H Function based Tamper-proofing Software Watermarking Scheme

Jianqi Zhu[1,2], Yanheng Liu[1,2], Aimin Wang[1,2*]
1. College of Computer Science and technology
2. Key Laboratory of Computation and Knowledge Engineering, Ministry of Education
Jilin University, Changchun, China
Emails: {zhujq, liuyh, wangam @jlu.edu.cn}

Kexin Yin[3]
3. College of computer and science and engineering
ChangChun University of Technology, Changchun, China
Email: yinkexin@126.com

***Abstract*—A novel constant tamper-proofing software watermark technique based on *H* encryption function is presented. First we split the watermark into smaller pieces before encoding them using CLOC scheme. With the watermark pieces, a many-to-one function (*H* function) as the decoding function is constructed in order to avoid the pattern-matching or reverse engineering attack. The results of the function are encoded into constants as the parameters of opaque predicates or appended to the condition branches of the program to make the pieces relevant. The feature of interaction among the pieces improves the tamper-proofing ability because there being one piece destroyed, the program will not work correctly. The simulation shows that the performance of the proposed scheme is good and can resist many kinds of attacks.**

***Index Terms*— constant tamper-proofing, CLOC encoding, opaque predicate, *H* function**

I. INTRODUCTION

Since the unauthorized use and modification of software are pervasive around the world, software piracy becomes an important issue [1]. Recent studies show that 35% of the software programs used today are pirated [2]. Software watermarking is an efficient technology for software protection [3] by inserting secret messages into the programs. In addition to applying watermarking techniques to protect the copyrights of software codes, combined with other techniques, software watermarking can also be used in database protection and information security problems.

Watermarks can be classified into two categories: static watermarks and dynamic watermarks [4]. A static watermark is stored inside program code in a certain format, and it does not change during the program execution. Static watermarking techniques are more fragile as they can be easily attacked by code optimizers or obfuscators. A dynamic watermark is inserted in the execution state of a software object. More precisely, in dynamic software watermarking, what has been embedded is not the watermark itself but some codes which cause the watermark to be expressed, or extracted, when the software is run. One of the most effective watermarking techniques proposed to date is the dynamic graph watermarking (DGW) scheme of Collberg et al [5] [6] [7]. The algorithm starts by mapping the watermark to a special data structure called Planted Plane Cubic Tree (PPCT), and when the program is executed the PPCT will be constructed.

The biggest advantage of DGW over static watermarking is that a dynamic watermark graph structure contains many pointers, and it is hard to analyze pointers at runtime. Also, because a DGW watermark is constructed dynamically, runtime information must be gathered to analyze the watermark structure. Hackers need more effort to analyze stack and heap dumps than to analyze plain language code. All of these features ensure that a DGW watermark gains a certain degree of protection simply by its method of construction. But, there is a weak point in DGW algorithm, the functionality of the candidate program (a software program that needs to be watermarked) does not depend on the watermark code. Moreover, so far, the technology of protecting DGW watermark in software has not received much academic attention.

Tamper-proofing technique has been suggested to protect DGW watermark. It can detect if a program has been altered, and if so, then the program will fail to function properly. In this paper, we present a new tamper-proofing technique based on PPCT and constant encoding by creating dependencies from a candidate program to constant. This algorithm can be an additional step in DGW watermarking system to protect watermarks against malicious attacks. The proposed scheme uses the watermark pieces to construct a many-to-one function and insert the results of the function into constants which would be distributed into the program in the form of parameters of opaque predicate. To a certain extent, this scheme resolves the problem that the decoding function is too simple that is easily be analyzed and attacked

This work is supported by the National Natural Science Foundation of China (No. 60973136) and Projects of International Cooperation and Exchanges of Ministry of Science and Technology of China (No. 2008DFA12140).

doi:10.4304/jsw.6.1.148-155

maliciously. The proposed technique has the following desirable features.

1. We split the watermark number into small pieces to ensure the stealth of DGW watermark.
2. Construct a many-to-one function with the watermark pieces, which increases the tamper-proofing of watermark.
3. *H* function realizes many-to-one mapping that effectively resists the pattern matching attack.

This paper is organized as follows. In section II, related works are explained. Section III describes the principle of DGW watermark. Section VI discusses our design considerations. Section V evaluates the proposed tamper-proofing technique with respect to resilience against attacks. Section VI concludes and discusses future work.

II. RELATED WORK

Tamper-proofing technique is widely used in data integrity and data confidentiality. Unfortunately, most of the work in this field are trade secrets and are not published. In 1999 Collberg and Thomborson [8] developed a watermark tamper-proofing technique using the Java reflection mechanism by checking the type consistency of the graphic nodes at runtime. This method verifies the intactness of the Java classes representing the watermark at run time. The authors pointed out that, this solution has a fairly obvious disadvantage in its lack of stealth.

Palsberg introduced an approach to protect dynamic watermarks, by using opaque predicates to guard the watermark representation [9]. Ideally, if the watermark representations are altered, the opaque predicates will switch the program flow into an error-making branch.

In 2002, Yong He [10] proposed a constant coding scheme to strengthen the resistance of the CT watermark. He presented a prototype design of the algorithm and successfully developed a codec which converts integers into PPCT structures. However, the decoding function is too simple to resist the pattern-matching attack. Moreover, the algorithm does not create dependencies between the watermark and the constants. Once the attacker figures out the location of the watermark building code, this tamper-proofed application will be no saver than untamper-proofed application.

III. DGW WATERMARKING

The CT algorithm is the most representative in DGW proposed by Collberg and Thomborson. The idea is to embed the watermark into the topology of graph structure dynamically that is constructed at runtime. The process [11] is as follows:

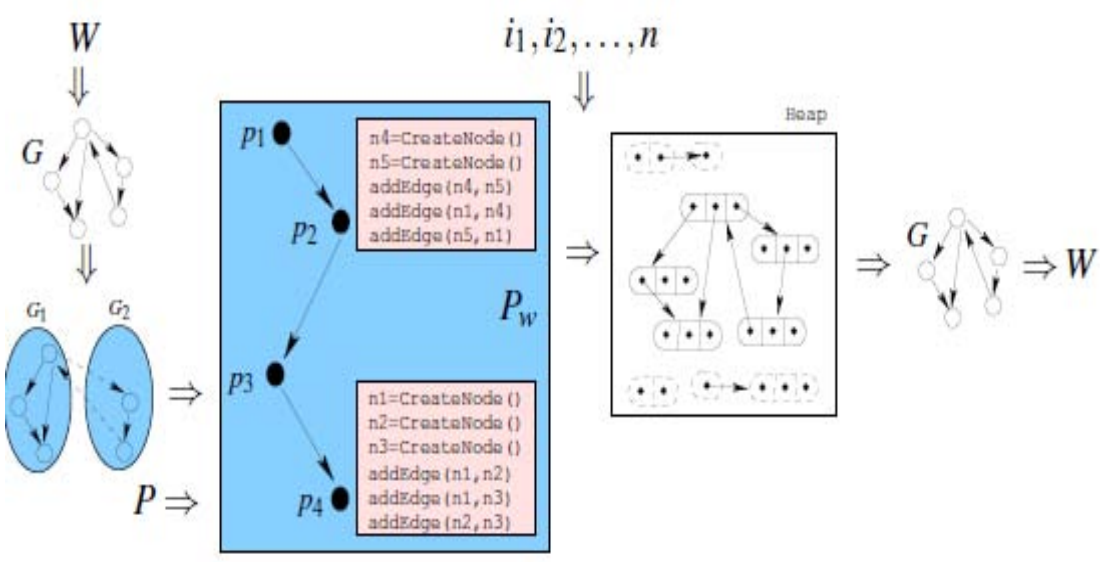


Figure1. Overview of the CT algorithm

A. Embedding:

Step1:*W* is embedded in the topology of graph *G*, *G* can be RPG（Reducible Permutation Graph）, Radix-*k* encoding linked list, parent-pointer tree, PPCT（Planted Plane Cubic Tree）or IPPCT（Intensify Planted Plane Cubic Tree）, etc;

Step2: Graph *G* is split into several components G_1, G_2,...;

Step3: Each G_i is converted into Java bytecode and embedded into the candidate program along the execution path.

B. Extracting:

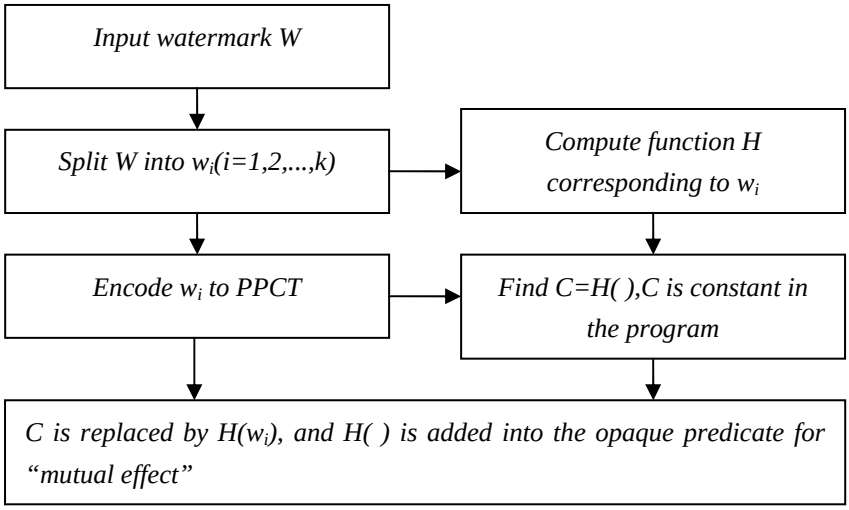
During extraction the candidate program is run with the same key *K* as input, the watermark graph gets built on the heap, the graph is extracted and the watermark number is recovered.

By now, there are three main encoding schemes in DG W: CLOC (Catalan Leaf-Oriented Conversion), BOC (Bit

-Oriented Conversion), CUIC (Catalan Unique Indexed C onversion）. In this paper, we use CLOC based on PPCT. It was proposed by Palsberg, and then improved by Yong He.

IV. DESIGN CONSIDERATIONS

As we discussed, a dynamic graph watermark is built at runtime in the program-controlled memory, and it is difficult to analyze or attack such watermarks. However, the DGW watermark does not relate closely to the functions of its candidate program. Thus, the DGW watermark still can be attacked by intensive analysis and modification to the watermarked program. In this section, we discuss a new tamper-proofing technique of DGW watermark. The process is described in Fig.2. PPCT has strong resistance against malicious attack, but its encoding range is small. If the watermark number is too big, the graph structure is also big and will be located easily. Different from the strategy of splitting PPCT into sub-trees [12], instead, we split the watermark number W into pieces $w_i(i=1,2,...,k)$ before encoding it. w_i is so smaller that the number of node in PPCT structure is smaller, then the watermark has good stealth.



Figture2. Tamper-proofing process

A. Watermark splitting

A watermark may take different forms, e.g., number, string or graph. Without loss of generality, we assume that a watermark (denoted as W) is a numerical value that the software owner selects. The aim of splitting the watermark is to ensure that the size of watermark piece is moderate, so it has good stealth and high efficiency. Attackers cannot get W from watermark pieces when they don't know the key. Secondly, it is difficult for attackers to resume W from these pieces so that it improves the security. The algorithm is as follows [13]:

(1) Compute the minimum exponent l so that W can be represented using k-1 digits of base 2^l;

(2) Split W into $w_1, w_2, \ldots, w_{k-1}$ pieces so that $W = \sum_{j=0}^{k-2} 2^{jl} w_j, 0 \le w_j < 2^l$;

(3) We get the following set, $S\{s_0, s_1, ..., s_{k-1}\}, s_0 = l-1, s_i = s_{i-1} + w_{i-1}$, where k, l works as part of secret keys to extract the watermark.

B. H function based constant tamper-proofing technique

Now, the most mature tamper-proofing method is to pick up some appropriate variables from the candidate program by decoding the PPCT instead of directly from the constant pool [14]. In this way, we can associate watermarks with constants, and if watermarks are attacked, the program cannot execute correctly. In order to avoid the pattern-matching attack, the decoding function should be many-to-one to prevent any reverse engineering attack. The many-to-one property is realized in our scheme.based on an encrypting algorithm [15] as following, where H is an encrypting function, $M_i(i=1,2,...,n)$ is plaintexts and $K_i(i=1,2,...,n)$ is secret key. It satisties:

(1) $H(M_i, K_i)(i=1,2,...n)$ are equal to each other;

(2) Given some (less than n) M_i and K_i, the plaintext information that is corresponding to the unknown secret keys cannot be calculated.

Proof: Choose an encryption function E (D is the decryption function), a random integer S and random values $r_i(i=1,2,...,n)$, it satisfies the following conditions:

E is the output function with fixed length, and the length is $|S|$.

$$| r_i | = | S | (i = 1, 2, ..., n)$$

Providing that

$$T=\bigoplus_{i=1}^{n}E(M_i,r_i)\oplus S,\ u_i=T\oplus E(M_i,r_i)$$

$$F(x,y,z)=E(x,z)\oplus y$$

$$G(x,y,z)=D(x\oplus y,z)$$

Apparently, for every $i(i=1,2,...,n)$, it has the following results:

$$\begin{aligned}F(M_i,u_i,r_i)&=E(M_i,r_i)\oplus u_i\\&=E(M_i,r_i)\oplus T\oplus E(M_i,r_i)\\&=T\end{aligned}$$

$$\begin{aligned}O(T,u_i,r_i)&=D(T\oplus u_i,r_i)\\&=D(T\oplus T\oplus E(M_i,r_i))\\&=D(E(M_i,r_i),r_i)\\&=M_i\end{aligned}$$

Apparently *F* and *G* form an encryption/decryption function pair. Where, *x* is the plaintext, (*y*, *z*) is the secret key. Function *F* satisfies the demands expressed above. For secret key (u_i,r_i) and plaintexts $M_i(i=1,2,...,n)$, we can get the same value using *F* to encrypt.

According to the calculation of *O*, we know that the security of *F* bases on the security of function *E* only if *E* satisfies condition (2), and *O* can satisfy condition (2). Q.E.D.

Choose $r_i=a(i=1,2,...,n)$ and calculate each secret key u_i for each plaintext, $H(M_i,a)=F(M_i,u_i,a)$, each $H(M_i,a)$ are equal. Select $H(M,a)$ to be a many-to-one function for tamper-proofing, *a* is a random constant.

The piece $w_i(i=1,2,...,k)$ is the parameter M_i of function *H*, we calculate $H(M,a)$ as the encoding constant. At the program is running, calculate $H(\text{int}(T_i),a)(1\le i\le k)$ as the constant decoding function. In this case, T_i is a random watermark structure tree, the form of $H(\text{int}(),)$ avoids the problem of easy exposure of int() and realizes the many-to-one property. Moreover, based on (2), attackers cannot calculate the left watermark information even they have found *k*-1 watermark pieces. The embedding process is as Fig. 3.

```
const int c = con;
public class A{
  PPCT Ti,Tj;
  public void print(){
  Ti = build_PPCT_watermark(wi);
  Tj = build_PPCT_watermark(wj);
  int a = c*2;
  system.out.println(c);
  }
}
```

(a) tamper-proofing code is not added

```
const int c = con;
pulbic class A{
  PPCT Ti,Tj;
  public void print(){
  Ti = build_PPCT_watermark(wi);
  Tj = build_PPCT_watermark(wj);
  int a = H(int(Ti),key)*2;
  system.out.println(H(int(Tj),key));
  }
}
```

(b) tamper-proofing code is added

Figure 3. Example of proposed temper-proofing technique

In this case, w_i , w_j are the watermark pieces, T_i , T_j are the root nodes of PPCT. In Fig. 3(b), constant *c* is replaced by the watermark decoding function with two different parameters.

The watermark pieces are added into the parameters of opaque predicate in the form of $(H(\text{int}(T_i))-H(\text{int}(T_j)))$, or appended to the branches of the program. Once there being one T_i attacked, the parameters of opaque predicate will change so that the program will not execute correctly as shown in Fig. 4.

```
class C{
  void m1(int a,int b){
     …
     if(a<=b)
     {…}
     else
     {…}
     …
  }
}
```

(a) no opaque predicate

```
class C{
  void m1(int a,int b){
     …
     if(a<=b)||H(int(Ti)-H(int(Tj)))
     {…  //1}
     else
     {…  //2}
     …
  }
}
```

(b) add opaque predicate

Figure 4. Watermark interaction

Once T_i or T_j is attacked and cause the watermark not to be recognized, codes 1 does not execute but code 2 does, in this way, the program does not produce the "dead codes", instead, it will enter into an incorrect execution state, and produces unexpected result. We call this effect as the "watermark pieces mutual effect". Once the watermark is attacked, we can felt it under the circumstance that the number of opaque predicates increases to c_k^2.

Refer to literature [16] that each watermark piece can interact with the two pieces before and behind it. So, the number of opaque predicate can be reduced to *k*, which can reduce the impact on the program performance. The interaction feature among pieces constructs a "cycle" that can be used to check and recover the tampered watermark. Obviously, this feature improves the tamper-proofing of watermark.

V. SECURITY ANALYSES

The watermark scheme in this paper is realized with Java language JDK1.6.0 in WindowsXP system and it is assessed in SandMark [17] [18] that is a standard watermark platform. The Benchmark applications in this experiment are TTT, calculator and mine-sweeping as shown in Table I.

TALBE I.

BENCHMARK APPLICATIONS

Program	The number of class	The number of method	size
TTT	12	51	11
Calculator	2	6	4
Mine-sweeping	8	38	42

A. Capacity of Information Hiding

The PPCT structure in this paper has a good performance of anti-attacks, but it is lower ability in coding capacity. In order to improve its stealth, the idea of watermark pieces is used. However, each piece resumes to the original watermark through exponential linear cumulation, so this scheme still gets a good amount of information hiding capacity. Table II shows the least number of leaf nodes *N* needed in coding watermark *W* with different piece number *k*.

TABLE II.

CODING ABILITY

Piece number *K*	Watermark *W*	Leaf nodes *N*
5	100	5
5	1000	6
10	100	4
10	1000	5

Table III lists the coding capacity contrast among a variety of commonly used software watermarking graph structures and IPPCT [19]. It shows that IPPCT with a higher data rate can be used to improve the capacity of information hiding. Table III lists encoding capability contrast conditions of some common used software watermark graph structure as well as IPPCT and PIPPCT (Planar IPPCT). PIPPCT is a dynamic graph structure proposed by the writer of this article which has relatively good data rate. Its structure is as Fig. 5(c), and Fig. 6 shows its node structure.

B. Stealth

Statistic analysis has been carried out on the byte code distribution of function $H(\text{int}(T_i), a)$ in system zl. Table IV shows the static code statistics: the number of instructions and types are 958 and 65 respectively, in which the proportion of instruction types more than 1% is 20, between 0.5% and 1% is 20 and others are have the proportion below 0.5%. Table V is the statistic window 2: the number of instructions and types are 933 and 269 respectively, in which the proportions of instruction types more than 1% is14, between 0.5% and 1% is 39 and others are have the proportion below 0.5%. In the statistic window 3: the number of instructions and types are 910 and 458 respectively, in which the proportion of instruction types more than 1% is 5, between 0.5% and 1% is 33 and others are have the proportion below 0.5%. In the statistic window 4: the number of instructions and types are 889 and 599 respectively, in which the proportion of instruction types more than 1% is 3, between 0.5% and 1% is 16 and others are have the proportion below 0.5%. In the statistic window 5: the number of instructions and types are 866 and 669 respectively, in which the proportion of instruction types more than 1% is 0, between 0.5% and 1% is 10 and others are have the proportion below 0.5%.

TABLE III.

COMPARISON OF THE ENCODING ABILITIES

Watermark *W*	Radix-*K*	PPCT	Permutation	RPG	IPPCT
897	6	18	8	12	10
9331	7	20	8	13	12
16631	7	20	9	15	12
169037	8	26	10	19	14
3524768	9	32	11	21	16

TABLE IV.

INSTRUCTION FREQUENCIES

Byte code	Proportion	Byte code	Proportion	Byte code	Proportion	Byte code	Proportion
aload	23.07%	return	1.15%	iconst_1	0.63%	ifge	0.1%
invokevirtual	7.1%	getstatic	1.04%	aload_3	0.63%	astore_1	0.1%
getfield	6.37%	athrow	1.04%	iload_1	0.63%	iload_3	0.1%
astore	6.05%	iload_2	0.94%	iastore	0.63%	imul	0.1%
invokeinterface	4.91%	if_icmpne	0.94%	ixor	0.52%	istore_3	0.1%
invokespecial	4.28%	Areturn	0.94%	ireturn	0.52%	if_acmpeq	0.1%
dup	3.76%	iinc	0.94%	ifeq	0.42%	f2i	0.1%
iconst	3.34%	ifne	0.94%	istore_2	0.42%	bastore	0.1%
new	3.24%	checkcast	0.84%	arraylength	0.42%	ifnull	0.1%
goto	3.03%	istore	0.73%	ldc	0.42%	fmul	0.1%
invokestatic	2.4%	iconst_0	0.73%	isub	0.42%	if_acmpne	0.1%
aload_0	2.4%	iaload	0.73%	iconst_2	0.31%	iconst_3	0.1%
putfield	1.67%	if_icmplt	0.73%	iconst_4	0.31%	iconst_5	0.1%
iload	1.57%	bipush	0.73%	newarray	0.31%	Putstatic	0.1%
pop	1.25%	aload_1	0.73%	istore_1	0.31%		
ifnonnull	1.15%	aconst	0.73%	astore_3	0.31%		
ldc_w	1.15%	if_icmpeq	0.73%	iadd	0.31%		

TABLE V.

MOST COMMON 2-GRAMS

Byte code	Proportion	Byte code	Proportion	Byte code	proportion
aload,aload	6.22%	……	…	invokespecial,new	0.11%
aload,getfield	4.72%	astore,getstatic	0.21%	bipush,istore_2	0.11%
aload,invokevirtual	4.07%	ifne,new	0.21%	iinc,iload_2	0.11%
astore,aload	3.43%	iconst_1,goto	0.21%	aload_0,iload_1	0.11%
new,dup	3.32%	aload,iconst	0.21%	dup,aload_1	0.11%
aload,invokeinterface	3.22%	invokespecial,aload_0	0.21%	if_acmpeq,aload	0.11%
goto,aload	2.57%	aload,aconst	0.21%	iload,iaload	0.11%
aload,invokestatic	1.39%	ifnonnull,pop	0.21%	ifeq,aload	0.11%
invokeinterface,astore	1.39%	if_icmpeq,aload	0.21%	ifnull,aload	0.11%
invokespecial,astore	1.39%	iconst_1,invokevirtual	0.21%	pop,iinc	0.11%
astore,goto	1.29%	getstatic,new	0.21%	iload_2,iload_1	0.11%
aload_0,getfield	1.29%	aload,ifnonnull	0.21%	iinc,iload_1	0.11%
invokespecial,athrow	1.07%	iconst,aload	0.21%	getfield,if_acmpeq	0.11%
invokevirtual,iconst	1.07%	……	…	aload_3,bipush	0.11%

Collberg had done statistic analysis on the 1132 static java byte code programs, and pointed out that the byte codes such as aload 0, invokevirtual, getfield, dup and invokespecial should have higher appearance rate, and others are below 1%, especially jsr w and goto w that are almost rarely been used. Based on the results in literature [20], the byte codes of our algorithm are frequently used, and most frequency distributions conform to the characteristics of the general applications, so it has good stealth.

C. Resistance

Table VI and VII are the experiment results of optimizer and obfuscator imposed on system zl that are provided by Sandmark. It shows that the algorithm in this paper can effectively fight against semantics-preserving transformation on Sandmark platform, and also can be immune of all obfuscators except split classes. And when it is attacked by split classes, the program will go into an incorrect execution condition.

TABLE VI.

THE EFFECTS OF OPTIMIZERS ON ZL

Optimizer	TTT	Calculator	Mine-sweeping
BLOAT	+	+	+
Dynamic Inliner	+	+	+
Inliner	+	+	+
Variable Reassigner	+	+	+

Subtractive attacks: if the watermark is located, attackers can try to remove it without changing the semantics. Essentially, the scheme in this paper is one kind of dynamic data structure watermark, and can completely resist this attack theoretically. The watermark is embedded in a tree structure that is generated dynamically and spread in the entire program, so attackers cannot locate the watermark.

Pattern-matching attacks: when attackers cannot know the program's behaviors, they will use debugger or other tools to trace every function's return values in the program, and replace the function whose return value is constant with the corresponding constant. The scheme cannot completely resist the pattern-matching attack, but the watermark is made up of pieces, so this attack needs high expense. And even finding all the pieces, the attacker cannot recover the original watermark without knowing the secret keys.

Distortive attacks: The attacker may find some pieces, and then distort them. However, the distorted watermark can hardly maintain the program semantics to a large extent, and the"mutual effect" cycle can locate the distorted watermarks and recover them in a certain extent.

TABLE VII.

THE EFFECTS OF OBFUSCATIONS ON ZL

Obfuscator	TTT	Calculator	Mine-sweeping
Array Folder	+	+	+
Array Splitter	+	+	+
BLOAT	+	+	+
Block Marker	+	+	+
Bludgeon Signatures	+	+	+
Boolean Splitter	+	+	+
Branch Inverter	+	+	+
Buggy Code	+	+	+
Class Encrypter	+	+	+
Class Splitter	+	+	+
ConstantPoolReorderer	+	+	+
Duplicate Registers	+	+	+
Dynamic Inliner	+	+	+
False Refactor	+	+	+
Field Assignment	+	+	+
Inliner	+	+	+
InsertOpaquePredicates	+	+	+
Integer Array Splitter	+	+	+
Interleave Methods	+	+	+
Irreducibility	+	+	+
Merge Local Integer	+	+	+
Method Merger	+	+	+
Objectify	+	+	+
Opaque Branch Insertion	+	+	+
Overlode Names	+	+	+
ParamAlias	+	+	+
PromotePrimitive Registers	+	+	+
Promote Primitive Types	+	+	+
Publicize Fields	+	+	+
Random Dead Code	+	+	+
Rename Registers	+	+	+
Reorder instructions	+	+	+
Reorder Parameters	+	+	+
Simple Opaque Predicates	--	--	--
Split Classes	+	+	+
Static Method Bodies	+	+	+
String Encoder	+	+	+
TransparentBranchInsertion	+	+	+
VariableReassigner	+	+	+

VI. CONCLUSION AND FUTURE WORK

Software watermark is an efficient technology for copyright protection, but the theory has yet to be mature. Because of the determinacy of software's behaviors itself, the software is easily attacked, and how to improve the robustness of software is one of key points, moreover, how to improve the ability of watermark's tamper-proofing is always a popular research problem. The constant tamper-proofing technology proposed in this paper based on *H* function, the watermark pieces are constructed into the constants and embed to the program's branches and conditions to form the "mutual effect" cycle to protect the program. In expanding the capability of encoding, this scheme also efficiently improves the watermark's robustness and tamper-proofing. Experiment analysis shows that this novel scheme has a strong ability to protect the program and has good performance.

Currently, some part of system zl still needs to be manual processes. How to realize the totally automatic model and to do accurate measurement evaluations on more samples, and even how to balance the relationships among stealth, data rate, and robustness are our future works.

ACKNOWLEDGMENT

This work is supported by both the National Nature Science Foundation of China (No.60573128) and Projects of International Cooperation and Exchanges of Ministry of Science and Technology of China (No. 2008DFA12140).

REFERENCES

[1] Zhu Jian-qi, Liu Yan-heng, Yin Ke-xin, "A Novel Dynamic Graph Software Watermark Scheme" [C]. *Proceedings of the ECS2009*, 7-8 March, 2009, Wuhan, China. pp. 775-780.

[2] Industrial Design and Construction (IDC) and Business Software Alliance (BSA), "Piracy study," July 2004. http://www.bsaa.com.au/downloads/ piracyStudy070704.pdf

[3] Zhang Li-he, Yang Yi-xian,et al, "A survey on software watermarking " [J], *Journal of Software*, 14(2): 268-277, 2003.

[4] Christian Collberg, Clark Thomborson, "Software watermarking: models and dynamic embeddings" [C], *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, p.311-324, January 20-22, 1999, San Antonio, Texas, United States.

[5] Zhu Jian-qi, Yin Ke-xin, Liu Yan-heng, "A Novel DGW Scheme Based on 2D_PPCT and Permutation" [C]. *The International Conference on Multimedia Information Networking and Security*, Nov 18-20, 2009, Wuhan, China. pp.109-113.

[6] William Zhu, Clark Thomborson, "Extraction in Software Watermarking" [A]. *In ACM Multimedia and Security Workshop [C]*, 26-27, September, Geneva, Switzerland, pp. 175-181, 2006.

[7] William Zhu, Clark Thomborson, "Recognition in Software Watermarking" [A]. *In 1st ACM Workshop on Multimedia , Content Protection and Security, in conjunction with ACM Multimedia 2006 [C]*, October 27th,Santa Barbara, CA, USA, pp. 29-3, 2006.

[8] Christian Collberg and Clark Thomborson, "Software watermarking:Models and Dynamic Embeddings" [A]. *In:Aiken A, et al., eds.Proceedings of the 26th Annual SIGPLAN-AIGACT Symposium on Principles of Programming Languages (POPL'99) [C]*.Association for Computing Machinery Press, pp. 311-324,1999.

[9] Jen Palsberg,Sowmya Krishnaswamy,et al, "Experience with Software Watermarking" [A]. In: Epstein J,et al., eds. *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC 2000) [C]*. New Orleans: IEEE Computer Society Press, pp. 308-316, 2000.

[10] Yong He, "Tamperproofing a Software Watermark by Encoding Constants" [D]. *Master's thesis*,Comp.Sci.Dept..Univ. of Auckland 2002.

[11] Collberg CS, Thomborson C, Townsend GM, "Dynamic graph-based software watermarking". *Technical report TP04-08*, April 28, 2004 [R].

[12] Christian Collberg, Clark Thomborson, and Gregg Townsend, "Dynamic graph-based software fingerprinting" [A]. *ACM Trans.Program.Lang.Syst [C]*. 29,6,Article 35 (October 2007),67 pages.

[13] Christian Collberg, Andrew Huntwork, et al, "Graph Theoretic Software Watermarks: Implementation, Analysis and Attacks", *Technical Report* TR04-06 2004 [R].

[14] Clark Thomborson, Jasvir Nagra, et al, "Tamper-proofing Software Watermarks" [A]. *In: Proc.Second Australasian Information Security Workshop. ed. P. Montague and C.Steketee. ACS. CRPIT 2004 [C]*.

[15] Zhu Zheng-ping, Zhong cheng, and Cheng Dong-yong, "Software Watermarking Algorithm Based on Hiding Execution Path of Watermark-functions" [J], *Application Research of Computers*, (12): 118-121, 2006.

[16] Liu Quan, Jiang Xue-mei, "Hierarchical semi-fragile digital watermarking algorithm for image tamper localization and recovery " [J]. *Journal on Communications*, 28(7): 105-110, 2007.

[17] Christian Collberg. Sandmark homepage [EB/QL]. http://www.cs.arizona.edu/sandmark.

[18] Christian Collberg, Ginger Myles, and Andrew Huntwork, "SandMark — A Tool for Software Protection Research" [J], *Journal of IEEE Magazine of Security and Privacy*, (1): 40-49, July-August 2003.

[19] Wang Yong, Yang Yi-xian, "A software watermark library scheme based on PPCT " [A]. *China information hiding workshop 2002 [C]*, The information security center (ISC), Beijing Univ. of Posts & Telecom, 2002.

[20] Christian Collberg, Ginger Myles, and Michael Stepp, "An empirical study of Java bytecode programs" [J]. *Published online 24 October 2006 in Wiley InterScience. Softw. Pract. Exper.* (37): 581-641, 2007.

ZHU Jian-qi was born in Zhenjiang, China, in 1976. He received the Ph.D. degree from Jilin University. His interests include software protection, obfuscation, software watermarking, and so on.

LIU Yan-heng was born in Jilin, China, in 1958. He received the Ph.D degree in mobile communication from Jilin University. His primary research fields includes: Network communication and protocol design, QoS mechanism in Mobile IP network, Policy-based network management and network intrusion detection system, and so on.

WANG Ai-min was born in Hebei province, China. He received the Ph.D degree in mobile communication from Jilin University. He research fields lies in network security, Ad Hoc and multimedia transmission.

YIN Ke-xin was born in Jilin, China in 1975. She received the Ph.D. degree from CUST. Her research interests lie in the information security and digital watermarking and so on.