participation of secret key, which is hard on preservation and distribution with software.

Wu Zhou et al. [8] proposed AppInk, which designs a dynamic graph based watermarking mechanism for Android Apps. This scheme involves secret key, and the key is needed before extract, therefore, the key cannot be embedded in watermark and is not suitable for negotiability. Yingjun Zhang and Kai Chen [7] proposed a picture-based watermark for Android Apps. The extraction of the scheme has to use the same sequence of events to find basic blocks in the execution path, which has the same problem with the former scheme. To solve these problems, we suggest text watermarking, which doesn't require keys in the process.

To satisfy the resistance of compilation and packaging, the watermark for Android copyright protection can based on text semantics. Some watermarks schemes depend on the semantics of the text. Mikhail J. Atallah et al. [4,5] first proposed the natural language watermarking scheme, using the syntactic structure of the text. This method preserves the inherent properties of the text while embedding. Hassan et al. [6] proposed the natural language watermarking algorithm by performing the morphosyntactic alterations to the text. Mercan et al. proposed an algorithm by using typing errors, acronyms and abbreviations like cursory text in char, emails etc. However, none of these methods may not be suitable for android applications, and the efficiency and capacity is still waiting to be improved.

3 Proposed Scheme - DroidMark

3.1 Design Goals

To insure the efficiency of the design, our scheme should achieve the following properties:

- ***Identifiability:*** It ensures that the scheme can identify the correct watermark in the carrier file. This is the basic property for extraction.
- ***Concealment:*** It guarantees that the attacker cannot distinguish the watermark information even if they obtain the carrier file. This property requires that the string added to carrier should be similar to the original content, and be as natural as possible.
- ***Transparency:*** After addition of the watermark information into the xml file, the APK should be able to operate normally, which indicates that the addition of watermark information should not be detected by operating the APP.

- ***Semi-fragile:*** It ensures the stability of the watermark in the circumstance of being recompiled and attacked. Besides, the modification or damage of the watermark can be able detected.
- ***Capacity:*** The high capacity will improve the robustness and applicability of the scheme, especially when watermarks have large volume but carriers have small volume. Achieving high capacity as possible is a necessary goal for the proposed scheme.
- ***Efficiency:*** It ensures the high performance of embedding and extracting in terms of computation overhead and timing cost.

3.2 Notations

For fast checking the short notation for Sect. 3, we list the major notations used in the remainder of the paper in Table 1.

Table 1. Notation

Symbol	Meaning
W	Array W[0,1 ... L−1] to store watermark in a format
L	Length of the array W
Loc	Array Loc[0,1 ... N−1] to save field location
N	Length of the array Loc
S_1, S_1'	String "</application>" and "<application> </application>"
S_2, S_2'	String "/>" and "</>"
S_3, S_3'	String "</activity>" and "<activity> android:name=Cache.i"
S_{3Name}	Content of Row With String "android:name"
$Cache$	String in the cache
APK_w, APK_o	Apk File with or without Watermark
CAR_w, CAR_o	AndroidManifest.xml file with or without Watermark
WTM_p, WTM_b	Plaintext or Binary watermark information
i, e, b	Random number, $i \in [1, 10]$, $e \in \{1, 3, 5, 7, 9\}$ and $b \in \{2, 4, 6, 8, 10\}$
$Seed$	Seed of random-number generator
T	Length of $Seed$
$H[j]$	Array to save random address

3.3 Scheme Construction

In this section, we will present our scheme construction for lightweight copyright protection for Android applications. The scheme is described as follows in algorithm form, followed by explanations:

3.3.1 Scheme Construction for DroidMark-XML

- **Embedding Procedures of DroidMark-XML**

1. Pretreat: Before embedding watermark into the APP, pretreat the watermark information and prepare the carrier file.
   a. $APK_o \rightarrow CAR_o$: get the AndroidManifest.xml file as the carrier file by decompiling the APK_o of the application to be marked.
   b. $WTM_p \rightarrow WTM_b$: turn the watermark into binary data, and save the data in $W[0, 1 \ldots L-1]$. Calculate the array length as L.
2. Embed: Scan AndroidManifest.xml file from beginning to the end successively for S_1, S_2 and S_3, and embed the binary data $W[0, 1 \ldots L-1]$ in turn by adding strings S_1', S_2' and S_3' respectively at the corresponding position in the following rules:
   a. f_{embed_1}: $S_1, S_2 \rightarrow CAR_o$. After find S_1 (or S_2), identify $W[j], j \in (0, L-1)$. If $W[j] = 0$, generate b, and put equivalent amount of string $S1'$ (or $S2'$) after $S1$ (or $S2$). While if $W[j] = 1$, generate e, and do the same as aforementioned.
   b. f_{embed_3}: $S_3 \rightarrow CAR_o$. At the process of scanning, $Cache$ will just record the latest string after string S_{3Name}. After find S_2 in the carrier, identify $W[j]$, and generate random number b or e. Also, add string $S3'$ after $S3$.
3. Recompile: $f_{compile_r}$. When $j = L$, which identifies the end of embedding, stop scanning and save the new AndroidManifest.xml. Recompile APK with the new file with watermark to get APK_w, in this case, the watermark will be embedded into an application.

- **Extracting Procedures of DroidMark-XML**

1. Decompile: Decompile APK to get AndroidManifest.xml file, which is the CAR_w, before extracting watermark from APK_w.
2. Extract: Scan CAR_w from beginning to the end successively for the specified strings S_1', S_2' and S_3', and extract the binary data in turn in the following rules:
   a. $f_{extract_1}$: $CAR_w \rightarrow S_1'$, S_2'. After find S_1' or S_2' in the CAR_w, count the amount of successive strings as x, an even x represents binary 0, while the odd x represents binary 1.
   b. $f_{extract_3}$: $CAR_w \rightarrow S_3'$. Set the initial value of $Cache$ as "null". At the process of scanning, compare the latest S_{3Name} with $Cache$. If S_{3Name} contains $Cache$, which means S_{3Name} is an anthropogenic watermark, get the rest part of S_{3Name} as ".i". Judge i if it's an even or an odd and get the binary data as above.
   c. Transform: Scan to the end of the CAR_w, and record the binary data successively in a new file. Finally, turn WTM_b into WTM_p. In this case, we can extract the watermark information.

3.3.2 Scheme Construction for DroidMark-DEX

- **Embedding Procedures of DroidMark-DEX**

1. Pretreat: Before embedding watermark into the APP, pretreat the watermark information and prepare the carrier file.
   a. $APK_o \rightarrow CAR_o$: Get the classes.dex file by decompressing APK_o to be marked.
   b. $WTM_p \rightarrow WTM_b$: Turn the plaintext or cryptographic text watermark into hexadecimal data.
   c. $CAR_o \rightarrow Loc$: For the present dex file, some fields keep empty or insignificance in the whole lifecycle of the application. For this reason, these fields can be used for watermarking or digital forensics. As far as we know, there's redundancy information exist between *Header_item* and *Map_list*. In this experiment, we scan for the field *unused* in Map_list. And store the corresponding location of file in the array *Loc*.
2. Embed:
   a. *Seed*: $H[j]$: generate a number as *Seed* to produce random numbers in the function $RAND(Seed)$, calculate the length of *Seed* as T. If the address is used, generate $H[j] = H[j] + 1$;
   b. $WM, H[j] \rightarrow CAR_o : WTM_b, Seed$: Embed *Seed* in the first T unit of *Loc*, and WM in the corresponding $Loc[H[j]]$.
3. Repackage: $CAR_w \rightarrow APK_w$: After finish embedding, recalculate checksum of dex file, and repackage the APK into APK_w. In this case, the watermark will be embedded into an application.

- **Extracting Procedures of DroidMark-DEX**

1. Decompress:Decompress its APK to get classes.dex file, which is CAR_w, before extracting watermark from APK_w.
2. Extract:
   a. CAR_w: $Seed, Loc$: Scan for satisfactory fields and number these locations in hexadecimal unit. Then store the corresponding location of file in the array *Loc*.
   b. *Seed*: $H[j]$; $CAR_w, H[j] \rightarrow WM : WTM_b$: Extract the first T unit of *Loc* to get *Seed*. Generate the random address using function. If the generation repeated, extract from one unit after. In this way, the WTM_b is extracted.
3. Transform: Finally, turn WTM_b into WTM_p. In this case, we can extract the watermark information.

4 Security Analysis and Performance Evaluation

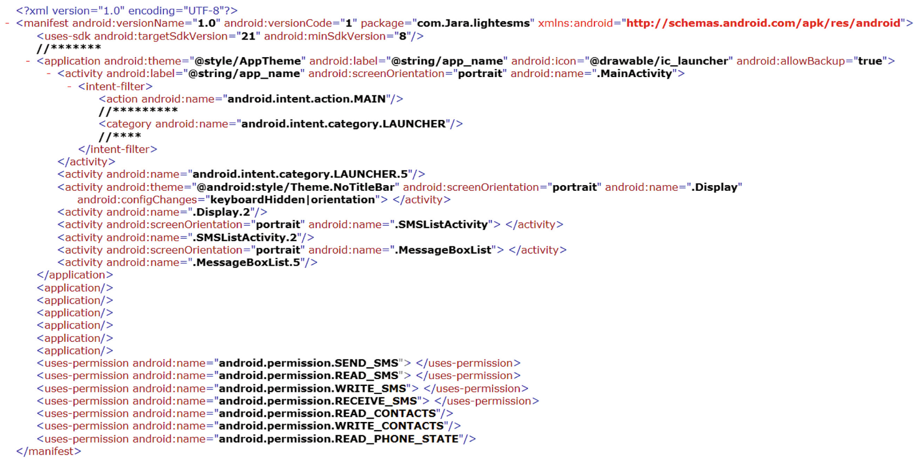
In this section, we evaluate the security and property performance of scheme in experiments. All the following experiments are based on language C and python, and tested on hardware AMD Athlon(tm) II X4 630 with Intel Pentium 2.7 GHz processor and 6 GB memory.

4.1 Security Analysis

To evaluate the security performance of the scheme, we analyze the four security properties in its design goal for DroidMark to evaluate its security performance. We analyze the security of our proposed scheme in the following two aspects under the assumption that APK is arranged:

- **Soundness and Concealment:**

DroidMark cannot be perceived by mainstream watermark detector.



```
  <?xml version="1.0" encoding="UTF-8"?>
- <manifest android:versionName="1.0" android:versionCode="1" package="com.Jara.lightesms" xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-sdk android:targetSdkVersion="21" android:minSdkVersion="8"/>
    //*******
  - <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@drawable/ic_launcher" android:allowBackup="true">
    - <activity android:label="@string/app_name" android:screenOrientation="portrait" android:name=".MainActivity">
      - <intent-filter>
          <action android:name="android.intent.action.MAIN"/>
          //*********
          <category android:name="android.intent.category.LAUNCHER"/>
          //****
        </intent-filter>
      </activity>
      <activity android:name="android.intent.category.LAUNCHER.5"/>
      <activity android:theme="@android:style/Theme.NoTitleBar" android:screenOrientation="portrait" android:name=".Display"
        android:configChanges="keyboardHidden|orientation"> </activity>
      <activity android:name=".Display.2"/>
      <activity android:screenOrientation="portrait" android:name=".SMSListActivity"> </activity>
      <activity android:name=".SMSListActivity.2"/>
      <activity android:screenOrientation="portrait" android:name=".MessageBoxList"> </activity>
      <activity android:name=".MessageBoxList.5"/>
    </application>
    <application/>
    <application/>
    <application/>
    <application/>
    <application/>
    <application/>
    <uses-permission android:name="android.permission.SEND_SMS"> </uses-permission>
    <uses-permission android:name="android.permission.READ_SMS"> </uses-permission>
    <uses-permission android:name="android.permission.WRITE_SMS"> </uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_SMS"> </uses-permission>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  </manifest>
```

Fig. 1. Effect of embedding in XML

We put an small-scale experiment on DroidMark-XML, and the effect is shown in Fig. 1. The binary data embedded is "1, 1, 0, 1, 0, 0, 1, 0", and the random number generated by the tool is "7, 9, 4, 5, 2, 2, 5, 6", which has the same oddity as the binary data. From Fig. 1 we can see that the embedding of watermark uses three methods to induces only slightly modification on the basis of original strings conform to the file semantics. In this way, the format of file embedded watermark is similar to the normal AndroidManifest.xml file. Therefore, the watermark is hard to be distinguished apart from the carrier. Watermark embedding and extracting rely on the detection and judgment of characteristic strings rather than the secret "Key". The scheme is free of insecurity in the process of "Key" distribution.

- **Identifiability:**

DroidMark can extract watermark accurately. DroidMark achieves good confusion by using the semantics of carrier files, and is secure in the mainstream watermark detector. Instead, DroidMark can extract the watermark embedded in carriers accurately.

- **Transparency:**

DroidMark has no influence on the operation of the APP. We did experiment on 100 different APKs of embedding watermark in the two carrier files. The experiment indicates the overhead on the installation and operation are the same in millisecond. Therefore, we conclude that DroidMark has no influence on APP's operation.

- **Semi-fragile:**

On the one hand, DroidMark can defend against the attack of decompilation and recompilation, on the other hand, the artificial modification will be aware of. We simulate attacks of recompiling and decompiling, turn out that the watermark in both xml and dex file are stable in the experiment. Therefore, the experiment justified that DroidMark can defend both attack. Besides, AndroidManifest.xml is a configuration file, while classes.dex is for execution, both are the essential part for an APP. If the adversary modifies the app artificially, both file will have to be modified incidentally. The destruction of the watermark will be simultaneous with the modification of carrier files. The damage of the watermark can thus both confirm the counterfeit of the APP.

4.2 Performance Analysis

The performance of DroidMark can be evaluated in two folders: efficiency of embedding and extraction, and capacity of watermark. Efficiency is related to the pretreatment of scanning for the location, and the process of embedding and extracting the watermark. Meanwhile capacity is influenced by the size and type of the carrier files, and the algorithm for watermarking as well.

4.2.1 Efficiency

During the process of embedding and extraction, the cost of computation is mainly induced by two aspects: carrier file scan, watermark implant and extraction.

In our algorithm, the length of watermark decide the time for scanning for embedding, while in the process of extraction, it's decided by the size of carrier file. Therefore, the efficiency cost grows along with the scale of the watermark and the length of carrier file. Further more, to inquire which element has more influence, we design two experiments as follows.

- **Efficiency of DroidMark-XML:**

Firstly, we embedded and extracted different watermark in the same Android-Manifest.xml file to control the environment variables of the length of the carrier file. Watermarks range from 1 to 1600 bits increasing by 100 bits, and the Fig. 2 shows that the overhead of the whole process of embedding and extracting watermarks. We can see that the cost of time is linearly other than exponentially increasing with the number of bit of watermarking, which indicates that the overhead increases along with the size of the watermark embedded is within a controllable scale. That is also the inevitable cost for embedding and extracting operations. Therefore, we can conclude that the cost of this proposed scheme is efficient and stable.

Secondly, the length of the file only has fatal effect on the scanning overhead of watermark's extraction. To explore the relationship between the scanning overhead and the length of the carrier file, we design the experiment to embed and extract the same watermark in 8 bits in a series of incremental size of carrier files. We keep the content of watermark as 8 bits to ensure the same cost of embedding and extracting, so that to achieve the influence of the scanning only. Besides, the increasing content of the carrier file is the repetition of central body in 1271 characters based on 1527 characters of the original file. We adopt this method to guarantee the validity of AndroidManifest.xml file, and the arithmetic progression is in favor of statistics of overhead as well. The result of the experiment is shown in the Fig. 3. The graph indicates that the overhead of scanning increases only less than 2 seconds in the file augment of more than 20 thousand characters, which is much less than the overhead of the process of embedding and extracting.

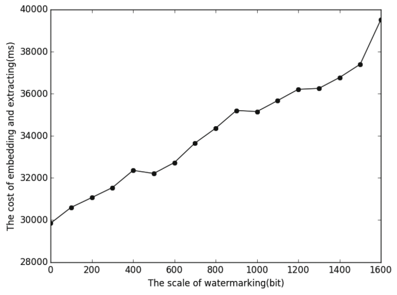


Fig. 2. Overhead in different length of watermark

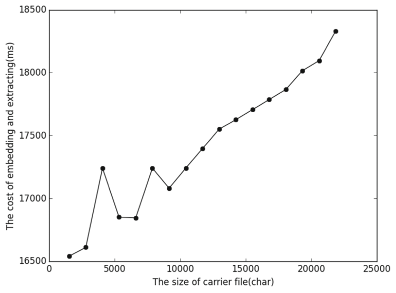


Fig. 3. Overhead in different size of file

- **Efficiency of DroidMark-DEX:**

In the process of DroidMark-Dex, the extra overhead compares to the DroidMark-XML is the confusion of embedding location, which is proved to be low-time-consuming operation. Hence the overhead of DroidMark-DEX has the similar tendency with DroidMark-XML. And for this circumstance and space limitations, the DroidMark-DEX experimental result is abridged.

Therefore, we can draw the conclusion that, the overhead mainly comes from the process of embedding and extraction, and the linearly growth with in a small scale justifies the efficiency and stability of our scheme.

4.2.2 Capacity

The capacity refers to how much bits of the original content are needed to hide one bit watermark information. Under the requirement of practicability and security, the higher capacity indicates the greater ability, applicability, and efficiency of watermark algorithm.

- **Capacity of DroidMark-XML:**

In order to detect the capacity of DroidMark-XML, we choose 100 different AndroidManifest.xml files decompiled from 100 Apps. And we calculate each characters of xml file as the x axis, while collect the totality location for watermark embedding as the y axis. The relationship between x and y is shown in Fig. 4. According to Fig. 4, we draw a baseline slopes 0.2, representing that the capacity of the AndroidManifest.xml is around 20 percent of the full text, which is pretty high compares to other android watermark algorithms. Therefore, we can conclude that the this proposed scheme is stable and high-capacity.

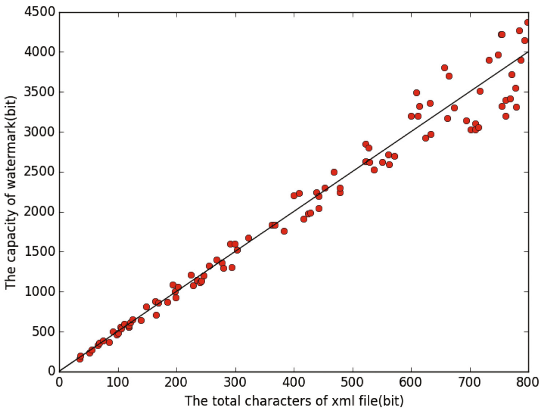


Fig. 4. Capacity of DroidMark-XML

- **Capacity of DroidMark-DEX:**

As for the capacity of DroidMark-DEX, the location in which the watermark is properly embedded is relatively fixed. Therefore, the capacity of DroidMark-DEX is fixed in a range rather than grows linearly. On account of redundancy between *Header_item* and *Map_list* in classes.dex, the available field of dex file we choose are "*unused*" fields in each "*map_list_item*[]" which is used to format the alignment. The amount of unused field is usually less than 20. Calculate this fields, we can conclude that the capacity of DroidMark-DEX is $20*2*8 = 320$ bits at most, which is much less than what in DroidMark-XML. But at the watermark in DroidMark-DEX has better effect of confusion and has tighter organization, which will offer better protection of watermark integrity.

5 Conclusions

In this paper, we proposed an watermark tool, called DroidMark, to protect the copyright of android applications. This method embeds the watermark in application configuration files, which achieves great concealment. At the same time, the semi-fragile property insure the authenticity for DroidMark. More over, the method achieves the disuse of secret key in the process of embedding and extracting, which caters applications property of circulation and lightweight. DroidMark is designed to achieve Android copyright protection, and can also be used in information hiding, secrete communication, provenance-based forensics, and key distribution. The watermarking algorithms and carrier selection guarantee the reliability and applicability of DroidMark. The achievements of security, high-performance and lightweight property of DroidMark is extensively analyzed and thoroughly verified by experiments.

Acknowledgements. The research was financially supported by the National Natural Science Foundation of China under Grant No.s 61310306028.

References

1. Sufatrio, Tan, D.J.J., Chua, T.-W., Thing, V.L.L.: Securing android: a survey, taxonomy, and challenges. ACM Comput. Surv. **47**, 58–102 (2015)
2. Choi, S., Jang, J., Jae, E.: Android applications copyright protection technology based on forensic mark. In: Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS 2012), pp. 338–339 (2012)
3. Kim, S.R., Kim, J.H., Kim, H.S.: A hybrid design of online execution class and encryption-based copyright protection for android apps. In: Proceedings of the 2012 ACM Research in Applied Computation Symposium (RACS 2012), pp. 342–343 (2012)
4. Atallah, M.J., McDonough, C., Nirenburg, S., Raskin, V.: Natural language processing for information assurance and security: an overview and implementations. In: Proceedings of the 2000 Workshop on New Security Paradigms (NSPW 2000), pp. 51–65 (2000)
5. Atallah, M.J., Raskin, V., Crogan, M., Hempelmann, C., Kerschbaum, F., Mohamed, D., Naik, S.: Natural language watermarking: design, analysis, and a proof-of-concept implementation. In: Moskowitz, I.S. (ed.) IH 2001. LNCS, vol. 2137, pp. 185–200. Springer, Heidelberg (2001). doi:10.1007/3-540-45496-9_14
6. Meral, H.M., et al.: Natural language watermarking via morphosyntactic alterations. Comput. Speech Lang. **23**, 107–125 (2009)
7. Zhang, Y., Chen, K.: AppMark: a picture-based watermark for android apps. In: Eighth International Conference on Software Security and Reliability (SERE 2014), pp. 58–67 (2014)
8. Zhou, W., Zhang, X., Jiang, X.: AppInk: watermarking android apps for repackaging deterrence. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security (ASIA CCS 2013), pp. 1–12 (2013)