Journal of Ambient Intelligence and Humanized Computing (2019) 10:295–304
https://doi.org/10.1007/s12652-017-0657-3

ORIGINAL RESEARCH

LMDGW: a novel matrix based dynamic graph watermark

Lingling Zeng[1,2] · Wei Ren[1,2] · Yuling Chen[2] · Min Lei[3]

Received: 9 September 2017 / Accepted: 12 December 2017 / Published online: 20 December 2017

Abstract
Android platform induces an open applicationdevelopment framework to attract more developers and promote larger market occupations. However, the open architecture also makes it easier to reverse engineering, which results in the property loss for both developers and companies, and increases the risks of mobile malicious code. Therefore, copyright protection for android application is of significant importance. A class of copyright protection methods are based on Android watermark. Different from static watermark, the process of embedding and extracting of dynamic graph watermark (DGW) is based on the function path and operation process of the app, which has better concealment. In this paper, we proposed a late-model dynamic watermark based on matrix, called *"LMDGW"*. This method is proposed to overcome the shortcoming of unintuitive and vulnerable property of traditional numberal DGW. We encode a matrix with low rank into a watermark graph, and embed the graph construction statements into smali code. With the containing of sensitive block characteristics, LMDGW is able to perceive and locate the changes in the specific block. Besides, LWDGW has great performance in tamper-proof attacks. Experiment results and analysis justified that LMDGW has great data rate and robust, and is available in sensitive code locating. LMDGW is proved to be an intelligent watermarking scheme, and is enlightening for intelligent security.

Keywords Dynamic graph watermark · Matrix watermark · Matrix encoding · Changes locating · Matrix recovery

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 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.

The existing watermark can be divided into two types: static watermark and dynamic watermark. The static watermark embeds the watermark information directly into App without execute the App, while the dynamic watermarks are constructed at the runtime of program and stored in the programs execution state. Collberg and Thomborson (1999) proposed the first dynamic watermarking algorithm based on graph, which is typical dynamic graph watermarking (DGW). It embeds watermark into the graph topology which is dynamically constructed by code execution with predetermined input sequence. And the later scheme of dynamic watermark are basically the improvements based on DGW, for example the two famous improvements: PPCT and IPPCT.

The DGW is one of the most effective watermarking techniques, and the key of DGW is the graph encoding. As the widely studied DGW, the great robustness of PPCT

✉ Wei Ren
weirencs@cug.edu.cn

Yuling Chen
61997525@qq.com

[1] School of Computer Science, China University of Geosciences, Wuhan, Hubei, People's Republic of China

[2] Guizhou Provincial Key Laboratory of Public Big Data, GuiZhou University, Guiyang, Guizhou, People's Republic of China

[3] Information Security Center, Beijing University of Post and Telecommunications, Beijing, Beijing, People's Republic of China

and IPPCT is due to structural advantages. However, there are still critical shortages.

The watermark information for PPCT and IPPCT is an unique sequence number. A developer firstly constructs a binary tree, and then calculate the (number) according to the tree, and makes the sequence number as the watermark of the app. However, the sequence number itself contains no meaning of copyright, which could bring the following concerns: 1. On the premise of without a trusted third party (TTP): the leakage of the sequence number will be a critical damage, anyone who knows the number could claim to be the owner of the App; 2. On the premise of having a trusted third party (TTP): when a new App application applies for acceptance, if there's a checking of whether there's another watermark contains in the new App, the problem lies in the methods to extract other watermarks without knowing the extracting key, which is the trigger statements (Zhou et al. 2013). The workload of this action is enormous. Otherwise, the TTP may offer the legal right to a repackaged App, which may cause the copyright problem.

In addition, the DGW is embedded to the smali code of the App. Therefore, the watermark may have the ability to detect the changes in the malicious changes of a repackaged App.

To meet the requirements of containing a visually identifiable watermark, and overcome the challenge of automatic detection for malicious changes in a repackage App, we proposed a new type of dynamic watermark called LMDGW. The contribution of this paper can be summarized as follows:

1. We put low-rank matrix picture into use of watermark protection, which, to the best of our knowledge, is the first scheme to combine the dynamic graph watermark with the matrix, and the matrix picture makes it possible for visually identifiability of a watermark, which makes it more difficult for counterfeiting.
2. We design the watermark to be semi-fragile, which is robust to normal operations and become fragile towards malicious distortion, for better detection of the suspicious repackaged malwares. Therefore the position of malicious modification can be preliminarily perceived by watermark noise extraction.
3. This paper realize an intelligent watermarking scheme, which can detect and locate the suspicious change of an repackaged Android application, and is enlightening for intelligent security.

The rest of the paper is organized as follows. Sect. 2 gives an overview on relevant prior work. And Sect. 3 provides some propaedeutics for the preparation of proposing LMDGW. Sect. 4 contains the detailed description of our proposed methods and algorithms. Analysis and evaluation for both security and performance of the scheme are provided in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Related work

Currently, there are four topologies that DGW adopts, including Radix-k, Permutation Graph, PPCT (Planted Plane Cubic Tree) and IPPCT (Improved PPCT). Collberg and Thomborson (2007) suggested two enumeration methods and families of graphs that can be used for embedding: Radix-k encoding based on a circular linked list of k-1 nodes and Enumeration encoding with a permutation of the numbers <0,..., n−1>, whose ideas are to index the watermark graph according to some enumeration method, and then use the index to represent the watermark number. Palsberg (2000) was the first to establish the relationship between PPCT with integer, and proposed effective PPCT enumeration encoding scheme. Wang Yong (2013) presented IPPCT, which integrates the advantages of Radix-k and PPCT. IPPCT not only greatly improved coding efficiency but also built a valuable software watermarking database which enhanced the ability to resisting the conspiracy attack.

Reference presented a way of representing a watermark with an indexed graph using a Parent-Pointer Tree. The advantage of Radix-k encoding enumeration and permutation coding over others is that the coding methods are simple to implement on the computer, and these watermarking schemes take full advantages of their all pointers within the corresponding graph which own higher data rate. However the performance in term of stealth and robustness is not satisfied. On the contrary, the PPCT method characters with better stealth, robustness but unsatisfied data rate. Therefore, many researchers proposed various new PPCT structures. Reference presented an improved PPCT structure (IPPCT), whose essence is to take advantages of Radix-k scheme and PPCT structure so that it can have better data rate. IPPCT combines radix-k circular linked list and PPCT structure, which takes full advantage of the right pointer of the free leaf nodes; however, there are unused middle nodes left. Therefore, schemes, such as IIPPCT, proposed improved IPPCT structure, which have the advantage of the IPPCT as well as coding based on the middle nodes and leaf nodes, so that the range of watermark number can be enlarged and receive a higher data rate.

However, the shortage to adopt the integer into watermark is the insignificance and non-restorability of the sequence number. Besides the combination of the tree with integers, there's should have a more meaningful method to overcome these detects. In this paper, we proposed a late-model DGW based on matrix graph, which makes the DGW become a readable watermark. And the research of the Matrix Recovery makes it possible for graph watermark's self-repair.

In Sect. 4 we will present the details of our DGW scheme, and evaluate our scheme in Sect. 5.

3 Preliminaries

In this section, we will briefly introduce the conception of grayscale graph, dynamic watermark and matrix recovery for the readers without background knowledge of these field. As for readers who are familiar with these knowledge, please omit this section and jump directly to the Sect. 4.

3.1 Dynamic watermark

The dynamic watermark is constructed at the runtime of the app. The first dynamic watermarking algorithm based on graph is proposed by Collberg and Thomborson, which is typical DGW. It embeds watermark into the graph topology which is dynamically constructed by the App execution. Currently, there're four basic topologies DGW adopts, including Radix-k, PG (Permutation Graph), PPCT (Planted Plane Cubic Tree) and IPPCT (Improved PPCT). We'll give a brief introduction of these encoding methods, which are instructive to us. And in the form of coding we make bold innovations in our scheme.

3.1.1 Radix-k encoding

Radix-k encoding is based on a circular linked nodes list. One of the pointer is used to maintain the list topology while the another pointer is used to encoding watermark graph. Radix-k encoding encodes 0 if pointer is null, encodes 1 to self-pointer, and encodes 2 to the pointer to the next node, etc. The encoding range with n nodes is $[0, (n+1)^n - 1]$. The example of Radix-6 nodes encoding graph is shown in Fig. 1, the encoding is "$61 * 73 = 3 * 6^4 + 2 * 6^3 + 3 * 6^2 + 4 * 6^1 + 1 * 6^0$".

3.1.2 Permutation graph encoding

Similar to the Radix-k graph topology, permutation graph encodes a watermark with a permutation of 1 to n-1, and with the circular linked nodes list. The encoding range with n nodes is [0,n!-1]. For example, integer 17 can be indicated by the permutation <3, 4, 2, 1>. Figure 2 shows the key diagram of permutation graph encoding method.

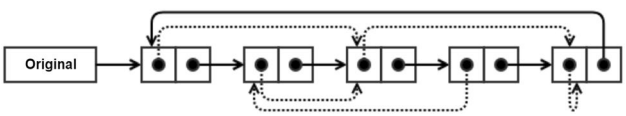

Fig. 1 Encoding methods for $LowR_M$

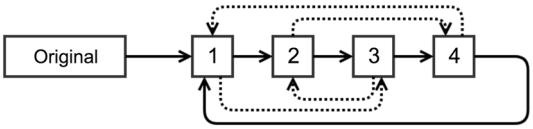

Fig. 2 Encoding methods for $LowR_M$

3.1.3 PPCT and IPPCT encoding

Fig. 3 illustrates the topology of PPCT and IPPCT. PPCT is essentially a binary tree with an original node, points to the root of the binary tree. Each leaf has a self-pointer and two other outgoing pointers, and all leaves, including the original node, constitute a circular linked list. According to Catalan number theory, the PPCT graph with n leaves encoding ranges from 0 to $\left[\frac{C_{n-2}^{n-1}}{n} - 1\right]$.

IPPCT is the combination with PPCT and Radix-k encoding method, and greatly improves coding efficiency. And the range with n nodes is up to $\left[\frac{C_{2n-2}^{n-1}}{n}\right]$.

3.1.4 Data rate

The data rate expresses the quantity of data embedded in the carrier. The definition is given by Collerg and Thomborson (1999). The information source entropy is also used in the definition of data rate. The entropy value is a concept in information theory that is used to measure the degree for the order of information. Suppose the watermark $w \in W$ and information source entropy $H(w) = -\sum_{i=1}^{n} p(w_i) log_2 p(w_i)$.

$|S(p)| = Max_{L \in dom(p)} |S(p, L)|$ represents the maximum resource occupied by P when the input sequence L has different value with double bytes unit, and $|P|$ represents the carrier size P with double byte unit.. In result, the static embedding

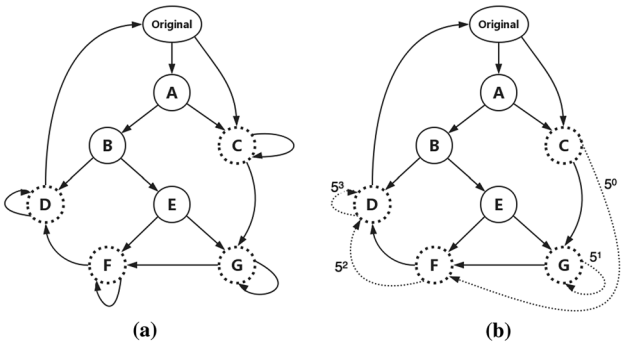

Fig. 3 Encoding methods for $LowR_M$

rate of the carrier P_w is: $\frac{H(w)}{max(1,|P_w|-|P|))}$, While the dynamic embedding rate is: $\frac{H(w)}{max(1,|S(P_w)|-|S(P)|)}$.

3.2 Matrix

A grayscale graph consists of pixels. Each pixel of the ordinary color image consists of three-primary colours: R (Red), G (Green) and B (Blue). The three pigments make up an ordinary color pixel, for example, the (R,G,B) value of color red is (255,0,0) while yellow is (255,255,0). As for the grayscale image, the value of RGB are the same, for example, the white is (255,255,255) while the black is (0,0,0).

In this paper, we use the grayscale image as the watermark graph. Therefore, each pixel can be expressed by just one value. With a strict array of pixels, the image shares the same structure with matrix. Therefore, to put the pixel value as the number in a matrix, the grayscale image can be transferred into a matrix.

3.3 Matrix recovery (Robust PCA)

Matrix recovery, namely Principal Component Analysis (PCA) has wide applications in scientific and engineering fields. It assumes that the given high-dimensional data lie near a lower-dimensional linear subspace, which is called low-ranking matrix. Therefore, the original matrix can be recovery based on the corresponding low-ranking matrix. To large extent, the goal of PCA is to efficiently and accurately estimate this low-dimensional subspace.

Suppose a matrix A of size $\mathbb{R}_{m\times n}$ with rank r, where $r \ll min(m,n)$. In the process of transformation, the matrix may be corrupted by errors or noises. In general, we model the observed matrix D to be a set of linear measurements on the matrix A, subject to noise and gross corruptions i.e., $D = L(A) + \eta$, where L is a linear operator, and η represents the matrix of corruptions. Matrix recovery is to recover the true matrix A from D.

Matrix recovery is widely used in the field of video and picture. And to the best of our knowledge, matrix recovery hasn't be used in the recovery of a watermark. And with the possibility of watermark corruption, the matrix recovery in the watermark field may have greatly utility value.

Until now, many schemes for matrix recovery have been proposed, and some of them are with impressive performance in theory and practice. YC Eldar and D Needell even proposed a }}*Benchmark*″ for precisely ascertaining the strength of these different methods. The mainstream methods for matrix recovery are based on four approaches: (1) the Accelerated Proximal Gradient Approach, (2) the Iterative Threshold Approach, (3) the Dual Approach and (4) the Methods of Augmented Lagrange Multipliers.

One of our goal in this paper is to put the use of low-ranking matrix, which make it possible to put the matrix recovery into use of Android watermark to recover the corrupted watermark graph, and extract noise of the repackaged App.

4 Proposed scheme

Fig. 4 depicts the system model of scheme architecture. The scheme consists of three components: watermark graph disposal, watermark embedding and watermark extracting, in which the watermark dispose contains watermark preparation, encoding and recovery modules.

For better expression of the proposed scheme, the notation of this paper is listed as follows:

In the process of watermark embedding, the smali file $Smali_O$ is decompiled from the original APK APK_O. At the same time, APK developer choose a watermark graph EmW_{Gph} or a Low-Rank Matrix $LowR_M$, representing the mark of the App. Then a Matrix graph M_{Gph} structure will be build out of $LowR_M$ according to the algorithm Alg.1, described in Sect. 4.2. The graph is build similar to the control flow graph of an App. Therefore, after confusion by

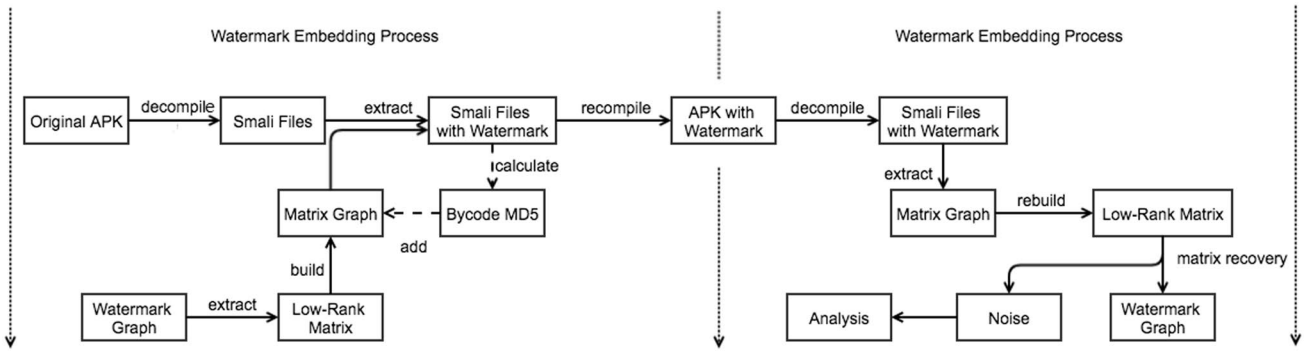

Fig. 4 System model of LMDGW

proguard, the graph has great concealment as the topology of PPCT and IPPCT. The M_{Gph} will be embedded in the $Smali_O$ according to the algorithm Alg.2, described in Sect. 4.3. And the Tri_{Str} will be extracted before embed watermark Nodes, and saved as the symmetric key for watermark embedding and extracting.

In the process of extracting, we firstly trigger the APK_W with Tri_{Str} to operate embedded statements for the matrix graph nodes generation (Table 1). Then the low-ranking matrix will be constructed according to the reverse algorithm Alg.1.

This scheme is focused on the matrix encoding and malicious locating. We adopt the graph matrix as the watermark. On the one hand, matrix can increase the relationship between the watermark and the host. And on the other hand, the APK_W file is under the suspicious of being repackaged. Therefore, the watermark information in APK_W may contain slight or severe noise. For this reason, we can recover the $LowR_M$ extracted from M_{Gph}. And the noise in ExW_{Gph}, which may contains the intention of the attacker or the repackaged App, can be extracted through the recovery at the same time. Therefore, through the analysis of the noise, we're able to locate the position of the sensitive modification.

In the rest of this section, we will give a detailed description of LMDGW our proposed scheme. And the experimental evaluation will be conclude in Sect. 6.

4.1 Pre preparation

DGW is embedded into the APK_O in a graph topology. The extraction of the DGW is driven by the event in Android application. Therefore, we choose smali file as the watermark host, and trigger the generation of the watermark graph by the execution pathes.

Table 1 Notations

Symbols	Meaning
APK_O	Original *APK* file without watermark
APK_W	*APK* file with watermark
$Smali_O$	Smali files extracted from APK_O
$Smali_W$	Smali files extracted from APK_W
EmW_{Gph}	Original watermark graph, representing the unique trademark of the developer
ExW_{Gph}	Extracted watermark graph from APK_W
$LowR_M$	Low-rank matrix of EmW_Gph
M_{Gph}	Digraph built from $LowR_M$
wmNode	Nodes for M_{Gph} encoding
W_{Nse}	Noise in ExW_{Gph}
Tri_{Str}	Trigger strings for DGW extraction

In the preparation of watermark embedding, given an APK_O file without watermark, we need to firstly decompile the file and get the corresponding $Smali_O$. We set the $Smali_O$ as the watermark carriers in the process of watermark embedding and extracting. In general, there're several $Smali_O$ files in one APK_O. After the watermark hosts are ready, a watermark graph is required, representing the unique mark of an APK_O. On the basis of containing clear information, the watermark graph should be low-rank matrix to meet the need of matrix recovery, for the watermark has the possibility of containing damages after being repackaged.

4.2 Watermark encoding

For better embedding into the $Smali_O$, and has better concealment, the matrix need to be efficiently encode before being embedded into the hosts. Therefore, before embed the matrix into the $Smali_O$, we need to transform the $LowR_M$ into M_{Gph}. For the weakness of the previous scheme, we proposed an encoding scheme for the low-rank matrix. And the scheme is describe in detail in Alg.1. And for better explanation, we take a model in Fig. 5 to explain the transformation in Alg.1.

We take a 8-order grayscale graph for example. We construct a graph matrix A, whose unique design and content is able to represent our intention for testing. We represent the pixel value from white to black as 0 to 8 for convenience. Each small squares of the matrix represents a pixel in the graph. The graph shown as Fig. 5 contains 100*100 pixels, however, for better explanation, we only display the central parts of the graph, which is 20*20 pixels, and the pixel value of the omitted part are all 0.

With each different line, there's a *wmNode* to save the simplified information for each line. Each *wmNode* contains four sections. We take the first section for the pointer to keep *wmNodes* in order, and the last section for the pointer to construct graph topology. We add a random number in each node, and the ending pointer will point to the corresponding order node to generate complication of the construction. In this case, the construction will be similar to Radix-k topology. The linked structures are very commonly used in Java applications, therefore, it is hard to distinguish these watermarking code from other code. The second section of *wmNode* is for a pointer points to an array that contains the pixel information of each row (or column), while the third section for the column number of the same *wmNode* information, and we take the structure of (m,n) when there're column number in a row, where m represents the start location while n represents the end. To make it more clear, we dispaly the results of Fig. 5 in Fig. 6.

The construction of the graph is under the assumption of low-rank matrix A. Therefore, the pixel value of each row or column remains many multiple relationships. To take this

opportunity, we simplify the matrix by a uniform representation of the rows who has pixel multiple relation. In this way, the amount of node is as same as the rank of a matrix. And the rank of A is far less then the matrix dimension.

The algorithm for low-rank matrix encoding is shown in graph below.

Algorithm 1: Matrix Topology Encoding

```
   Input: graphmatrix  A = ℝ_{m×n}
   Output: watermarkNode[order]
 1 watermarkNode Node struct{
 2       Node.node1 = order;
 3       Node.node2 = pixel imformation in a row;
 4       Node.node3 = column;
 5       Node.node4 = random graph;
 6 }
 7 for (column i from 1 to m) do
 8     order = 1;
 9     flag = loc_start = 0;
10     for (row j from 1 to n) do
11         k = 0;
12         if (pv[i-1][j] != pv[i][j]) then
13             flag = 1;
14         end
15         if (pv[j]!= pv[j+1]) then
16             stringInfo[k] = [loc_start; pv[j]];
17             k++;
18             loc_start = j;
19             continue;
20         end
21     end
22     if (flag) then
23         watermarkNode Node[order];
24         Node[order].node1 = order;
25         for (k' from 0 to k) do
26             Node[order].node2 = stringInfo[k'];
27         end
28         Node[order].node3 = column;
29         seed = seed[j];
30         Node[order].node4 = seed[j];
31         break;
32     end
33     add i ∈ Node[order].node3;
34     order ++;
35 end
36 return watermarkNode[order];
```

4.3 Watermark embedding

The algorithm for watermark embedding is shown in the Alg.3 below. We firstly scan for the methods block array $LowR_{Em[m]}$, which contains sensitive statement according to the rules in Alg.2, and get the execution path for control events statements (including clicking button, menu and list items), and save the execution path to the $Trigger[m]$. We set the number for $LowR_{Em[m]}$ in M, which $M < order$. We set *wateramrkNode* for each $LowR_{Em[m]}$ in the array $loc[k]$.

Algorithm 2: Location Scanning

```
   Input: Smali
   Output: LowR_{Em[m]},Trigger[m]
 1 while (Smali) do
 2     i = 0;
 3     if (method block Block contains sensitive
         statement) then
 4         if (upward statement or method contains
             click() function) then
 5             click()executionpath → Trigger[i]
 6             Block → LowR_{Em[i]}
 7             i++;
 8         end
 9     end
10 end
11 return LowR_{Em[m]},Trigger[m];
```

We calculate md5 value of the smali bycode for each $Loc_{Em[m]}$ as its checkout value. We reason we only check the method contains sensitive statement is that, for the significant concealment, the malicious action added by attackers may lies in the nearby location with original sensitive to make use of the original invoking. Therefore, the malicious changes in the corresponding location in $Loc_{Em[m]}$ block can be identified by the checkout value, and the value will be shown in W_{Nse}.

In the process of embedding, we split topology graph into $G_1, G_2 ... G_m$, and embed the *watermarkNode*[*order*] in the Alg.3. The statement to embed DWG should be written in smali syntax.

Algorithm 3: Watermark Embedding

```
   Input: Loc_{Em[m]},watermarkNode[order],n,loc[k]
   Output: M_{Graph}
 1 for (i=0 to M) do
 2     Loc_{Em[i]} = method block i for embedding;
 3     mMD5[i] = MD5(Loc_{Em[i]});
 4     watermarkNode[loc[i]].node2[1][3] =
         (watermarkNode[loc[i]].node2[1][3] +
         mMD5%256)%256;
 5 end
 6 watermarkNode[1].node4 = watermarkNode[3];
 7 watermarkNode[2].node4 = watermarkNode[6];
 8 ...
 9 watermarkNode[7].node4 = watermarkNode[4];
10 return watermarkNode[order];
```

After finishing the embedding of DWG, we recompile the smali file into an APK_W with Apktool. In this way, the APK_W can be released to the APP Market.

4.4 Watermark extracting

When the verification of an application is demanded, we can extract the watermark from the application. Unlike the

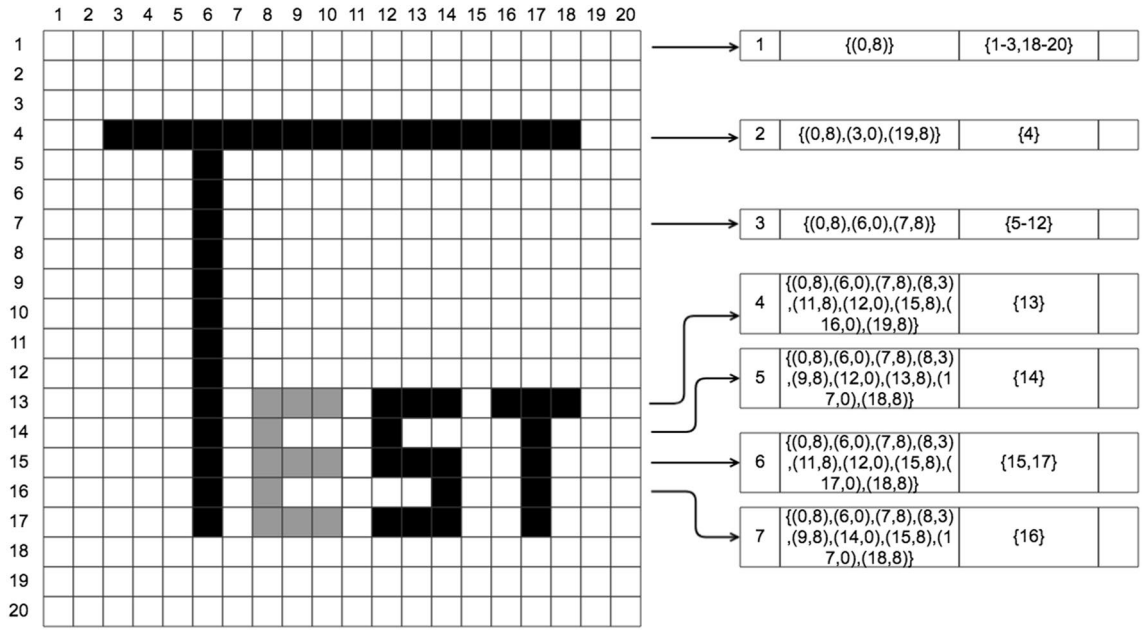

Fig. 5 Encoding for $LowR_M$ – Graph to Nodes

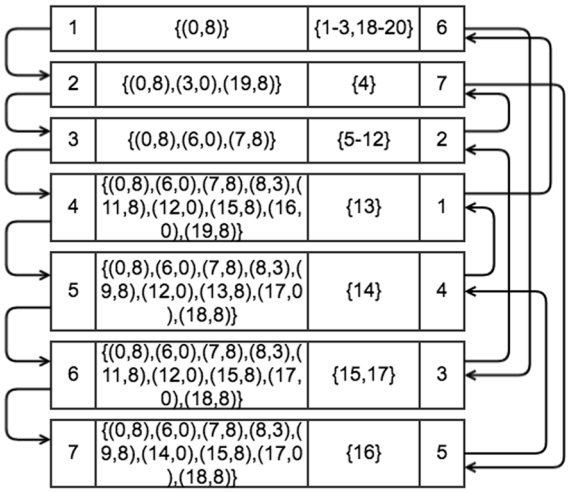

Fig. 6 Encoding for $LowR_M$ – Nodes to Digraph

static watermark, the extraction of dynamic graph watermark is in the process of App operation. Input the watermark $Trigger[m]$, we execute the trigger operation in the extended Dalvik virtual Machine (DalvikVM) to record and capture the watermark generated in the runtime. And we remould the DalvikVM to increase the md5 calculation for the specified method block $Loc_{Em[m]}$ to generate watermark automatically. The graph can be generated and extracted by analysing of the app running on the memory stack structures.

After the graph is extracted, the low-rank matrix graph can be constructed according to the reverse algorithm in Alg.1. And the low-rank matrix graph is a discernible watermark for developer and the trusted third party.

4.5 Watermark recovery

A repackaged malware could contains sensitive operations nearby the original invoke of sensitive methods. Therefore, to locate and identify the malicious modification of the repackaged App, and realize the danger perception. matrix recovery can be used in the danger warning.

When the satisfied matrix A is damaged, it can be restored and extract the noise graph. On the one hand, the reduction of the image can play a more convincing role in the verification of watermark. On the other hand, the noise of the graph can locate the modification of the sensitive block. Combined with $Trigger[m]$, we can locate and analysis the malicious modification in the repackaged App without the source code.

When the accuracy of the matrix recovery is beyond the acceptable range, we can start with the compare between the EmW_{Gph} and ExW_{Gph} for W_{Nse}.

If the future accuracy of matrix recovery can reach the available threshold, it is possible to extract W_{Nse} without original matrix.

5 Evaluation

The simulation of our proposed algorithm has been implemented in the framework $LMDGW - SYSTEM$. The system consists of watermark encoding, embedding and extracting watermark, and attacks simulation. In this paper, we apply the described methods in front to evaluate the performances of new encoding method of DGW schemes. The benchmark applications are the four methods mentioned in Sect. 3. The experimental analysis shows the efficiency of the methodology in terms of the data rate, robustness and ability of malicious code locating.

The system for experiments is based on programming language C, DalvikVM command, python and matlab, and tested on processor of 2.2 GHz Intel Core i7 with 16 GB memory.

5.1 Data rate

Table 2 shows the encoding range and data rate of current topologies. And we draw a graph indicates the relationship between data rate and the number of leaf nodes according to Table 2 in Fig. 7. According to the Fig. 7, the data rate of our scheme shows great performance as Radix-k, and even shows slightly better effect when the number of leaf is more than "7". And when the number of leaf is less than "4", LMDGW shows worse performance than Radix-k, but is still far better than other encoding algorithms as long as the nodes is more than "1". Therefore, from Table 2 and Fig. 7, we can conclude that our algorithm makes effective improvements on encoding range and data rate, and despite the form and content of a node, the topology of our scheme is as same as the Radix-k scheme.

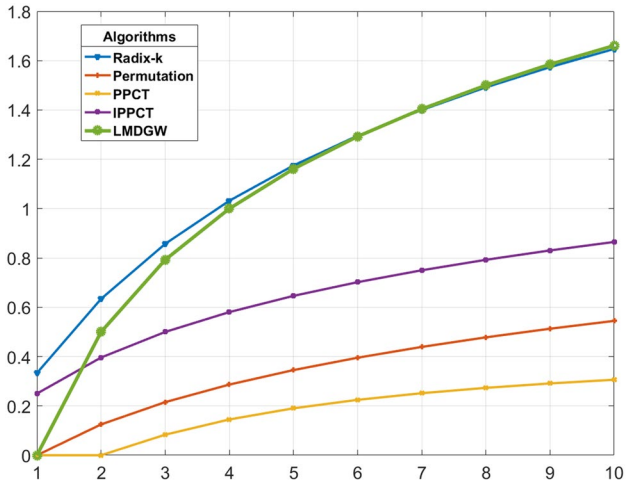

Fig. 7 Data rate comparison

5.2 Robustness

To verify the robustness of LMDGW, we make experiments in a few basic realistic scenarios requirements on the four former DGW schemes and LMDGW, and add a index of small-scale corruption of DGW. The performance of the scheme, which is shown in the Table 3, indicates that our scheme is able to handle basic scenarios well along with other four methods. However, only LMDGW is able to recover from controllable corruption of the DGW because of the research achievement in low-rank matrix recovery. Therefore, LMDGW has better robustness than other four DGW encoding schemes.

5.3 Malicious change locating

We firstly construct a graph EmW_{Gph} as is shown in Fig. 8. The graph is a watermark of our test experiment. And we embed the graph into a test APK_O. We firstly decompile the APK_O info smali files, and transfer the graph EmW_{Gph} into the matrix $LowR_M$. Then we encoding the matrix into M_{Gph} and scan smali files for the satisfactory location for embedding. Then embed the graph into smali files in the algorithms in Sect. 4, and recompile the files into APK_O.

Table 2 Data Rate

Algorithm	Nodes	Data Rate
Radix-k	n	$\frac{log_2(n+1)^n}{2n+1} \approx \frac{log_2 n}{2}$
Permutation	n	$\frac{log_2 n!}{4n}$
PPCT	n	$\frac{log_2 C_{2n-2}^{n-1}/n}{4n}$
IPPCT	n	$\frac{log_2(n+1)^n}{4n} \approx \frac{log_2 n}{4}$
LMDGW	n	$\frac{log_2 n^n}{2n} = \frac{log_2 n}{2}$

Table 3 Robustness of LMDGW

DGWs	Attacks				
	Confusion	Decompilation	Recompilation	Compress	Small-Scale Corruption of DGW
Radix-d	√	√	√	√	×
Permutation	√	√	√	√	×
PPCT	√	√	√	√	×
IPPCT	√	√	√	√	×
LMDGW	√	√	√	√	√

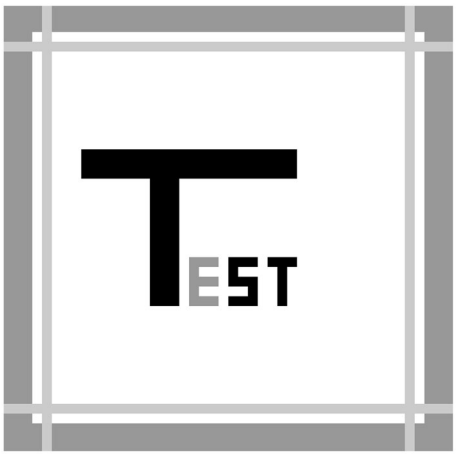

Fig. 8 Fig 2: Encoding methods for $LowR_M$

When the APK_W is ready, we pretend as attackers, and add four malicious codes into the APK_W, three of which are added under the existing trigger points, while the other one is in the independent path.

After added the malicious code, we extract the watermark from the repackaged APK_W, and the watermark is shown in Fig. 9. From the noise we can conclude that our scheme shown great performance in detecting the malicious change under the path of existing trigger points. However, the watermark is not able to notice the change under the independent path or path without sensitive actions in the APK_O. On the other hand, such changes show less aggressive actions and changes under the independent path is easy to be noticed during the use. Therefore, our proposed scheme shows value in malicious changes locating.

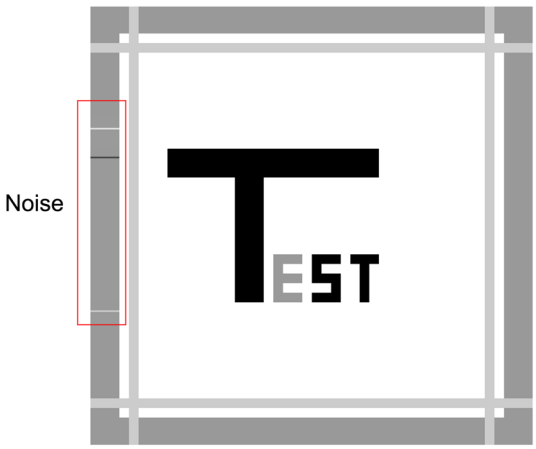

Fig. 9 Fig 2: Encoding methods for $LowR_M$

6 Conclusion

In this paper, we proposed a late-model dynamic graph watermark based on matrix, called LMDGW. LMDGW can be used to protect the copyright of android applications, and locate malicious changes at critical area in repackaged Apps. This method encodes matrix with low rank into watermark graph, and embeds the watermark graph into smali files as dynamic watermark according to the basic dynamic graph watermark embedding method. The matrix watermark make it possible for matrix recovery, which insure the integrality of the watermark graph, and achieves to overcome the unintuitive and vulnerable property of traditional watermarks like PPCT and Radix-d. Moreover, the noise of the watermark extracted from repackaged App has ability to analyse the malicious changes in the sensitive sections, which is a extended ability of LMDGW. The data rate and robust of the LMDGW shows great performance, and a few damages of watermark can be recovery by the theory of matrix recovery.

Acknowledgements This work is financially supported by Open Funding of Guizhou Provincial Key Laboratory of Public Big Data, No. 2017BDKFJJ006

References

Chasaki D, Mansour C (2015) Security challenges in the internet of things. In: Journal International (ed) of. Services and Data Management, Space Based and Situated Computing, Special Issue on Future Internet

Chen M, Lin Z, Ma Y, Wu L (2010). The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. Eprint Arxiv, 9

Chen Y, Jalali A, Sanghavi S, Caramanis C (2013) Low-rank matrix recovery from errors and erasures. IEEE Trans Inf Theory 59(7):4324–4337

Cheng C, Zeng R (2016). An improved watermarking scheme based on ppct coding. Comput Knowl Technol

Cheng J, Song Y (2012). Dynamic map based on ppct structure software watermark protection. In: World automation congress, pp 133–136

Collberg C, Myles G, Huntwork A (2003) Sandmark-a tool for software protection research. Secur Priv IEEE 1(4):40–49

Collberg C, Thomborson C (1999). Software watermarking: models and dynamic embeddings. In: ACM Sigplan-Sigact Symposium on Principles of Programming Languages, pp 311–324

Collberg CS, Thomborson C (2000) Watermarking, tamper-proofing, and obfuscation—tools for software protection. Softw Eng IEEE Trans 28(8):735–746

Collberg CS, Thomborson C, Townsend GM (2007) Dynamic graph-based software fingerprinting. ACM Trans Program Lang Syst 29(6):35

Hamilton J, Danicic S (2011). A survey of static software watermarking. In: Internet security, pp 100–107

In: Proc of CIHW (2013) A software watermark database scheme based on ppct. 2004:1–12

Khalid SKA, Deris MM, Mohamad KM (2013) Anti-cropping digital image watermarking using sudoku. Int J Grid Util Comput 4(2/3):169–177

Kuzuno H, Magata K (2016) Detecting and characterising of mobile advertisement network traffic using graph modelling: Comput Int J Space Based Situat 6(2):90

Kuzuno H, Tonami S (2013). Signature generation for sensitive information leakage in android applications. In: IEEE international conference on data engineering workshops, pp 112–119

Liu Y, Liu C, Zou H (2016) A new structure tensor based image inpainting algorithm. Int J Grid Util Comput 7(4):294–303

Luo YX, Cheng JH, Fang DY (2008). Dynamic graph watermark algorithm based on the threshold scheme. In: International symposium on information science and engineering, pp 689–693

Palsberg J, Krishnaswamy S, Kwon M, Ma D, Shao Q, Zhang Y (2000). Experience with software watermarking. In: Computer security applications, 2000. ACSAC '00. Conference, pp 308–316

Ren W, Huang S, Ren Y, Raymond KK (2016a) Lipisc: a lightweight and flexible method for privacy-aware intersection set computation. Plos One 11(6):e0157752

Ren W, Liu R, Lei M, Choo KKR (2016b) Segoac: a tree-based model for self-defined, proxy-enabled and group-oriented access control in mobile cloud computing. Comput Stand Interfaces 54:29–35

Steinbauer M, Kotsis GA (2016) Dynamograph: extending the pregel paradigm for large-scale temporal graph processing. Int J Grid Util Comput 7(2):141

Wang Y (2012). Improved ppct hybrid coding scheme. Computer Engineering & Applications

Xiong L, Xu Z, Shi Y. Q (2017). An integer wavelet transform based scheme for reversible data hiding in encrypted images. Multidimens Syst Signal Process, pp 1–12

Zhang H, Chen D (2012). An improved dynamic graph watermark algorithm. In: Multimedia information networking and security, pp 568–571

Zhang Y, Chen K (2014). Appmark: A picture-based watermark for android apps. In: Eighth international conference on software security and reliability, pp 58–67

Zhilyakova LY (2015) Dynamic graph models and their properties. Autom Remote Control 76(8):1417–1435

Zhou W, Zhang X, Jiang X (2013). Appink:watermarking android apps for repackaging deterrence. pages 1–12

Zhou Z, Wang Y, Wu QMJ, Yang CN, Sun X (2016) Effective and efficient global context verification for image copy detection. IEEE Trans Inf Forensics Secur 12(1):48–63