基于混沌不透明表达式的不透明谓词混淆技术研究

苏 庆 孙金田
(广东工业大学计算机学院 广州 510006)

摘 要 为了更好地进行代码混淆,提出了一种基于混沌映射和二次映射的混沌不透明表达式构造方法。根据混沌不透明表达式的定义,采用具有初值敏感依赖性、伪随机性、状态空间均匀分布性、多分支性和无特殊符号等性质的混沌映射。以二维帐篷映射为例,创建与之相匹配的二次映射,将混沌映射的运行状态空间映射至表达式的结果空间,以进行混沌不透明表达式的构造。将混沌不透明表达式与不透明谓词相结合,形成了一种新的不透明谓词构造方法,同时提出了一种新构造谓词与原有谓词融合于一体的不透明谓词插入方法,两者结合形成了一种新型的不透明谓词混淆技术。实验结果表明,该技术令各项软件的复杂度指标都有明显的提升,并且增加的程序开销较小。

关键词 代码混淆,混沌映射,混沌不透明表达式,二维帐篷映射,不透明谓词
中图法分类号 TP309.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.023

Research on Opaque Predicate Obfuscation Technique Based on Chaotic Opaque Expression

SU Qing SUN Jin-tian
(School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China)

Abstract In order to improve the code confusion, a chaotic opaque expression construction method based on chaotic map and quadratic map was proposed. According to the definition of chaotic opaque expression, the chaotic map with the properties of initial value sensitive dependence, pseudo-randomness, uniform distribution of state space, multiple branching and non-special symbols is used. Taking two-dimensional tent map as an example, the matched quadratic map maps the running state space of the chaotic map to the result space of the expression to construct a chaotic opaque expression. Combining chaotic opaque expression with opaque predicate, a new method for constructing opaque predicate is formed. At the same time, an opaque predicate insertion method was proposed in which a new construction predicate is merged with an original predicate. The combination of the both forms a new opaque predicate obfuscation technique. The experimental results show that the technique has obvious improvement in various software complexity indexes, and the cost of the program is relatively lower.

Keywords Code obfuscation, Chaotic map, Chaotic opaque expression, Two-dimensional tent map, Opaque predicate

1 引言

代码混淆指将原程序进行等价转换,使转换后得到的新程序与原程序在功能上等价,但新程序中的数据和代码结构变得复杂且可读性差,从而导致破解者的逆向分析工作变得更加困难[1]。在主流的代码混淆技术中,不透明谓词混淆技术具有形式简单且不显著增加程序开销的优点,成为了热点的控制流混淆技术之一[2]。Collberg等人[3]首先提出别名构造和并发性构造两种不透明谓词的构造方法,一定程度地增加了不透明谓词的破解难度。Arboit[4]使用二次剩余理论来构造不透明谓词,并结合软件数字水印技术设计了一种基于不透明谓词的Java程序水印方案,但经过分析和评估验证,该方案的密码安全性不佳[5]。后来Yuan Zheng等人[6]提出利用初等数论中的同余方程来构造不透明谓词,相比于二次剩余方式,其在安全性和代价开销方面都更具优越性。近年来,随着混沌理论的发展和应用,苏庆等人[7]提出了基于混沌映射Logistic及其改进的En_Logistic混沌映射来构造不透明谓词簇的方法,并通过实验验证了整个混淆转换系统对密钥的高度敏感性,证明了该方法密码安全性较强的特点。Xie Xin等人[8]针对静态反汇编可以准确分析控制流图的问题,提出了基于超混沌不透明谓词的重叠指令和自修改代码的混合混淆技术,可以有效增加静态反汇编的难度。

事实上,不透明谓词是一种特殊的不透明表达式。当不透明表达式的结果空间为{true,false}时,不透明表达式退化为不透明谓词。Wang[9]在实现压扁控制流算法的过程中首先提出了一种利用全局数组构造不透明表达式的算法思想。

到稿日期:2016-10-16 返修日期:2017-04-22 本文受国家自然科学基金(61273118),广东省科技计划(2016A010101027,2013B022200004),广州市科技计划(201605101034176)资助。

苏 庆(1979-),男,博士,副教授,主要研究方向为软件安全与保护,E-mail:99537380@qq.com;孙金田(1989-),男,硕士生,主要研究方向为软件安全。

Collberg[10]受其启发给出了一组具体利用全局数组构造不透明表达式的规则，并将其应用于不透明谓词构造算法中，但该算法通用性不高。Andreas Moser 等人[11]提出了一种依赖于不透明表达式的二进制代码混淆技术，并通过分析及实验证明了此种技术抵抗静态分析的能力较强。林水明[12]提出了一种利用 Arnold 混沌映射构造 N 态不透明谓词的自动化过程，其本质也是一种不透明表达式的构造算法，虽然该方法提高了不透明表达式的安全性，但存在构造效率过低的问题。

本文采用二维帐篷混沌映射，给出了一种构造效率较高的不透明表达式构造算法，并将其与数论表达式相结合来构造混沌不透明谓词，从而提出了一种新的不透明谓词插入方法，形成了一种新的不透明谓词混淆技术。

2　混沌不透明表达式的构造

定义 1(不透明表达式)　在代码混淆中，若一个表达式 $P=F(I)$ 的值 $p\in\Theta$(Θ 为表达式结果空间)在嵌入程序之前已确定，但攻击者却很难根据表达式本身推断该值；或者能够迷惑攻击者，令其不能洞察程序设计者使用常量 p 的意图，则称 P 是不透明表达式。对于 $P=F(I)$，I 为输入空间，采用某种实现机制 $f\in F$，可得 $p=f(i)$，其中 $i\in I$。当不透明表达式的取值为 K 时，记作 P^{-K}。

混沌[13]指确定性动力学系统因对初值敏感而表现出的不可预测且类似随机性的运动。常见的混沌映射可用确定性的非线性差分方程来描述，其不包含任何随机因素，但其轨迹可能是完全随机的。混沌映射有很强的不确定性和复杂性，并已证明存在与生俱来的密码安全性，因此可以利用混沌映射的特性构造高安全性的不透明表达式。

定义 2(混沌不透明表达式)　运用混沌映射 *chaos* 来构造混沌不透明表达式。由于混沌映射的运行状态空间不一定与表达式结果空间相同，因此需要构造另一个映射 *map* 来将混沌映射的运行状态空间映射至表达式结果空间。混沌不透明表达式的一般形式为：$E=M(C(I))$，其映射过程为 $I\xrightarrow{chaos}C(I)\xrightarrow{map}\Theta$。为表述方便，使用三元组 $E(C,I,M)$ 来对其进行表示，其中 C 为混沌映射，I 为输入初值空间，M 为二次映射方法。

2.1　混沌映射的选择

一个可应用于构造混沌不透明表达式的混沌映射应满足以下条件[7]：

1)对初值具有敏感依赖性。该特性有利于增加整个系统的不确定性。

2)混沌序列具有良好的伪随机特性。伪随机性越高，迭代生成的实数序列的统计特性就越不明显，从而增大了攻击者根据当前状态推测下一个状态或者表达式结果的难度。

3)混沌序列在状态空间中均匀分布。该特性有利于生成均匀的覆盖全局空间的状态序列，从而提高混沌不透明表达式的构造效率。

4)混沌映射公式较为复杂，最好有多条分支。混沌序列产生的过程越复杂，破解者对不透明表达式进行静态分析的难度越大。

5)混沌映射公式中不能包含特殊的数学符号或数学公式，否则隐蔽性较差，容易被发现和破解。

袁赣南等人[13]通过扩展一维帐篷映射提出了一种二维帐篷映射，并证明了该映射具备混沌特性，其定义如下。

设 $I_0=[0,1]\times[0,1]$ 为平面上的单位方体，定义平面上方体的帐篷映射：$f_{\alpha,\beta}:I_0\to I_0$ 为：

$$f_{\alpha,\beta}(x,y)=\begin{cases}\dfrac{x}{\alpha},\dfrac{y}{\beta}, & x\in[0,\alpha],y\in[0,\beta]\\ \dfrac{1-x}{1-\alpha},\dfrac{y}{\beta}, & x\in[\alpha,1],y\in[0,\beta]\\ \dfrac{x}{\alpha},\dfrac{1-y}{1-\beta}, & x\in[0,\alpha],y\in[\beta,1]\\ \dfrac{1-x}{1-\alpha},\dfrac{1-y}{1-\beta}, & x\in[\alpha,1],y\in[\beta,1]\end{cases}\tag{1}$$

其中，$0<\alpha<1$，$0<\beta<1$。

本文研究发现，参数 α 和 β 的取值对生成的混沌序列的影响较大，若参数取值不理想，则可能会使生成的混沌序列不够均匀。图 1 给出了当 α 和 β 分别取不同值时，初值 (x_0,y_0) 取(0.342781,0.543981)并经过 500 次迭代后产生的混沌序列在空间的分布情况。

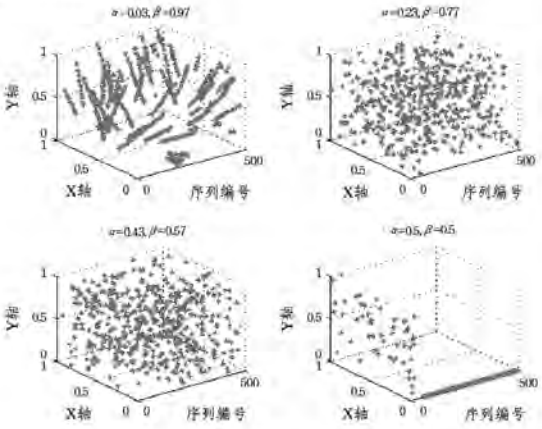


图 1　不同参数下二维帐篷映射混沌序列的空间分布

从图 1 可以看出，当 α 和 β 的取值靠近区间(0,1)的两端时，生成的混沌序列在空间的分布不够均匀，因此 α 和 β 应尽量取(0,1)中间部分的值。但当 α 和 β 同取 0.5 时，混沌序列在迭代一定次数之后也出现了不均匀的现象。本文取 $\alpha=0.43$，$\beta=0.57$，此时，二维帐篷混沌映射的公式为：

$$f_{\alpha,\beta}(x,y)=\begin{cases}\dfrac{x}{0.43},\dfrac{y}{0.57}, & x\in[0,0.43],y\in[0,0.57]\\ \dfrac{1-x}{0.57},\dfrac{y}{0.57}, & x\in[0.43,1],y\in[0,0.57]\\ \dfrac{x}{0.43},\dfrac{1-y}{0.43}, & x\in[0,0.43],y\in[0.57,1]\\ \dfrac{1-x}{0.57},\dfrac{1-y}{0.43}, & x\in[0.43,1],y\in[0.57,1]\end{cases}\tag{2}$$

由式(2)可知，该混沌映射公式较为复杂，且具备多条分支，满足条件 4)。此外，该混沌映射未包含特殊的数学符号或数学公式，因此也满足条件 5)。

图 2 给出了分别采用初值 (0.342781, 0.543981) 和

(0.342782,0.543982),并迭代 300 次时形成的混沌序列空间分布情况。从图 2 可看出,虽然初值接近,但在经过有限次迭代后,两个迭代序列分布迥异,并且均表现出无规律可循的伪随机现象。因此,该混沌映射具备良好的初值敏感性和伪随机特性,满足条件 1)和条件 2)。

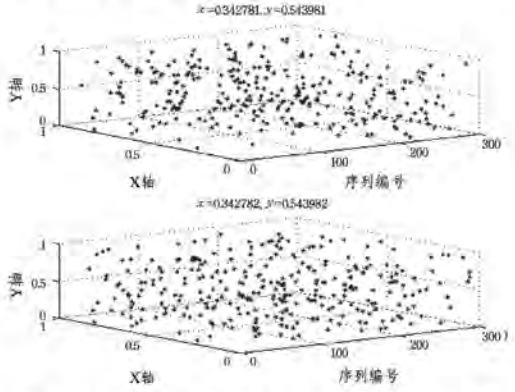


图 2 初值相近的二维帐篷映射混沌序列的空间分布

图 3 给出了对平面帐篷映射迭代 1×10^5 次并将平面分为 10×10 的小平面方体时,落入各个小平面方体点的个数的分布图,其初值为(0.342781,0.543981)。

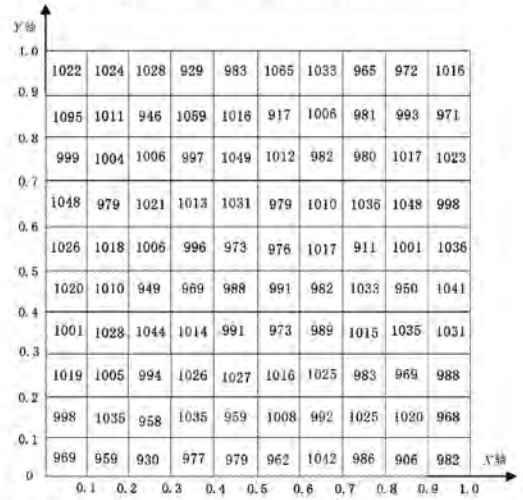


图 3 迭代 10^5 次后各个小平面方体点的个数分布图

在显著性水平为 5%时,对落入各个小平面方体点的个数进行 χ^2 均匀分布检验[15]得到的检验值为:

$$\chi^2=\sum_{i=1}^{10}\sum_{j=1}^{10}\frac{(n_{i,j}-N/100)^2}{N/100}=113.116 \tag{3}$$

当 $\alpha=0.05$ 时,通过 χ^2 分布临界值表可知其临界值为 124.34,因此二维帐篷混沌映射能够通过显著性水平为 5% 时的均匀分布假设检验。此外,落入各个小平面方体点的个数的最大值和最小值分别为 1095 和 906,最大值与最小值之差较小。由上可知,二维帐篷混沌序列表现出良好的均匀性,满足条件 3)。

综上,二维帐篷混沌映射符合不透明表达式构造的各项要求,因此选用二维帐篷混沌映射来进行构造。

2.2 混沌不透明表达式构造算法

定义 3(二维帐篷映射混沌不透明表达式) 该表达式为一个五元组 $E(T,x_0,y_0,h[\ ],fun)$,其中,T 为二维帐篷混沌映射,$h(x_0,y_0)$为 T 的输入初值,$h[\ ]$为迭代步长数组,fun 为将 T 的运行状态空间 Ω 映射至表达式结果空间 Θ 的二次映射。

设表达式结果为 Γ^n,其有 n 位。由(x_0,y_0)和 Γ^n 经过 T 映射和 fun 映射可以计算 $h[\ ]$,其计算公式为:

$$h[\ ]=fun*(T*(x_0,y_0,\Gamma^n)) \tag{4}$$

不失一般性,假设需构造的表达式的结果空间为 Z,Z 为任意 n 位整数集合。对于 $\forall(z_1,z_2,\cdots,z_n)\in Z$,对应的不透明表达式构造算法描述如下。

Step1 随机生成二维帐篷映射 T 的初始状态向量$\vec{X}_0=(x_0,y_0)$,其中 $0<x_0,y_0<1,x_0,y_0\in R$,R 为实数集合。

Step2 由式(2)迭代产生下一个状态向量$\vec{X}_i=(x_i,y_i)$ $(i=1,2,\cdots,i$ 为迭代步长),通过二次映射函数 fun 将状态向量$\vec{X}_i$映射至整数 N_i,即 $N_i=fun(\vec{X}_i)$,其中 $N_i\in Z\&0\leqslant N_i\leqslant9$,将当前迭代步长 i 与整数 N_i 存入映射集合,即 $U=U\cup\{(i,N_i)\}$。

Step3 判断迭代步长 i 是否已经达到阈值 V:

1)若 $i<V$,则重复 Step2,直至 $z_1z_2\cdots z_n$ 每一位数 $z_j(j=1,2,\cdots n)$在 U 中都存在相应的元素(i,N_i)使得 $z_j=N_i$,此时称 U 已完备,并可用于构造不透明表达式,转 Step4。

2)否则,放弃本次构造,重新转向 Step1。

Step4 建立一个存储迭代步长的数组 $h[\ ]$。对于整数 $z_1z_2\cdots z_n$ 每一位上的数字 $z_j(j=1,2,\cdots,n)$,分别在 U 中使用随机或其他策略定位一个集合元素(i,N_i)使得 $z_j=N_i$,并从(i,N_i)中将 i 取出,且依次存入数组 $h[\ ]$中。构造完毕。

按照如下方式构造不透明表达式构造算法中采用的二次映射函数 fun。

将(0,1)×(0,1)平面均分为 10×10 的小平面体。在保证平均分配的情况下将这 100 个小平面方体任意分为 10 组,并使每组小平面方体依次映射 0 到 9 之间的不同值。图 4 给出了其中的一种划分和映射方式,例如,假设$(x_i,y_i)=(0.226323,0.249763)$,由该图可以查出其映射值为 9,对应于图中的阴影方格。

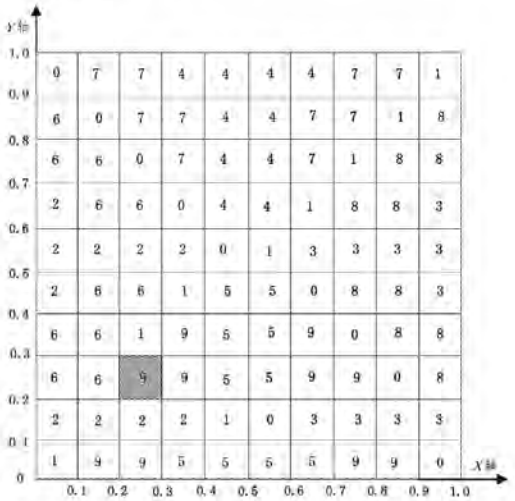
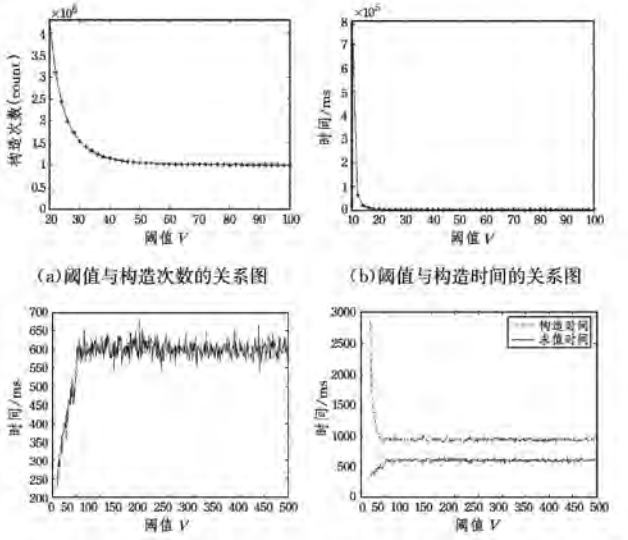


图 4 二次映射函数举例

阈值取值的大小会影响不透明表达式的构造效率。仍然

假设表达式结果空间为 Z（Z 为 n 位整数集合）。对于 $\forall(z_1 z_2\cdots z_n)\in Z(0\leqslant z_i\leqslant 9)$，每一位取值都是 0～9 之间的整数，最坏情况下 $z_1 z_2\cdots z_n$ 中的 $z_i(i=1,2,\cdots n)$ 包含了 0～9 中的所有可能，此时集合 U 达到最大完备状态。譬如，若执行某次算法时能够构造出整数 1234567890，则可以构造出任意整数。图 5 给出了在不同阈值 V 下对整数 1234567890 成功构造 1×10^6 次时阈值与构造次数、构造时间和求值时间之间的关系图。



(a)阈值与构造次数的关系图　(b)阈值与构造时间的关系图

(c)阈值与求值时间的关系图　(d)阈值与构造时间、求值时间的关系图

图 5　阈值与构造次数、构造时间和求值时间之间的关系

从图 5(a)可以看出，随着 V 的增大，构造算法需执行的次数 *count* 逐渐收敛于 1×10^6，即基本上每次执行构造算法都能成功构造出整数 1234567890。

图 5(b)给出了阈值 V 与构造时间之间的关系，随着 V 的增大，构造时间快速减少，在 $V=50$ 附近，构造时间收敛。此时基本上每次都可以成功构造不透明表达式。另外，只有 $V\geqslant n$（n 为表达式结果的最大位数）时，方有可能成功构造不透明表达式，因此在[10，50]区间内，V 越大，构造效率越高。图 5(c)给出了阈值 V 与求值时间的关系，随着 V 的增大，求值时间快速上升，同样收敛于 $V=50$ 附近。其原因在于在 $V=50$ 附近，构造成功所需的迭代次数相对稳定，因此求值时同样所需的迭代次数也相对稳定。但总体而言，求值时间小于构造时间（见图 5(d)），构造时间与求值时间大致上是平行的折线，其原因在于混沌不透明表达式的构造算法复杂度远远大于求值算法。这也说明一旦不透明表达式构造完成并插入程序后，所增加的程序运行时间开销远小于构造开销。

3　基于混沌不透明表达式的不透明谓词混淆技术

不透明谓词作为控制流混淆的一种热点技术，其生成和插入都是研究重点。初期基于数论中的数学表达式构造的不透明谓词形式简单，且易被破解[3]；近期直接应用混沌映射而构造的不透明谓词的安全性则大大增强。本文提出一种基于混沌不透明表达式的不透明谓词混淆方法，该方法既可以与数论表达式相结合，又能发挥混沌映射的安全性特点，为不透明谓词混淆技术提供了一种新思路。

3.1　混沌不透明表达式与不透明谓词的结合

数论中存在永真和永假的表达式，这些表达式都可以用于构造不透明谓词。数论表达式的形式一般较简单，应用这些表达式构造的不透明谓词的输出容易被判断，因此通过逆向分析将其破解的概率较高。运用混沌不透明表达式替换简单数论表达式中的常量，可将数论表达式的形式复杂化。在混沌不透明表达式替换常量后的表达式中，由于常量被等价转换为函数形式，因此大大增加了攻击者对表达式进行攻击的工作量和难度，提高了安全性。

表 1 列举了部分永真和永假表达式及其应用混沌不透明表达式替换常量后的形式。*chaos*( )是不透明表达式的求值函数，将值为 *num* 的混沌不透明表达式表示为 $chaos(key^{=num})$。

表 1　表达式举例

表达式类别	简单数论表达式	混沌不透明表达式替换常量后的表达式
永真表达式	$7*y^2-x^2\neq1$	$chaos(key^{=7})*y^2-x^2\neq chaos(key^{=1})$
	$x^2>0$	$x^2>chaos(key^{=0})$
	$x(x+1)\%2=0$	$x(x+chaos(key^{=1}))\%chaos(key^{=2})=chaos(key^{=0})$
永假表达式	$x(x+1)\%2\neq0$	$x(x+chaos(key^{=1}))\%chaos(key^{=2})\neq chaos(key^{=0})$
	$x(x+1)(x+2)\%3\neq0$	$x(x+chaos(key^{=1}))(x+chaos(key^{=2}))\%chaos(key^{=3})\neq chaos(key^{=0})$

3.2　一种不透明谓词插入方法

在攻击者的角度，攻击不透明谓词 $P(x_1,x_2,\cdots,x_n)$ 的过程为[10]：

(1)定位构成不透明谓词 $P(x_1,x_2,\cdots,x_n)$ 的指令。

(2)确定 P 的输入参数，即 $x_1,x_2,\cdots,x_n$。

(3)确定各个输入参数的取值范围 $R_1,R_2,\cdots,R_n$。

(4)根据 $x_1,x_2,\cdots,x_n$ 所有的取值进行排列组合，计算 P 的所有取值。如果无法对 P 的所有可能输入逐一进行测试，那么可借助统计的方法在一定概率上确定 P 的结果。

(5)若 P 的所有取值总为真，则将条件转移指令替换成无条件转移指令；若 P 的所有取值总为假，则将包括条件转移指令在内的所有组成谓词的指令全部忽略。

由于攻击者首先需要定位不透明谓词，因此本文提出一种新的不透明谓词插入方法，该方法分拆不透明谓词 Q，并与程序中原有的表达式 P 合并，从而组成一个新的谓词 $P\&Q$，大大增加了定位不透明谓词的难度。

此不透明谓词的插入过程如下：

Step1　寻找分支语句中的表达式 P，以比较运算符 op 为界，分解 P_1，op 和 P_2 3 部分：

$P=(P_1\ op\ P_2)$

Step2　构造混沌永真不透明谓词 Q，以运算符“=”为界，同样分解为 Q_1，= 和 Q_2 3 部分：

$Q=(Q_1=Q_2)$

Step3　将 Q_1 和 Q_2 分别添加到 P 的左右两端，从而构成一个新的谓词 $P\&Q$。

$P\&Q=((P_1+Q_1)op(P_2+Q_2))$

由于 Q_1 和 Q_2 的等价性，$P\&Q$ 的真值表与 P 完全相同，因此若 P 不是永真式，则 $P\&Q$ 根据不同输入表现出时真时

假的特性，大大增加了不透明谓词被发现和破解的难度，同时也使得程序原有的表达式 P 更加复杂，从而起到进一步掩盖作者意图的作用，使得混淆后的程序具有更高的安全性。

例如，令：

$P=(x>y)$

$Q=x(x+chaos(key^{-1}))\%chaos(key^{-2})=chaos(key^{-0})$

则

$P\&Q=x+x(x+chaos(key^{-1}))\%chaos(key^{-2})$
$>y+chaos(key^{-0})$

不透明表达式插入源程序中的效果如图 6 所示。

混淆前：	混淆后：
int x=某复杂表达式；	int x=某复杂表达式；
int y=某复杂表达式；	int y=某复杂表达式；
if($x>y$){	if($x+x*(x+chaos(key=1))\%chaos(key^{-2})>y+chaos(key^{-0})$){
执行代码段；	执行代码段；
}	}

图 6 代码混淆前后的对比图

在此实例中，攻击者无法将不透明谓词 Q 从 $P\&Q$ 中区分开来，只能对整个较为复杂的 $P\&Q$ 进行攻击，大大增加了难度和工作量。

4 实验结果及分析

为了评估基于混沌不透明表达式的不透明谓词混淆技术在实际应用中的性能，本文以闰年统计程序 YearsToKnow 为测试用例，通过比对混淆前后的各项实验结果，来对混淆后程序的各种常用程序复杂度、控制流复杂度以及代价开销等进行评价。

4.1 常用复杂度分析

Sandmark 工具[16]可以计算多种常用的程序复杂度。表 2 列出了 YearsToKnow 程序在混淆前和混淆后(插入不透明谓词的个数分别为 1,2,3 和 10)的各种复杂度值。

表 2 混淆前后程序的复杂度比较

	混淆前	混淆后(不同的不透明谓词插入个数)			
		1	2	3	10
Halstead Measure	997	9373	11657	14810	18753
McCabe Cyclomatic metrics	10	17	17	18	24
Harrison Measure	132	387	445	490	834
Kafura Measure	1156	1600	2116	2704	5972
Munson Measure	21	44	44	44	44
CK Measure	18	20	21	23	35

从表 2 可以看出，Halstead Measure，McCabe Cyclomatic metrics，Harrison Measure，Kafura Measure 和 CK Measure 都会随着不透明谓词插入个数的增多而增大。而 Munson Measure 是数据结构复杂度，本文在第一次插入不透明谓词时引入了额外的数据结构，因此其复杂度也只在第一次插入不透明谓词时有明显提升，之后不变。

4.2 控制流复杂度分析

Sandmark 工具提供了生成程序控制流程图的功能，并且可以对流程图进行缩放，以方便用户查看和比较。图 7 给出了混淆前 YearsToKnow 程序的控制流程图。

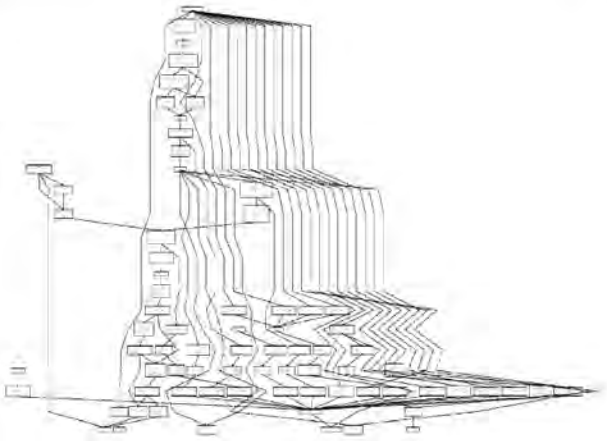


图 7 混淆前 YearsToKnow 程序的控制流程图

图 8 给出了 YearsToKnow 程序经过一次不透明谓词混淆后的程序的控制流程图。与图 7 相比，混淆后程序的流程结构的主要变化是增加了不透明表达式的求值过程及插入的不透明谓词的求值过程，大致对应于图中矩形线框内的区域。

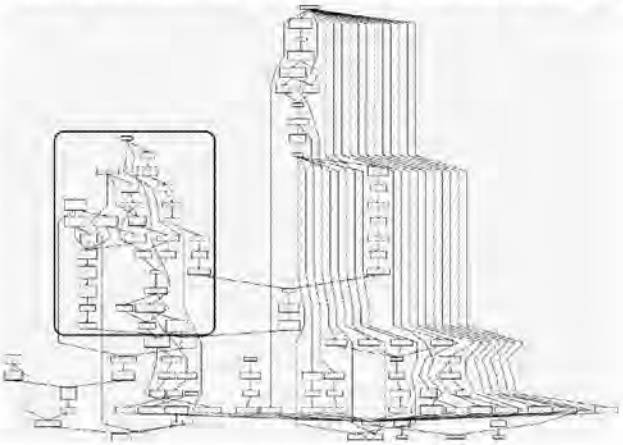


图 8 经过混沌不透明谓词混淆后的控制流程图

对比图 7 和图 8 可知，混沌不透明谓词混淆能够有效地改变原代码的流程结构，使得控制流结构更加复杂，增加了破解者进行逆向分析的难度。

4.3 代价开销分析

混淆前后的静态分析结果如表 3 所列。从表 3 可以看出，将误差考虑在内，插入有限数量的不透明谓词对程序运行时间造成的影响较小，说明本文所用不透明谓词的构造算法的效率较高；由于第一次插入不透明谓词时需要引入混沌不透明表达式求值函数，因此会给程序带来额外的容量增长，除此之外，插入不透明谓词的个数与程序容量的增长呈线性相关，且平均每次插入不透明谓词只会给程序容量带来较小幅度的增长。总体来看，本文使用的混淆算法给程序带来的代价开销较小。

表 3 混淆前后的静态分析结果

	混淆前	混淆后(不同的不透明谓词插入个数)			
		1	2	3	10
运行时间/ms	183.57	183.43	182.58	184.15	183.37
程序容量/kB	4.39	5.04	5.09	5.13	5.18

结束语 本文将混沌映射与二次映射相结合，提出了一

种混沌不透明表达式的构造方法，并通过统计与分析验证了该构造方法具有较高的构造效率。随后基于混沌不透明表达式提出了一种新的不透明谓词构造方法和不透明谓词插入方法，增强了不透明谓词的安全性，并通过实验验证了该技术在各项软件复杂度指标中都有明显的提升，同时给程序增加的代价开销较小。未来将继续研究混沌不透明表达式中关于混沌映射以及二次映射的最优构造方案。

参 考 文 献

[1] PREDA M D, GIACOBAZZI R. Control Code Obfuscation by Abstract Interpretation[C]// IEEE International Conference on Software Engineering and Formal Methods. IEEE Xplore, 2005: 301-310.

[2] ZHAO Y J, TANG Z Y, WANG I. Evaluation of code obfuscating Transformation[J]. Journal of Software, 2012, 23(3): 700-711.

[3] COLLBERG C, THOMBORSON C D. Low manufacturing cheap, resilient, and stealthy opaque constructs[C]// Proceeding of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. California: ACM, 1998: 184-196.

[4] ARBOIT G. A method for watermarking java programs via opaque predicates[C]// The Fifth International Conference on Electronic Commerce Research (ICECR-5). 2002: 102-110.

[5] MYLES G, COLLBERG C. Software watermarking via opaque predicates: implementation, analysis, and attacks[J]. Electron Commerce Research, 2006, 4(6): 155-171.

[6] YUAN Z, WEN Q, MAO M. Constructing Opaque Predicates for Java Programs[C]// 2006 International Conference on Computational Intelligence and Security. 2006: 895-898.

[7] SU Q, WU W M, LI Z L, et al. Research and Application of Chaos Opaque Predicate in Code Obfuscation[J]. Computer Science, 2013, 40(6): 155-160.

[8] XIE X, LIU F, LU B, et al. Mixed Obfuscation of Overlapping Instruction and Self-Modify Code Based on Hyper-Chaotic Opaque Predicates[C]// Tenth International Conference on Computational Intelligence and Security. IEEE Computer Society, 2014: 524-528.

[9] WANG C X. A security architecture for survivability mechanisms[D]. Charlottesville: University of Virginia, 2001.

[10] COLLBERG C, NAGRA J. Surreptitious Software Obfuscation, Watermarking, and Tamperproofing for Software Protection [M]. Beijing: Post & Telecom Press, 2012: 204-205.

[11] MOSER A, KRUEGEL C, KIRDA E. Limits of Static Analysis for Malware Detection[C]// Computer Security Applications Conference, 2007. IEEE, 2007: 421-430.

[12] WU W M, LIN S M, LIN Z Y. chaotic-based opaque predicate control flow flatten algorithm[J]. Computer Science, 2015, 42(5): 178-182.

[13] GONZALOALVAREZ, SHUJUNLI. Some Basic Cryptographic Requirements for Chaos-Based Cryptosystems[J]. International Journal of Bifurcation & Chaos, 2006, 16(8): 2129-2151.

[14] YUAN G N, QIN H L, LAI D S. Construct of Tent Map Stream Cipher on Square[J]. Computer Engineering and Applications, 2002, 38(13): 124-126.

[15] SCHUMACKER R, TOMEK S. Understanding Statistics Using R[M]. Springer Publishing Company, Incorporated, 2013.

[16] COLLBERG C. A Tool for the Study of Software Protection Algorithms[EB/OL]. http://sandmark.cs.arizona.edu.

(上接第 109 页)

索玮岚. 基于扩展 VIKOR 的不确定语言多属性群决策方法. [J]. 控制与决策, 2013, 28(9): 1431-1440.

[16] ZHANG S, WANG T, GU X P. Synthetic Evaluation of Power Grid Operating States Based on Intuitionistic Fuzzy Analytic Hierarchy Process[J]. Automation of Electric Power System, 2016 (4): 41-49. (in Chinese)
张尚，王涛，顾雪平. 基于直觉模糊层次分析法的电网运行状态综合评估[J]. 电力系统自动化, 2016(4): 41-49.

[17] XIE M, DENG J L, LIU M B, et al. Temperature-lowering Load Estimation Method Based on Meteorological Data and Entropy Weight Theory[J]. Automation of Electric Power System, 2016 (3): 135-139. (in Chinese)
谢敏，邓佳梁，刘明波，等. 基于气象信息和熵权理论的降温负荷估算方法[J]. 电力系统自动化, 2016(3): 135-139.

[18] WANG C. Trust Evaluation Based on User's Behavion in Cloud Computing[D]. Baotou: Inner Mongolia University of Science & Technology, 2015. (in Chinese)
王超. 云计算环境下基于用户行为的信任评估研究[D]. 包头：内蒙古科技大学, 2015.

[19] LOURENZUTTI R, KROHLING R A. A generalized TOPSIS method for group decision making with heterogeneous information in a dynamic environment[J]. Information Sciences, 2016, 330: 1-18.

[20] LI C B, YUAN J H, QI Z Q. Investment Risk-Based Decision of Distributed Generation Based on Grey Cumulative Prospect Theory [J]. East China Electric Power, 2014, 42(5): 993-998. (in Chinese)
李存斌，苑嘉航，祁之强. 基于灰色累积前景理论分布式电源投资风险型决策[J]. 华东电力, 2014, 42(5): 993-998.

[21] XU W H, CHEN H Y, ZHANG Y P, et al. Fuzzy Comprehensive Evaluation Method Based on Measure of Medium Truth Degree [J]. Computer Science, 2016, 43(2): 204-209. (in Chinese)
徐文华，陈海燕，张育平，等. 一种基于中介真值程度度量的模糊综合评价方法[J]. 计算机科学, 2016, 43(2): 204-209.