软件安全与漏洞分析

4.3 软件水印与软件胎记

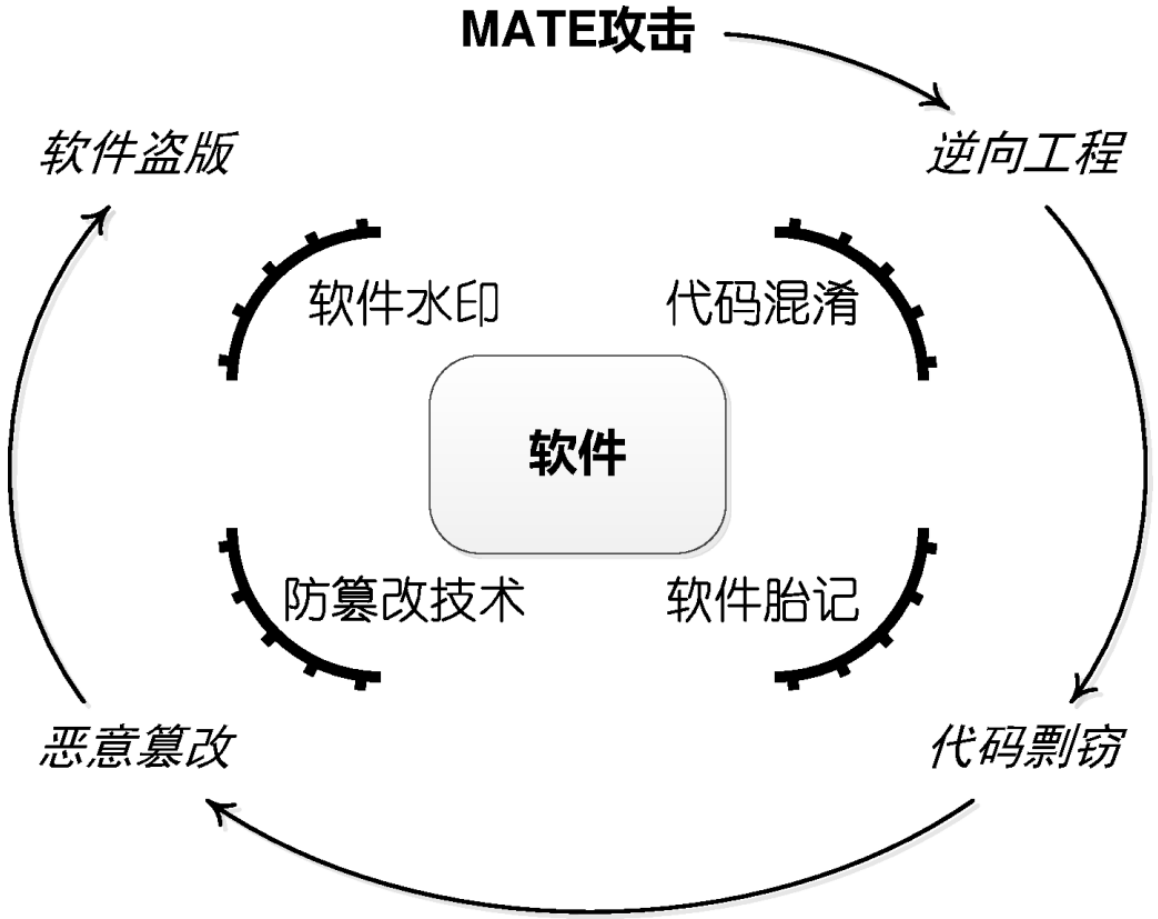
Previously in Software Security

- 代码混淆
  - 代码混淆的可能性
  - 现有的主要代码混淆方法
- 软件防篡改

代码混淆的基本理论

- 本节主题1 – 软件胎记
  - 方法与应用
- 本节主题2 – 软件水印
  - 一些关键性的构造思路
  - 几种主要的设计

回顾：软件自我保护技术的应用场景



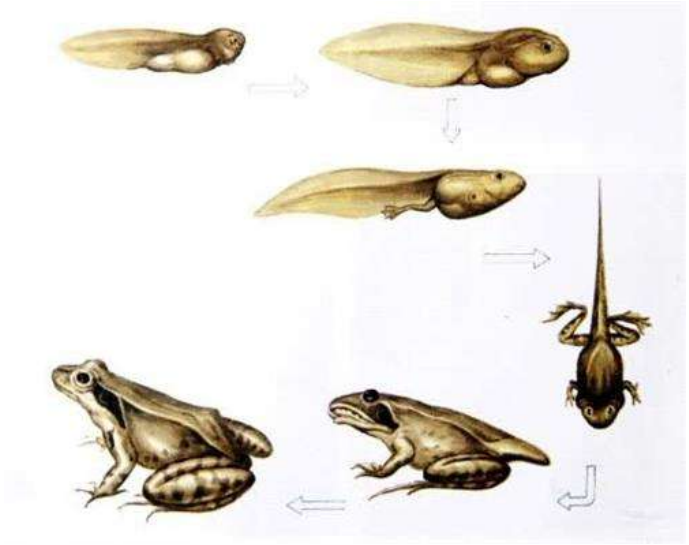
软件胎记

- “胎记”的实际含义
  - 一类对象的本质特征
  - 与生俱来
  - 独一无二

乔治•居维叶：利用一颗牙齿，就可以恢复一个动物的全貌，乃至更多信息

一些题外话

- 软件保护方法中的“仿生学”
  - 代码多态（polymorphism）
  - 代码变态（metamorphism）
  - 代码拟态（mimimorphism）



- 参考文献：
  - Wu Z, Gianvecchio S, Xie M, et al. Mimimorphism: A new approach to binary code obfuscation[C]//Proceedings of the 17th ACM conference on Computer and communications security. ACM, 2010: 536-546.

软件胎记

- 软件胎记的广义安全价值
  - 反代码剽窃（未经授权使用共享库，或违反协议使用开源代码）
  - 检测恶意代码（特别是经过各种伪装保护的）
  - 检测移动app的重包装
- 构造形式分类
  - 静态/动态（取决于胎记所依赖的特征类型）

软件胎记

- 静态构造举例：基于JAVA的栈行为模式

Category	Opcode	*act*
NORMAL	dastore lastore	−4
	aastore bastore castore dcmpg dcmpl fastore iastore lcmp sastore	−3
	dadd ddiv dmul drem dreturn dstore dstore_*n* dsub ladd laload land ldiv lmul lor lrem lreturn lstore lstore_*n* lsub lushr lxor pop2	−2
	aaload areturn astore astore_*n* athrow baload caload d2f d2i fadd faload fcmpg fcmpl fdiv fmul frem freturn fstore fstore_*n* fsub iadd iaload iand idiv imul ior irem ireturn ishl ishr istore istore_*n* isub iushr ixor l2f l2i lshl lshr monitorenter monitorexit multianewarray pop saload	−1
	anewarray arraylength checkcast d2l daload dneg f2i fneg i2b i2c i2f i2s iinc ineg instanceof l2d lneg newarray nop ret return swap wide	0
	aconst_null aload aload_*n* bipush dup dup_x*n* f2d f2l fconst_*n* fload fload_*n* i2d i2l iconst_*n* iload iload_*n* ldc ldc_w new sipush	+1
	dconst_*n* dload dload_*n* dup2 dup2_x*n* lconst_*n* ldc2_w lload lload_*n*	+2
BRANCH	if_acmpeq if_acmpne if_icmpeq if_icmpne if_icmplt if_icmpge if_icmpgt if_icmple	−2
	ifeq ifne iflt ifge ifgt ifle ifnonnull ifnull lookupswitch tableswitch	−1
	goto goto_w	0
	jsr jsr_w	+1
OBJECT	getfield getstatic putfield putstatic	See (1)
INVOKE	invokeinterface invokespecial invokestatic invokevirtual	See (2)

软件胎记

- 静态构造举例：基于JAVA的栈行为模式

$$act(x) = \begin{cases} -1 - sz(\texttt{fv}) & \text{if } x = \texttt{putfield}, \\ 0 - sz(\texttt{fv}) & \text{if } x = \texttt{putstatic}, \\ -1 + sz(\texttt{fv}) & \text{if } x = \texttt{getfield}, \\ 0 + sz(\texttt{fv}) & \text{if } x = \texttt{getstatic}, \end{cases} \tag{1}$$

$$act(x) = \begin{cases} sz(r) - \sum_i sz(ar_i) & \text{if } x = \texttt{invokestatic}, \\ sz(r) - \sum_i sz(ar_i) - 1 & \text{otherwise.} \end{cases} \tag{2}$$

软件胎记

- 静态构造举例：基于JAVA的栈行为模式

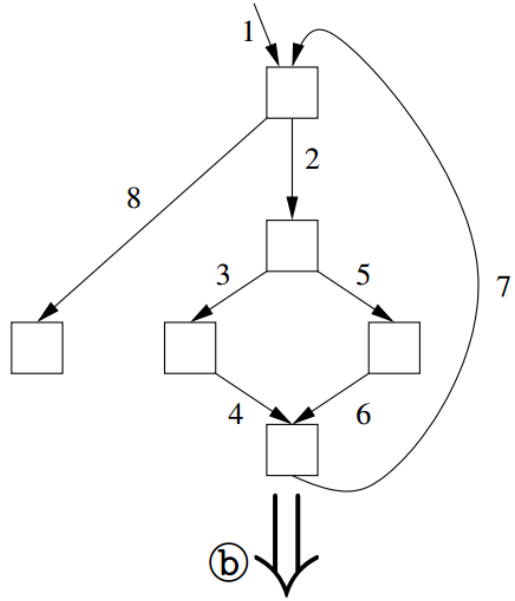
Stack Pattern a			Stack Pattern b		
Bytecode	Stack	w	Bytecode	Stack	w
`iload_0`	[•]	7	`iload_0`	[•]	7
`iload_0`	[••]	7	`iconst_1`	[••]	1
`iconst_1`	[•••]	1	`isub`	[•]	4
`isub`	[••]	4	`invokestatic`	[•]	1
`invokestatic`	[••]	1	`iload_0`	[••]	7
`imul`	[•]	6	`iconst_2`	[•••]	3
`ireturn`	[]	2	`isub`	[••]	4
			`invokestatic`	[••]	1
			`iadd`	[•]	3
			`ireturn`	[]	2

软件胎记

- 动态构造1：基于执行路径

```
int a;
for(int i=0; i < 5; i++){
    if(i < 3)
        a = 1;
    else
        a = 2;
}
```

ⓐ



ⓑ

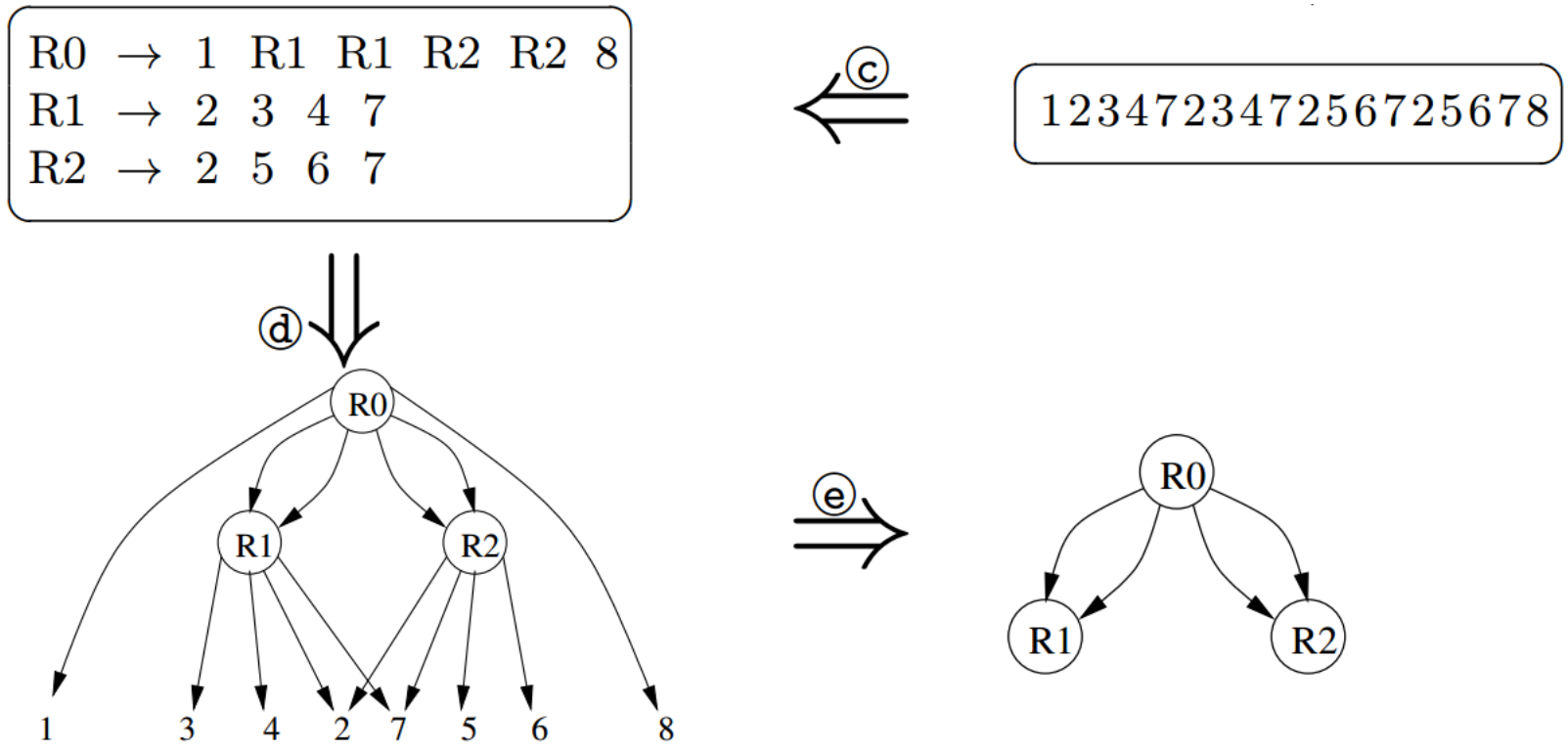
```
1234723472567256 78
```

ⓒ

```
R0 → 1 R1 R1 R2 R2 8
R1 → 2 3 4 7
R2 → 2 5 6 7
```

软件胎记

- 动态构造1：基于执行路径



软件胎记

- 动态构造2：基于程序内的系统调用

```
1.  S0;
2.  If  (i==1) {
3.     S1;
4.     for (j=0;j<3;j++)
    {
5.        S2;
6.        S3;
7.        S4;
8.     }
9.     S5;
10.} else {
11.   S6;
12.}
```

(a)

$S_0\ S_1\ S_2\ S_3\ S_4\ S_2\ S_3\ S_4\ S_2\ S_3\ S_4\ S_5$

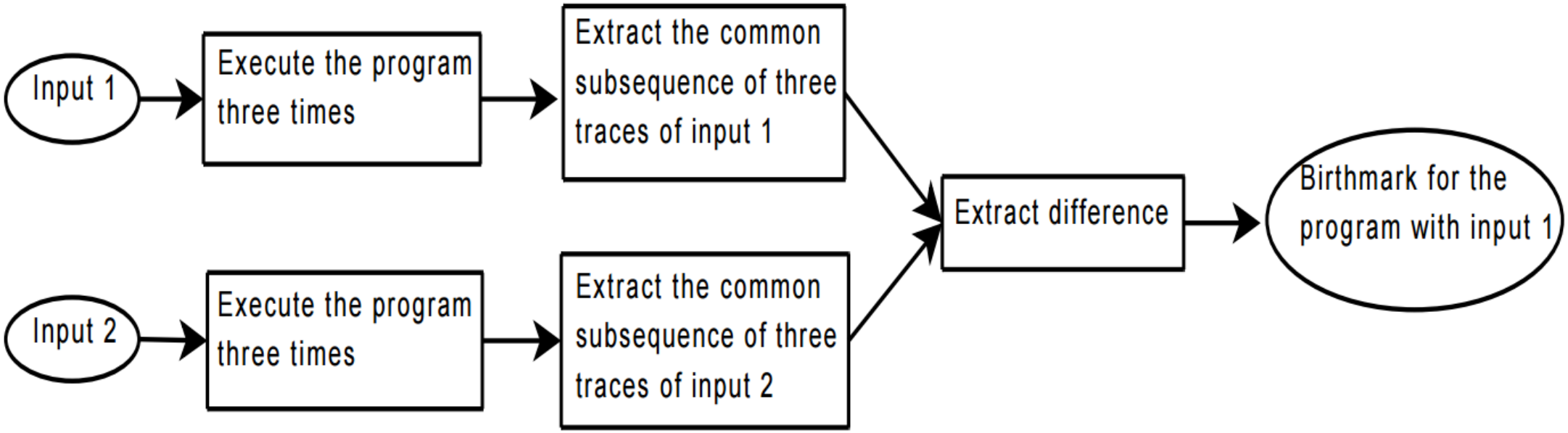
(b)

$S = \{\ S_0S_1S_2S_3,\ S_1S_2S_3S_4,\ S_2S_3S_4S_2,\ S_3S_4S_2S_3,\ S_4S_2S_3S_4,\ S_2S_3S_4S_5\}$

(c)

软件胎记

- 动态构造2：基于程序内的系统调用



软件胎记

还有更多……

软件胎记

- 软件胎记的不足之处
  - 基于相似度，**仅指出可能性**，而非完全准确
  - **需要**有标准样本作为**比较依据**存在
  - **离线工作**，在防范恶意代码/app重包装时**存在滞后性**

软件胎记

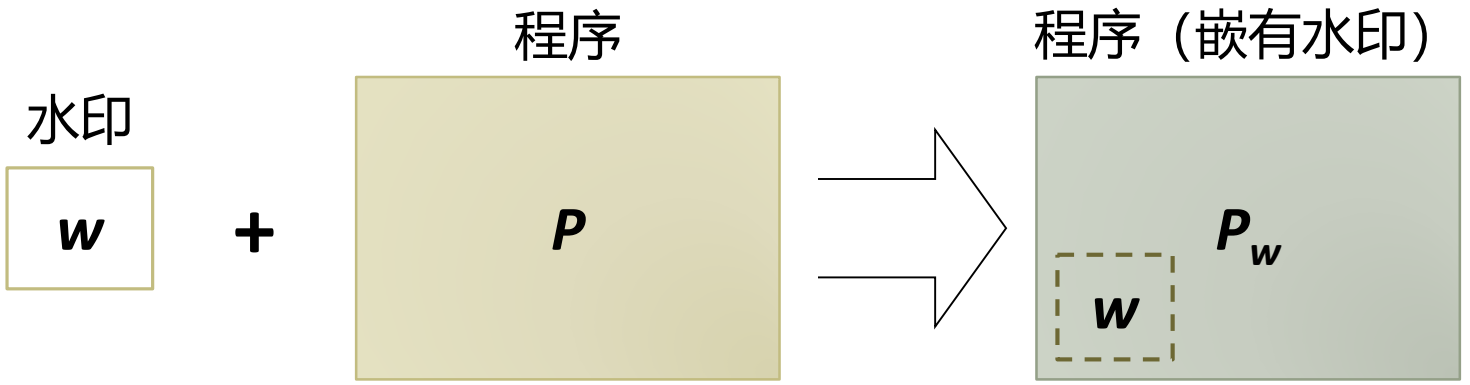
- 参考文献（too many…）
  - Lim H, Heewan P, Seokwoo C, et al. Detecting theft of java applications via a static birthmark based on weighted stack patterns[J]. IEICE transactions on information and systems, 2008, 91(9): 2323-2332.
  - Myles G, Collberg C. Detecting software theft via whole program path birthmarks[C]//International Conference on Information Security. Springer Berlin Heidelberg, 2004: 404-415.
  - Wang X, Jhi Y C, Zhu S, et al. Detecting software theft via system call based birthmarks[C]//Computer Security Applications Conference, 2009. ACSAC'09. Annual. IEEE, 2009: 149-158.

软件胎记

参考文献（and more…）

- Zhang F, Huang H, Zhu S, et al. ViewDroid: Towards obfuscation-resilient mobile application repackaging detection[C]//Proceedings of the 2014 ACM conference on Security and privacy in wireless & mobile networks. ACM, 2014: 25-36.
- Huang H, Zhu S, Liu P, et al. A framework for evaluating mobile app repackaging detection algorithms[C]//International Conference on Trust and Trustworthy Computing. Springer Berlin Heidelberg, 2013: 169-186.
- Guan Q, Huang H, Luo W, et al. Semantics-based repackaging detection for mobile apps[C]//International Symposium on Engineering Secure Software and Systems. Springer International Publishing, 2016: 89-105.

软件水印

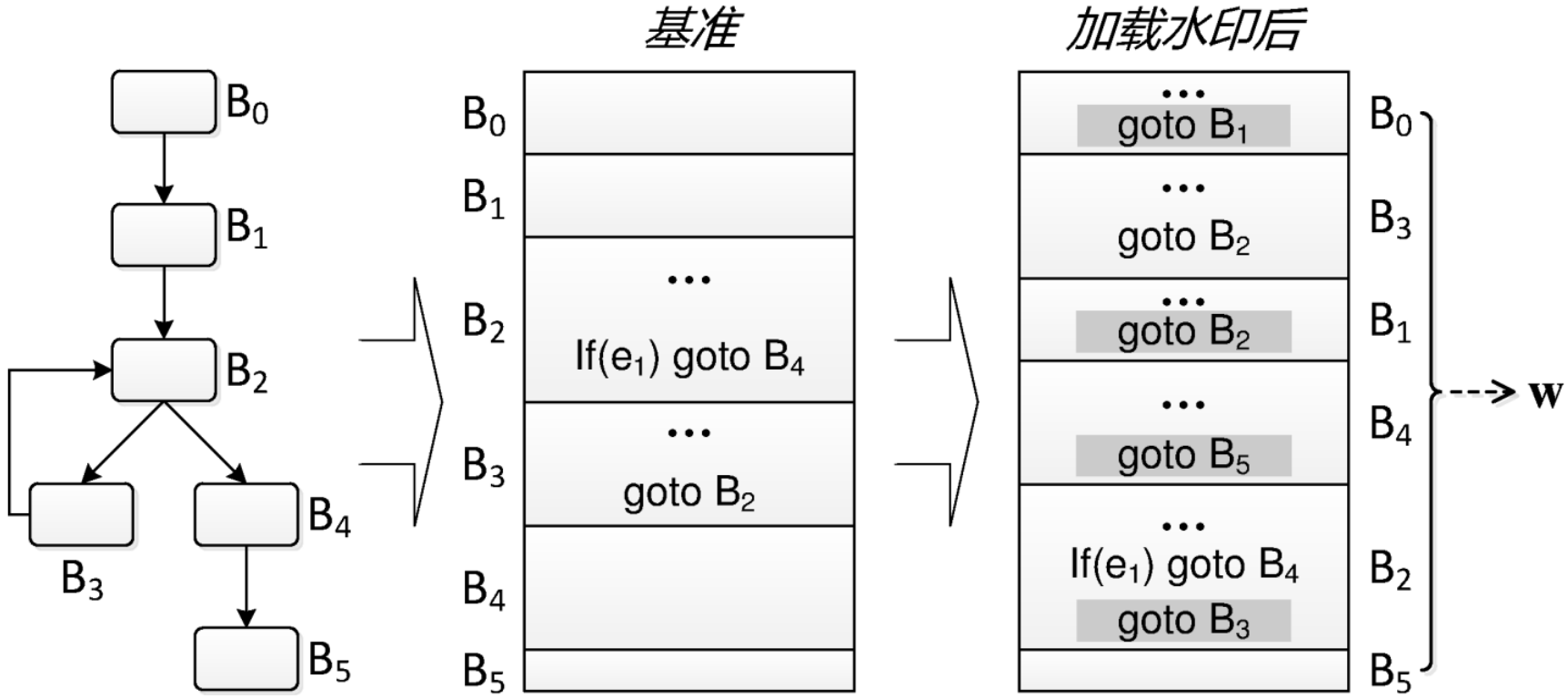


软件水印

- 软件水印的应用形式
  - 反盗版 – 通过声明版权
  - 反盗版 – 通过追溯盗版母盘的来源
- 软件水印的形式分类
  - 静态/动态（取决于水印的构造方式）

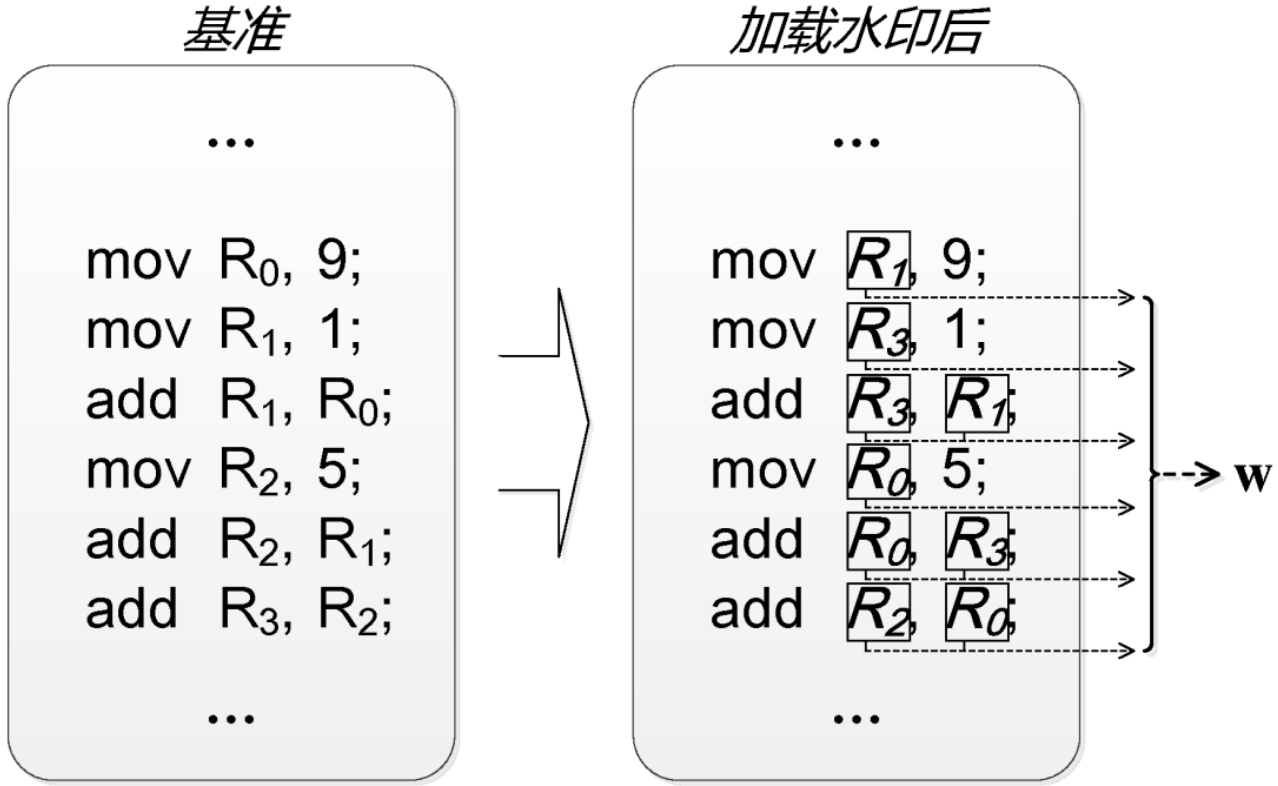
软件水印

- 静态构造举例：基本块重排序



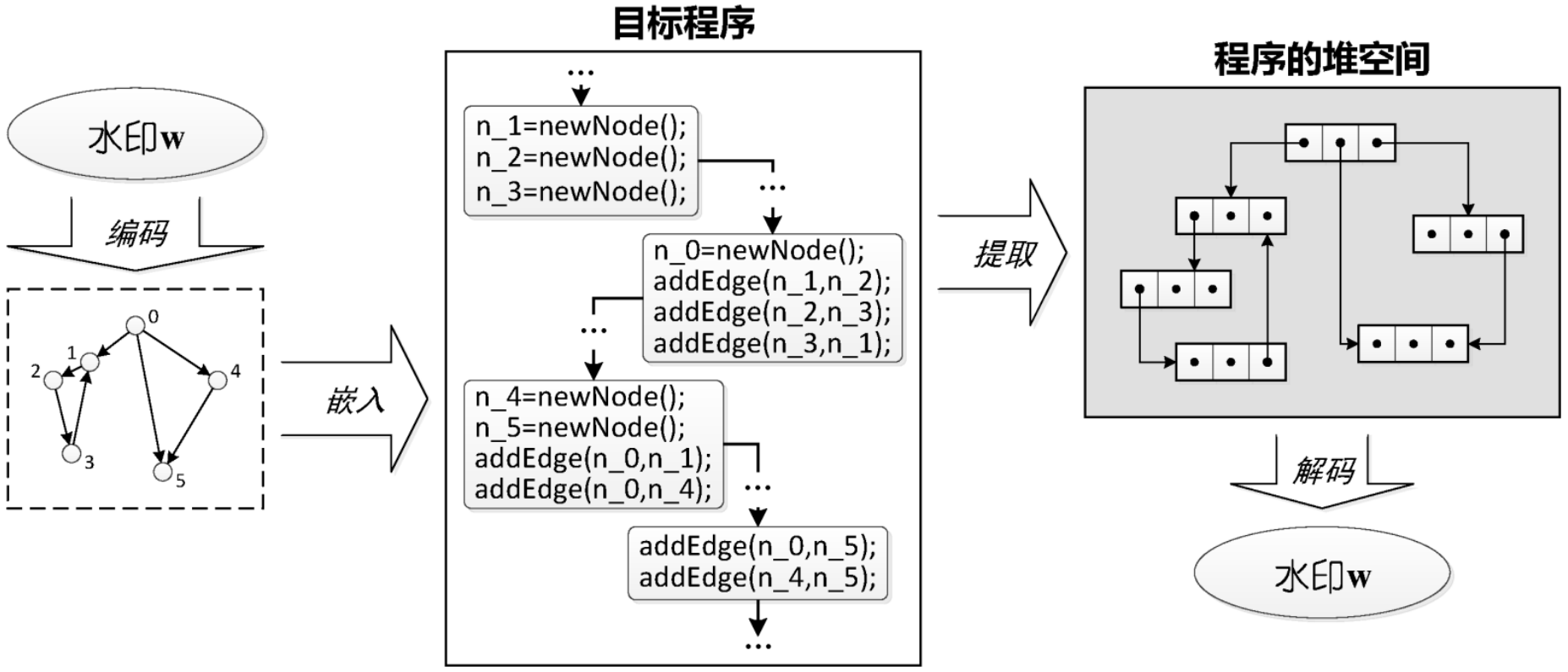
软件水印

- 静态构造举例：寄存器占用重分配



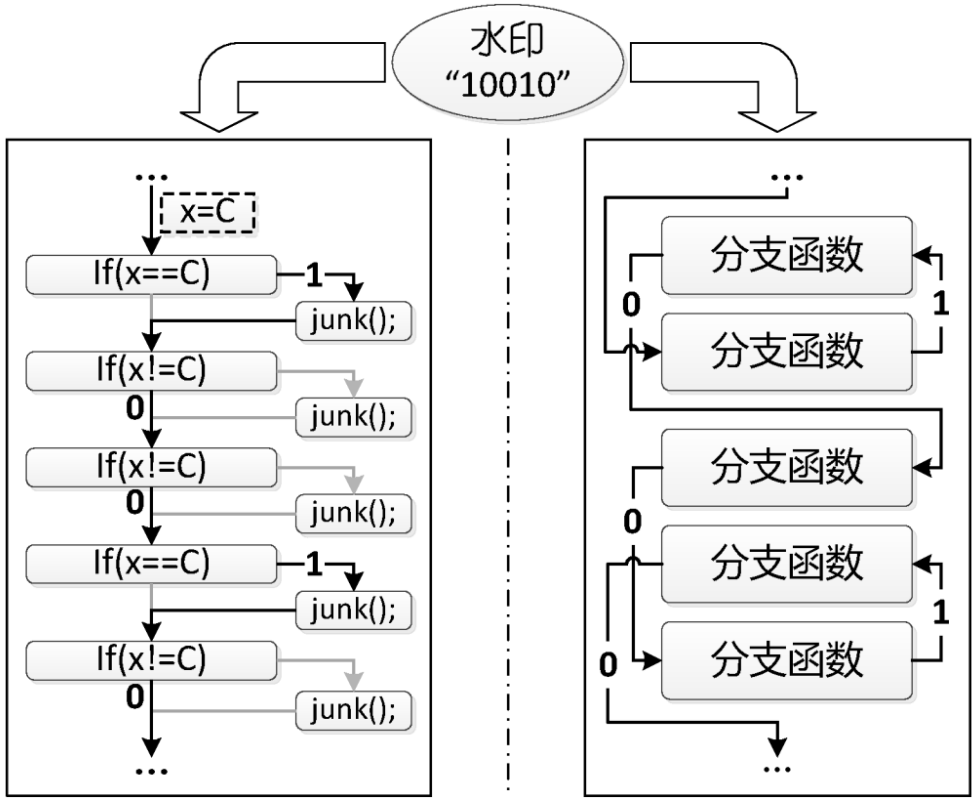
软件水印

- 传统动态构造1：基于动态生成的图对象



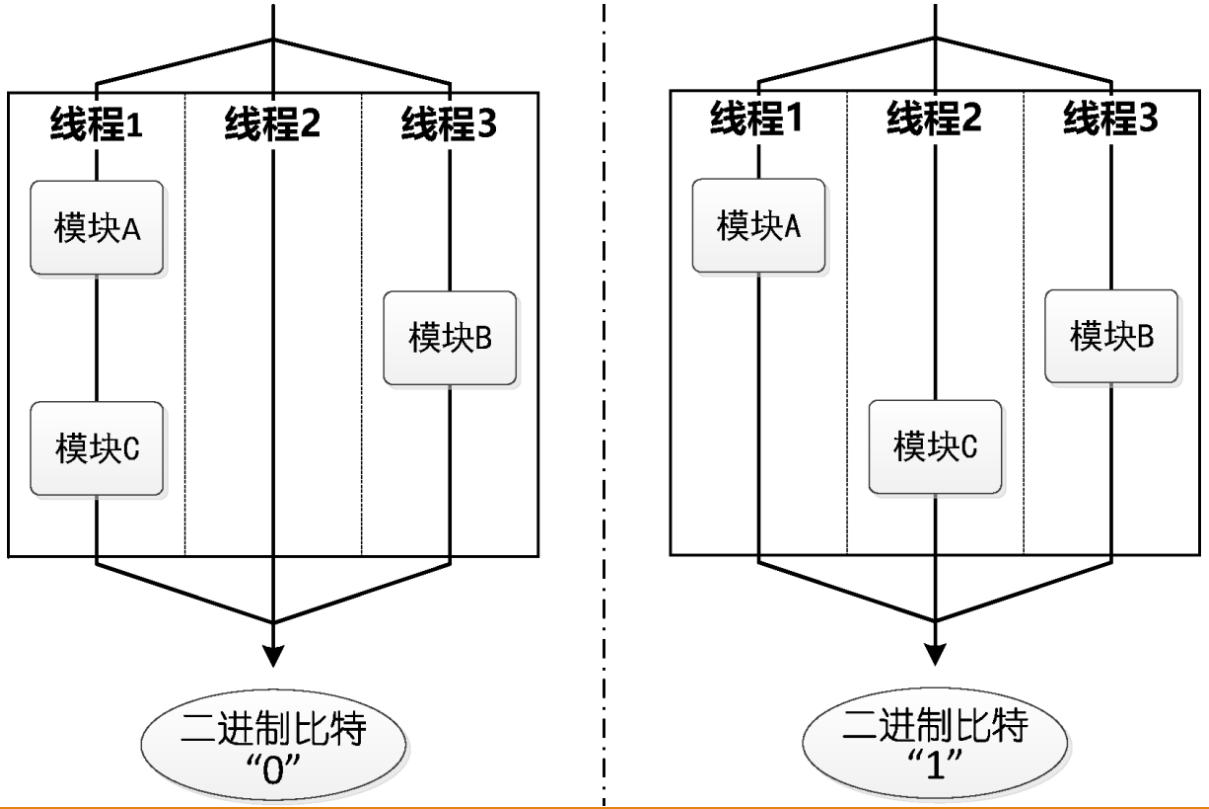
软件水印

- 传统动态构造2：基于执行路径上的分支行为



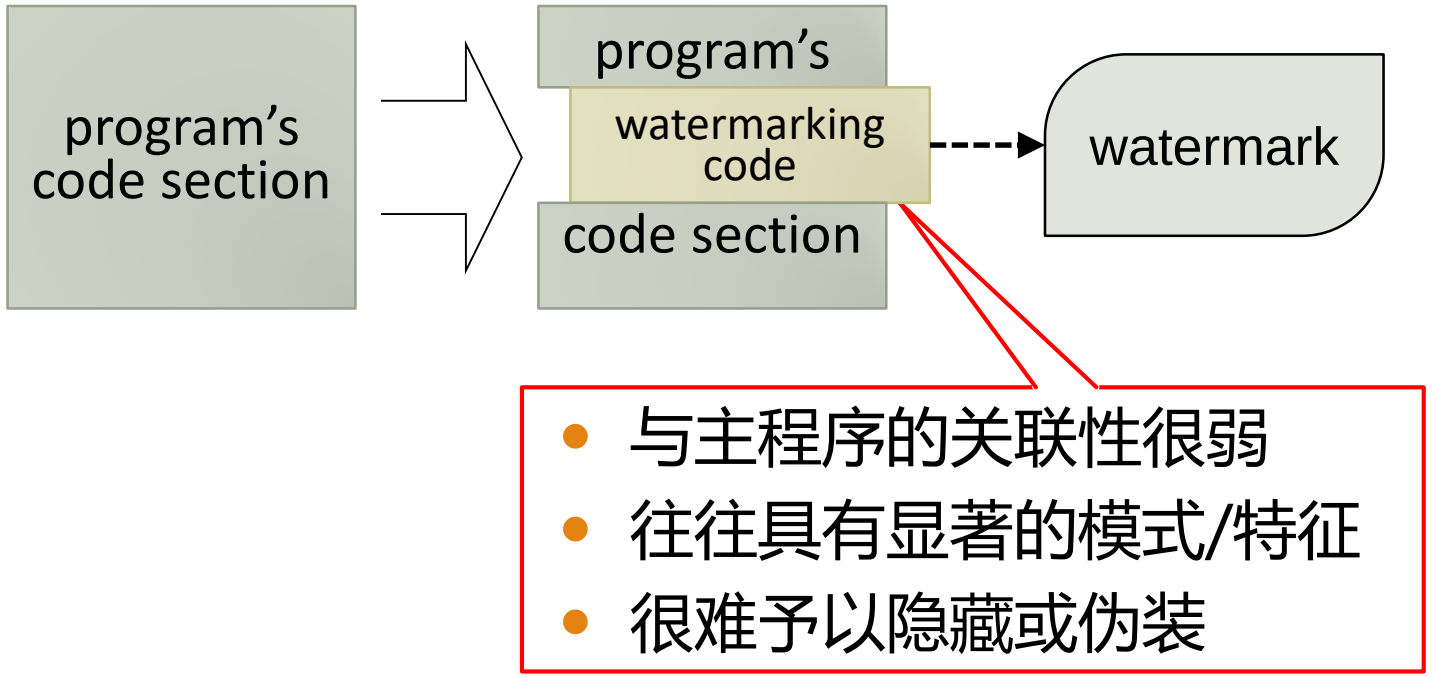
软件水印

- 传统动态构造3：基于多线程的同步行为



软件水印

- 传统动态构造的问题1：



软件水印

- 一个特别的设计：基于抽象解释的水印
  - 在正常维度上，水印组件服务于载体软件的原本功能
  - 在预设的**秘密维度**上，水印组件展示出隐藏的信息

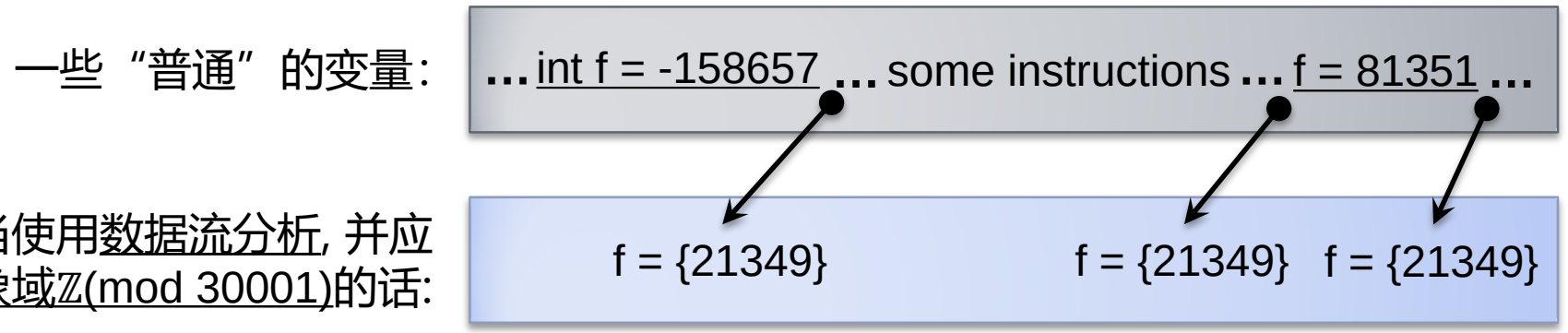
具体域：　$-1717 \times 15 =$

抽象域[$+, -, ?$]：　$(-) \times (+) = ?$

待验证命题:
is positive?

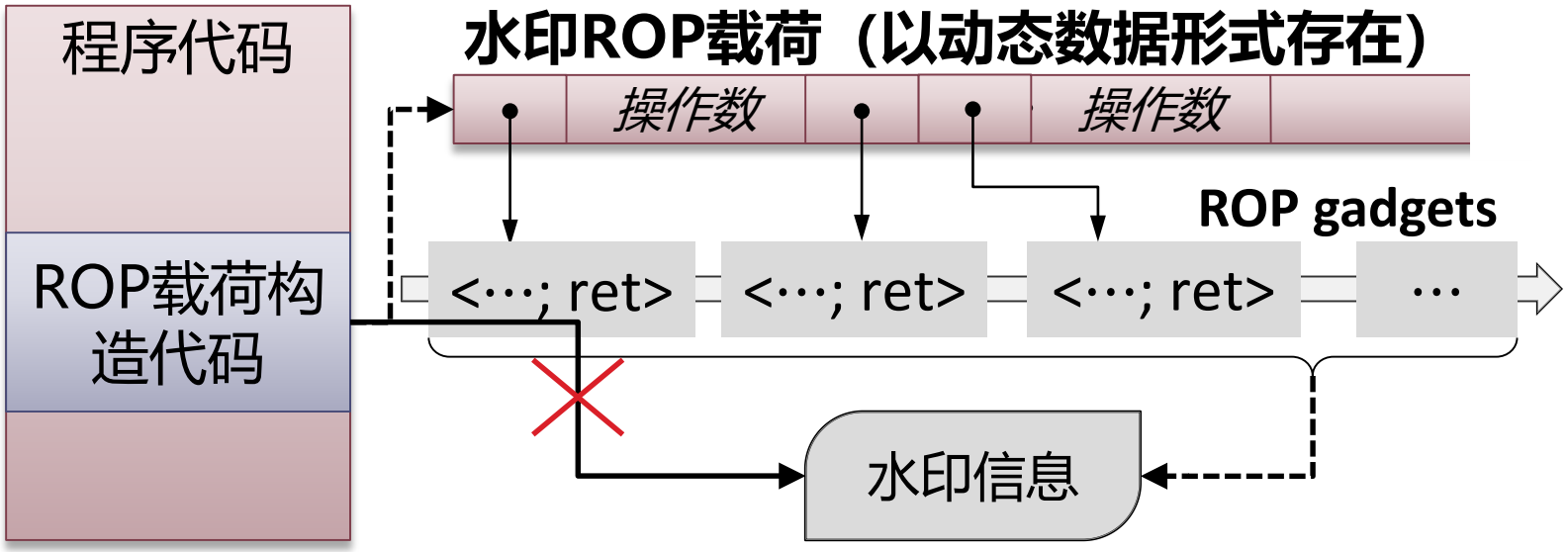
软件水印

- 一个特别的设计：基于抽象解释的水印
  - 在正常维度上，水印组件服务于载体软件的原本功能
  - 在预设的**秘密维度**上，水印组件展示出隐藏的信息



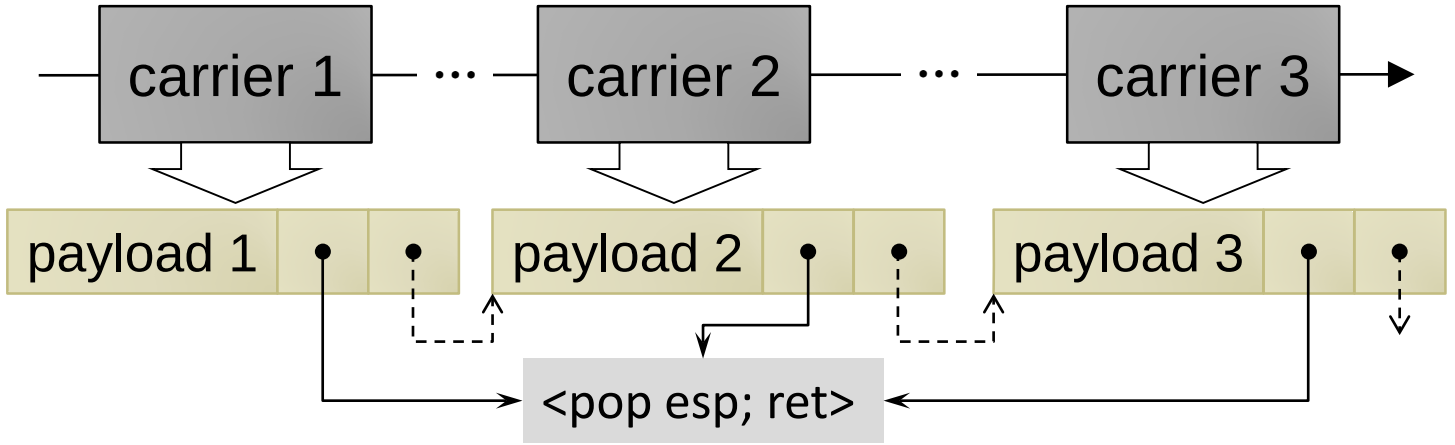
软件水印

- 改进动态构造1：利用返回导向编程



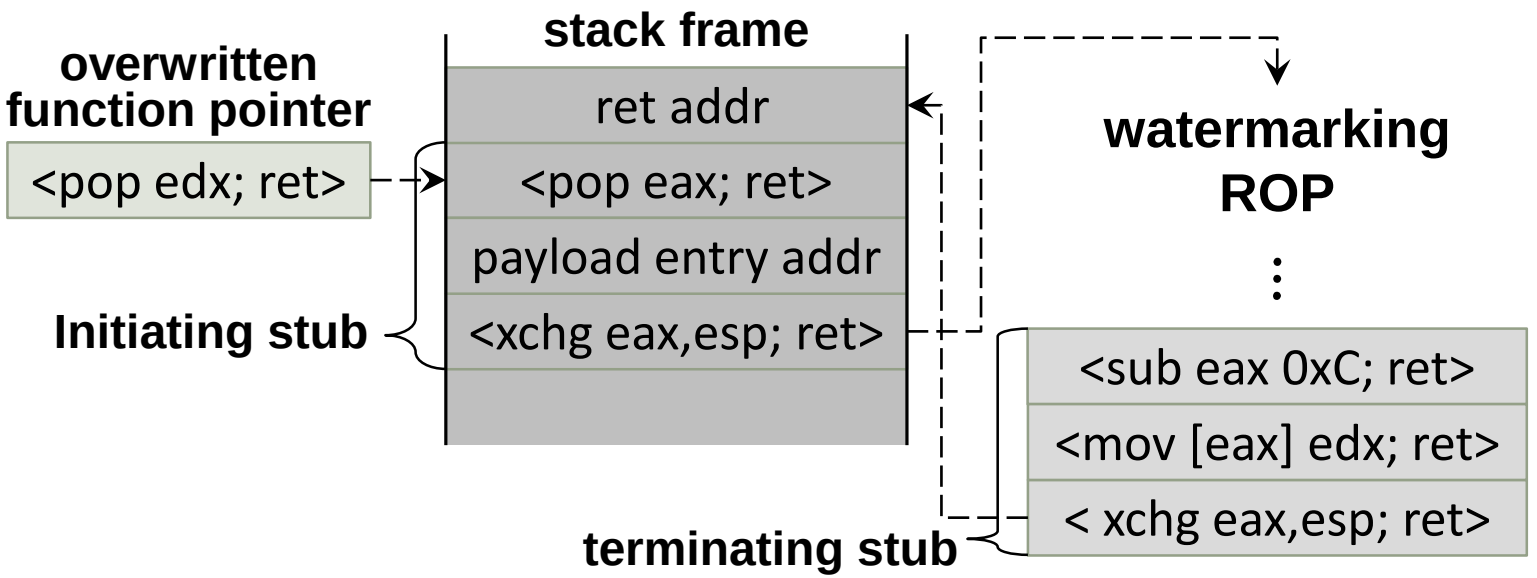
软件水印

- 改进动态构造1：利用返回导向编程



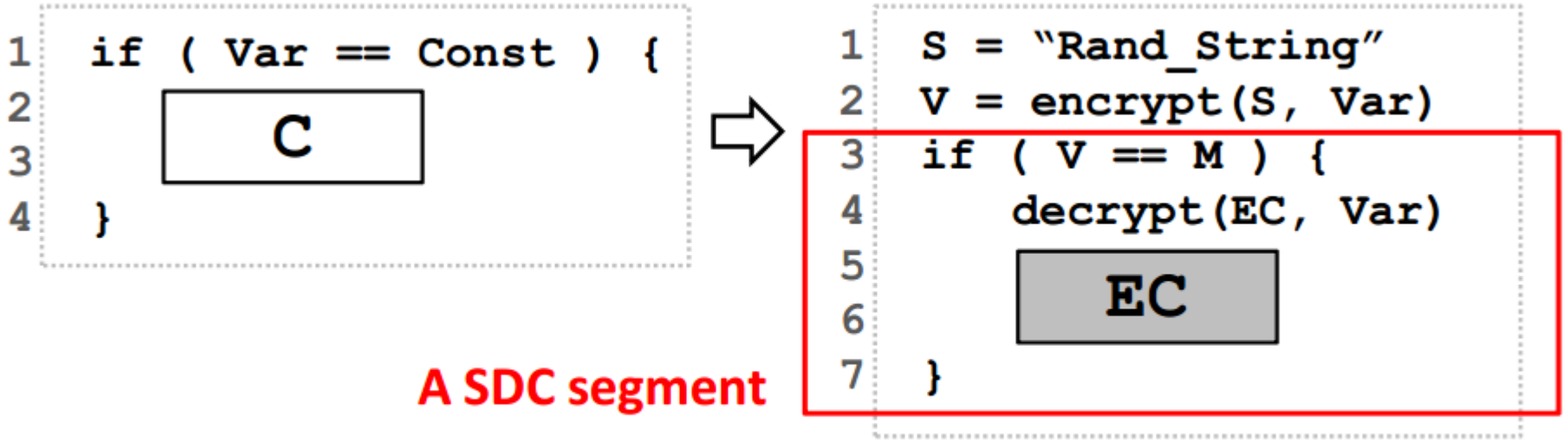
软件水印

- 改进动态构造1：利用返回导向编程



软件水印

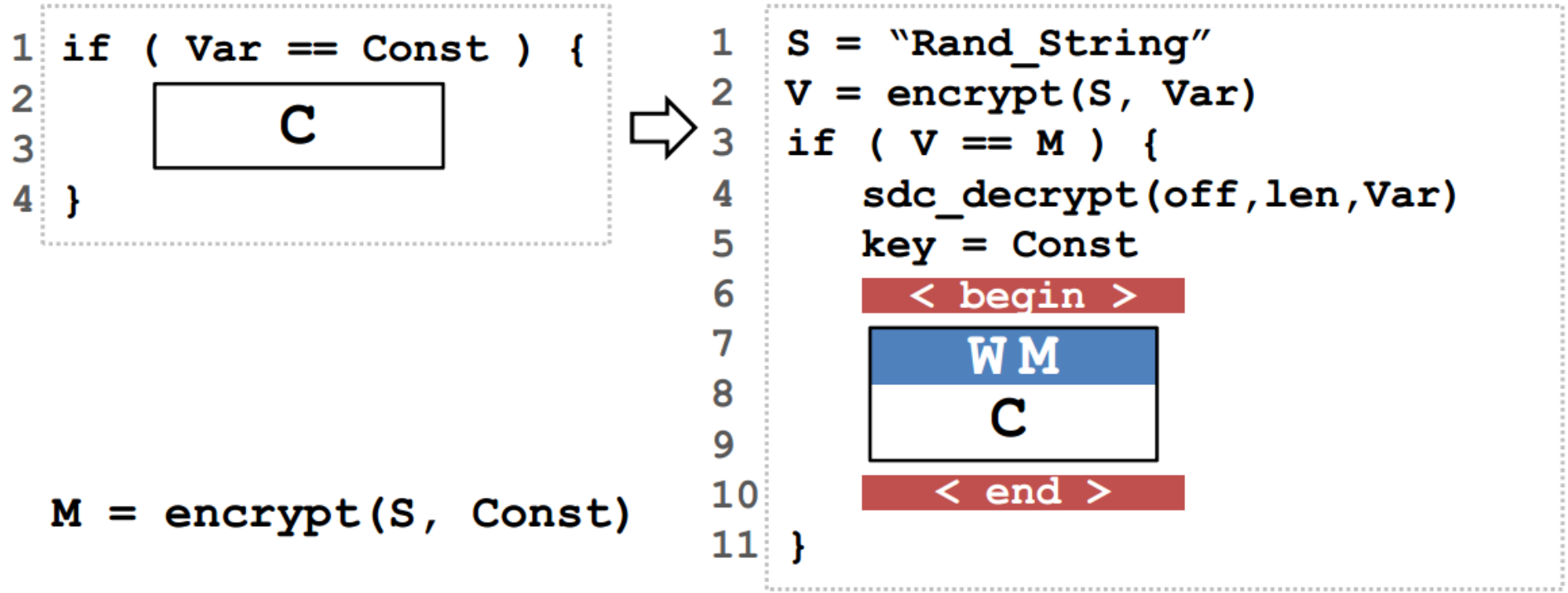
- 改进动态构造2：利用代码混淆



Where, M= encrypt(S, Const),EC = encrypt(C, Const)

软件水印

- 改进动态构造2：利用代码混淆

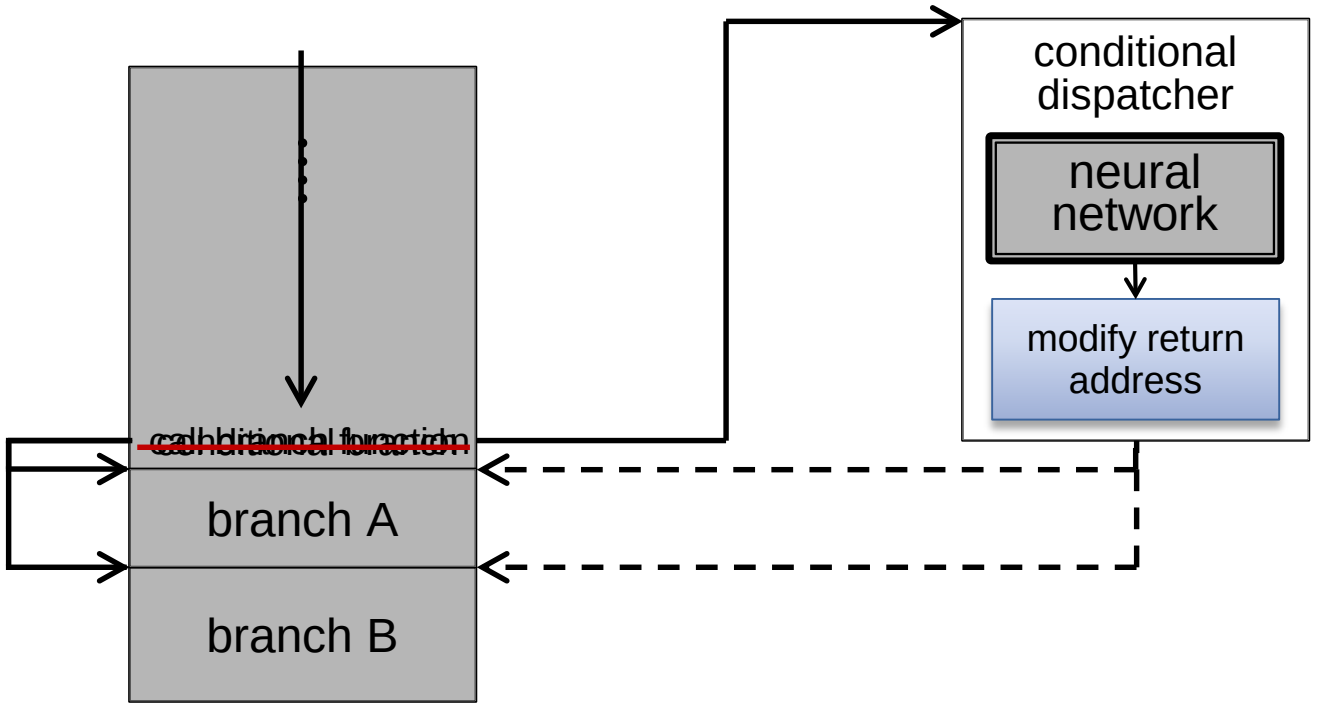


软件水印

- 传统动态构造的问题2：

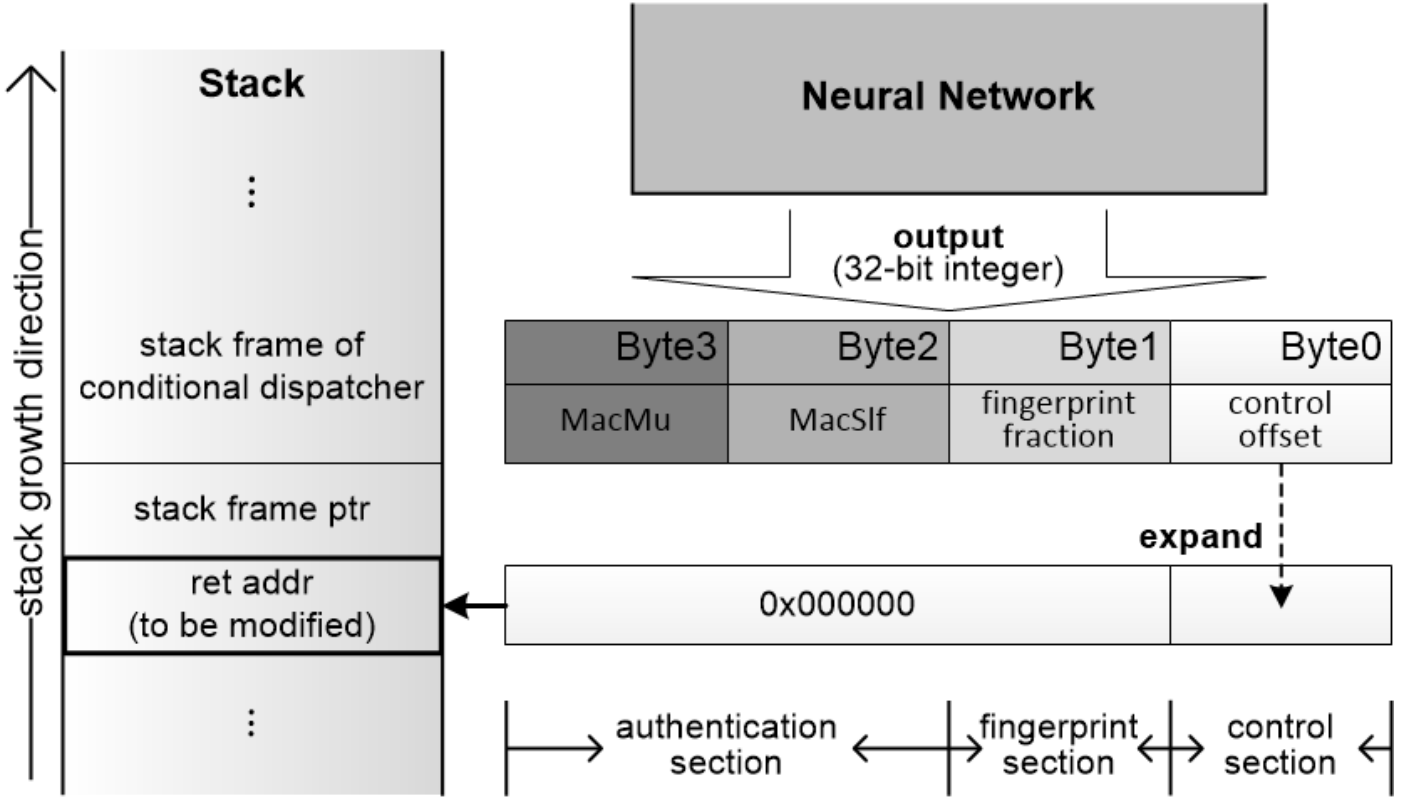
软件水印

- 改进动态构造3：利用神经网络



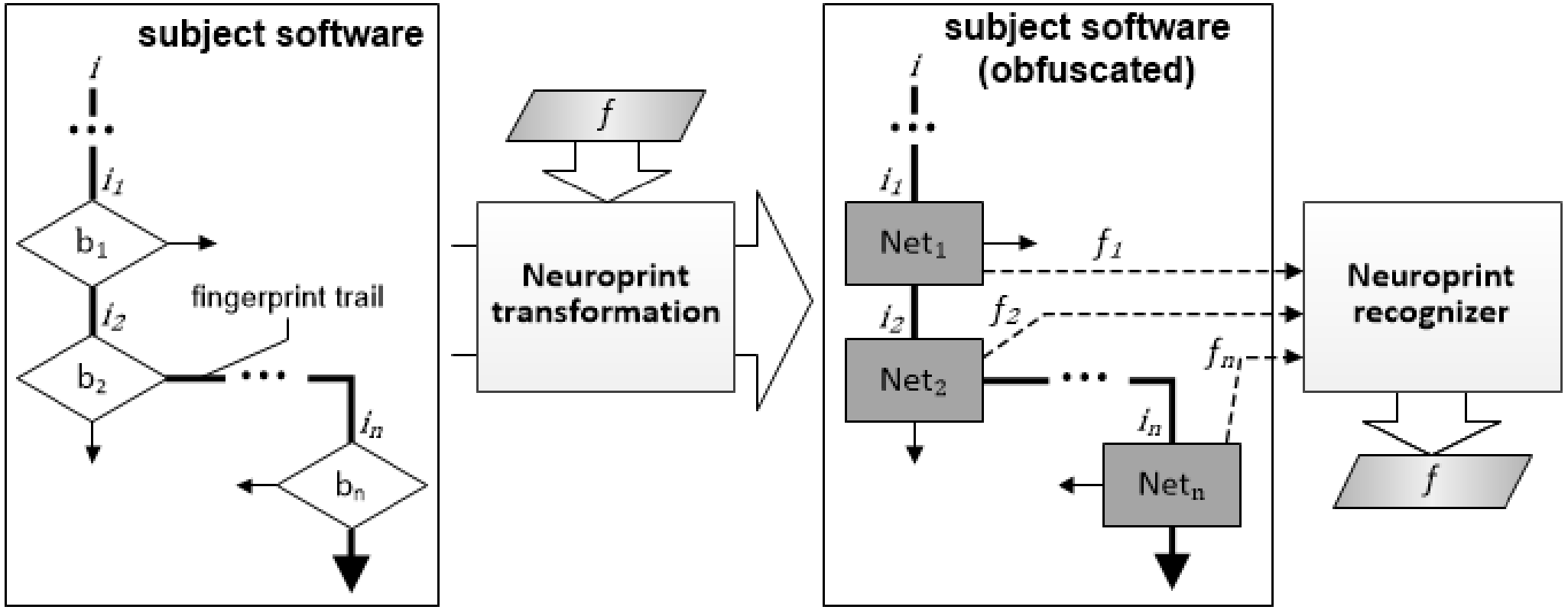
软件水印

- 改进动态构造3：利用神经网络



软件水印

- 改进动态构造3：利用神经网络



软件水印

- 软件水印仍然存在的不足之处
  - 没有在真正意义上实现隐蔽性
  - 缺乏有效的定性/定量评估标准（特别是隐蔽性这一安全度量）
  - 数据嵌入率很差
  - 没有解决工业化、自动化实施的问题

软件水印

- 软件水印仍然存在的不足之处

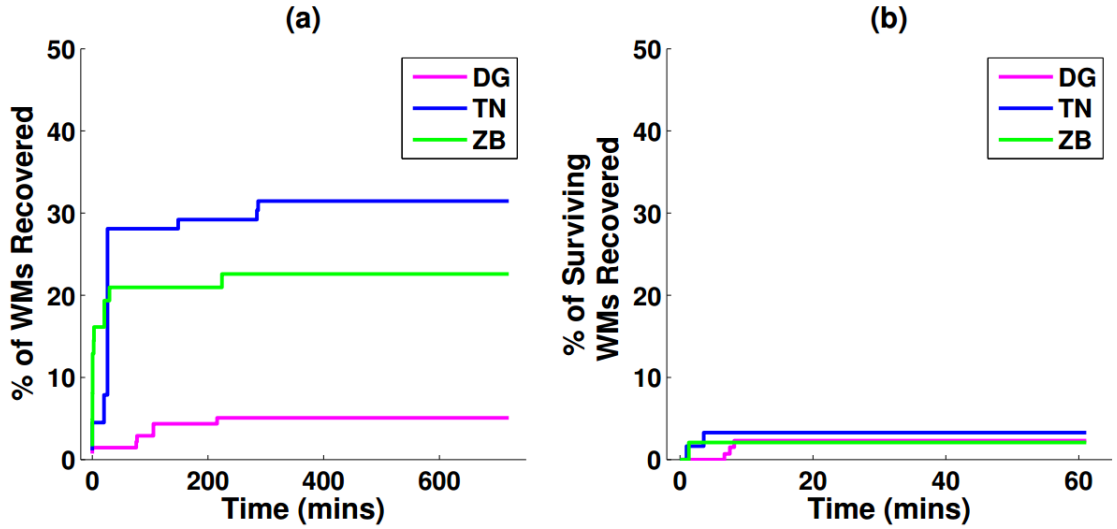


Figure 10: (a) Percentage of watermarks removed by attacker using Monkey. (b) Percentage of surviving watermarks later recovered by human testers.

代码混淆的方法

- 参考文献（still a lot…）
  - Collberg C, Thomborson C. Software watermarking: Models and dynamic embeddings[C]//Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1999: 311-324.
  - Myles G, Collberg C. Software watermarking through register allocation: Implementation, analysis, and attacks[J]. Information security and cryptology-ICISC 2003, 2004: 274-293.
  - Collberg C S, Thomborson C, Townsend G M. Dynamic graph-based software fingerprinting[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2007, 29(6): 35.

代码混淆的方法

- 参考文献（and more…）
  - Collberg C, Carter E, Debray S, et al. Dynamic path-based software watermarking[J]. ACM Sigplan Notices, 2004, 39(6): 107-118.
  - Nagra J, Thomborson C. Threading software watermarks[C]//International Workshop on Information Hiding. Springer Berlin Heidelberg, 2004: 208-223.
  - Cousot P, Cousot R. An abstract interpretation-based framework for software watermarking[C]//ACM SIGPLAN Notices. ACM, 2004, 39(1): 173-185.

代码混淆的方法

- 参考文献（finally…）
  - Ma H, Lu K, Ma X, et al. Software Watermarking using Return-Oriented Programming[C]//Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security. ACM, 2015: 369-380.
  - Ren C, Chen K, Liu P. Droidmarking: resilient software watermarking for impeding android application repackaging[C]//Proceedings of the 29th ACM/IEEE international conference on Automated software engineering. ACM, 2014: 635-646.
  - Ma H, Li R, Yu X, et al. Integrated Software Fingerprinting via Neural-Network-Based Control Flow Obfuscation[J]. IEEE Transactions on Information Forensics and Security, 2016, 11(10): 2322-2337.

The End