第41卷第7期 中国科学技术大学学报 Vol. 41, No. 7
2011年7月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jul. 2011

文章编号:0253-2778(2011)07-0599-08

一种基于代码加密的防篡改软件水印方案

汤战勇,房鼎益,苏 琳

(西北大学信息科学与技术学院,西北大学-爱迪德信息安全联合实验室,陕西西安 710127)

摘要:提出一种基于代码加密的防篡改软件水印方案,并对隐藏水印信息的PPCT结构进行改进.水印嵌入过程中,利用秘密分享方案,对代表版权信息的大数进行分割,以提高水印信息的隐蔽性.通过修改软件的源代码和目标代码实现水印的嵌入过程,根据特定的策略对部分目标代码进行加密,并提高加解密密钥与程序自身的关联聚合度,从而增强软件水印的鲁棒性以及防篡改能力.

关键词:软件水印;动态图水印;广义中国剩余定理;代码加密;防篡改

中图分类号:TP311 **文献标识码**:A doi:10.3969/j.issn.0253-2778.2011.07.006

A tamper-proof software watermark using code-based encryption

TANG Zhanyong, FANG Dingyi, SU Lin

(*School of Information Science and Technology, Northwest University, and NWU-Irdeto Network-Information Security Joint Lab (NISL), Xi'an 710127, China*)

Abstract: Utilizing a modified PPCT structure, a tamper-proof software watermark solution with code-based encryption was proposed. The general Chinese remainder theorem was exploited to split the watermark which was represented as a big number into pieces to enhance stealthiness. Changes to the source and object code were made to embed the watermark, and according to certain policies some parts of the object code was encrypted with an en/decryption key that was highly coupled with the object code to increase robustness and tamper-proof capability.

Key words: software watermark; dynamic graph; general Chinese remainder theorem; code encryption; tamper-proof

0 引言

软件水印(software watermark)是数字水印技术的分支,是近年来出现的软件产品版权保护技术.软件所有者通过在软件里秘密嵌入一个代表版权的信息用来标识作者、发行商、所有者、使用者等.利用携带的版权保护信息和身份认证信息,可以鉴别出非法复制和盗用的软件产品.软件水印采取的是被动防御方式,将水印作为盗版行为发生后的一种证据,当发生版权纠纷时,所有者从嵌入水印信息的软件中提取出水印,来保护自己的版权.

软件水印在实际使用中面临两个难题:一是需要考虑在保证和提高嵌入信息的抗攻击性的同时如何提高数据率;其次是当恶意攻击发生时,如何有效地感知水印被篡改,及时触发篡改响应机制执行错误的控制流或者终止程序运行,从而提供对程序及其水印的双重保护.

PPCT(planted plane cubic tree)树是一种具有

收稿日期:2011-04-28;**修回日期**:2011-06-22

基金项目:国家自然科学基金(61070176),陕西省科技攻关项目(2011K06-07),陕西省教育厅产业化示范项目(2010JC24,2010JC25),西北大学研究生创新基金(09YZZ62,10YZZ16)资助.

作者简介:汤战勇,男,1979年生,博士. 研究方向:网络与信息安全和软件安全与保护. E-mail: zytang.nwu@gmail.com

通讯作者:房鼎益,博士/教授. E-mail: dyf@nwu.edu.cn

良好性质的拓扑结构，从形式上看，它是一个特殊的二叉树，对这种特殊的二叉树稍作改进就可生成一种能用以表示动态数据结构软件水印的PPCT水印编码结构.研究表明[1-2]，同基于K基数的循环链表、排列图等传统的动态图软件水印拓扑结构相比，PPCT树抗攻击性较好，但是数据率较低.本文通过对PPCT进行了改进得到新的MPPCT结构，使其在数据率上具有明显的优势，且能有效抵抗共谋攻击，是一种良好的水印拓扑图结构.针对软件水印被恶意篡改的难题，现有的软件水印在防篡改方面进行了部分尝试，Moskowitz等[3]提出将水印信息以及程序代码嵌入到应用程序的特殊资源中(例如图片和音乐)，程序在执行时，从资源里提取代码并执行，一旦隐藏水印信息和代码的资源被篡改，程序就会终止运行，以此达到防篡改的目的.问题是这种凭空产生和执行的代码的不寻常性会使水印失去隐蔽性，引起攻击者的注意.Collberg等[1]提出采用Java反射(Reflection)技术实现水印的防篡改，主要思想是在程序的运行过程中检查水印拓扑图节点类的字段类型以及字段的个数，当发现字段的类型或者个数与既定的不一致时，说明水印图已被篡改，但这种方案的通用性差，隐蔽性有限，而且仅用于Java程序.此外，Yong[4]提出一种基于常量编码的软件防篡改方案；Holmes[5]提出在程序中包含校验和计算的防篡改措施；文献[6-9]也研究了基于常量的水印防篡改方案.但是，这些方案都存在一定的不足，其主要缺点是解码函数的形式较单一，难以抵抗模式匹配攻击.

围绕传统软件水印中有限的防篡改能力和较低的数据率、鲁棒性等问题，本文在深入研究软件水印关键技术的基础上，给出了改进的水印拓扑图结构，同时提出了将代码加密防篡改技术与软件水印技术相结合的软件保护方案.水印嵌入过程中，使用基于广义中国剩余定理的数据分割法，对代表版权信息的大数进行分割，以提高水印信息的隐蔽性.通过修改软件的源代码和目标代码实现水印的嵌入过程，根据特定的策略对部分目标代码进行加密，并提高加解密密钥与程序自身的关联聚合度，从而增强软件水印的鲁棒性以及防篡改能力.

1 软件水印及其安全性分析

1.1 软件水印

一个完整的软件水印框架[1,12]由用户密钥、原始程序、水印信息、水印的嵌入算法以及提取算法组成，具体包括水印的嵌入过程以及提取过程.

软件水印的嵌入是根据用户密钥即程序的特定输入序列，以及嵌入算法，将水印信息编码后插入到原始程序中，形成一个包含水印信息的新程序，表示为Embed(P,W,Key)⇒Pw.

软件水印的提取是根据用户密钥即程序的特定输入序列，以及提取算法，选择是否使用原始程序，从包含水印的程序中提取水印信息，表示为Recognize(Pw,Key)⇒W，或者Recognize(Pw,Key,P)⇒W.

1.2 软件水印的评价

度量软件水印算法好坏的指标主要包括[10-11]：嵌入水印后程序的正确性、软件水印的数据率、鲁棒性、隐蔽性以及给程序带来的性能过载.

①正确性：这是软件水印的最基本要求.正确性是指嵌入水印前后的程序在行为上必须保持一致.换句话说，就是要求相同的输入产生相同的输出.

②数据率：表示嵌入在程序中的水印代码隐藏的信息量.要求软件水印应该包括足够多的信息量.

③鲁棒性：表示嵌入水印后的程序，能够抵抗攻击的能力.一般要求软件水印具有较高的鲁棒性.

④隐蔽性：表示攻击者得到一份程序后，通过各种方法仍不能确定该程序是否嵌入水印，或者无法定位水印嵌入位置的能力.

⑤性能过载：嵌入水印后的程序与嵌入水印前的程序相比，不能明显地影响程序的性能，包括对占用空间以及执行时间的影响.一般要求软件水印具有较低的性能过载.

一个好的软件水印算法要求嵌入水印后的程序具有较高的数据率、较强的鲁棒性、较高的隐蔽性和较低的性能过载.但高的数据率往往意味着低的鲁棒性以及差的隐蔽性.因此设计一个水印算法时，往往需要对数据率、鲁棒性、隐蔽性以及性能过载进行折衷考虑.

1.3 针对软件水印的攻击

要评估软件水印算法的好坏，必须了解软件水印可能遭受的攻击[10-11].

(Ⅰ)减少攻击(subtractive attack)

攻击者使用反编译、反汇编等技术，定位到软件水印的位置后，删除部分或者全部水印信息，但是程序的功能保持不变，称为减少攻击.

(Ⅱ)添加攻击(additive attack)

攻击者在包含水印信息的软件中添加自己的水印信息，如果新添加的水印信息能够完全覆盖原始的水印信息，或者不能证明原始的水印信息嵌入时间比自己早，称这种攻击为一个有效的添加攻击.

（Ⅲ）扭曲攻击(distortive attack)

当攻击者不能够定位到水印位置时，他会通过保留语义转换程序，使得程序能够正常运行，但软件的所有者无法提取水印这种攻击手段称为扭曲攻击. 当攻击者无法定位软件水印位置时，他常常使用保留语义转换的方法来破坏水印信息.

（Ⅳ）共谋攻击(collusive attack)

共谋攻击主要是针对软件指纹的. 攻击者得到软件所有者包含指纹的多个程序，通过比较程序得出软件指纹的嵌入位置，继而破坏指纹的过程称为共谋攻击.

2　基于代码加密的防篡改软件水印方案

2.1　水印数据的分割

在秘密分享方法的使用上，本文采用 GCRT 分割法对水印数据进行分割. GCRT 分割法的基本思想是建立在广义中国剩余定理(general Chinese remainder theorem)之上的，只要攻击者修改的数据不超过水印子数据的一半，还是能够恢复出原始的数据. GCRT 分割法将水印数据 W 分割成一系列的子数据，分割步骤如下[13]：

① 选取 r 个两两互质的质数，并且 $W<\prod_{k=1}^{r}P_k$.

② 将 W 分割 $r(r-1)/2$ 块，每一块用 $W\equiv X_k \bmod P_iP_j\ (0<i<j\leqslant r, 0\leqslant X_k<P_iP_j)$ 形式表示，求出 X_k.

③ 每一个 $W\equiv X_k \bmod P_iP_j$ 都可以通过某一表达式转换成一特定的整数值. 表达式如 $W_k=X_k+\sum_{n=1}^{i-1}\sum_{m=n+1}^{r}P_nP_m+\sum_{m=1}^{j-1}P_iP_m\ (n<m, i<m)$，$W_k$ 即为分割后的水印子数据. 恢复原始的水印数据时，根据水印子数据 W_k 以及相同的枚举表达式求出 X_k，然后根据 $W\equiv X_k \bmod P_iP_j$ 以及广义中国剩余定理求出 W.

2.2　改进的 PPCT 结构

PPCT 结构具有以下一些特征，使之可以成为一种具有良好鲁棒性的水印载体. 其特征描述如下[2]：

（Ⅰ）有一个根节点(Origin 节点)，该节点有两个指针，左指针指向这棵树的最右下叶节点，右指针指向二叉树的根节点.

（Ⅱ）每个节点都拥有左右两个指针，非叶节点的左右指针指向自己的左右孩子，叶节点的右指针指向自身，其左指针满足如下规则：非叶节点的右子树的左下节点指向其左子树的右下节点，整棵树的最左叶节点指向 Origin 节点.

（Ⅲ）一个具有 n 个叶节点的 PPCT 结构共有 $C(n)=\frac{1}{n}C_{2n-2}^{n-1}$ 种表示方法.

在抗攻击性方面，PPCT 结构的抗攻击性最好，但是其数据率较低. 因此，本文提出一种改进 PPCT 结构的方法，在不影响隐蔽性与抗攻击性的前提下，提高其数据率. 从图 1(a)可以看出，PPCT 结构所有叶节点的右指针都指向自身，因此，可以对叶节点的右指针进行改进，使其包含一定的信息，将改进后的 PPCT 结构称为 MPPCT 结构.

如图 1(b)，MPPCT 结构有一个 Origin 节点，即它的根节点，Origin 节点的左指针指向最右叶节点，右指针指向二叉树的根节点. 每个叶节点都有两个指针，左指针指向前一叶节点，右指针满足下述原则：对于某一整数，计算它的二进制数据中连续 1 和 0 的长度，修改叶节点的右指针，从最右叶节点开始，叶节点的右指针分别指向序号为这些长度的节点，通过这种方式来表示该二进制数据. 将 MPPCT 结构解码为十进制数据时，首先沿着根节点，找出最右叶节点，解码该叶节点表示的二进制串中 1 的个数，依次类推，直到最左叶结点. 最后将解码出的二进制串转化为十进制数据形式.

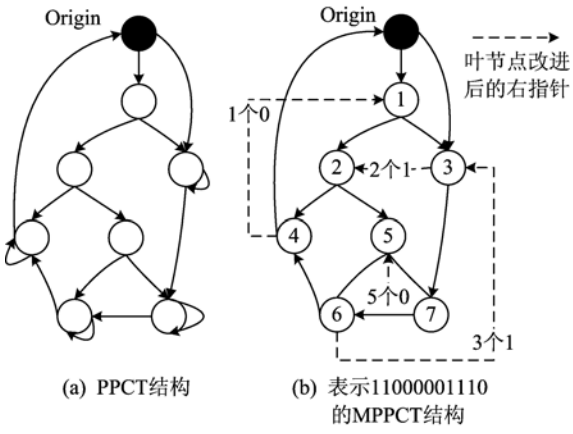

图 1　具有 4 个叶节点的 PPCT 与 MPPCT 结构

Fig. 1　PPCT and MPPCT with 4 leaf nodes

由于具有 M 个节点的 MPPCT 结构有 $M/2$ 个

叶节点，理论上，每个叶节点的右指针可指向的节点有 A_{M-1}^{1}种，因此具有 $M/2$ 个叶节点的 MPPCT 所能表示的数据个数 $C(M)=(A_{M-1}^{1})^{M/2}=(M-1)^{M/2}$.

2.3 算法基本思想

算法将代码加密防篡改技术与软件水印技术相结合，实现具有防篡改功能的软件水印系统. 主要思想是：如图 2，在程序中插入具有特定功能的水印分支函数（watermark branch function，WBF），修改 WBF 的返回地址，当在特定的输入序列下执行程序时，找出程序执行过程中能够执行到的一些无条件跳转指令，修改这些无条件跳转指令的目标指令，使它们跳转至新插入的分支函数，由分支函数通过内部计算后执行 ret 指令时，返回到真正被替代的目标指令. 为了实现防篡改功能，以及提高新插入的分支函数与程序的关联度，本文使用交错和互相保护机制来增强水印系统的安全性，选择部分被替代的目标函数进行加密，将程序中敏感性代码的校验和以及分支函数内部计算的结果共同作为加密密钥因子.

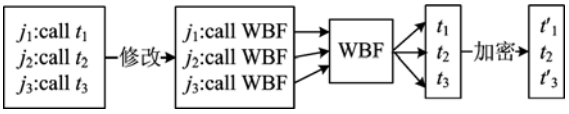

图 2 修改无条件跳转指令

Fig. 2 Change function calls

WBF 函数负责寻找无条件转移指令的目标函数，并将其转向目标函数的入口地址处执行. 具体来说，它负责如下的工作：①从水印子数据 w_i 对应的 MPPCT 结构中提取 w_i 的值；②产生下一个密钥 k_i. 根据公式 $k_i=\mathrm{SHA1}[k_{i-1}\oplus w_i]$，计算下一个密钥 k_i；③根据 k_i 以及完美 hash 函数 h，计算出 $h(k_i)$ 的值，然后根据 $h(k_i)$的值查找存放在位移表 T 中的偏移地址，即 $h:\{k_1,k_2,\cdots,k_n\}\rightarrow\{1,2,\cdots,m\}$ $(n\leqslant m)$.

本算法中的位移表 T 具有 3 列，每列存储的数据如下：①存储 WBF 函数的返回地址与被替代目标指令间的偏移地址 d_i. 例如对于将指令“s_i: call t_i”改为“s_i: call WBF”，则 WBF 原始的返回地址为 Address(s_i 下一条指令)，原始的目标指令为 t_i，偏移地址 $d_i=$ Address(t_i) − Address(s_i 下一条指令)，新的返回地址为 Address(s_i 下一条指令)$+d_i$，即 WBF 可以返回到原始的目标指令处执行. ②存储程序中待计算校验和代码段的起止地址，这两处地址为相对于 PE 文件头的偏移地址.

2.4 水印的嵌入

水印的嵌入过程分为水印数据的处理、源代码的处理、目标代码的处理以及加密器对目标代码加密几个子步骤.

2.4.1 水印数据的处理

水印数据的处理包括：①选取两个大质数 P 与 Q，计算它们的乘积 W；②使用 GCRT 数据分割法，将 W 分割成一系列的子数据 W_i；③随机生成一个整数 k_0；④将 k_0 和 W_i 编码为 MPPCT 结构，并且构造生成 MPPCT 结构的代码.

2.4.2 源代码处理

（Ⅰ）标记与追踪

目的是根据特定的执行序列，找出只会在 $I_0\,I_1\,I_2\cdots$下执行时触发的函数. 水印将会嵌入在这些函数中，同时待加密函数也将从这些函数中选取.

（Ⅱ）源程序插入

将校验和计算函数、加解密函数、完美 hash 函数和生成 MPPCT 结构的代码插入到源程序中. 校验和计算函数是为了检查程序的完整性；加解密函数负责程序在执行过程中对密文代码使用之前的解密以及使用之后的加密；完美 hash 函数的作用是将字符串 k_i 映射成唯一的整数，即 hash$\{k_1,k_2,\cdots,k_n\}\rightarrow\{1,2,\cdots,m\}$，其中 $n\leqslant m$；MPPCT 结构的代码负责程序运行过程中，生成 MPPCT 结构，水印信息隐藏在这些 MPPCT 结构中.

（Ⅲ）源程序标记

找出将要被加密的目标函数，修改这些函数. 找出待计算校验和的函数，并在函数里添加标记. 为了使外部加密器对待加密函数进行加密时，能够在目标文件中找出这些函数的位置，同时使得程序能够正确地执行这些函数，需要对这些函数进行修改，修改方式如下：①在每个函数的开始处依次插入调用解密函数操作，以及待加密段的起始标记；②在每个函数 return 语句前插入待加密段的结束标记，以及调用加密函数操作. 敏感性的代码不仅包括生成 MPPCT 结构的代码，也包括其他关键代码.

2.4.3 目标代码处理与加密

主要从以下 5 个方面修改目标代码：

（Ⅰ）构造并插入 WBF 函数

根据上述对 WBF 的功能介绍，可以通过以下几步来构造 WBF 函数：

①构造从 w_i 对应的 MPPCT 结构中提取 w_i 的方法，这个方法负责计算 w_i. WBF 根据公式 $k_i=\mathrm{SHA1}[k_{i-1}\oplus W_i]$，计算下一个密钥 k_i，而 k_0 的值比较特殊，它将从 k_0 对应的 MPPCT 结构中提取出 k_0.

②构造根据 k_i 计算出偏移地址 d_i 的方法. 为了使程序能够按照原始的逻辑执行，WBF 需要计算出原始被替代函数的入口地址，由于位移表 T 的第一列存放了 d_i，同时使用完美 hash 函数 h，使得每个 k_i 对应表中的不同行，从而取出 d_i 的过程，表示为 $d_i=T[h(k_i)][0]$，对于每一个被替代的函数，都有一个 k_i 与 d_i 的映射关系，即 $\Phi\{k_1\rightarrow d_1,\ k_2\rightarrow d_2,\ \cdots,k_n\rightarrow d_n\}$.

（Ⅱ）修改转调指令

将符合条件的“call 目标函数”指令改为“call WBF”指令. 修改追踪阶段找出的每个函数 f，将函数 f 里的“s：call 目标函数”指令改为“s：call WBF”指令. 所选的目标函数可为用户自定义的函数，也可为系统函数. 对于每个被替代的目标指令 A，计算密钥 k_i 以及 A 与 s 下一条指令间的偏移地址 d_i，并建立 k_i 与 d_i 的对应关系.

（Ⅲ）构造并插入位移表 T

构造表 T，并将其插入到 PE 文件的数据段中. 根据完美 hash 函数 h 计算出 $h(k_i)$的值，这个值即为 k_i 对应于表 T 的行号，最后根据 Φ，将偏移地址 d_i，相应的待校验代码块的起止地址填充到 T 表里的 $h(k_i)$ 行中，表示为 $T[h(k_i)][0]=d_i$，$T[h(k_i)][1]=$待计算校验和代码段的起始位置，$T[h(k_i)][2]=$待计算校验和代码段的结束位置，这两处位置可以根据处理源代码时，在待计算校验和代码段里插入的标记得到.

（Ⅳ）修改 WBF 函数的返回地址

修改 WBF 函数的返回地址，使 WBF 返回到原始的目标指令处执行. 可以通过如下指令来修改返回地址：

① pop ECX

② add ECX，d_i

③ push ECX

其中语句①表示将当前栈指针指向的返回地址出栈，并存入 ECX 寄存器里；语句②表示将 ECX 寄存器里的值增加偏移地址 d_i；语句③表示将新计算的值重新进栈，此时栈顶里的数据为新的返回地址.

（Ⅴ）外部加密器加密目标代码

加密过程处于水印嵌入的最后一步，是由外部加密器实现. 为了提高水印系统的鲁棒性，同时实现防篡改功能，算法建立敏感性代码段的校验和、水印分支函数 WBF 产生的 k_i 与加解密密钥的关联聚合度，来增强水印系统的安全性. 图 3 为代码加密示意图. 图中，符号 Check 代表计算校验和的函数；Hash 代表完美 hash 函数；Q_i 代表程序中的敏感性代码段，由校验和函数计算它们的短消息摘要 $V(Q_i)$；C_i 代表程序中的密文代码；Encrypt 代表加密算法，用来加密 C_i；Decrypt 代表解密算法，用来解密 C_i；W_i 代表嵌入在程序里的 MPPCT 结构的代码. 本文将水印数据分割成若干个子数据，每个子数据对应一个 MPPCT 结构；$V_i\oplus k_i$ 代表加密密钥因子，密钥是由 Q_i 的校验和 V_i、WBF 产生的 k_i 经过 SHA1 运算得出；WBF 代表新插入程序中的水印分支函数. 结合图 3，使用外部加密器，对目标代码的加密过程可以描述为，遍历所有待加密函数，对每个函数做如下处理：

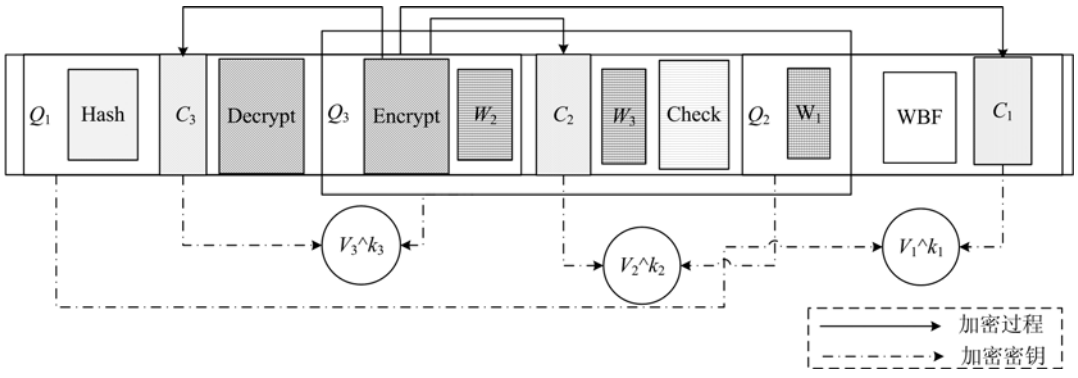

图 3　具有防篡改功能的代码加密示意图

Fig. 3　Schematic diagram of the tamper-proofing encryption code

①根据插入的标记，找出待加密代码段相对于 PE 文件头的偏移地址 offset1 和 offset2.

②使用外部加密器，对 offset1，offset2 之间的代码进行加密，加密过程如下：

（ⅰ）根据 k_i 以及完美 hash 函数 h，查找位移表 T，得出 Q_i 代码段的起止地址；

（ⅱ）根据校验和计算函数，计算出相应的 Q_i 代码处的校验和 V_i；

（ⅲ）根据 $Key_i=SHA1[k_i \oplus V_i]$，计算 Key_i；

（ⅳ）将 Key_i 作为密钥，对目标文件中的 offset1 与 offset2 之间的代码进行加密.

2.5 水印的提取

可按照如下步骤提取水印信息：

①按照特定的输入来执行程序，当输入完毕后，在堆中会生成表示水印子数据的 MPPCT 结构，根据每个 MPPCT 结构的根节点，得到 MPPCT 完整的图结构，然后将 MPPCT 结构解码为相应的水印子数据 w_i.

②根据 GCRT 分割原理，将一系列的 w_i 还原为最初的水印数据 W.

3 实验与数据分析

3.1 性能过载分析

水印的嵌入肯定会给程序带来性能影响，本文将从两个方面来衡量这种性能影响：嵌入水印后的程序在运行空间上的过载以及执行时间上的过载. 下面通过实验数据来分析嵌入水印前后，宿主程序在空间和执行时间上的变化.

（Ⅰ）测试环境

操作系统：Windows XP；处理器：Pentium 4 CPU 2.80 Hz；内存：1.5 G

水印数据：38911；实验时，分别将其分割为 3 块与 15 块

加密算法：AES 加密算法

（Ⅱ）实验结果

表 1 针对空间过载进行了分析，实验数据表明，将一个代表版权的大数 W 分割成 $r(r-1)/2$ 个子数据，构造这些子数据的 MPPCT 结构，并将它们插入程序中，会增加程序的大小，而且分割的子数据越多，带来的空间开销越大；为了实现防篡改功能而在程序中添加的计算校验和的代码、加解密的代码以及完美 hash 函数也会给程序带来一些空间开销；实验研究还表明，嵌入水印前的程序越小，则嵌入水印引起的空间过载越大.

表 2 针对时间上过载进行了分析，实验表明将水印数据分割为多个子数据块，会影响程序的执行时间，给程序带来时间过载. 当修改的 call 指令很少时，水印分割块数的多少对程序执行时间的影响差别不大. 但是如果修改的 call 指令较多，程序执行过程中会不断地将 MPPCT 结构解码为相应的水印子数据，而且程序运行过程中 WBF 会频繁地查表，这些都会影响到程序的执行时间，给程序带来时间过载. 当执行到密文函数时，对密文函数的执行前解密与执行后加密操作，也会影响到程序的执行时间，给程序带来时间过载.

3.2 鲁棒性分析

（Ⅰ）添加与减少攻击

本文采用一个交错和互相保护机制来增强系统的安全性. 如果攻击者使用他们的水印来替代原始的水印，或者删除部分水印信息，势必造成 WBF 函数计算出错误的密钥 k_i，使得 WBF 查找错误的位移，程序执行错误的控制流. 同时，也会使解密函数计算了错误的解密密钥，导致程序运行终止. 因此，本文的软件水印方案可以有效地抵抗添加攻击与减

表 1 嵌入水印前后程序大小改变情况（n 为水印分割块数）

Tab. 1 Size change after embedding watermark

测试程序	原始大小/kB	修改的 call 指令/个	加密函数/个	嵌入水印后大小/kB		增长比例/%	
				$n=3$	$n=15$	$n=3$	$n=15$
GCRT	524	1	1	596	616	13.7	17.5
		5	1	596	616	13.7	17.5
Encrypt	532	3	2	604	624	13.5	17.29
		5	3	604	624	13.5	17.29
Conzilan	2032	25	8	2100	2124	3.3	4.5
		38	10	2100	2124	3.3	4.5

表 2　嵌入水印前后程序执行时间改变情况(执行 10 次取平均值)

Tab. 2　Execution time change after embedding watermark

测试程序	原始时间/ms	修改的 call 指令/个	加密函数/个	嵌入水印后时间/ms		增长比例/%	
				$n=3$	$n=15$	$n=3$	$n=15$
GCRT	14	1	1	15	15	7	7
		5	1	15	15.3	7	7.2
Encrypt	16	3	2	18	18.2	12.5	13
		5	3	19	19.3	18.7	20
Conzilan	102	25	8	160	161	56.8	57.8
		38	10	137	138	34.3	35.2

少攻击.

(Ⅱ) 扭曲攻击

程序里有一张存放位移的表,程序运行时 WBF 函数根据 k_i 查找表,得到原始目标函数地址及其返回地址之间的偏移地址,从而计算出目标函数的地址.当对程序进行保留语义转换时,目标函数的地址发生改变,导致 WBF 函数返回到错误的地址处,程序执行错误的控制流.因此,本文提出的软件水印方案可以有效抵抗扭曲攻击.另外,使用 GCRT 分割法分割水印数据时,若不超过一半的水印子数据被修改,原始水印数据还能被恢复.综合以上分析,本文提出的软件水印方案具有较强的鲁棒性.

3.3　数据率分析

如表 3 所示,在给定节点数目情况下,从 3 种图结构所能编码的数据容量对比情况可以看出:MPPCT 结构的数据编码容量高于 PPCT 结构,虽然低于 K 基数循环链表,但是由于自身结构的特点,它的抗攻击性要高于 K 基数循环链表.在水印系统中,使用一个交错和互相保护机制来增强系统的安全性,如果攻击者修改了 MPPCT 结构,势必造成 WBF 函数计算出错误的密钥 k_i,从而使得 WBF 查找错误的位移,程序执行错误的控制流.另外,也会使得解密函数计算出错误的解密密钥,导致程序运行终止,这种机制在一定程度上增强了 MPPCT 结构的抗攻击性.另外如果给定 MPPCT 结构的节点数目,会产生多种 MPPCT 结构,因此 MPPCT 结构也可以生成一个能够抵抗共谋攻击的水印库.所以采用 MPPCT 结构作为隐藏水印信息的拓扑图结构,是一种良好的水印编码方案.

表 3　3 种不同水印拓扑图结构编码容量对比表

Tab. 3　Encoding volume comparison of 3 different watermark topological structures

节点数目	K 基数循环链表	PPCT 结构	MPPCT 结构
M	$M^{M-1}-1$	$\frac{2}{M}C_{M-2}^{\frac{M-2}{2}}$	$(M-1)^{M/2}$
2	1	1	1
10	1×10^{9}	14	5.9×10^{4}
100	1×10^{198}	5.09×10^{26}	6.05×10^{99}
200	2×10^{398}	2.27×10^{56}	7.67×10^{229}

3.4　安全性分析

该算法的安全性是由程序执行过程中的保密性与完整性保证的.

(Ⅰ) 程序执行过程中的保密性

本算法对部分代码进行加密,保护了算法的私有性.程序正常执行过程中,系统无法识别处于加密态的目标代码,因此处于加密态的代码在被解密前是安全的.系统对受保护的代码使用完毕后,立即对其进行加密,虽然损失了执行效率,但是使内存中不会出现整个代码的明文内容,增加了破解难度.

(Ⅱ) 程序执行过程中的完整性

为了预防攻击者对水印代码等敏感性代码进行篡改,使用了基于代码加密的防篡改技术,将敏感性代码的短消息摘要以及水印分支函数产生的密钥共同作为密钥因子.如果敏感性代码段 Q_i 被修改为 Q_i',则 $\mathrm{hash}(Q_i')\neq\mathrm{hash}(Q_i)$,最终导致受保护的代码解密错误.在水印系统的实现过程中,可以将待检测校验和的代码覆盖整个程序.另外使用的交错和相互保护机制决定攻击者不能修改任何一处代码,否则势必导致解密失败.如果攻击者想要成功删除水印信息,需找出所有的被加密代码及其对应的校验和代码段,同时需要知道每个代码段的密钥,而且随着被加密代码个数的增多,破解会变得更加困难.

(Ⅲ) 面向逆向工程的安全性分析

攻击者通过逆向工程技术分析出软件的架构,利用篡改机制对软件代码进行修改.本部分从逆向

攻击的角度对本方案在安全性上进行验证.借鉴文献[14]的相关方案,利用 IDA[15]分析工具在攻击过程中对攻击开销的计算评测嵌入水印后系统的安全性,在逆向攻击中主要以攻击开销对测试体进行评测,对比逆向前后攻击开销的变化来评测攻击者进行篡改和分析的难度.从表 4 的实验结果来看,通过 WBF 函数加密过的程序针对敏感性代码的攻击开销明显增长.

表 4 不同算法攻击开销比较

Tab. 4 Comparison on attack cost of different algorithms

开销	攻击过程		
	WBF 算法	FBW 算法	GTW 算法
反汇编	22	21	19
查找关键代码	35	23	23
控制流分析	24	26	24
数据流分析	28	29	24

4 结论

软件水印技术解决的是软件版权保护问题.本文围绕传统软件水印中有限的防篡改能力和较低的数据率、鲁棒性等问题,在深入研究软件水印关键技术的基础上,给出了改进的水印拓扑图结构,同时提出了将代码加密防篡改技术与软件水印技术相结合的软件保护方案,最后分析验证了该方案的可行性和有效性.

参考文献(References)

[1] Collberg C, Thomborson C. Software watermarking: Models and dynamic embeddings[C]// Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 1999:311-324.

[2] Collberg C, Thomborson C, Townsend G. Dynamic graph-based software watermarking[R]. Tucson, AZ, USA: University of Arizona, 2004: TR04-08.

[3] Moskowitz S, Cooperman M. Method for stega-cipher protection of computer code: US, 5745569 [P]. 1998-04-28[2011-04-15].

[4] He Yong. Tamperproofing a Software Watermark by Encoding Constants [D]. Auckland, New Zealand: University of Auckland, 2002.

[5] Holmes K. Computer software protection: US, 5287407 [P]. 1994-02-15[2011-04-15].

[6] Khiyal M, Khan A, Amjad S, et al. Evaluating effectiveness of tamper proofing on dynamic graph software watermarks [J]. International Journal of Computer Science and Information Security, 2009, 6 (3):57-63

[7] Li Honglei. Research on Software Watermarking and Tamper-proofing [D]. Guangzhou: Guangdong University of Technology, 2005.
李红蕾. 软件水印及其防篡改技术研究与实践[D]. 广州:广东工业大学, 2005.

[8] Shen Jingbo. Protect Software Using Watermarking Techiniques [D]. Xi'an: Northwest University, 2006.
沈静博. 基于数字水印的软件保护技术研究[D]. 西安:西北大学, 2006.

[9] Yang Zhigang. Tamper-proofing Software Watermarking Technology Based on Constant Encoding [D]. Changchun: Jilin University, 2009.
杨志刚. 基于常量编码的防篡改软件水印技术[D]. 长春:吉林大学, 2009.

[10] Collberg C, Jha S, Tomko D, et al. UWStego: A general architecture for software watermarking [EB/OL]. Madison, WI, USA: University of Wisconsin-Madison, 2001[2011-04-15]. http://www.cs.wisc.edu/hbwang/watermark/TR.

[11] Myles G. Software theft detection through program identification[D]. University of Arizona, Tucson AZ, 2006:56-68.

[12] Zhang Lihe, Yang Yixian, Niu Xinxin, et al. A survey on software watermarking [J]. Journal of Software, 2003,14(2):268-277.
张立和,杨义先,钮心忻,等. 软件水印综述[J]. 软件学报, 2003,14(2):268-277.

[13] Collberg C, Jasvir N. Surreptitious Software Obfuscation, Watermarking, and Tamperproofing for Software Protection [M]. Addison-Wesley Professional, 2009:526-528.

[14] Ceccato M, Di Penta M, Nagra J, et al. Towards experimental evaluation of code obfuscation eechniques [C]// Proceedings of the 4th ACM workshop on Quality of protection. New York: ACM,2008:39-46.

[15] The IDA Pro Disassembler and Debugger[DB/OL]. Belgium: Hex-Rays[2011-04-15]. http://www.hex-rays.com/idapro/.