软件安全与漏洞分析

4.2 代码混淆的基本理论

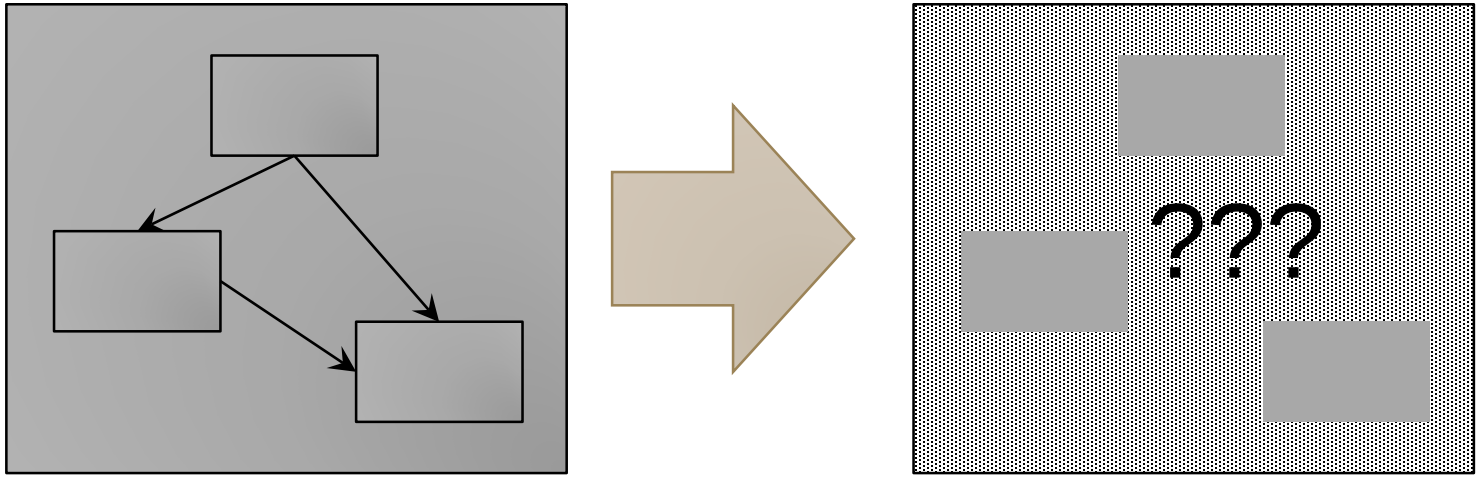
Previously in Software Security

- 软件自我保护技术概述
  - Man-At-The-End攻击模型
  - 代码混淆
  - 软件防篡改
  - 软件水印

代码混淆的基本理论

- 本节主题1 – 代码混淆
  - 代码混淆的可能性
  - 现有的主要代码混淆方法
- 本节主题2 – 软件防篡改

回顾：代码混淆的基本概念

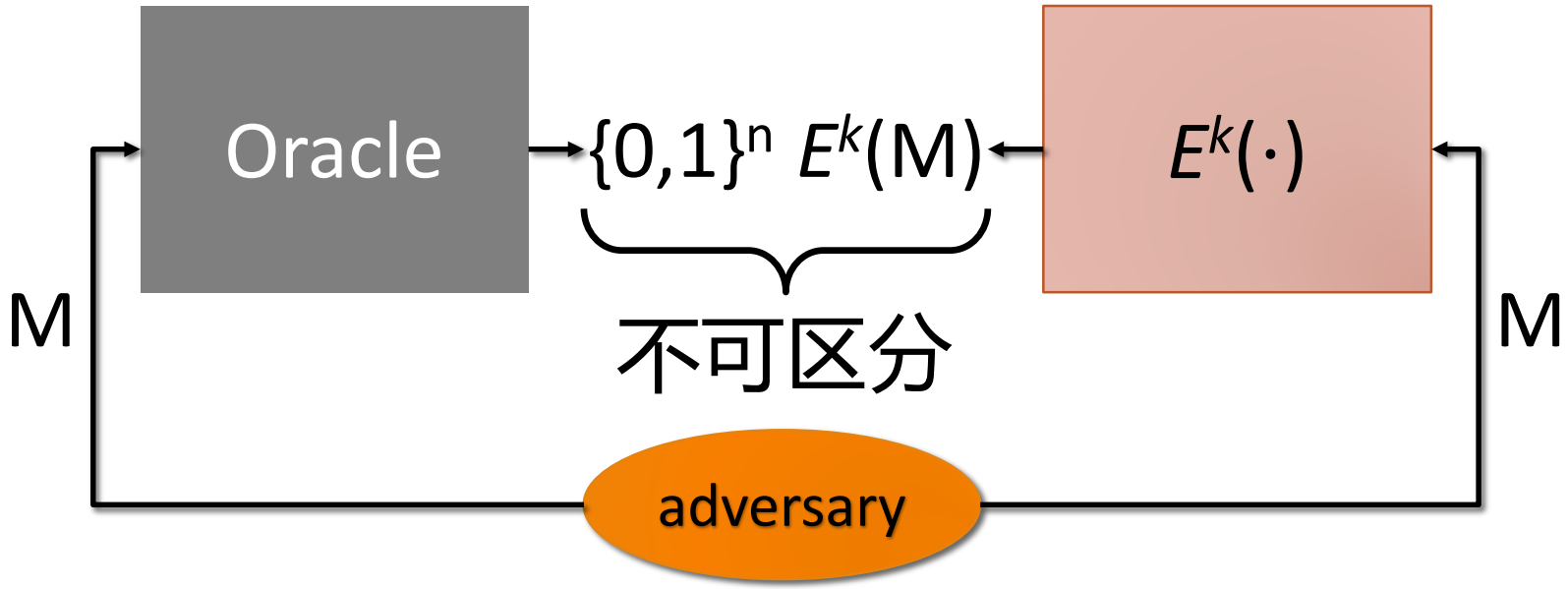


回顾：代码混淆的基本概念

- 目的 -- 阻止对程序的非授权逆向工程
- 实现思想 -- 使得对程序的分析变得困难
- 要求 -- 混淆不允许令程序的执行语义产生变化

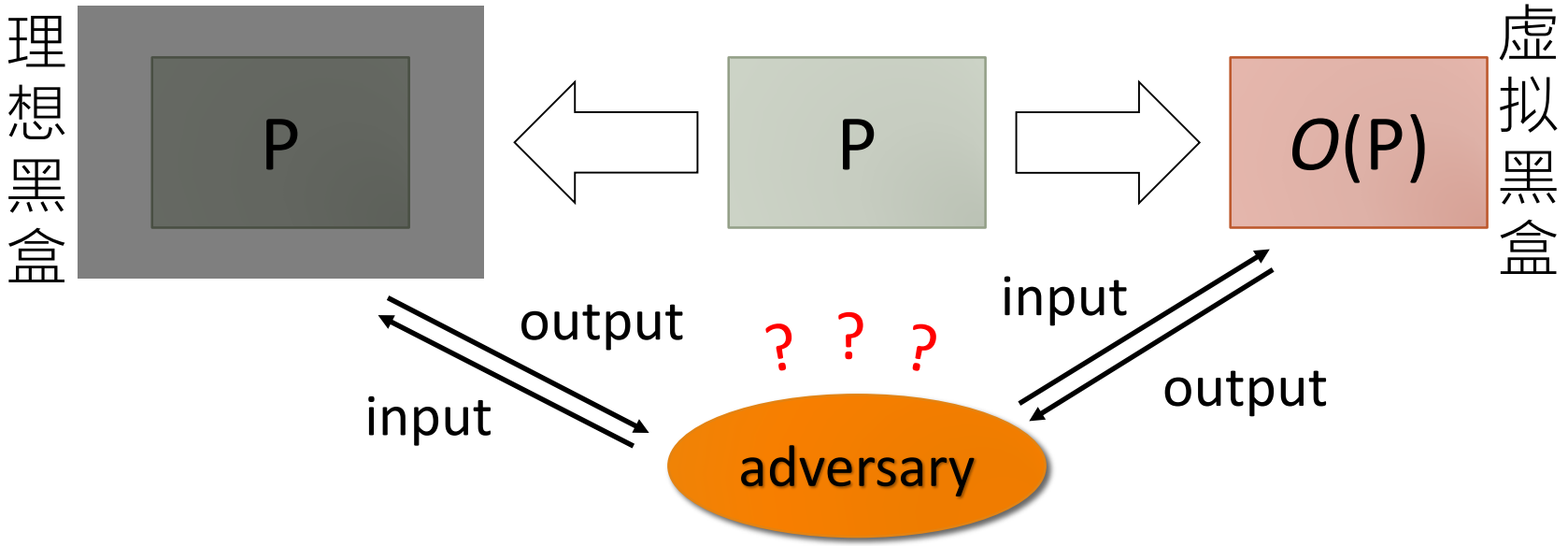
代码混淆的可能性

- 借鉴密码学算法的安全模型
- 目标：**可证明安全性**



代码混淆的可能性

- 借鉴密码学算法的安全模型
- 目标：**可证明安全性**



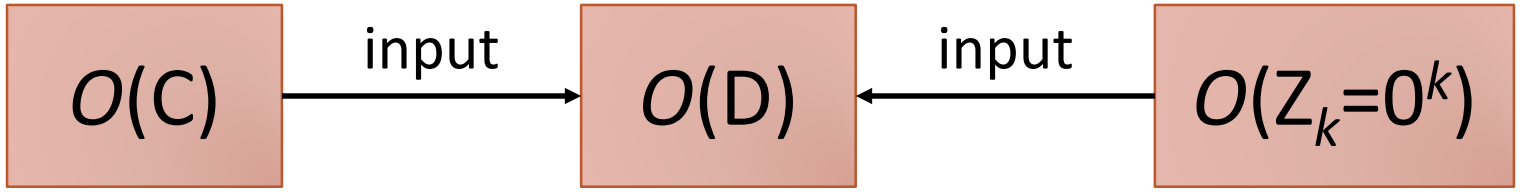
代码混淆的可能性

- 然而，“虚拟黑盒”式的混淆器真的可以实现吗？
- 研究结论：**不可能**
- 核心因素：**程序执行与Oracle访问有着本质区别**
  - 程序是对函数的简明描述
  - Oracle访问只给出函数的输入-输出映射关系
  - 函数的功能往往无法通过Oracle访问的方式予以精确学习

代码混淆的可能性

$$\alpha, \beta \in \{0,1\}^k$$

$$C_{\alpha,\beta}(x) \stackrel{\text{def}}{=} \begin{cases} \beta & x = \alpha \\ 0^k & \text{otherwise} \end{cases} \qquad D_{\alpha,\beta}(C) \stackrel{\text{def}}{=} \begin{cases} 1 & C(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}$$

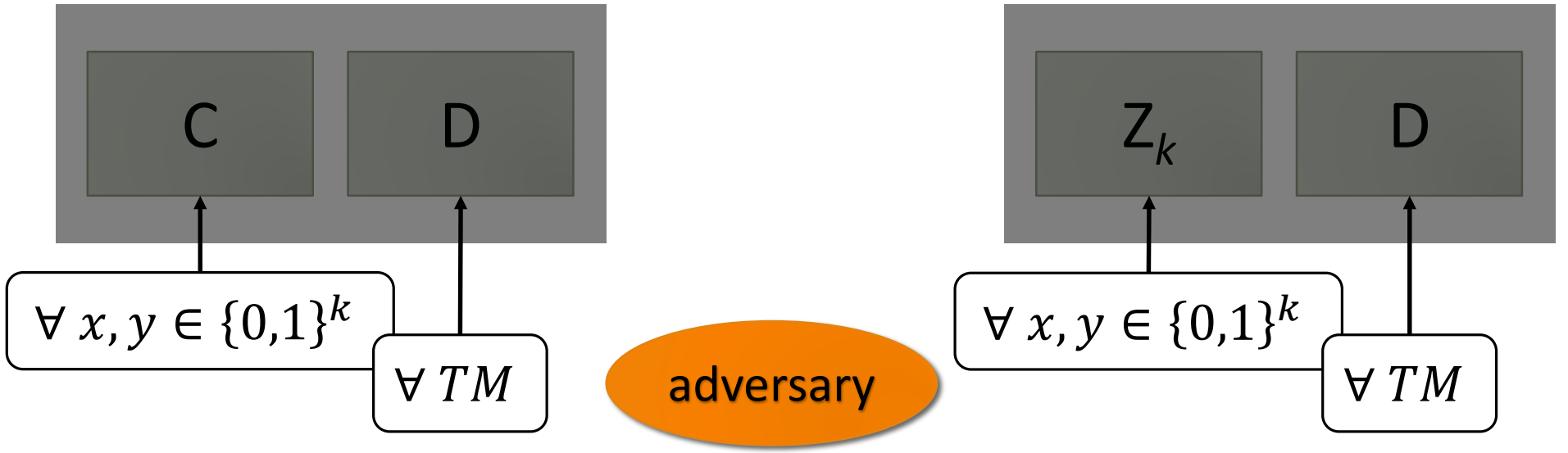


$$\Pr\left[A(\mathcal{O}(C_{\alpha,\beta}), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 1$$

adversary

$$\Pr\left[A(\mathcal{O}(Z_k), \mathcal{O}(D_{\alpha,\beta})) = 1\right] = 0$$

代码混淆的可能性



$$C_{\alpha,\beta}(x) \stackrel{\text{def}}{=} \begin{cases} \beta & x = \alpha \\ 0^k & \text{otherwise} \end{cases}$$

$$D_{\alpha,\beta}(C) \stackrel{\text{def}}{=} \begin{cases} 1 & C(\alpha) = \beta \\ 0 & \text{otherwise} \end{cases}$$

$$\left| \Pr\left[S^{C_{\alpha,\beta}, D_{\alpha,\beta}}(1^k) = 1 \right] - \Pr\left[S^{Z_k, D_{\alpha,\beta}}(1^k) = 1 \right] \right| \leq 2^{-\Omega(k)}$$

代码混淆的可能性

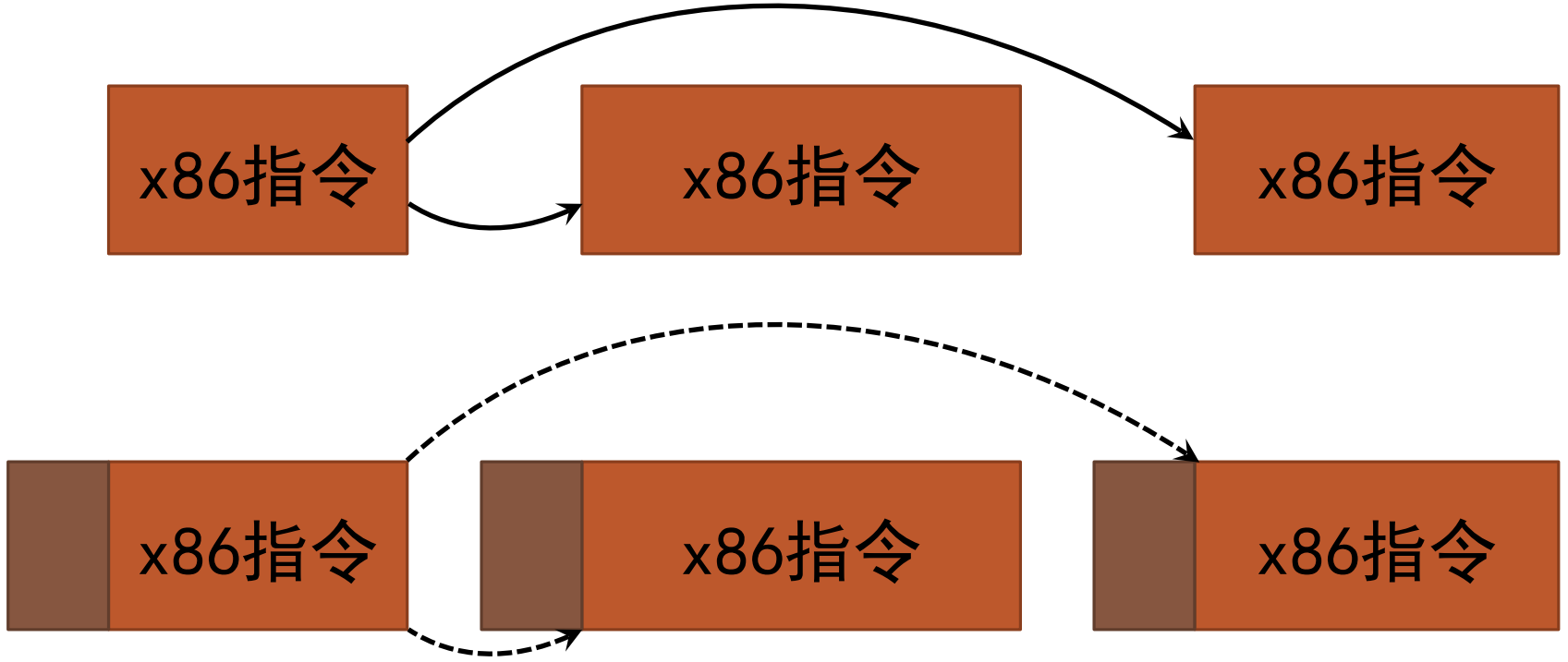
- 参考文献： Barak B, Goldreich O, Impagliazzo R, et al. On the (Im)possibility of Obfuscating Programs[M]// Advances in Cryptology — CRYPTO 2001. Springer Berlin Heidelberg, 2001:1-18.
- 扩展阅读：学习论文中各个不可能性定理的详细证明过程

代码混淆的可能性

- 代码混淆的实际能力
  - 做不到：让程序的执行逻辑变得不可知
  - 做得到：使程序的执行逻辑变得难以理解

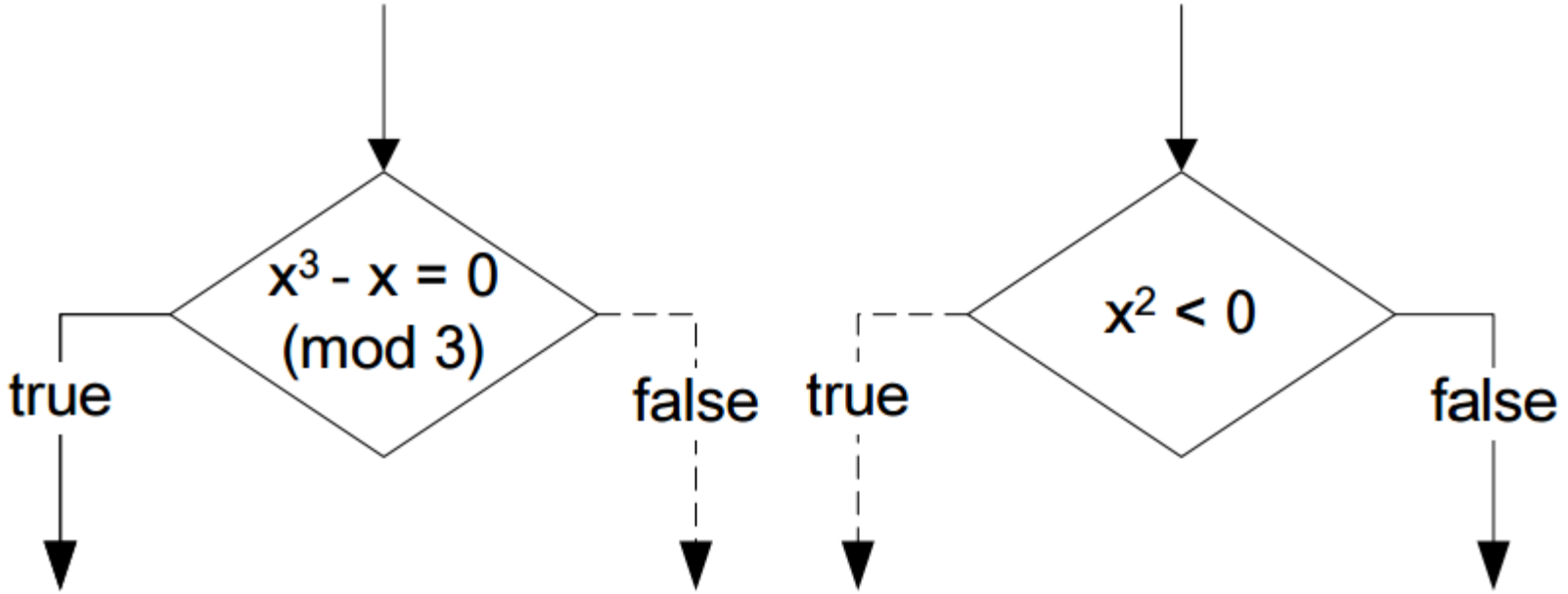
代码混淆的方法

- 简易的代码混淆举例



代码混淆的方法

- 不透明谓词（opaque predicate）



代码混淆的方法

- 不透明谓词（opaque predicate）
  - 较强的构造方法：基于3SAT问题

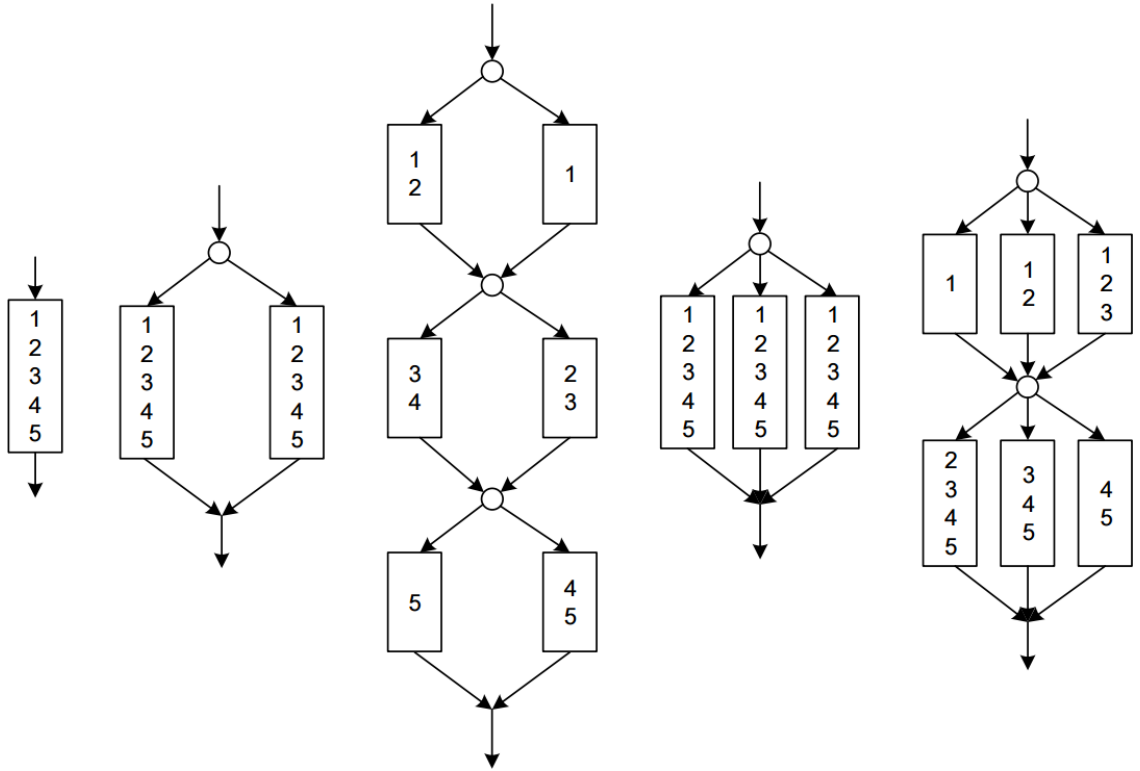
问题描述：给定$C = C_1 \wedge C_2 \wedge \ldots \wedge C_n$，其中任意$C_i$为3变量的析取范式，则是否存在一组真值赋值，使得C的取值为真?

举例： $C = C_1 \wedge C_2 \wedge C_3$，其中
$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$,
$C_2 = \neg x_1 \vee x_2 \vee \neg x_3$,
$C_3 = \neg x_1 \vee \neg x_2 \vee x_3$.

$\forall x, y \in \mathbb{Z}$	$7y^2 - 1 \neq x^2$
$\forall x \in \mathbb{Z}$	$2 \lvert \lfloor \frac{x^2}{2} \rfloor$
$\forall x \in \mathbb{Z}$	$2 \lvert x(x+1)$
$\forall x \in \mathbb{Z}$	$x^2 \geq 0$
$\forall x \in \mathbb{Z}$	$3 \lvert x(x+1)(x+2)$
$\forall x \in \mathbb{Z}$	$7 \nmid x^2 + 1$
$\forall x \in \mathbb{Z}$	$81 \nmid x^2 + x + 7$
$\forall x \in \mathbb{Z}$	$19 \nmid 4x^2 + 4$
$\forall x \in \mathbb{Z}$	$4 \lvert x^2(x+1)(x+1)$

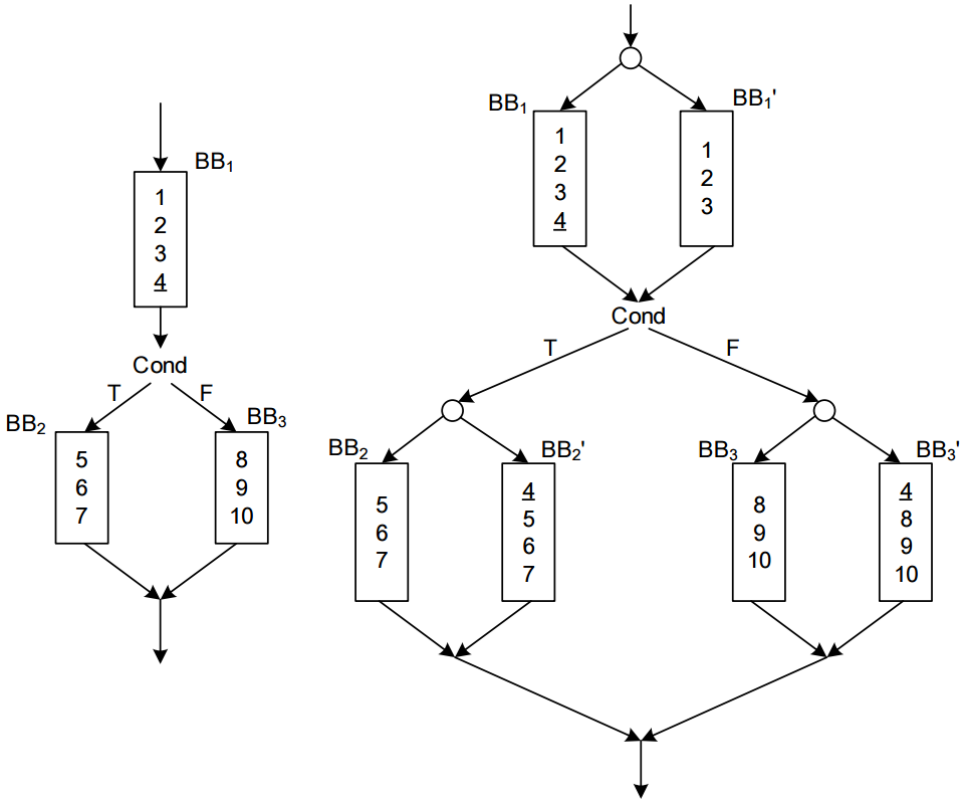
代码混淆的方法

- 不透明谓词（opaque predicate）



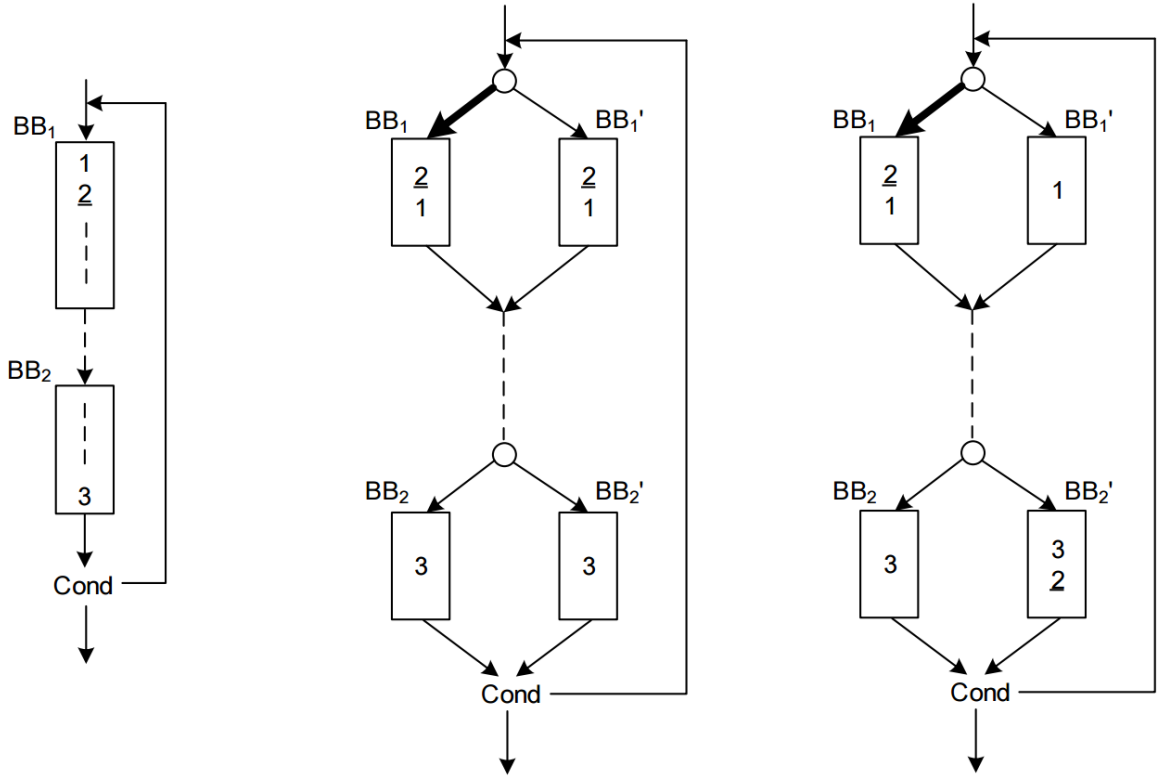
代码混淆的方法

- 不透明谓词（opaque predicate）



代码混淆的方法

- 不透明谓词（opaque predicate）



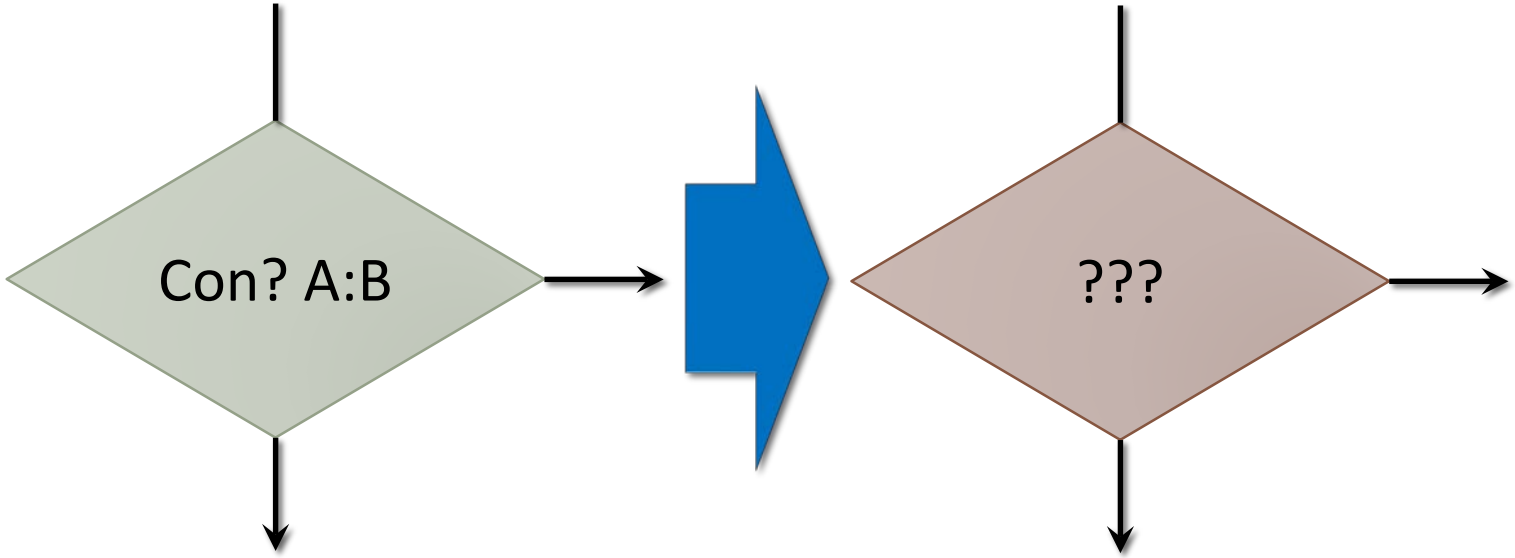
代码混淆的方法

- 问题：证明显示不透明谓词在计算上是不安全的
- 参考文献：

  - Xu D, Ming J, Wu D. Generalized Dynamic Opaque Predicates: A New Control Flow Obfuscation Method[C]//International Conference on Information Security. Springer International Publishing, 2016: 323-342.
  - Sheridan B, Sherr M. On Manufacturing Resilient Opaque Constructs Against Static Analysis[C]//European Symposium on Research in Computer Security. Springer International Publishing, 2016: 39-58.

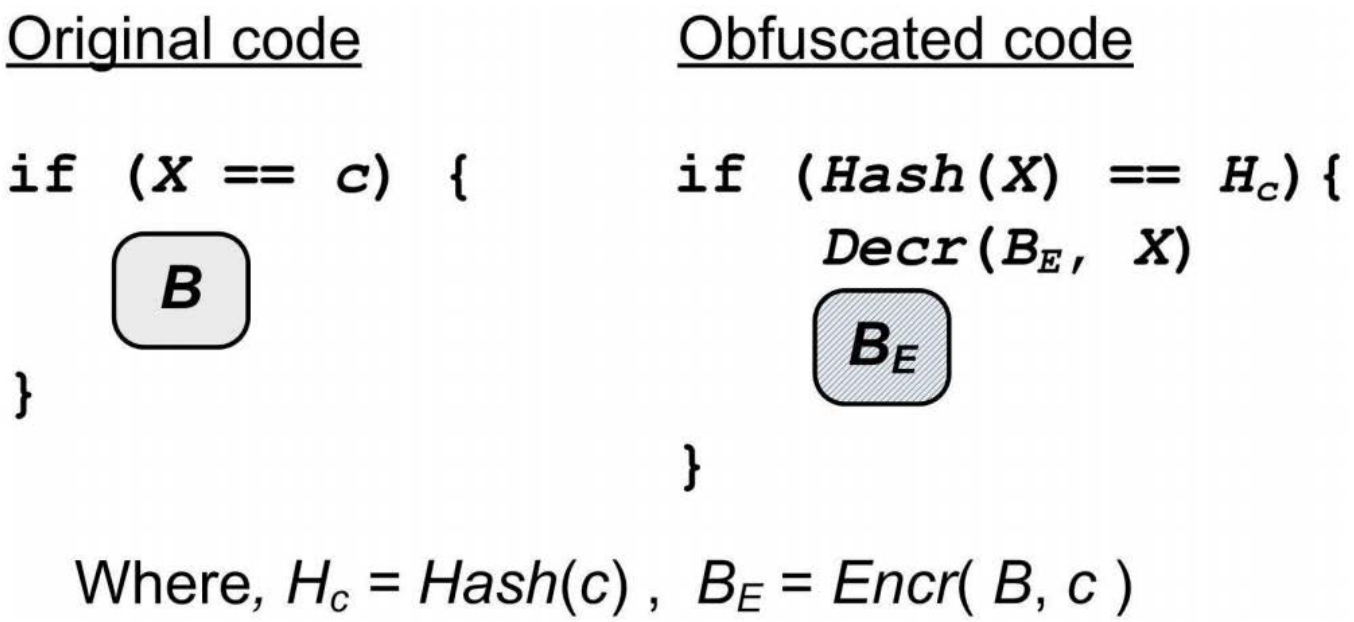
代码混淆的方法

- 条件分支混淆



代码混淆的方法

- 条件分支混淆
  - 方法1：单向函数



代码混淆的方法

- 条件分支混淆
  - 方法2：利用分析技术弱点

```
if (x == 30) {
  do_m();
}
```

$$a_i = \begin{cases} n & \text{for } i = 0 \\ f(a_{i-1}) & \text{for } i > 0 \end{cases}$$

$$\text{where } f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2 \\ 3n+1 & \text{if } n \equiv 1 \pmod 2 \end{cases}$$

```
// x: user input
// y: spurious variable

y = x + 1000;
while(y > 1){
  if (y % 2 == 1){
    y = 3 * y + 1;
  }
  else{
    y = y / 2;
  }
  if ((x - y > 28) &&
    (x + y < 32)){ // cond.
    do_m();
    break;
  }
}
```

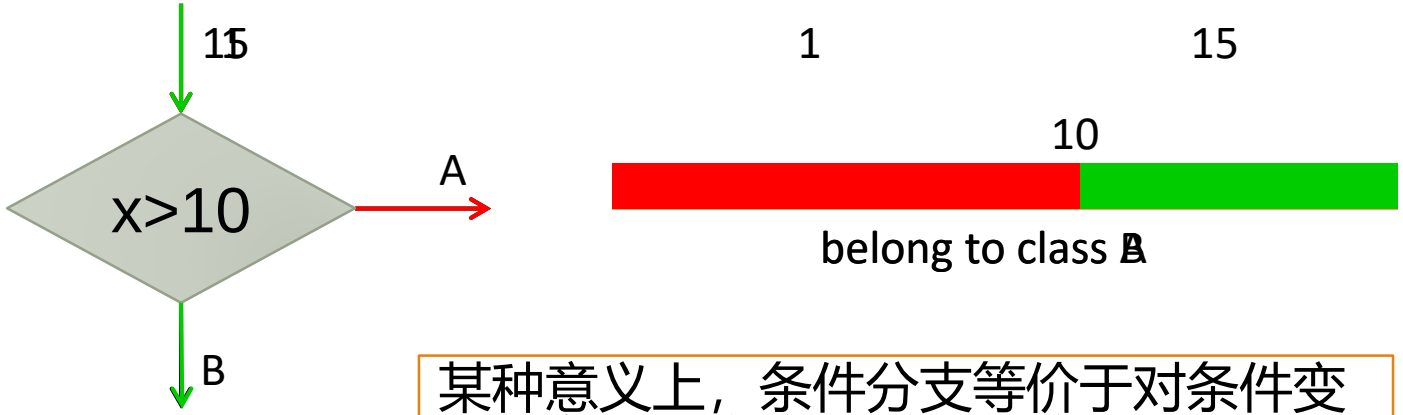
代码混淆的方法

- 条件分支混淆
  - 方法2：

y = 1030			y = 1158			
iteration	y	y % 2 == 1	iteration	y	y % 2 == 1	STP result
1	1030	false	1	1158	false	
2	515	true	2	579	true	
3	1546	false	3	1738	false	
...	...	...	...	...	...	
9	145	true	9	163	true	
10	436	false	10	490	false	
11	218	false	11	245	true	true
...	...	...	...	...	...	...
123	4	false	30	4	false	false
124	2	false	31	2	false	false
125	1	true	32	1	true	false

代码混淆的方法

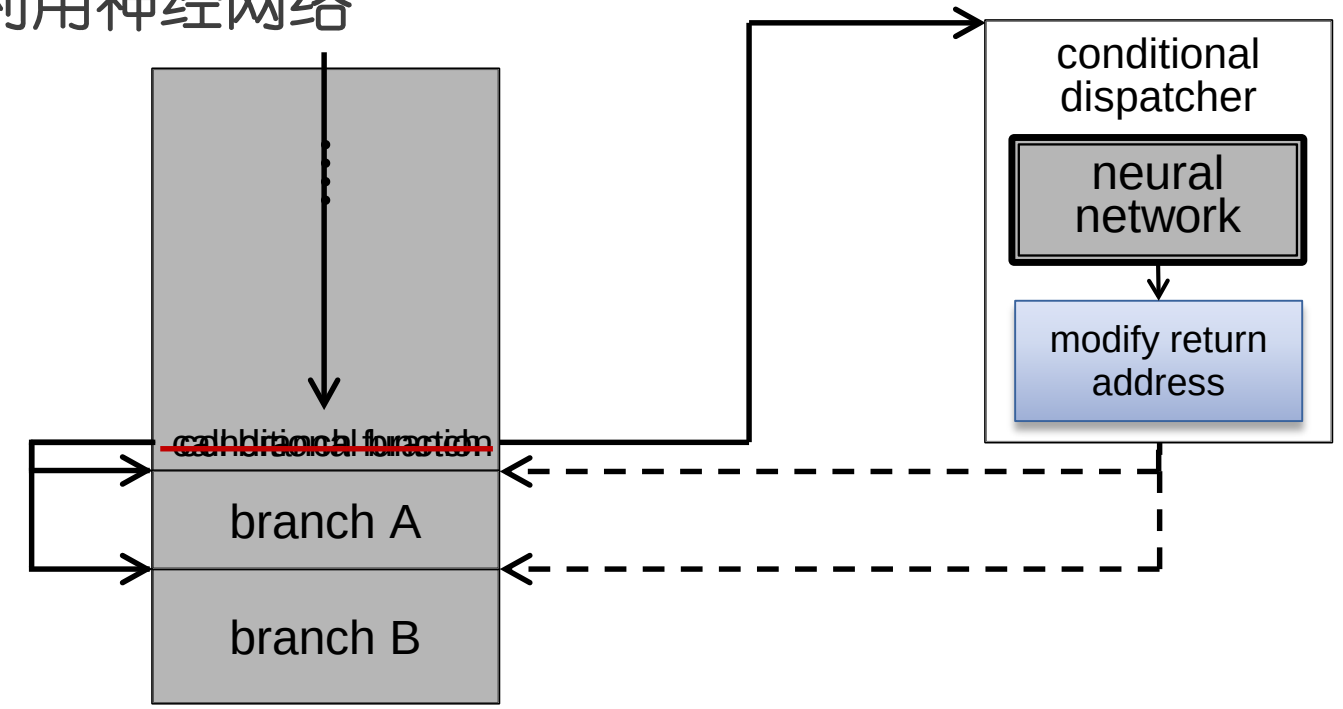
- 条件分支混淆
  - 方法3：利用神经网络



某种意义上，条件分支等价于对条件变量所有取值之集合的一个分类

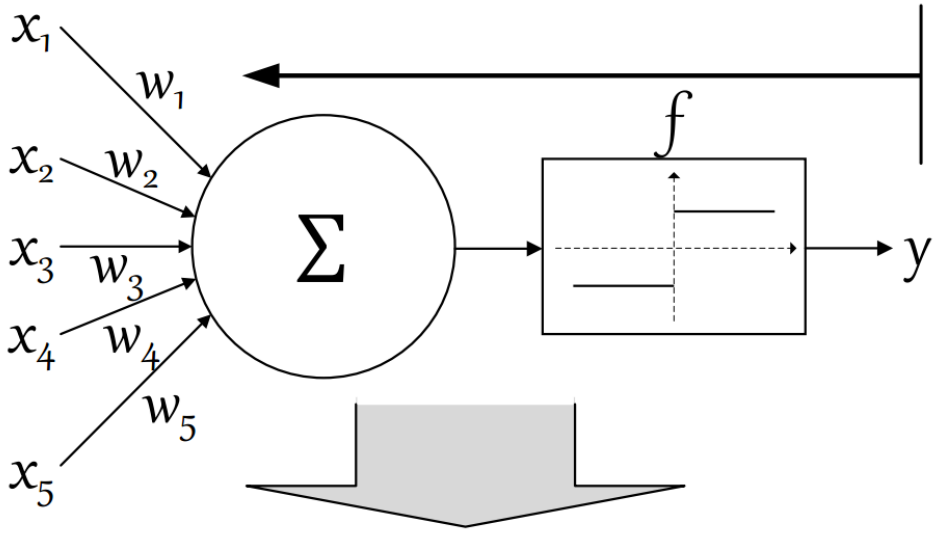
代码混淆的方法

- 条件分支混淆
  - 方法3：利用神经网络



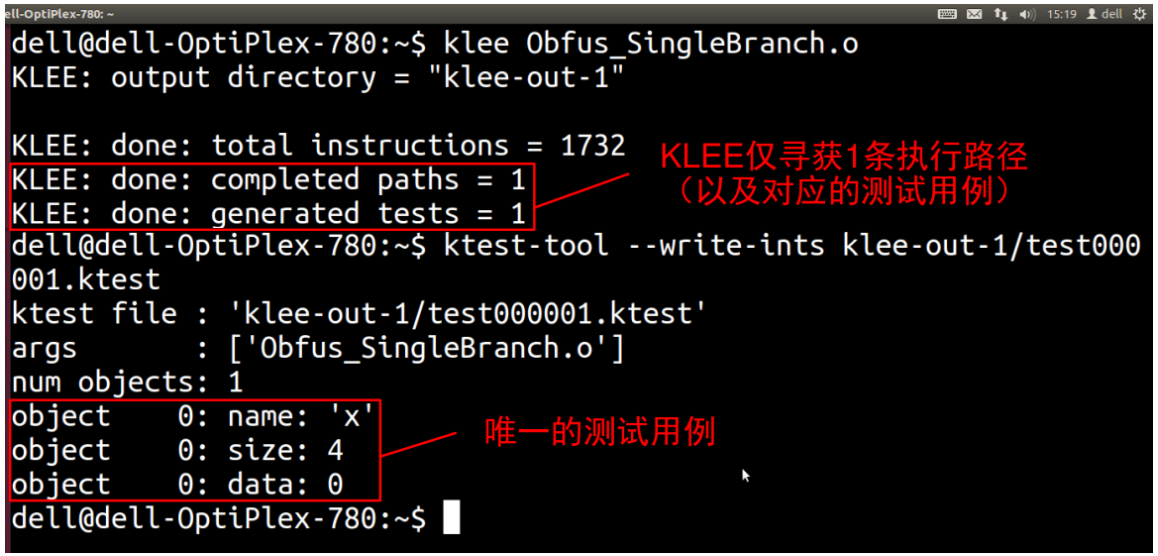
代码混淆的方法

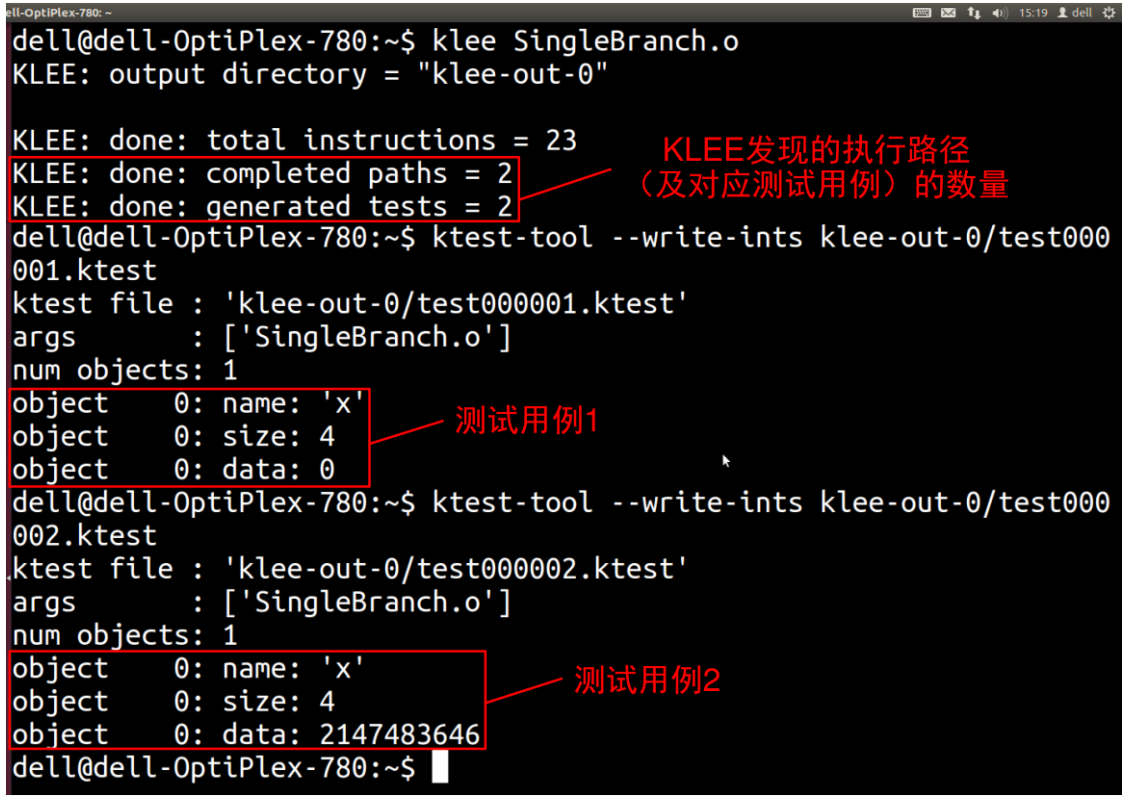
- 条件分支混淆
  - 方法3：利用神经网络



代码混淆的方法

- 条件分支混淆
  - 方法3：利用神经网络





代码混淆的方法

- 条件分支混淆
  - 方法3：利用神经网络

statement (unequal)		>16	>6	>4	>2	≤ 29	≤ 11	≤ 20	≤ 13
input value		-16							
verification result	original	-16							
	obfuscated	NA							
# of constraints	original	733							
	obfuscated	13751	18759	28931	31584	12809	18759	28931	31658
statement (equal)		$=16$	$=6$	$=4$	$=2$	$=29$	$=11$	$=20$	$=13$
input value		16	6	4	2	29	11	20	13
verification result	original	16	6	4	2	29	11	20	13
	obfuscated	NA							
# of constraints	original	2566							
	obfuscated	12809	18759	28931	31584	12809	18759	28931	31584

代码混淆的方法

- 参考文献：
  - Sharif M I, Lanzi A, Giffin J T, et al. Impeding Malware Analysis Using Conditional Code Obfuscation[C]//NDSS. 2008.
  - Wang Z, Ming J, Jia C, et al. Linear obfuscation to combat symbolic execution[C]//European Symposium on Research in Computer Security. Springer Berlin Heidelberg, 2011: 210-226.
  - Ma H, Ma X, Liu W, et al. Control flow obfuscation using neural network to fight concolic testing[C]//International Conference on Security and Privacy in Communication Systems. Springer International Publishing, 2014: 287-304.

软件防篡改

- 目标：
  - 使得软件的内部逻辑无法被篡改
  - 当篡改发生时，完成自我诊断/修复
- 意义：**阻止**对软件的**破解**

软件防篡改

Denuvo的最新相关信息

正版游戏守护神——Denuvo是如何被攻破的? 百家号 9小时前

对游戏厂商而言,Denuvo保证了游戏在黄金期的销售;对Denuvo公司来说,他们凭借技术获得了庞大收益;对黑客而言,挑战Denuvo则是难得的锻炼技术的机会。...

冒险游戏《RIME》正式宣布移除Denuvo DRM 网易游戏 13小时前

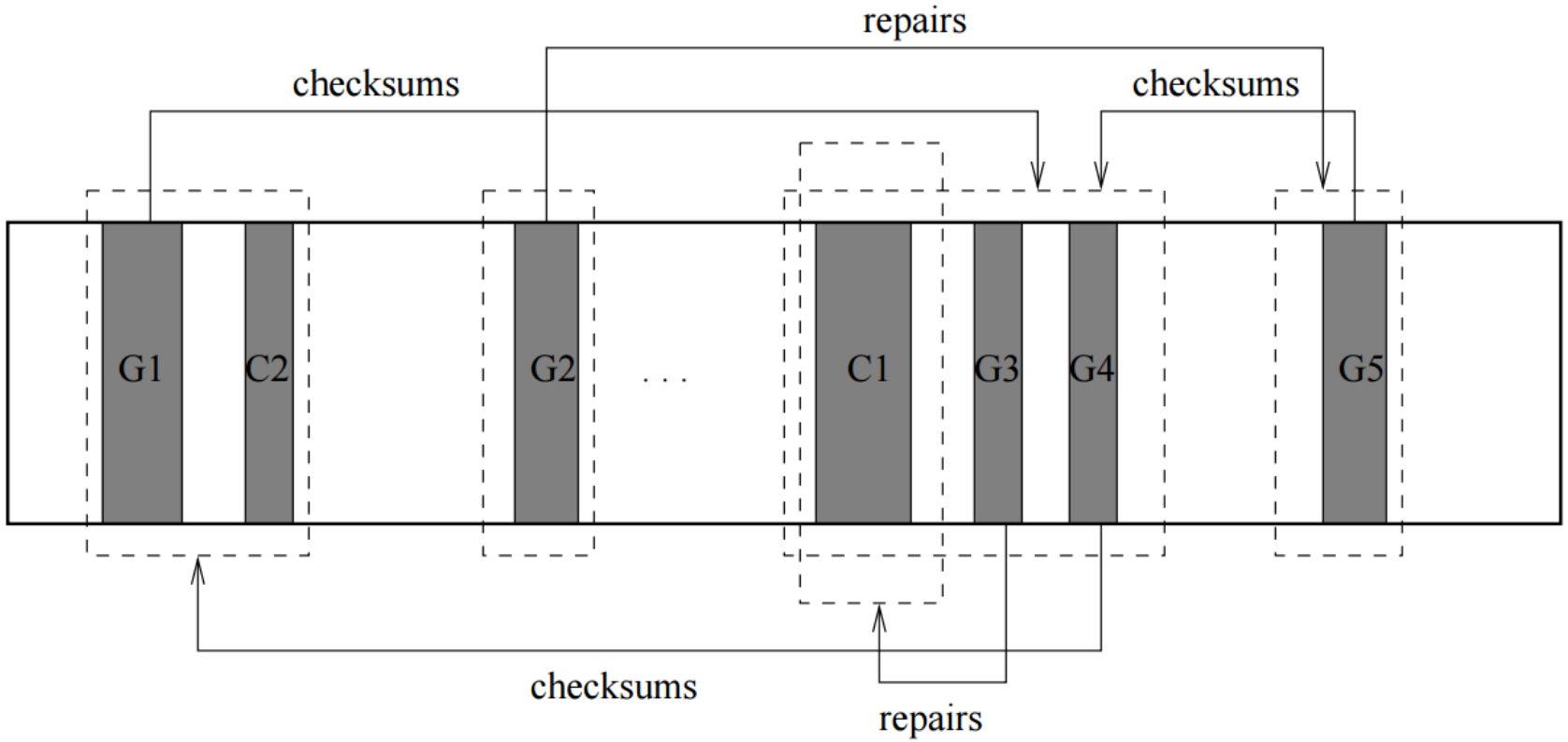
《RIME》破解版仅5天就发布 Denuvo是游戏卡顿元凶 着迷网 2天前

《掠食》被破解,BT加密Denuvo已沦陷 百家号 5月19日

《掠食》上市十天便遭破解 BT加密Denuvo已沦陷 太平洋游戏网 5月17日

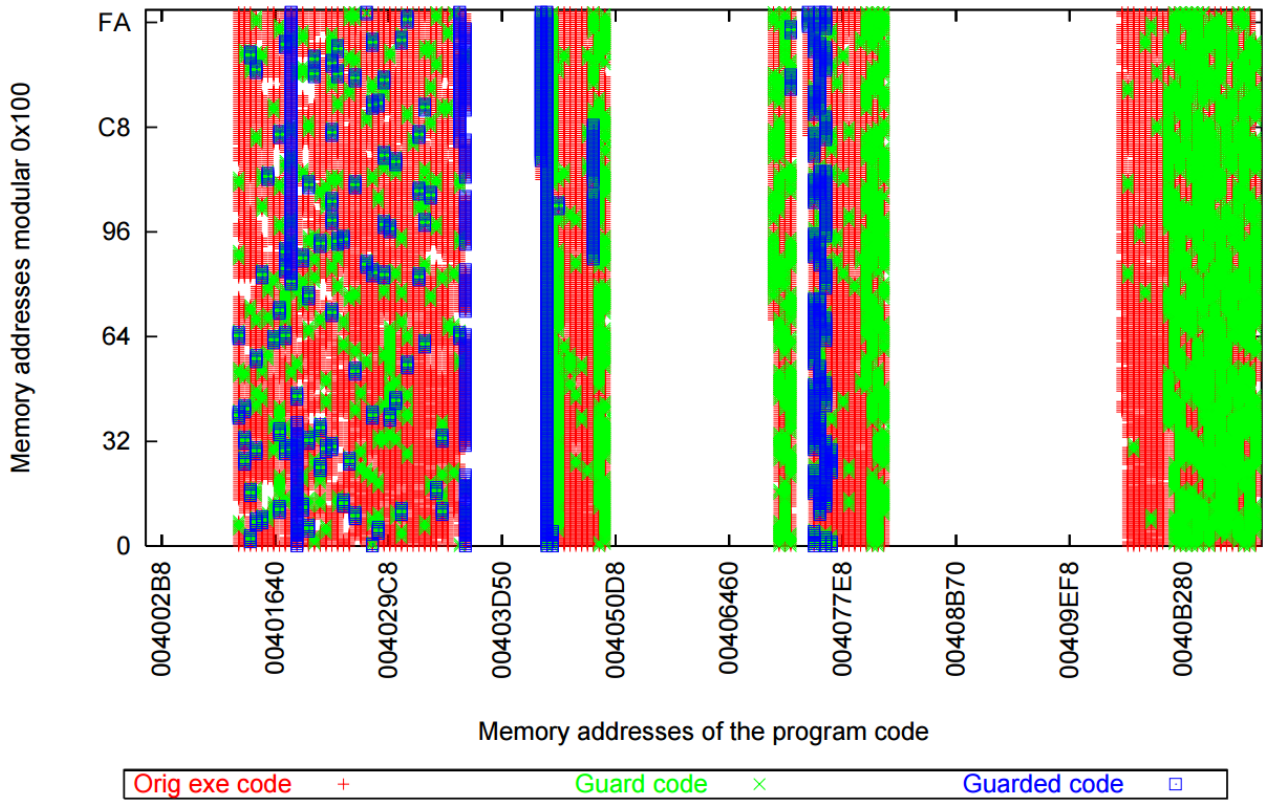
软件防篡改

- 方法1：内省自检（introspection）



软件防篡改

- 方法1：内省自检（introspection）



软件防篡改

- 方法2：Oblivious Hashing

十个人站成一列纵队
从十顶黄帽子和九顶蓝帽子中，取出十顶分别给每个人戴上。

站在最后的人说：“我虽然看见了你们每个人头上的帽子，但仍然不知道自己头上的帽子的颜色。你们呢？”

第九个人说：“我也不知道。”
第八个人说：“我也不知道。”
第七个、第六个……直到第二个人，依次都说不知道自己头上帽子的颜色。

最后，第一个人说：“我知道自己头上帽子的颜色了。”

软件防篡改

- 方法2：Oblivious Hashing

```
int x = 123;

if (GetUserInput() > 10)
{
    x = x + 1;
}
else
{
    printf("Hello\n");
}
```

```
INITIALIZE_HASH(hash1);

int x = 123;
UPDATE_HASH(hash1, x);

if (GetUserInput() > 10)
{
    UPDATE_HASH(hash1, BRANCH_ID_1);
    x = x + 1;
    UPDATE_HASH(hash1, x);
}
else
{
    UPDATE_HASH(hash1, BRANCH_ID_2);
    printf("Hello\n");
}

VERIFY_HASH(hash1);
```

代码混淆的方法

- 参考文献：
  - Chang H, Atallah M J. Protecting software code by guards[C]//ACM Workshop on Digital Rights Management. Springer Berlin Heidelberg, 2001: 160-175.
  - Jacob M, Jakubowski M H, Venkatesan R. Towards integral binary execution: Implementing oblivious hashing using overlapped instruction encodings[C]//Proceedings of the 9th workshop on Multimedia & security. ACM, 2007: 129-140.

What’s next?

- 软件水印
- 代码胎记