计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2019.20190368
56(10):2183-2192,2019

一种基于分支条件混淆的代码加密技术

耿 普 祝跃飞
(战略支援部队信息工程大学 郑州 450002)
(23015636@qq.com)

A Code Encrypt Technique Based on Branch Condition Obfuscation

Geng Pu and Zhu Yuefei
(*Strategic Support Force Information Engineering University*, *Zhengzhou* 450002)

Abstract Code encryption based on path branch obfuscation of equal condition can make the key be apart from the encrypted program, so this encryption can oppose static and dynamic programming analysis method at the same time, but it can't be used with branches controlled by other comparative relationships. In this paper, using Lagrange interpolation method to produce input-preprocess function, we not only resolve the uniqueness problem of key which produced by multi-inputs of the branch, but also preserve the security of obfuscation on branch conditions, so the conditional code of multi-inputs branch can be encrypted. Using the method which resolves the uniqueness problem of key, code encryption based on equal condition branch obfuscation can be extended to branch of greater-than and less-than condition branch, and complicate condition branch formed by blending of interval condition and equal condition.

Key words conditional code encryption; path branch obfuscation; Lagrange interpolation method; multi-input branch; code protection

摘 要 基于分支条件混淆的代码加密技术,实现了秘钥和程序的分离,能够对抗程序静态和动态分析手段,但仅能用于相等条件分支.通过引入拉格朗日插值法,生成输入处理函数,在保证分支条件混淆安全的前提下,解决了多输入分支条件下通过输入产生秘钥的问题,实现多输入分支下的条件代码加密;把多输入分支下生成唯一秘钥方法应用到等于条件取或分支、大小比较条件分支和复杂条件分支,实现了基于分支条件混淆的代码加密技术从相等条件分支到区间条件分支和复杂条件分支的扩展.

关键词 条件代码加密;路径分支混淆;拉格朗日插值法;多输入分支;代码保护

中图法分类号 TP311

代码加密[1-7]和代码混淆都是重要的代码保护技术,传统的代码加密技术在加密代码的粒度上越来越细,从文件、程序节区粒度到函数[7-8]、基本块粒度,甚至到单个机器指令[5]的粒度;另外在解密函数上也引入了多态[4]的方法,加密代码每次解密执行后的再次加密会通过秘钥变化和解密函数变化来增加逆向分析的难度;但是在隐藏解密秘钥方面还缺少相关研究.分支条件混淆[9-13]是代码混淆[14]研究的一个新方向,通过混淆分支条件,能够隐藏程序执行逻辑、对抗符号执行[11,15-16],特别是在对触发条件的保护上具有较好的效果,能够阻碍生成满足触发条件的程序输入.

虽然分支条件混淆能够保护程序的触发条件,但是无法保护基于触发条件的程序代码.于是 Sharif

收稿日期:2019-06-11;**修回日期**:2019-08-09

等人[9]结合分支条件混淆和代码加密,提出了一种基于分支条件混淆的条件代码加密技术,使用满足分支条件的输入作为代码加解密秘钥对条件代码进行加密,因此在程序内存中不需要存储秘钥,而是在运行时直接获取分支条件的输入作为秘钥;另外由于混淆使得逆向分析者难以获取满足分支条件的输入,也就无法获取加解密秘钥,因此结合分支条件混淆的代码加密实现了解密秘钥的隐藏.然而该分支混淆方法均只能应用于等于条件;另外,受限于解密秘钥唯一性要求,基于分支条件混淆的代码加密技术只能应用于单个等于条件的分支;这些限制极大阻碍了基于分支条件混淆的代码加密技术的应用.

王志等人[11]在 Sharif[9]分支条件混淆方法的基础上,提出了一种基于保留前缀加密的分支混淆技术,把分支条件混淆从等于判断的条件扩展到大小判断的条件.然而在非等条件下,分支条件为真的输入不具备唯一性要求,因此不能使用分支输入作为秘钥对条件代码进行加密.在分支条件取值为真的输入不唯一的条件下,为解决使用分支输入产生唯一性秘钥的要求,在多输入分支条件下,结合分支条件混淆,本文提出了一种基于拉格朗日插值法的秘钥生成技术,实现了解密秘钥和分支条件混淆的结合,隐藏了代码解密秘钥.

本文的主要贡献有 4 个方面:

1) 基于拉格朗日插值法,在分支条件取值为真的输入不唯一的条件下,提出了一种使用分支条件输入生成代码加解密秘钥的方法;

2) 通过对多个等于条件取或形成的复合条件的分支输入进行处理,完成了多输入条件下的秘钥生成,实现条件代码加密,避免了代码复制和复合条件拆分带来的空间损耗;

3) 完成了基于分支混淆的条件代码加密方法从等于条件分支到区间条件分支的扩展;

4) 针对等于条件和区间条件通过逻辑运行形成的复杂条件,实现了分支混淆和复杂条件代码加密的结合,实现了复杂条件分支下的基于分支混淆的代码加密.

1 基础知识

1.1 前缀算法

前缀算法是一种把一个整数区间转变为一个前缀集合的算法,并且保证区间中的每一个整数均在前缀集合能找到一个与之匹配的前缀,反之在集合中没有匹配到前缀的整数一定不属于该区间.并且前缀算法得到的前缀集合元素个数有 2 个特点,其中 n 表示整数的二进制位数:

1) 区间前缀集合元素个数最大值为 $2n-2$;

2) 区间前缀集合元素个数平均值为 $((n-2)\times 2^{2n-1}+(n+1)2^n+1)/(2^{2n-1}+2^{n-1})$,在 32 b 或者 64 b 程序中,该值都接近为 $n-2$.

前缀算法伪代码如算法 1 所示:

算法 1. 一个整数区间到前缀集合的转换算法.

输入:区间起始值的二进制表示 $a_1a_2\cdots a_n$;区间结束值的二进制表示 $b_1b_2\cdots b_n$;

输出:区间的前缀集合 $Prefix$.

```
Prefix Search_Prefix(a1a2…an,b1b2…bn)
{
  for (int k=1;(k≤n) && (ak=bk);
    k++)
    if (k=n+1)
      return {a1a2…an};
    endif
  endfor
  if ((akak+1…an=00…0) && (bkbk+1…bn
    =11…1))
    if (k=1)
      return { * };
    else
      return{a1a2…ak-1};
    endif
  endif
  PrefixA=Search_Prefix(ak+1ak+2…an,11
    …1);
  PrefixB=Search_Prefix(00…0,bk+1bk+2…
    bn)
  return{a1a2…ak-10+PrefixA,a1a2…ak-11
    +PrefixB};
}
```

1.2 保留前缀 Hash 加密

保留前缀加密[17].假设 $a=a_1a_2\cdots a_n$,$b=b_1b_2\cdots b_n$ 为 2 个 n 位整数,如果 $a_1a_2\cdots a_k=b_1b_2\cdots b_k$,其中 $k<n$ 且 $a_{k+1}\neq b_{k+1}$,则称整数 a 和整数 b 是 k 位前缀匹配的.F 是一个定义在 $\{0,1\}^n$ 到 $\{0,1\}^n$ 上的一对一的加密函数,对任意 2 个给定的整数 a,b,如果 a,b 是 k 位前缀匹配时一定有 $F(a)$ 和 $F(b)$ 也是 k 位前缀匹配,则称 F 为保留前缀加密哈数.

标准形式定理(canonical form theorem)[17].假设 f_i 是一个从 $\{0,1\}^i$ 到 $\{0,1\}$ 上的函数,其中 $i=1,2,\cdots,n-1$,并且 f_0 是一个常量函数,F 是一个 $\{0,1\}^n$ 上的函数,其定义对任意给定的整数 $a=a_1a_2\cdots a_n$,令:

$$F(a):=a'_1a'_2\cdots a'_n,\quad a'_i=a_i\oplus f_{i-1}(a_1a_2\cdots a_{i-1}),\tag{1}$$

其中,$\oplus$表示异或运算,$i=1,2,\cdots,n$,f_0 为常数,可以得出结论:1)F 是一个保留前缀加密函数;2)任意保留前缀加密函数必定有 F 的表示形式.定理的证明参见文献[6],此处省略.

保留前缀 Hash 加密[11]是一种把 Hash 函数引入到保留前缀加密计算的加密算法,表示为 $Fh(a):=a'_1a'_2\cdots a'_n$,$a'_i=a_i\oplus f_{i-1}(a_1a_2\cdots a_{i-1})$,其中 $f_{i-1}(a_1a_2\cdots a_{i-1})=T(\mathrm{Hash}(a_1a_2\cdots a_{i-1}))$,其中 T 为取位函数,即选择 Hash 结果的某一位.

1.3　Sharif 提出的条件代码加密方法介绍

Sharif 提出的基于分支条件混淆的条件代码加密如图 1 所示,主要是通过分支输入生成条件代码的加解密秘钥,从而保证程序中不需要存储秘钥,实现了解密秘钥隐藏和分支条件混淆的结合.逆向攻击者解密代码的难度等同于找到使得分支条件取值为真的输入的难度,而分支条件混淆的目的正是阻碍攻击者获取使得分支条件取值为真的输入.

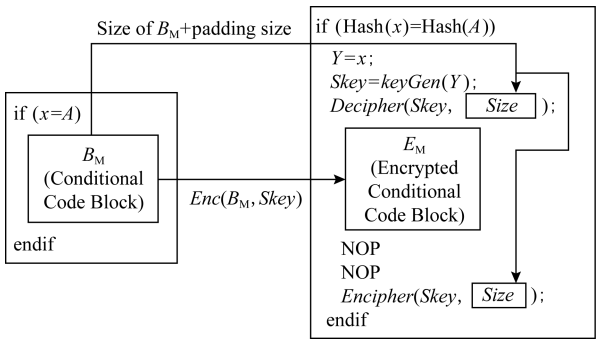


Fig. 1　Example of conditional code encryption

图 1　条件代码加密示意图

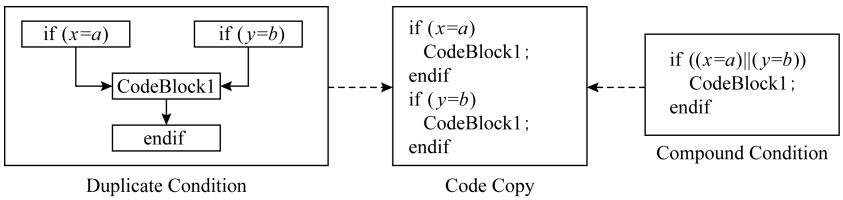


Fig. 2　The process of duplicate condition and compound condition

图 2　重复条件代码和复合条件代码的复制处理

另外,本文把形如 $a\leqslant x\leqslant b$ 的条件称为区间条件,把单个的等于比较条件、单个的区间条件成为基本条件.多个基本条件通过逻辑运算组合成的条件成为复合条件.

在 Sharif 提出的基于分支条件混淆的代码加密方法中,对重复条件和多个等于条件取或的复合条件,需要使用条件代码复制的方法,把复合条件拆分为多个简单条件,然后再对各个简单条件的代码进行加解密,增加了混淆的损耗,同时降低了混淆的隐蔽性.如图 3 所示:

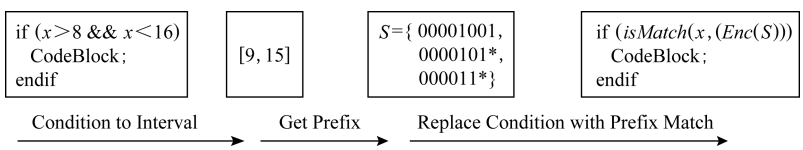


Fig. 3　The example of branch obfuscation based on prefix-preserving algorithm

图 3　基于保留前缀 Hash 加密的分支条件混淆示意图

1.4 基于保留前缀加密的分支混淆方法介绍

Sharif 等人[9]提出的分支条件混淆仅针对等于条件,通过使用 Hash 值比较代替明文比较,使得攻击者难以从 Hash 值比较中恢复出明文比较关系,达到分支条件混淆的目的.如对分支条件 if($x=a$),其中 x 是任意类型的值,实质就是一片内存的内容,通过分支条件混淆后变为 if(Hash(x)=Hash(a)).

王志等人[11]提出了基于保留前缀加密和 Hash 函数的路径分支混淆技术,使用加密前缀匹配替代分支条件,实现分支条件混淆.例如分支条件 if($9\leqslant x\leqslant 15$),首先把条件转换为整数区间[9,15];然后使用前缀算法获取区间的前缀集合 $S=\{1001, 101^{*}, 11^{*}\}$;接下来使用保留前缀 Hash 加密算法对 S 进行加密得到 ES;最后使用加密前缀匹配函数替换分支条件,即 if($9\leqslant x\leqslant 15$)变为 if($isMatch(x,ES)$),其中 $isMatch(x,ES)$如算法 2 所示.

算法 2. isMatch 算法.

输入:整数 x、加密前缀集合 ES;

输出:true 或者 false,true 表示 x 满足分支条件,false 表示 x 不满足分支条件.

```
bool isMatch(x,HS)
{int encPrefixofInput=encInput(x);
 int tmpFlag[32]={0};tmpFlag[0]=1≪
   31;
 for(int i=1;i<32;i++){tmpFlag[i]=
   tmpFlag[i-1]|(1≪(31-i));}
 for(int j=0;j<NumofES;j++)
 {tmpLen=ES[j].prefixLen;
  tmpPrefix = encPrefixofinput&tmpFlag
    [tmpLen];
  if (tmpPrefix = ES[j]. prefix ){return
    true;}
 }
   return fasle;
}
```

2 多输入条件下的秘钥生成

基于分支条件混淆的条件代码加密,由于使用分支条件的输入作为加解密秘钥,导致该加密方法仅能用于单个等于条件的分支.对于具有多个输入值使得分支条件取值为真的分支,由于输入不具有唯一性,导致秘钥不能直接使用分支输入.因此需要对输入进行处理,使得多个输入 $x_1,x_2,x_3,\cdots$ 输入经过处理后得到相同的输出 y,使用 y 作为解密秘钥,从而保证秘钥的唯一性.

2.1 拉格朗日插值法

拉格朗日插值法是 18 世纪法国数学家约瑟夫·拉格朗日发明的一种多项式插值方法.用于寻找二维平面上经过 $n+1$ 个点的 n 次多项式,使得多项式在给定的 $n+1$ 个观测点能取到给定的值.经过 $n+1$ 个点的次数不超过 n 的多项式只有一个,即拉格朗日多项式.本文使用拉格朗日插值保证多输入条件下的输出唯一性.

对给定的 $n+1$ 个点$(x_0,y_0),(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$,首先根据每个点得出 $n+1$ 个拉格朗日基本多项式,其中第 i 个点(x_i,y_i) 的拉格朗日基本多项式为

$$L_i(x)=\prod_{j=0,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}=\frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)},$$

显然该多项式在 x_i 处的取值为 1、在其他点 x_j(其中 $j\neq i$)处的取值为零;令拉格朗日插值多项式 $L(x)=\sum_{i=0}^{n}y_iL_i(x)$,显然 $L(x)$经过 $n+1$ 个给定的点.在本文中,由于需要对 n 个输入 $x_1,x_2,\cdots,x_n$ 计算得到相同的输出 y,因此取 $y_i=y$,其中 $i=1,2,\cdots,n$.

2.2 输入处理多项式生成

一方面,拉格朗日多项式的系数不是整数,即存在除法,因此在使用计算机计算的过程中可能会出现实际值与计算值有出入的问题;另外一方面,拉格朗日多项式具备鲜明的特征,给攻击者从拉格朗日多项式中获取分支输入提供了便利;最后,如果多项式的次数过低,则在某个分支输入已知的情况下会导致其他分支输入泄露,从而会导致分支条件混淆被还原.因此需要对拉格朗日多项式进行变换,以实现多项式系数的整数化、消除拉格朗日多项式的特征和提高多项式次数.

首先,通过对插值多项式 $L(x)$进行去分母和模运算变化,使得多项式的系数整数化,同时消除拉格朗日多项式的特征.即取

$$L'(x)=\Big(\prod_{j=0}^{n}C_j\Big)L(x)=$$

$$\sum_{i=0}^{n}y\Big(\prod_{j=0,j\neq i}^{n}C_j\Big)f_i(x)\equiv\sum_{i=0}^{n}a_ix^i \bmod IMAX,$$

其中，$c_j=\prod_{i=0,i\neq j}^{n}(x_j-x_i)$，乘法和加法均为模 $IMAX$ 运算，$IMAX$ 取值为 n 位整数的最大值加1，比如整数位长为32时，$IMAX$ 取值为 2^{32}.$IMAX$ 的取值是为了方便编程实现，因为在计算机的整数运算过程中，由于整数表示范围有限，导致加减乘除运算在不对溢出位进行处理时都可以看成是模 $IMAX$ 的运算.通过模运算去分母后，多项式在插值点处的取值发生了改变，但是在每个插值点仍然取到同样的唯一值，仅仅是该值从 y 变为

$$y\times(\prod_{j=0}^{n}C_j)\bmod IMAX.$$

其次，如果 $L(x)$ 的次数较低时，通过添加一个在插值点处取值为零的高次多项式的方式增加多项式 $L(x)$ 的次数.令插值点为 $x_0,x_1,\cdots,x_k$，取 m 个随机数 $x_{k+1},\cdots,x_{k+m}$，令 $R(x)=\prod_{i=0}^{k+m}(x-x_i)$，取 $L'(x)=L(x)+R(x)\equiv\sum_{i=0}^{k+m}a_ix^i \bmod IMAX$，则多项式次数变为 $k+m$，如果不需要改变多项式次数，则取 $R(x)=0$.由于 $R(x)$ 在插值点处取值为零，因此变换后的输入预处理函数与次数变换前的多项式在插值点处保持相同的取值.

经过变换后得到的多项式成为输入处理多项式函数，记为 $inputProcess=\sum_{i=0}^{m}a_ix^i$.该函数在输入 x_i 处取值均为 $y'=y\times(\prod_{j=0}^{n}C_j)\bmod IMAX$，使用 y' 生成秘钥，则保证了在多点输入条件下使用输入产生的秘钥具有唯一性.并且 inputProcess 函数是一个普通的多项式函数，不会泄露 $\{x_0,x_1,\cdots,x_n\}$ 和 y' 的值；即使在某一个或者某几个 x_i 暴露的情况下，虽然 y' 已知，多项式函数 inputProcess 可以理解为一元高次方程 $f(x)=\sum_{i=0}^{m}a_ix^i-y'=0$，但是高次方程的求解困难保证了其余解不会泄露，即其余的输入不会因输入处理多项式已知而泄露.

3　基于分支条件混淆的条件代码加密

使用拉格朗日多项式进行多输入分支的条件代码加密密钥生成，解决了大小比较分支的输入不唯一问题，使得条件代码加密和分支混淆紧密结合在一起，对条件代码进行更好的混淆保护.同时通过密钥构造解决 Sharif 条件代码加密方法中的代码复制问题，提升混淆的效率和隐蔽性.

3.1　Sharif 条件代码加密方法的改进

重复条件和复合条件需要使用代码复制变转换为多个简单条件，根本原因在于能够执行到条件代码的分支输入不唯一，因此不能直接使用分支输入作为秘钥.通过对输入进行处理，使得多个输入生成相同的输出，则不需要使用代码复制的手段，减少了加密导致的空间占用.在 Sharif 分支条件混淆的基础上，对多输入进行处理的步骤如图4所示.

1）判断等于条件的输入 x,y 是否为整数，若不是整数，则对输入进行加盐的 Hash 计算，然后取 Hash 值的前4个字节作为拉格朗日插值法的插值点的横坐标；若 x,y 为整数，则直接取 x,y 作为插值点横坐标；然后取一个随机整数作为多项式在插值点 x 和 y 处的取值，生成拉格朗日多项式.

2）使用拉格朗日多项式变换，得到输入预处理函数 inputProcess.

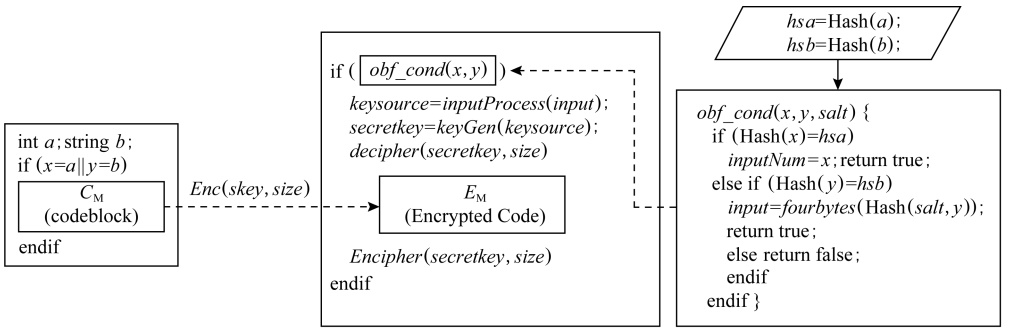
3）在使用 Hash 值比较替代整数比较的过程中，依次判断输入 x,y 是否使得分支条件取值为真，使用第一个使分支条件取值为真的输入作为 inputProcess 函数的参数计算该分支的输出值 $keysource$，使用 $keysource$ 生成解密秘钥，最后进行条件代码的解密和再次加密.

3.2　大小比较分支的条件代码加密

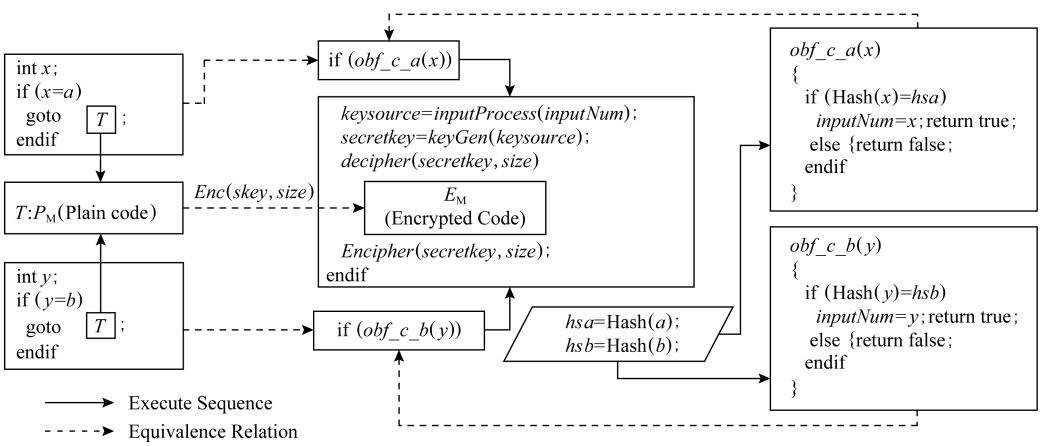
对分支条件 if($a\leqslant x\leqslant b$)，应用基于保留前缀加密 Hash 算法的分支条件混淆技术混淆后，分支条件为 if($isMatch(x,ES)$)，其中 $ES=\{es_1,es_2,\cdots,es_m\}$ 为区间$[a,b]$对应前缀集合应用保留前缀 Hash 加密算法 Fh 计算后得到加密前缀数据.对区间条件在基于保留前缀 Hash 加密的分支混淆后，条件代码加密过程如图5所示，加密步骤如下：

1）对 n 位整数区间$[a,b]$，计算其前缀集合 $S=\{s_1,s_2,\cdots,s_k\}$，对 j 位前缀元素 s_i，取随机数 r_i 的尾部 $n-j$ 比特与 s_i 一起组合成 n 位整数 t_i，使用 $t_1,t_2,\cdots,t_k$ 作为拉格朗日插值点的横坐标，取随机数 tmp 作为所有插值点的纵坐标，计算输入预处理函数 inputProcess.保存随机数数组 $R=\{r_1,r_2,\cdots,r_k\}$.

2）在算法2中，若 x 使得 isMatch 返回 true，则在 isMatch 中记录与 x 相匹配的加密前缀 es_i 的下标 i，获取 es_i 对应前缀的位长，假设为 m；然后取 x 的前 m 位和 r_i 的后 $n-m$ 位组合成 inputProcess 函数的输入 $inputNum$，计算得到分支的输出



(a) Example of compound condition process



(b) Example of duplicate condition process

Fig. 4 The process of compound condition and duplicate condition

图 4 复合条件和重复条件处理示意图

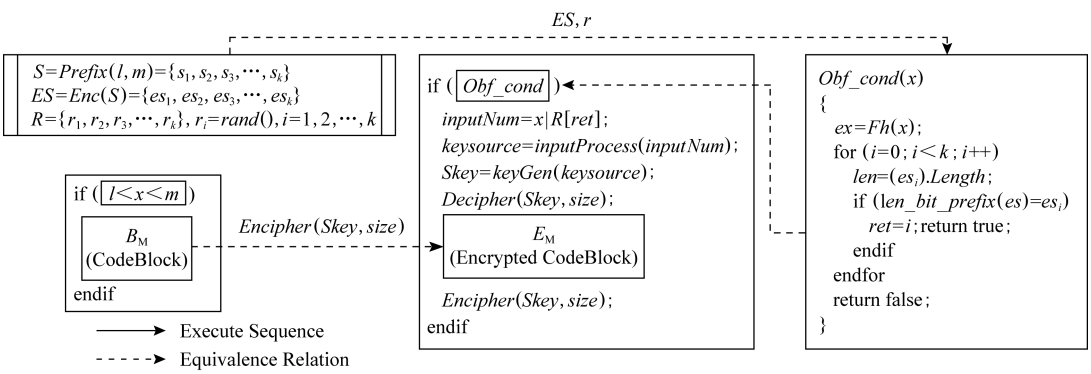


Fig. 5 Example of interval condition process

图 5 区间条件分支处理示意图

keysource,然后使用 *keysource* 生成解密秘钥 *key*,使用 *key* 进行条件代码的解密和再次加密.

3.3 复杂条件分支的条件代码加密

由等于条件和区间条件通过逻辑运算形成的复杂条件中,复杂条件的混淆是通过对构成复杂条件的每个基本条件进行混淆而完成;秘钥生成则是通过对每个基本条件进行单独的输入处理,但是需要保证每个基本条件在其输入使得条件取值为真时的输出都是相等的.为统一表示,令等于条件 if($x=a$)混淆后为 if(*isMatch*(x,*ES*)),但是在等于条件下,isMatch 算法变为{return *memcmp*(Hash(x),*ES*,*hashLen*);},其中 *ES* 为 a 的 Hash 值;令等于条件 if($x=a$)的输入处理函数为 *inputProcess*(x),等于条件下的 inputProcess 函数伪代码如下:

```
inputProcess(x)
{
  if (x is interger){return x;}
  else {int tmp=Hash(x);
```

return $firstFourBytes(tmp)$;}

}

下面介绍复杂分支条件下的代码加密过程,加密示意图如图 6 所示.

1) 根据与运算和或运算的结合律:$(c_{11})\&\&(c_{12}\|c_{13})=(c_{11}\&\&c_{12})\|(c_{11}\&\&c_{13})$,把复杂条件转换为基本条件与运算结合成的复合条件的或运算式,即形如$(c_{11}\&\&c_{12})\|(c_{21}\&\&c_{22})\|(c_{31}\&\&c_{32})$,其中 c_{ij} 表示基本条件;记无或运算相隔的与运算条件为一组,即 $group[0]=\{c_{11},c_{12}\}$,$group[1]=\{c_{21},c_{22}\}$,$group[2]=\{c_{31},c_{32}\}$共 3 组.

2) 对分支条件 if$((c_{11}\&\&c_{12})\|(c_{21}\&\&c_{22})\|(c_{31}\&\&c_{32}))$的每个基本条件进行混淆,混淆后的分支条件为 if$(isMatch(x_{11},ES_{11})\&\&isMatch(x_{12},ES_{12}))\|(isMatch(x_{21},ES_{21})\&\&isMatch(x_{22},ES_{22}))\|(isMatch(x_{31},ES_{31})\&\&isMatch(x_{32},ES_{32}))$.

3) 对每个基本条件 c_{ij} 分别计算输入其输入处理函数,并记为 $inputPrecess[i][j]$;设函数 $inputPrecess[i][j]$的分支输出为为 Y_{ij},$randA$ 为一随机整数数,并令 $R_{ij}=randA-Y_{ij}$,修改 $inputPrecess[i][j]=inputPrecess[i][j]+R_{ij}$,则修改后所有基本条件的预处理函数 $inputProcess[i][j]$,在其输入为使得条件 C_{ij} 取值为真的分支输入值时,预处理函数的取值均为 $randA$.

4) 在混淆后复杂条件执行时,按照顺序计算每组条件与运算的取值,找到第 1 组取值为 true 的条件.假设第 i 组条件与运算的取值为 true,则随机选择第 i 组条件中的任意一个条件代表该复杂分支条件进行分支输出计算,假设选中条件为 C_{ij},则选取 $inputProcess[i][j]$为复杂分支条件的输出计算函数,选取 $x[i][j]$为 $inputProcess[i][j]$的计算参数,计算结果则为 $randA$,使用 $randA$ 计算秘钥,然后对条件代码进行解密和 2 次加密.

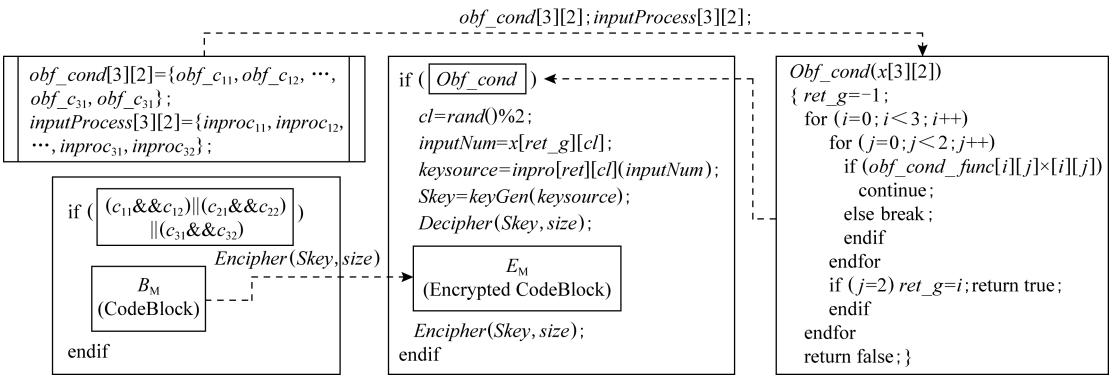


Fig. 6　Example of complicated condition process

图 6　复杂条件分支处理示意图

3.4　多线程处理

如果需要加密的代码是一段多个线程共用的代码,则在解密和 2 次加密的过程中需要进行加解密的操作同步,否则会导致加解密错乱,程序执行出现不可控错误.本文采用临界区对加加解密操作进行同步,解密操作在进入临界区后首先判断加密代码

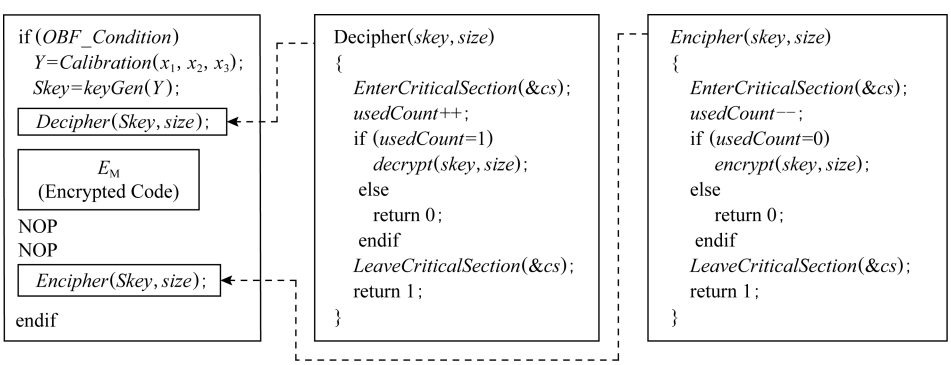


Fig. 7　The encipher and decipher example of multi-thread code

图 7　多线程代码加解密示意图

是否在使用中.如果否,则解密;如果是,则不需要解密、直接返回,退出临界区.

同样,在2次加密的过程中,进入临界区后首先判断需要加密的代码是否在使用中.如果否,则对代码进行2次加密;如果是,则加密函数直接返回,退出临界区.处理过程如图7所示.

4 分析与评估

4.1 条件代码加密的时间和空间消耗

由于本文提出的代码加密是基于分支条件混淆,因此时间和空间的消耗都是基于分支条件已经被混淆的程序来计算的.

首先是加密算法消耗的存储空间,本文采用AES-128加密算法对条件代码进行加密,AES算法源代码通过编译得到的obj文件占用空间为58 807 B;其次inputProcess函数通过把系数和最高次数记为参数后,该函数的算法占用416 B.上述算法是所有的分支条件代码加密所共用的代码,因此随着被选择加密的分支条件越多,每个分支占用的共用空间就越少.另外由于每个分支条件需要存储自己的多项式系数,需要增加加密和解密的外包函数,因此每个分支条件单独占用的空间平均是$30\times4+83+83=283$ B.即对单个区间条件构成的分支,对m个分支条件的代码进行加密,平均每个分支条件的代码加密需要消耗$283+(416+58\,807)/m$字节的空间.如果是m个复杂条件,且每个复杂条件由k个基本条件构成,则每个复杂分支平均消耗$120\times k+(416+58\,807)/m$字节的空间用于代码加密.

其次是时间消耗,时间消耗主要在于AES128一次加密和解密的时间,即平均每个分支多消耗0.003 3 ms(由10 000次AES加密和解密消耗32.8 ms时间计算得到单次加解密时间).

Table 1 Consumption Data of Time and Space

表1 加密消耗的时间和空间数据

Properties Types	Branch Shaped with Three Equal Conditions	Branch Shaped with Single Size Comparison Condition	Branch Shaped with Six Basic Conditions
Number of Branches	m	m	m
Per-branch Average Space Consumption/B	$59\,223/m$	$59\,223/m+283$	$59\,223/m+120\times6$
Per-branch Average Time Consumption/ms	0.003 3	0.003 3	0.003 3

4.2 与其他加密方法的比较

Sharif第1次提出了基于分支条件混淆的代码解密方法,通过分支条件混淆实现了秘钥隐藏;通过逆向分析获取加解密秘钥的的难度,等同于找到使得被混淆分支条件取值为真的程序输入的难度.即相比其他的加密方法,基于分支条件混淆的代码加密方法能更好的对抗解密秘钥获取攻击.

在基于分支条件混淆的代码加密研究方面,除了Sharif在基于等于条件混淆的代码加密研究外,还没有相关的研究.与Sharif的加密方法相比,本文提出的方法在2个方面具有优势:

1) 把Sharif的方法从等于条件扩展到了区间条件和复杂条件,具有更好的通用性;

2) 对多个等于条件取或的分支,本文提出的方法不需要代码复制,从而节省了代码占用的空间.

综上所述,本文提出的代码加密方法在时间和空间消耗上是可接受的;由于解密秘钥通过分支混淆隐藏,使得加密后代码能较好对抗逆向分析,因此

Table 2 Comparison of Different Methods

表2 加密方法优势对比

Method of Encryption	Time Consumption	Space Consumption	Key Concealed	Application Range
Sharif Method	Time of Single Encryption and Decryption	Cipher Algorithm	Concealed	Conditional Code Shaped with Equal Condition
Our Method	Time of Single Encryption and Decryption	Cipher Algorithm and Polynomial	Concealed	Conditional Code Shaped with Equal Condition and Size Comparison Condition
Methods That Are Not Based on Branch Obfuscation	Time of Single Encryption and Decryption	Cipher Algorithm	Not Concealed	Almost All Code

该加密方法能够实际使用,且具有比较好的代码保护功能.

5 结　论

基于分支条件混淆的条件代码加密,通过满足分支条件的输入产生秘钥,同时利用分支混淆隐藏分支条件,从而隔离秘钥和程序,极大地提高了解密难度.由于秘钥生成和分支条件紧密联系在一起,只有在加密的条件代码所在分支因正确的输入而得到执行时,通过分支输入可以计算出秘钥,实现解密,否则就会导致解密错误,产生不可预测的错误导致程序运行终止.虽然基于分支条件混淆的条件代码解密具有较好的抗程序分析效果,但是当前研究仅能在等于条件分支上实现该类加密方式,极大地限制了该方法的使用.本文通过构建输入预处理函数,对具有多个正确输入的分支进行处理,完成了多输入分支下利用分支输入产生具有唯一性秘钥的方法.利用该方法,对多个等于条件取或的分支和重复条件代码分支的条件代码加密进行了优化,减少了加密对空间的消耗;区间条件分支使用基于保留前缀加密和 Hash 函数的分支混淆方法进行分支条件混淆,本文使用整数区间对应的前缀集合生成输入预处理函数,实现了区间条件分支下基于分支输入产生唯一性秘钥的方法,从而对区间条件分支的条件代码进行加密.最后对等于和区间条件复合而成的复杂条件分支进行处理,对该分支同样实现了基于分支输入产生唯一性秘钥的方法,进而对条件代码进行加密.通过以上处理完成了基于分支条件混淆的条件代码加密从等于条件分支到区间条件分支和复杂条件分支的扩展,提高了代码保护的范围,增强了代码安全性.

参　考　文　献

[1] Schrittwieser S, Katzenbeisser S, Kieseberg P, et al. Covert computation: Hiding code in code for obfuscation purposes [C] //Proc of the 8th ACM SIGSAC Symp on Information, Computer and Communications Security. New York: ACM, 2013: 529-534

[2] Balachandran V, Emmanuel S. Potent and stealthy control flow obfuscation by stack based self-modifying code [J]. IEEE Transactions on Information Forensics and Security, 2013, 8(4): 669-681

[3] Cappaert J, Preneel B, Anckaert B, et al. Towards tamper resistant code encryption: Practice and experience [C] //Proc of the 4th Int Conf on Information Security Practice and Experience. Berlin: Springer, 2008: 86-100

[4] Wu Zhenyu, Steven Gianvecchio, Xie Mengjun, et al. Mimimorphism: A new approach to binary code obfuscation [C] //Proc of the 17th ACM Conf on Computer and Communications Security (CCS'12). New York: ACM, 2012: 536-546

[5] Vrba Z. Cryptexec: Next-generation runtime binary encryption using on-demand function extraction [EB/OL]. [2019-06-11]. http://www.phrack.org/issues/63/13.html #article

[6] Vrba Ž, Halvorsen P, Griwodz C. Program obfuscation by strong cryptography [C] //Proc of 2010 Int Conf on Availability, Reliability and Security. Piscataway, NJ: IEEE, 2010: 242-247

[7] Cappaert J, Kisserli N, Schellekens D, et al. Self-encrypting code to protect against analysis and tampering [C/OL] //Proc of the 1st Benelux Workshop on Information and Sytem Security (WISSec 2006). 2006 [2019-06-10]. https://www.esat.kuleuven.be/cosic/publications/article-811.pdf

[8] Balachandran V, Keong N W, Emmanuel S. Function level control flow obfuscation for software security [C] //Proc of the 8th Int Conf on Complex, Intelligent and Software Intensive Systems. Piscataway, NJ: IEEE, 2014: 133-140

[9] Sharif M, Lanzi A, Giffin J, et al. Impeding malware analysis using conditional code obfuscation [C] //Proc of the Network and Distributed System Security Symp. Rosten, VA: Internet Society, 2008: 321-333

[10] Jia Chunfu, Wang Zhi, Liu Xin, et al. Branch obfuscation: An efficient binary code obfuscation to impede symbolic execution [J]. Journal of Computer Research and Development, 2011, 48(11): 2111-2119 (in Chinese)
(贾春福, 王志, 刘昕, 等. 路径模糊: 一种有效抵抗符号执行的二进制混淆技术[J]. 计算机研究与发展, 2011, 48(11): 2111-2119)

[11] Wang Zhi, Jia Chunfu, Liu Weijie, et al. Branch obfuscation to combat symbolic execution [J]. Acta Electronica Sinica, 2015, 43(5): 870-878 (in Chinese)
(王志, 贾春福, 刘伟杰, 等. 一种抵抗符号执行的路径分支混淆技术[J]. 电子学报, 2015, 43(5): 870-878)

[12] Lin Hong, Zhang Xiaohua, Ma Yong, et al. Branch obfuscation using binary code side effects [C] //Proc of the 1st Int Conf on Computer, Networks and Communication Engineering (ICCNCE 2013). Amsterdam, Netherlands: Atlantis Press, 2013

[13] Banescu S, Collberg C, Pretschner A. Predicting the resilience of obfuscated code against symbolic execution attacks via machine learning [C] //Proc of the 26th USENIX Security Symposium (USENIX Security 17). Berkeley, CA: USENIX Association, 2017: 661-678

[14] Avidan E, Feitelson D G. From obfuscation to comprehension [C] //Proc of the 23rd Int Conf on Program Comprehension. Piscataway, NJ: IEEE, 2015: 178-181
[15] Yadegari B, Debray S. Symbolic execution of obfuscated code [C] //Proc of the 22nd ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2015: 732-744
[16] Banescu S, Collberg C, Ganesh V, et al. Code obfuscation against symbolic execution attacks [C] //Proc of the 32nd Annual Conf on Computer Security Applications. New York: ACM, 2016: 189-200
[17] Fan Jinliang, Xu Jun, Amma M H, et al. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme [J]. Computer Networks, 2004, 46(2): 253-272

Geng Pu, born in 1982. PhD candidate. His main research interests include code obfuscation and information security.

Zhu Yuefei, born in 1962. PhD, professor and PhD supervisor. His main research interests include cyberspace security and elliptic curve cyphers.

2017 年《计算机研究与发展》高被引论文 TOP10

排名	论文信息
1	施巍松, 孙辉, 曹杰, 张权, 刘伟. **边缘计算:万物互联时代新型计算模型**[J]. 计算机研究与发展, 2017, 54(5): 907-924 Shi Weisong, Sun Hui, Cao Jie, Zhang Quan, Liu Wei. Edge Computing—An Emerging Computing Model for the Internet of Everything Era [J]. Journal of Computer Research and Development, 2017, 54(5): 907-924
2	黎建辉, 沈志宏, 孟小峰. **科学大数据管理:概念、技术与系统**[J]. 计算机研究与发展, 2017, 54(2): 235-247 Li Jianhui, Shen Zhihong, Meng Xiaofeng. Scientific Big Data Management: Concepts, Technologies and System [J]. Journal of Computer Research and Development, 2017, 54(2): 235-247
3	张玉清, 周威, 彭安妮. **物联网安全综述**[J]. 计算机研究与发展, 2017, 54(10): 2130-2143 Zhang Yuqing, Zhou Wei, Peng Anni. Survey of Internet of Things Security [J]. Journal of Computer Research and Development, 2017, 54(10): 2130-2143
4	祝烈煌, 高峰, 沈蒙, 李艳东, 郑宝昆, 毛洪亮, 吴震. **区块链隐私保护研究综述**[J]. 计算机研究与发展, 2017, 54(10): 2170-2186 Zhu Liehuang, Gao Feng, Shen Meng, Li Yandong, Zheng Baokun, Mao Hongliang, Wu Zhen. Survey on Privacy Preserving Techniques for Blockchain Technology [J]. Journal of Computer Research and Development, 2017, 54(10): 2170-2186
5	李敏, 孟祥茂. **动态蛋白质网络的构建、分析及应用研究进展**[J]. 计算机研究与发展, 2017, 54(6): 1281-1299 Li Min, Meng Xiangmao. The Construction, Analysis, and Applications of Dynamic Protein-Protein Interaction Networks [J]. Journal of Computer Research and Development, 2017, 54(6): 1281-1299
6	王继业, 高灵超, 董爱强, 郭少勇, 陈晖, 魏欣. **基于区块链的数据安全共享网络体系研究**[J]. 计算机研究与发展, 2017, 54(4): 742-749 Wang Jiye, Gao Lingchao, Dong Aiqiang, Guo Shaoyong, Chen Hui, Wei Xin. Block Chain Based Data Security Sharing Network Architecture Research [J]. Journal of Computer Research and Development, 2017, 54(4): 742-749
7	陈龙, 管子玉, 何金红, 彭进业. **情感分类研究进展**[J]. 计算机研究与发展, 2017, 54(6): 1150-1170 Chen Long, Guan Ziyu, He Jinhong, Peng Jinye. A Survey on Sentiment Classification [J]. Journal of Computer Research and Development, 2017, 54(6): 1150-1170
8	高玉凯, 王新华, 郭磊, 陈竹敏. **一种基于协同矩阵分解的用户冷启动推荐算法**[J]. 计算机研究与发展, 2017, 54(8): 1813-1823 Gao Yukai, Wang Xinhua, Guo Lei, Chen Zhumin. Learning to Recommend with Collaborative Matrix Factorization for New Users [J]. Journal of Computer Research and Development, 2017, 54(8): 1813-1823
9	傅艺绮, 董威, 尹良泽, 杜雨晴. **基于组合机器学习算法的软件缺陷预测模型**[J]. 计算机研究与发展, 2017, 54(3): 633-641 Fu Yiqi, Dong Wei, Yin Liangze, Du Yuqing. Software Defect Prediction Model Based on the Combination of Machine Learning Algorithms [J]. Journal of Computer Research and Development, 2017, 54(3): 633-641
10	刘洋. **神经机器翻译前沿进展**[J]. 计算机研究与发展, 2017, 54(6): 1144-1149 Liu Yang. Recent Advances in Neural Machine Translation [J]. Journal of Computer Research and Development, 2017, 54(6): 1144-1149

数据来源:CSCD、中国知网;统计日期:2018 年 12 月 3 日