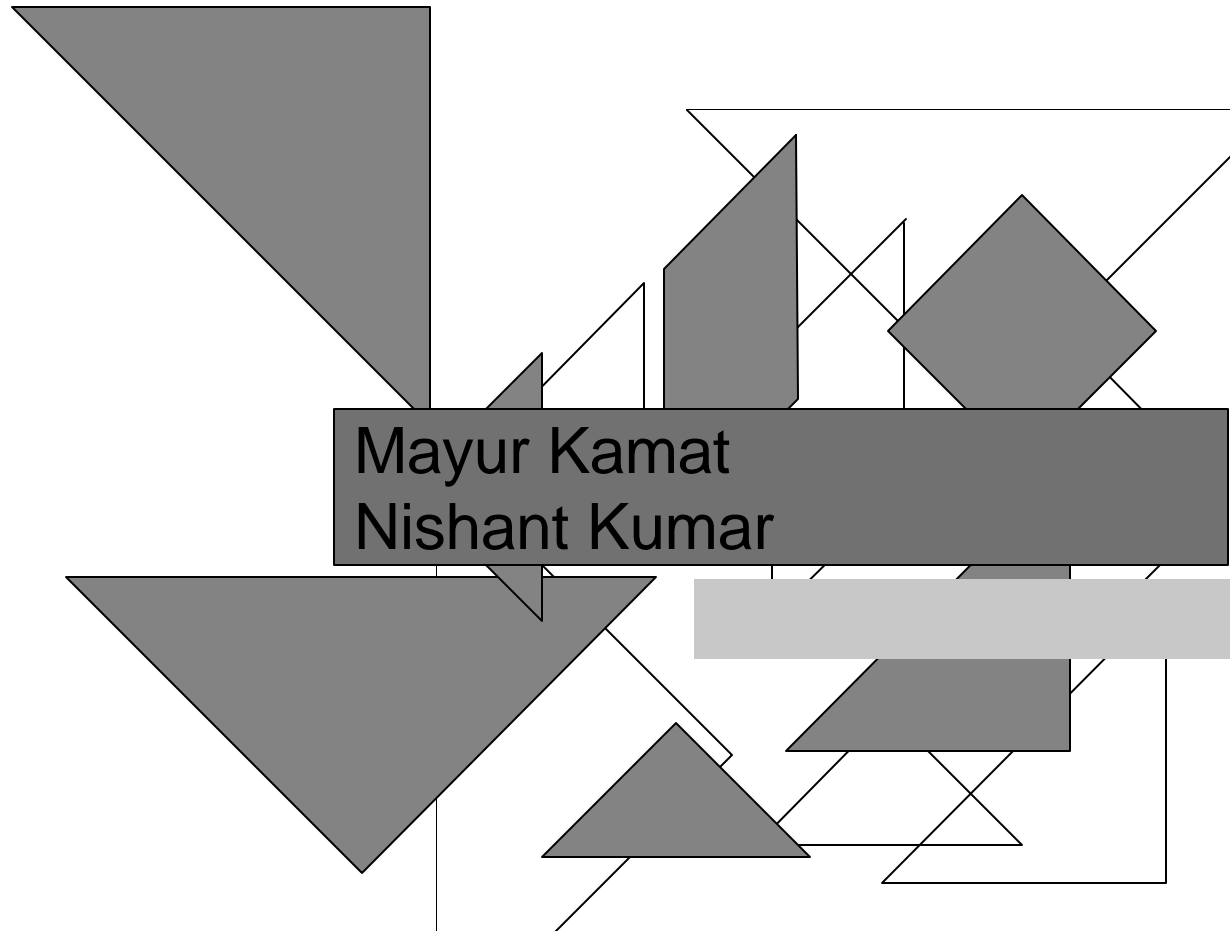


Code Obfuscation

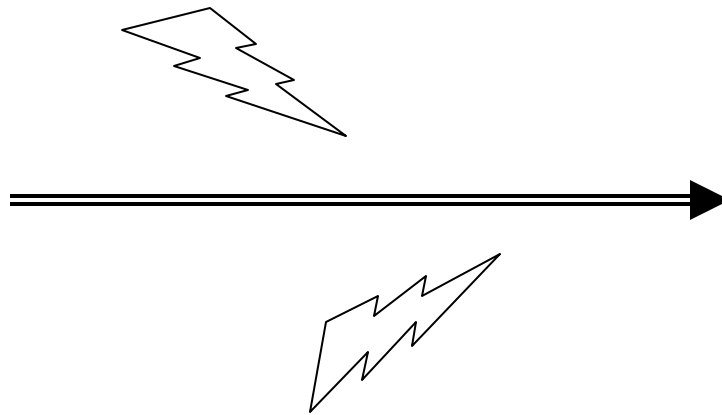


Mayur Kamat
Nishant Kumar

Agenda

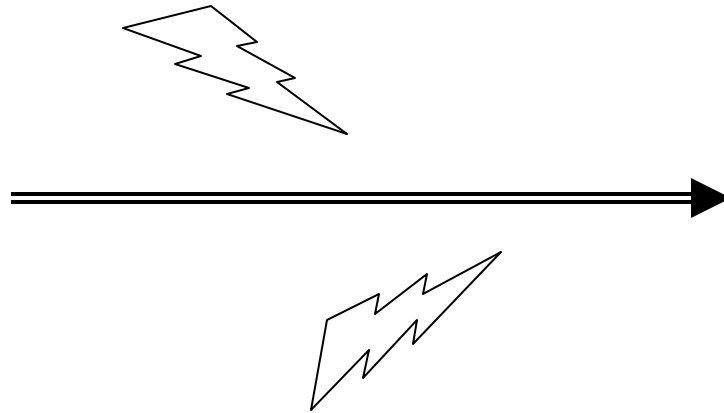
- ◆ Malicious Host Problem
- ◆ Code Obfuscation
- ◆ Watermarking and Tamper Proofing
- ◆ Market solutions

Traditional Network Security Problem



Hostile Network

Malicious Host



Hostile Network

Hostile User

Motivation

◆ Intellectual Property Protection

■ Under threat in various forms

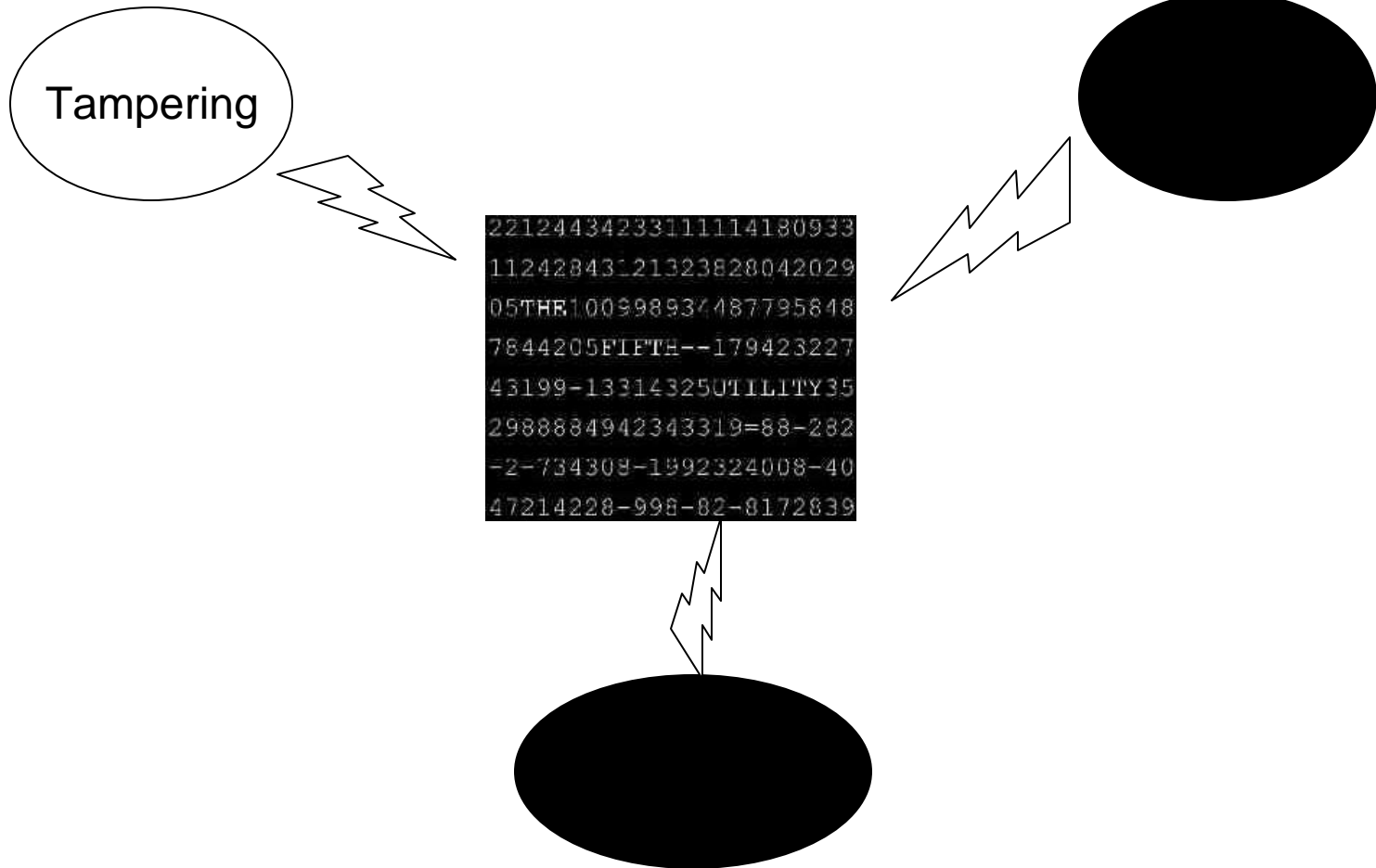
- ◆ Reverse engineering

- ◆ Tampering

- ◆ Software Piracy

■ Loss of revenue and finally an overall loss

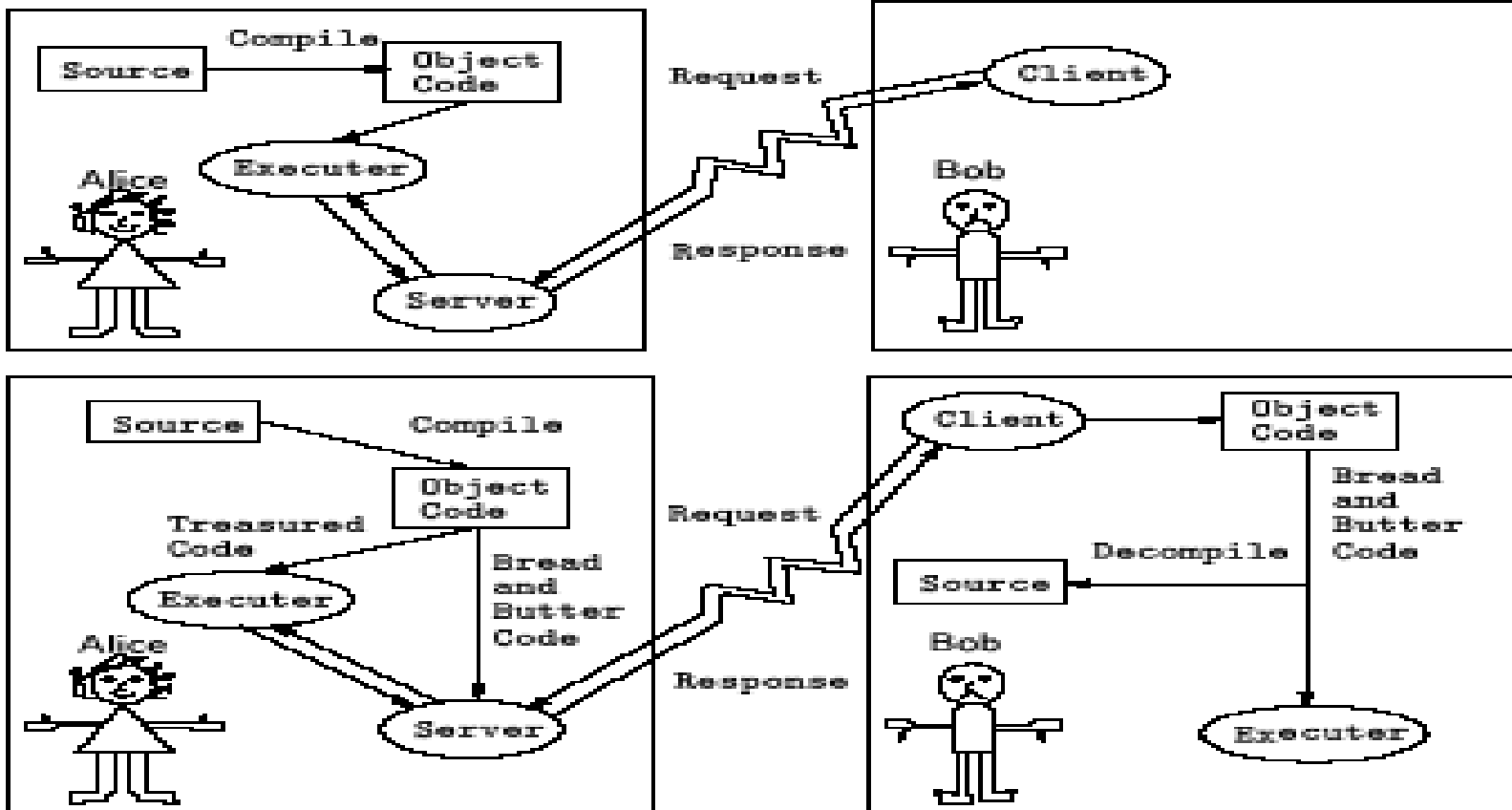
Code under Attack



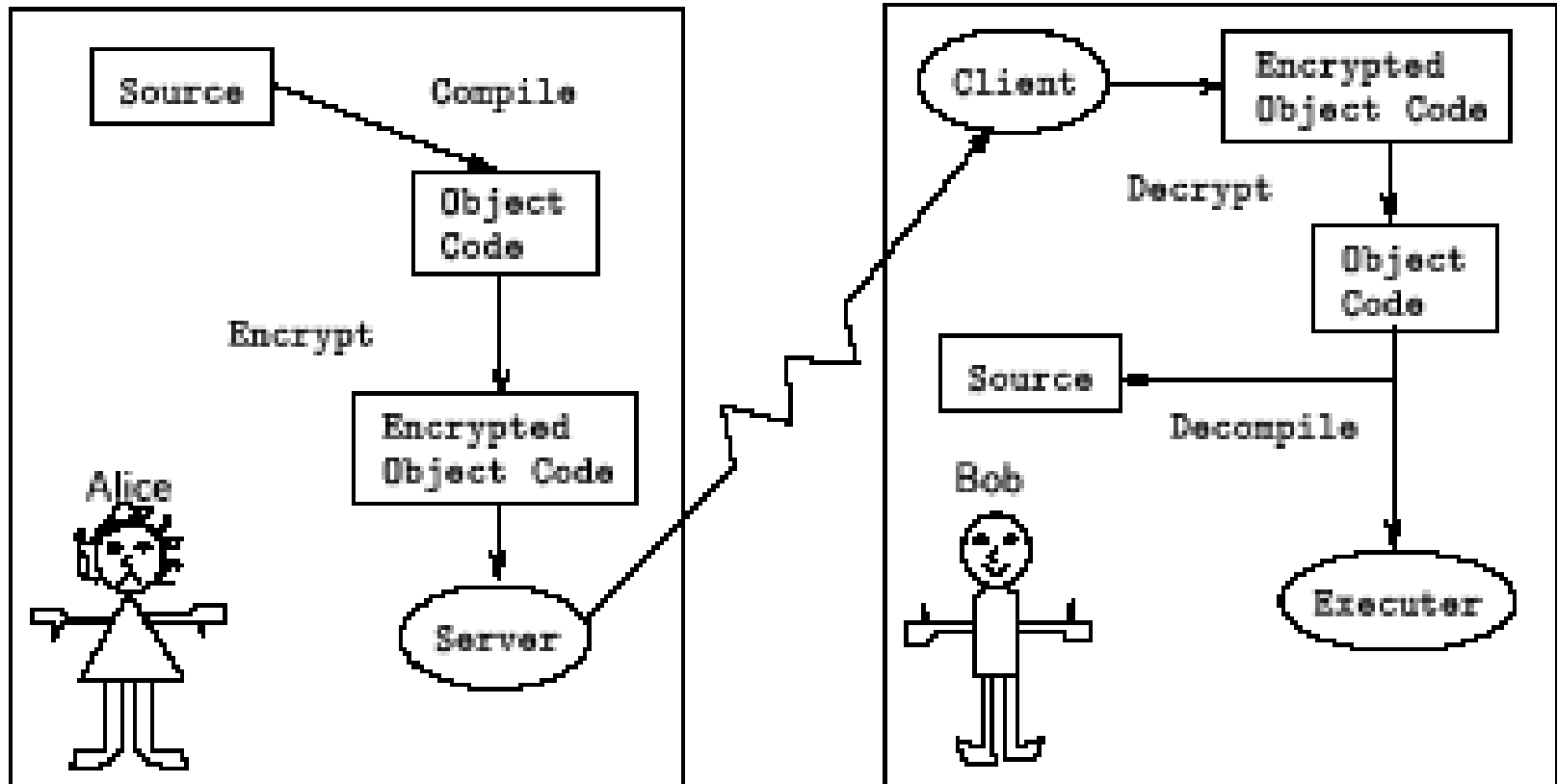
Protection

- ◆ Legal battles
 - No Electronic Theft (NET)
 - Digital Millennium Copyright Act (DMCA)
- ◆ Technical Protection
 - Server-Side Protection
 - Encryption
 - Native Code
 - Obfuscation

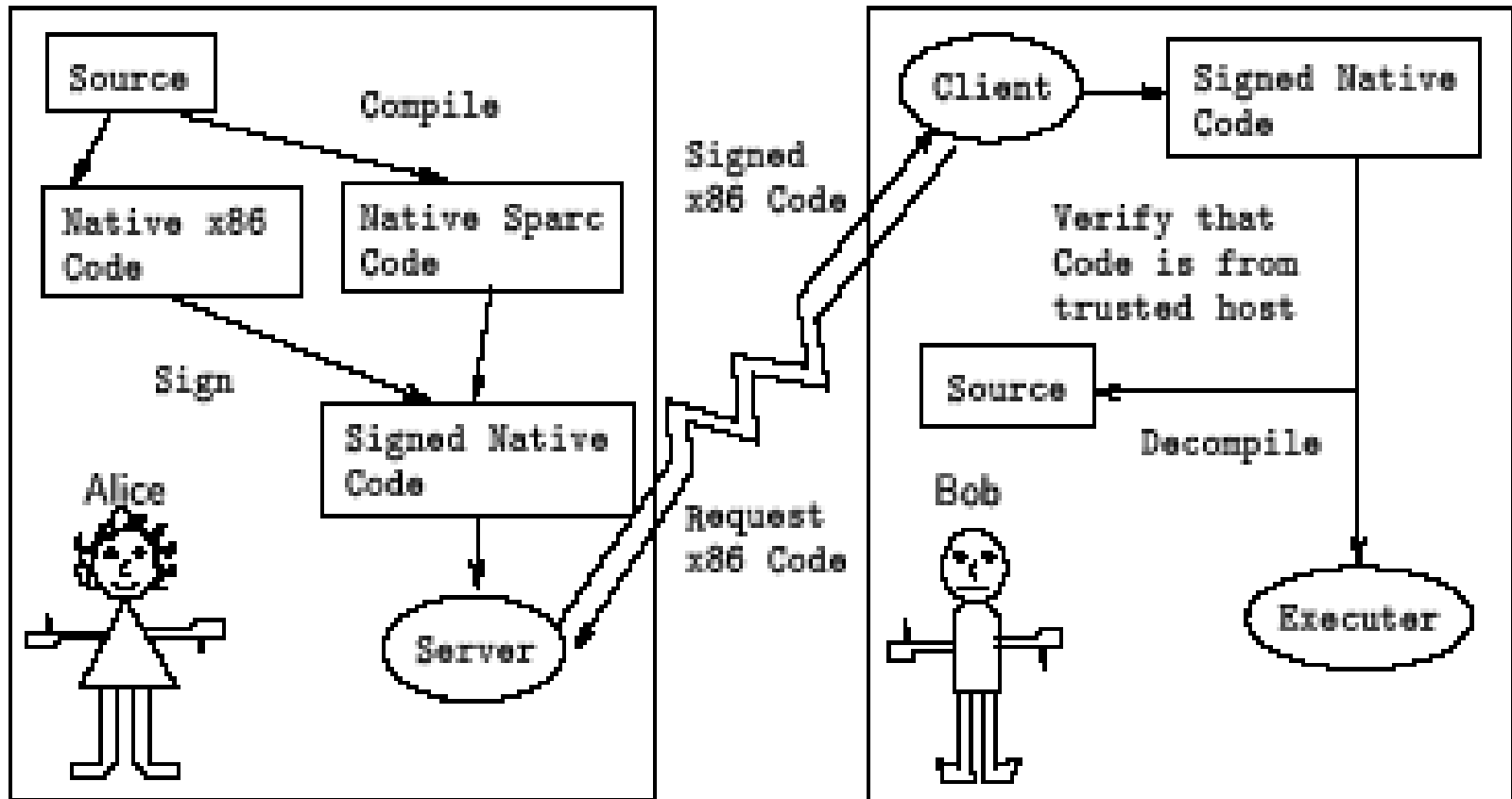
Server-Side Protection



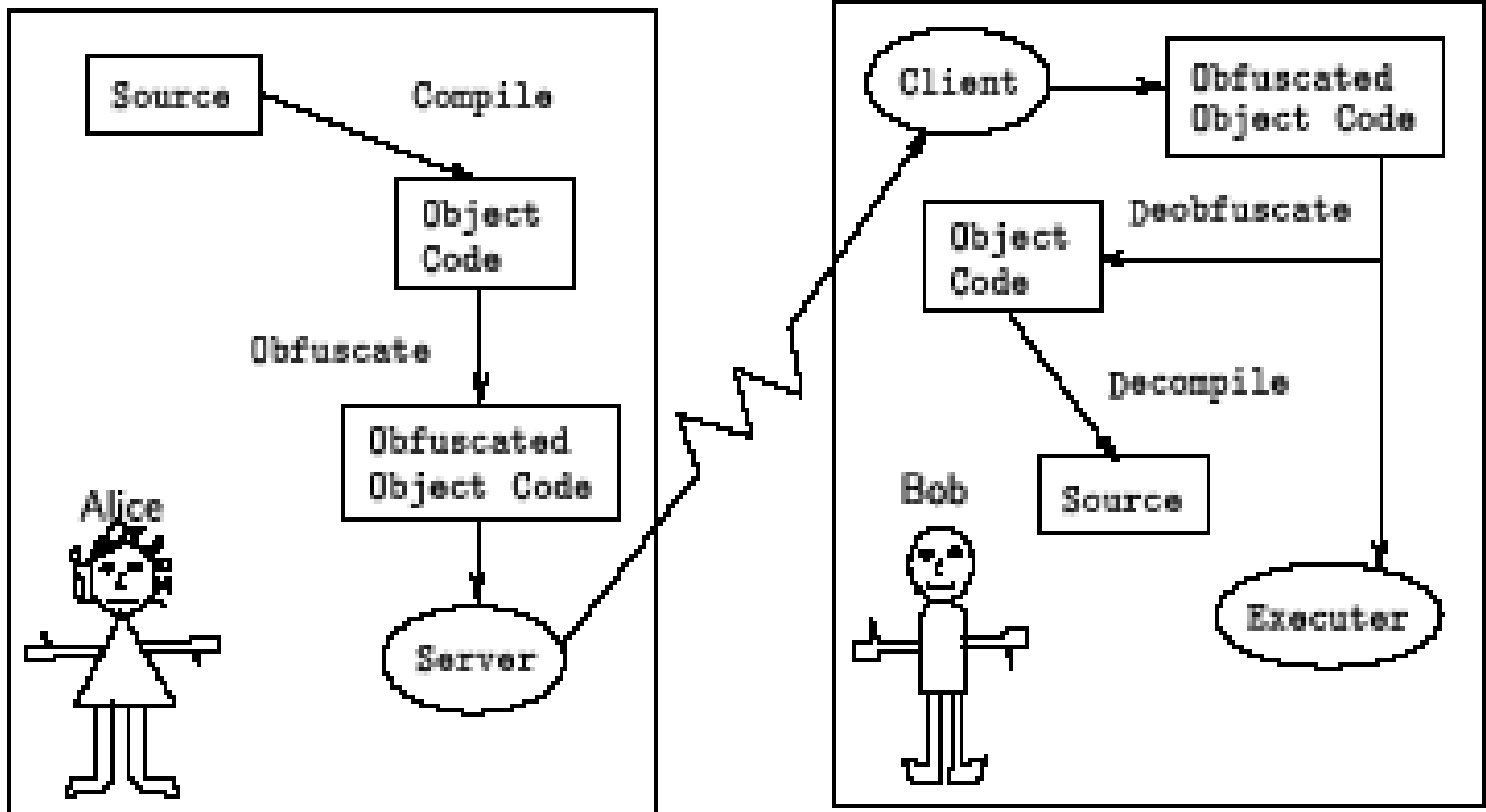
Encryption



Native Code



Obfuscation



Definition

T

$P \rightarrow P'$ is an obfuscating transformation if

- If P fails to terminate or terminates with error, then P' may or may not terminate
- Otherwise, P' must terminate and produce same output as P

Measuring Obfuscation

Obfuscation can be measured as sum of the following:

- ◆ **Potency** ($E(P')/E(P) - 1$)
- ◆ **Resilience** (trivial, weak, strong, full, 1-way)
- ◆ **Cost** (free, cheap, costly, dear)
- ◆ **Stealth**

Forget about good
software engineering
principles

Software metrics

Complexity of software increases:

- ◆ Program length
 - No. of operators and operands.
- ◆ Data flow complexity
 - Number of inter-block variable references.
- ◆ Cyclomatic complexity
 - Number of predicates in a function.
- ◆ Nesting complexity
 - Number of nesting level of conditionals in a program.

Software metrics...

◆ Data structure complexity

- Complexity of the static data structures in the program like variables, vectors, records.

◆ OO Metrics

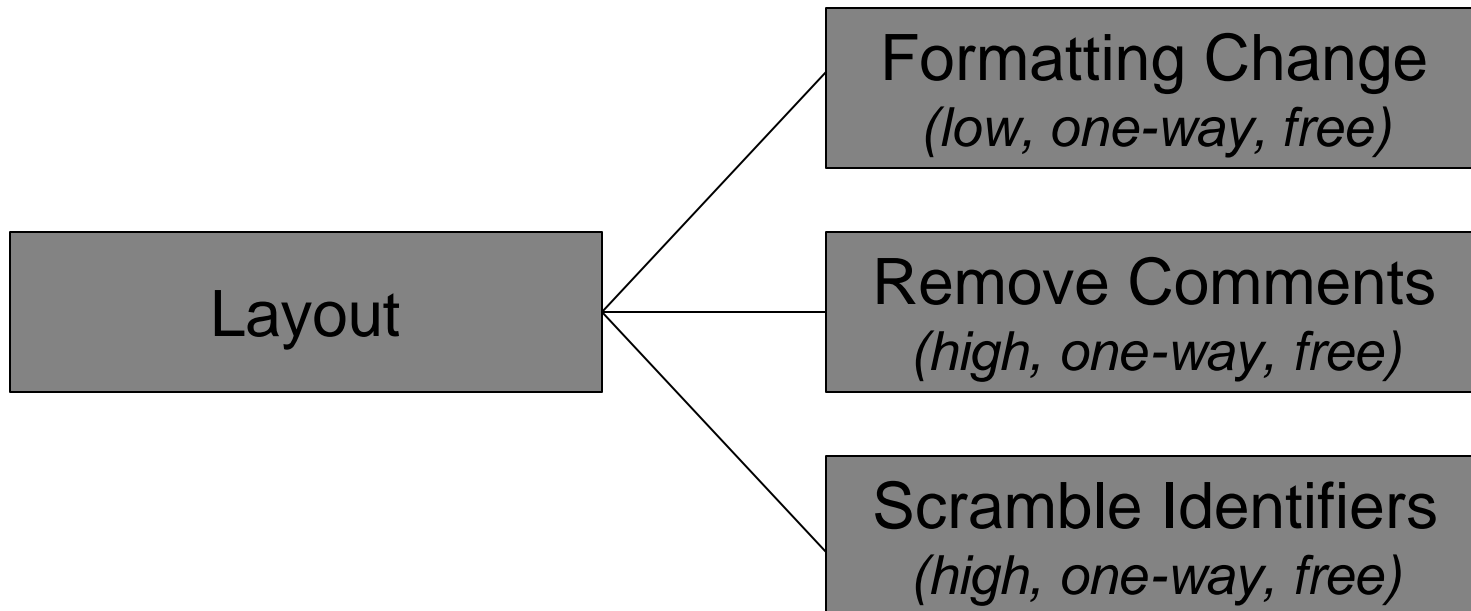
- Level of inheritance
- Coupling
- Number of methods triggered by another method
- Non-cohesiveness

Obfuscation Categories

- ◆ Layout Obfuscation
- ◆ Data Obfuscation
- ◆ Control Obfuscation
- ◆ Preventive Obfuscation

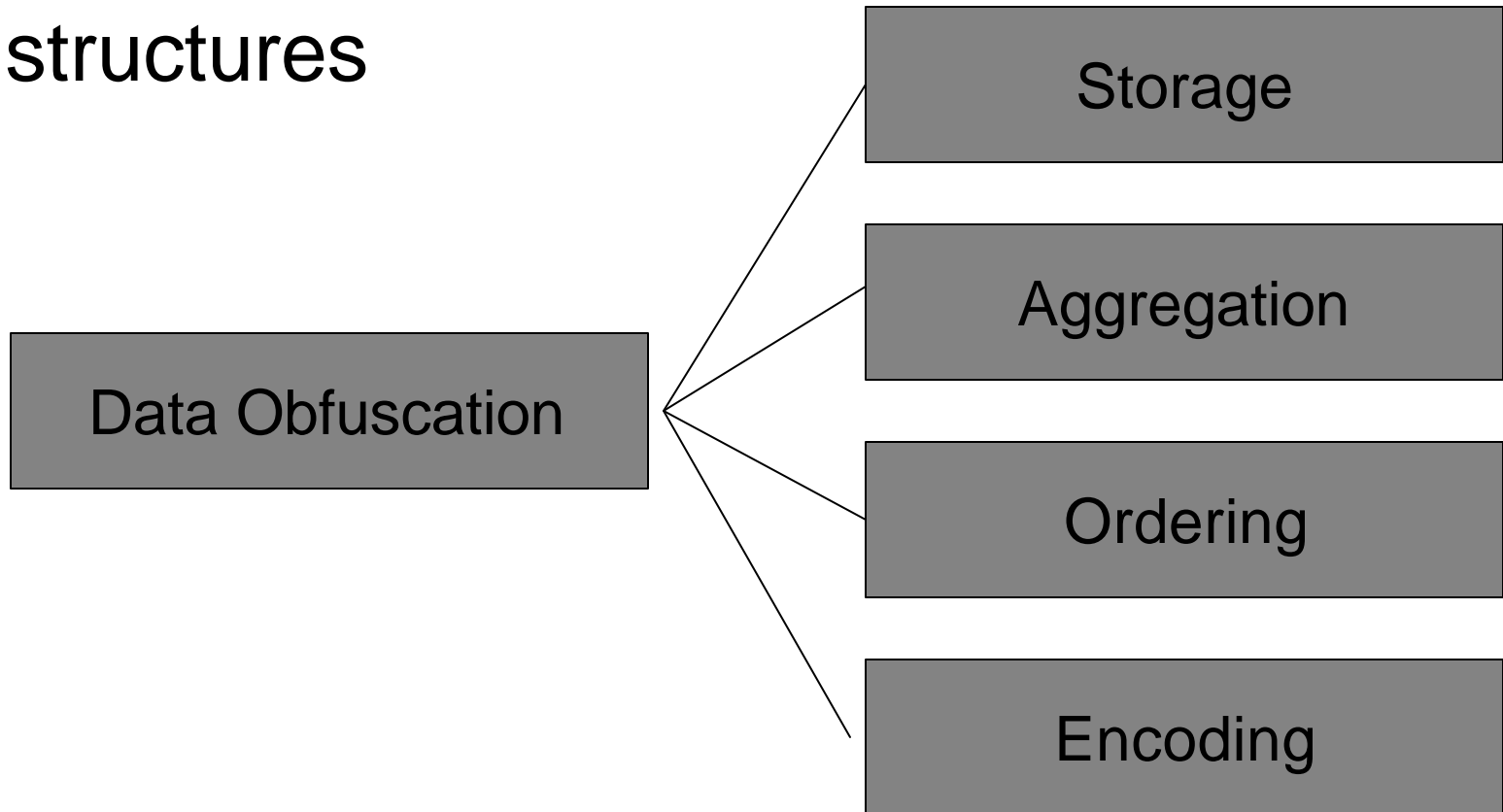
Layout Obfuscation

- ◆ Aimed at making the code unreadable



Data Obfuscation

- ◆ Aimed at obscuring data and data structures



Storage & Encoding - 1

◆ Split variables

- Transform single variable : $v = [(v1, v2.. vk)]$
- Increases Potency
- Required to provide extra functions for variable mapping (higher cost)

Original	Obfuscated
Bool a, b, c	Short a1, a2, a3, b1, b2, c1, c2
a = true, b = false, c = true	a1 = a2=a3 = true,
c = a & b	c1 = ((a1 ^ a2) ^ a3) & (b1 ^ b2) c2 = c1

Storage and Encoding- 2

◆ Convert Static into Procedural

- Lots of information conveyed through static variables
- Convert into function call
- High potency and resilience (depends on function)

Actual	Obfuscated
String t = "Net"	String retStr(int i) { string s; s[1] = "N"; s[2] = "e"; s[3] = "t";}

Storage and Encoding- 3

- ◆ Other important methods are
 - Scalars to Objects
 - Changing encoding
 - Changing Variable Lifetimes....


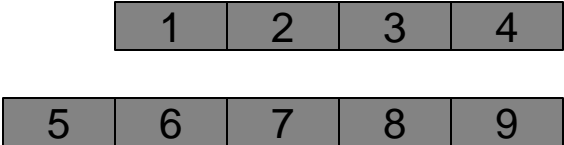
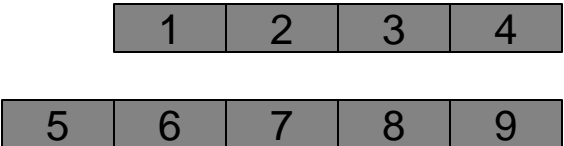

Aggregation & Reordering - 1

- ◆ Array Transformation
- ◆ Modifying Inheritance
- ◆ Merging scalar variables
- ◆ Reordering execution sequence

Aggregation & Reordering - 2

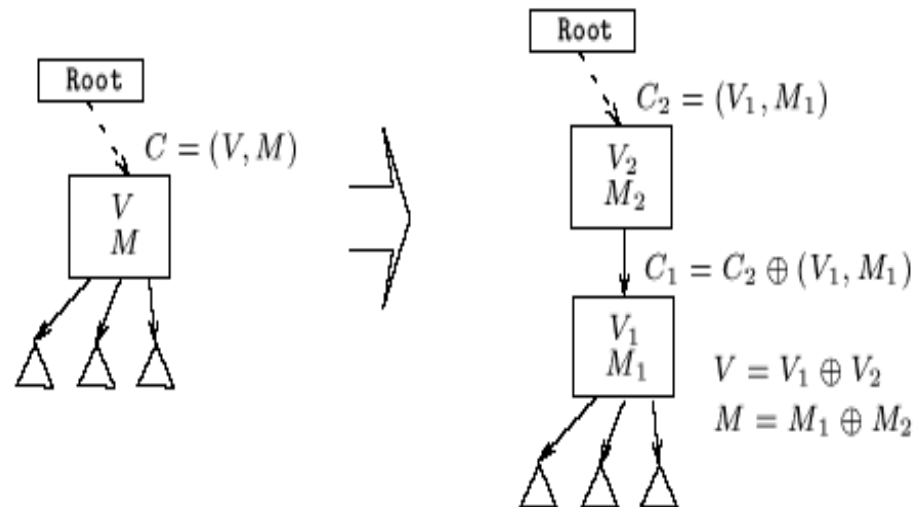
◆ Array Transformation

- Splitting & folding (increases potency)
- Merging & flattening (decreases potency)

Actual	Obfuscated
	
	

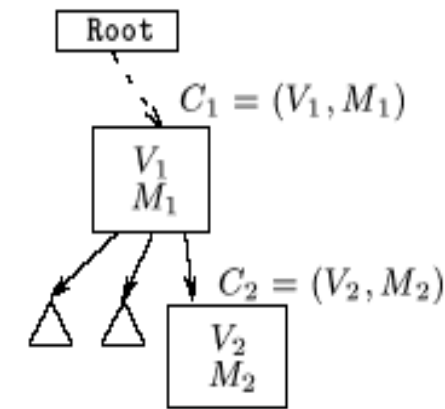
Aggregation & Reordering - 3

- ◆ Inheritance
 - Split a class

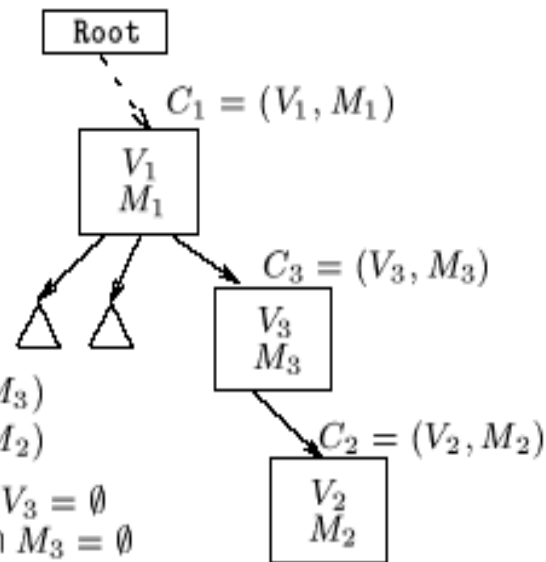


Aggregation & Reordering - 4

- ◆ Inheritance
 - Insert a class



$$C_2 = C_1 \oplus (V_2, M_2)$$



$$C_3 = C_1 \oplus (V_3, M_3)$$

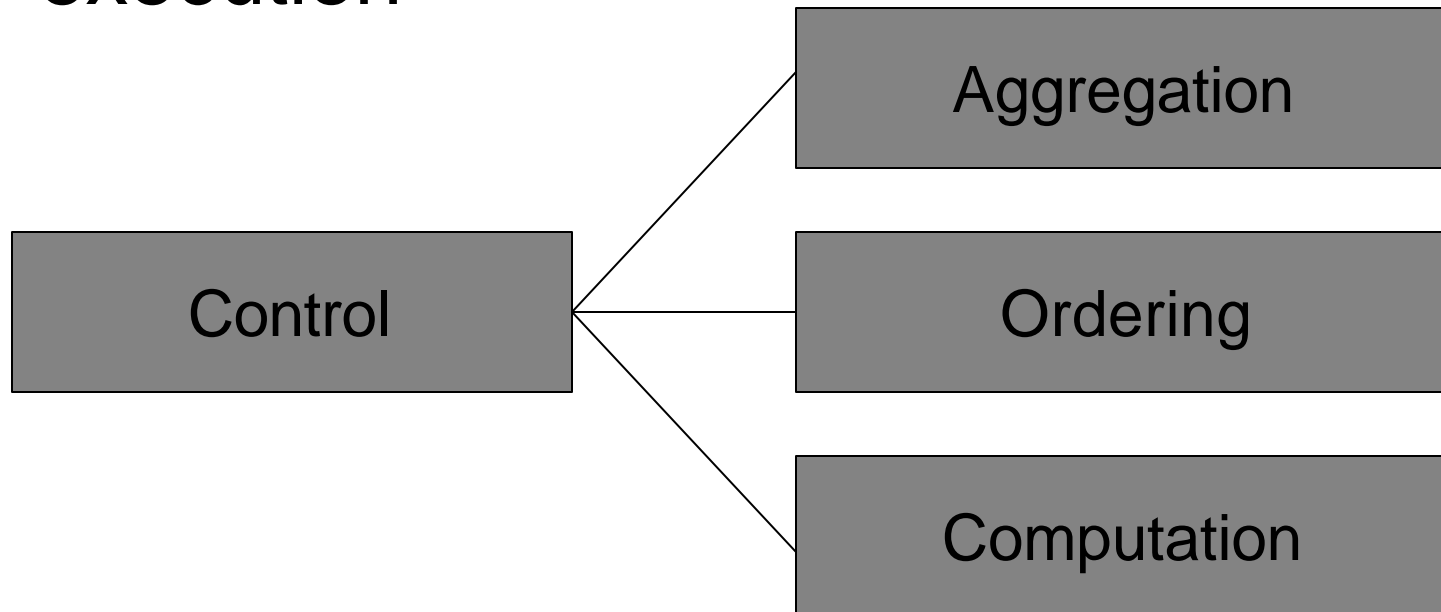
$$C_2 = C_3 \oplus (V_2, M_2)$$

$$V_1 \cap V_3 = \emptyset$$

$$M_1 \cap M_3 = \emptyset$$

Control Obfuscation

- ◆ Aimed at obfuscating the flow of execution



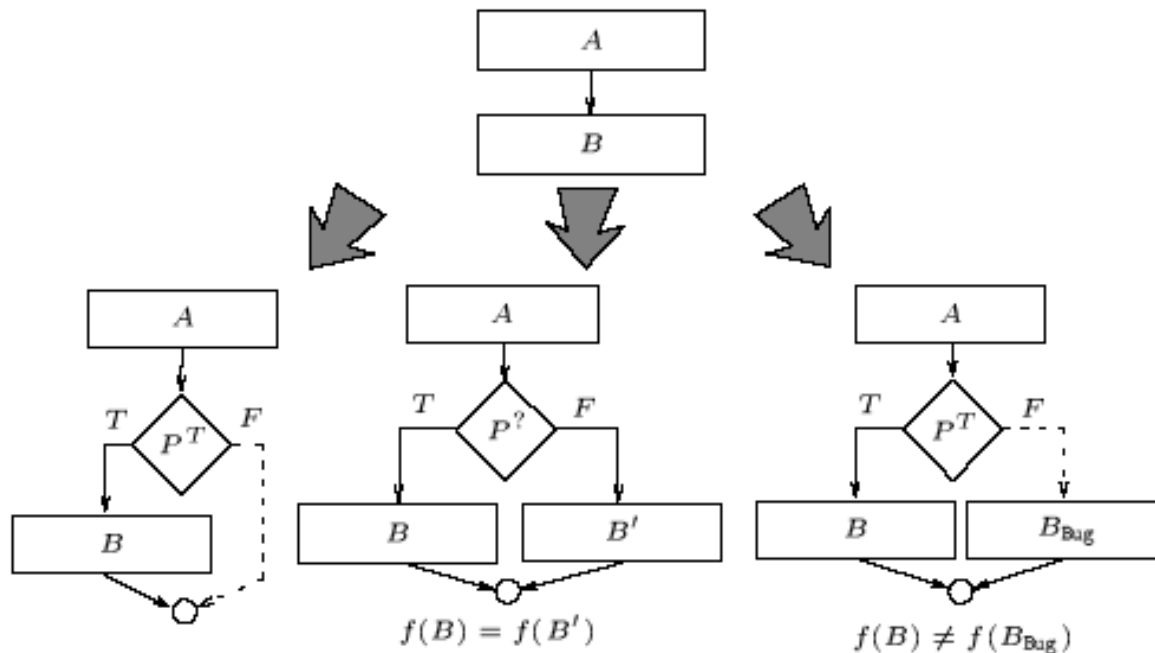
Opaque Construct

- ◆ Obfuscator knows the outcome but very hard for de-obfuscator to predict.
- ◆ Makes it resilient
- ◆ Makes it costly – adding additional code.

```
{  
    int v, a= 5, b= 6;  
    v11 = a + b;  
    if ( b > 5 )T.....  
    if ( random ( 1, 5) < 0 )F ....  
}
```

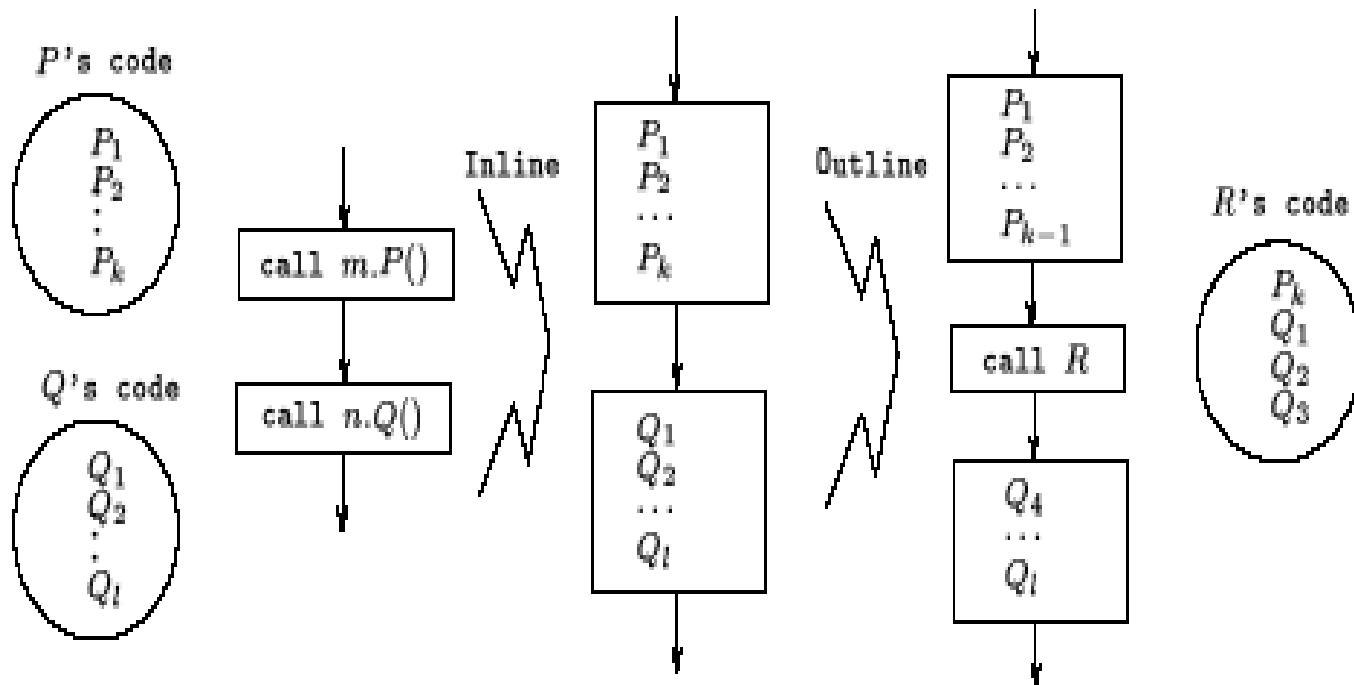
Redundant Code

- ◆ Introducing opaque predicates
- ◆ Introducing multiple obfuscated loops
- ◆ Introducing buggy loops



Inline and Outline

- ◆ Inline: removes procedural abstraction.
- ◆ Outline: creates bogus procedural abstraction.
- ◆ High resilience

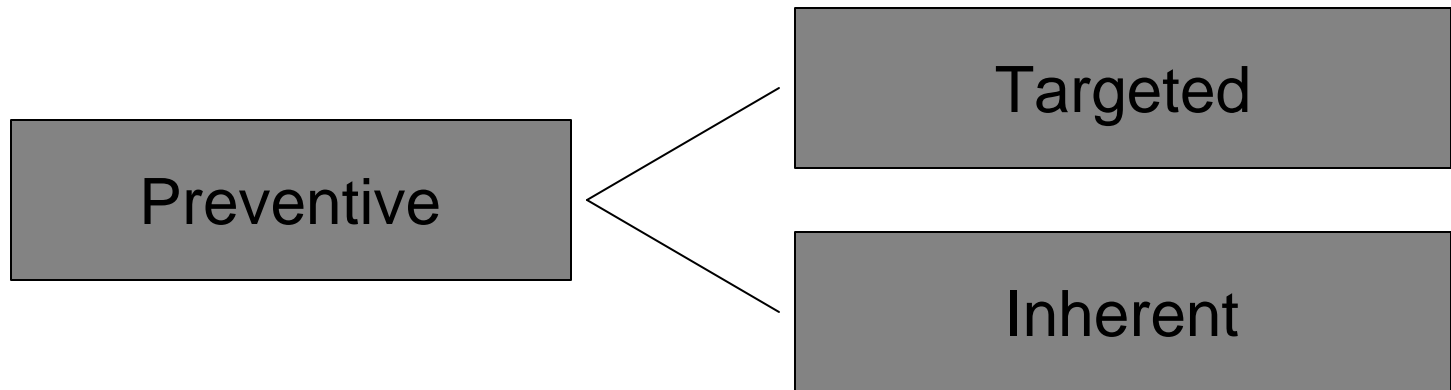


Other Control Obfuscation methods

- ◆ Cloning
- ◆ Interleaving Methods
- ◆ Non-reducible Flow Graphs

Preventive Obfuscation

- ◆ Works by trying to break the known de-obfuscation techniques.



Inherent

- ◆ In the example
 - Added bogus data
 - Prevented de-obfuscator from automatic analysis

Actual Code	Obfuscated Code
<pre>For (i = 1; i<= 10; i++) A[i] = I;</pre>	<pre>Int B[50] For (i=10; i>= 1; i--) { A[i] = I; B[i] += B[*i/2];}</pre>

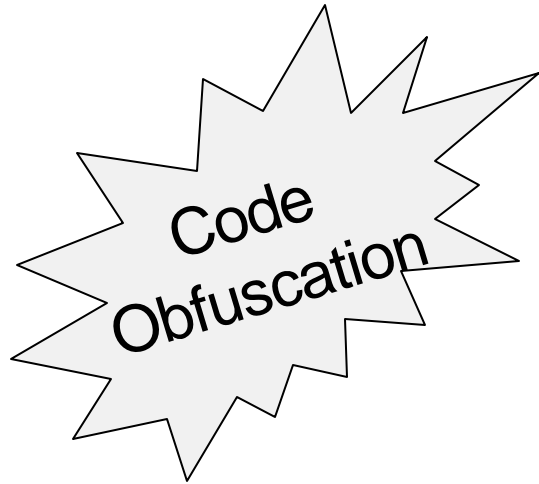
Target

- ◆ It targets specific de-obfuscating programs
- ◆ E.g. HoseMocha example
 - Crashes if we add any code after return statement

Why don't we use obfuscation

- ◆ Cost of obfuscation
 - Execution time
 - High program complexity
 - Effort
 - More \$\$\$
- ◆ Ignorance

Solutions to Malicious Host

A light gray starburst shape with a black outline, containing the text "Code Obfuscation".

Code
Obfuscation

A light gray starburst shape with a black outline, containing the text "Digital Rights Management".

Digital
Rights
Management

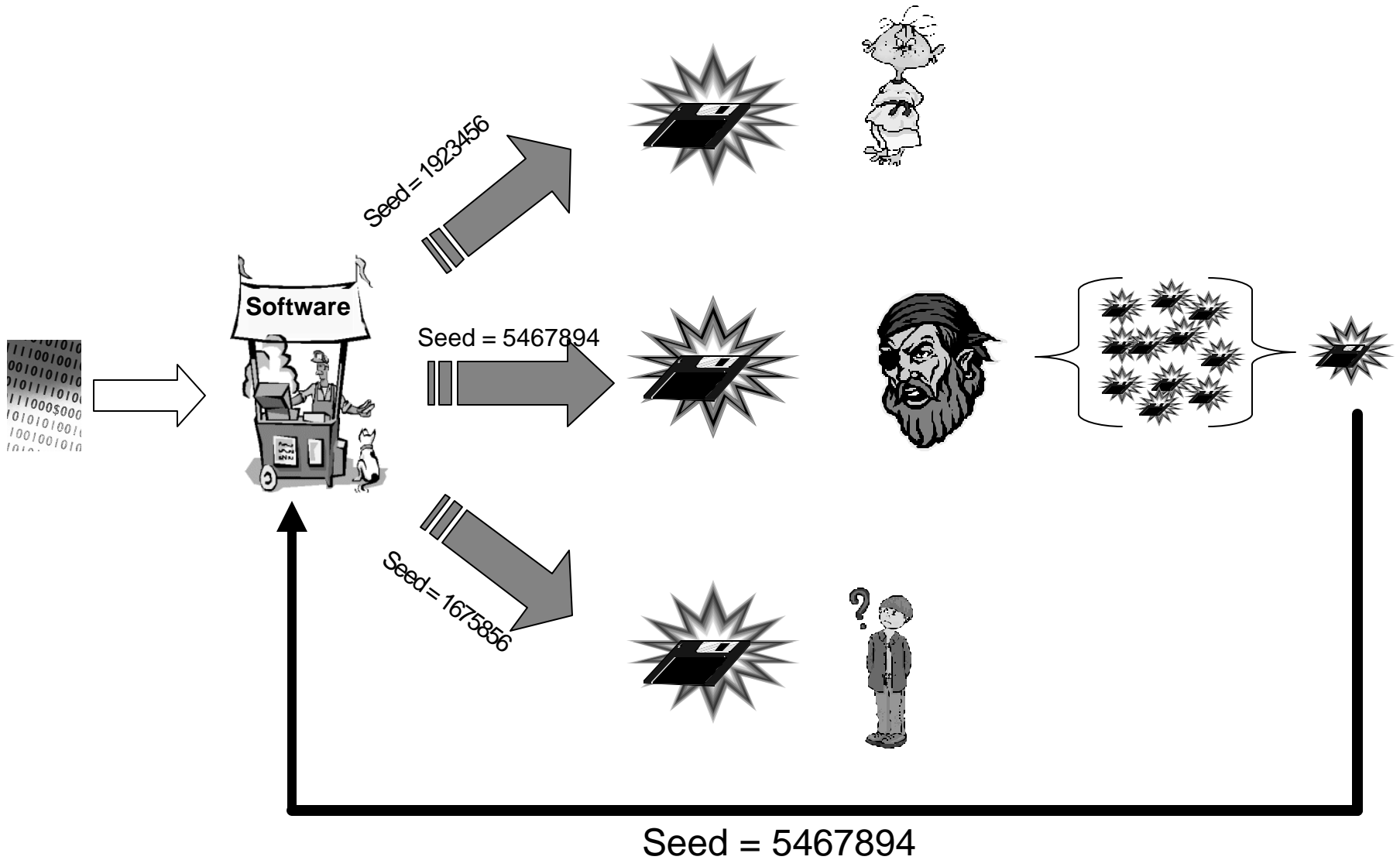
A dark gray starburst shape with a black outline, containing the text "Tamper-Proofing".

Tamper-Proofing

A dark gray starburst shape with a black outline, containing the text "Watermarking".

Watermarking

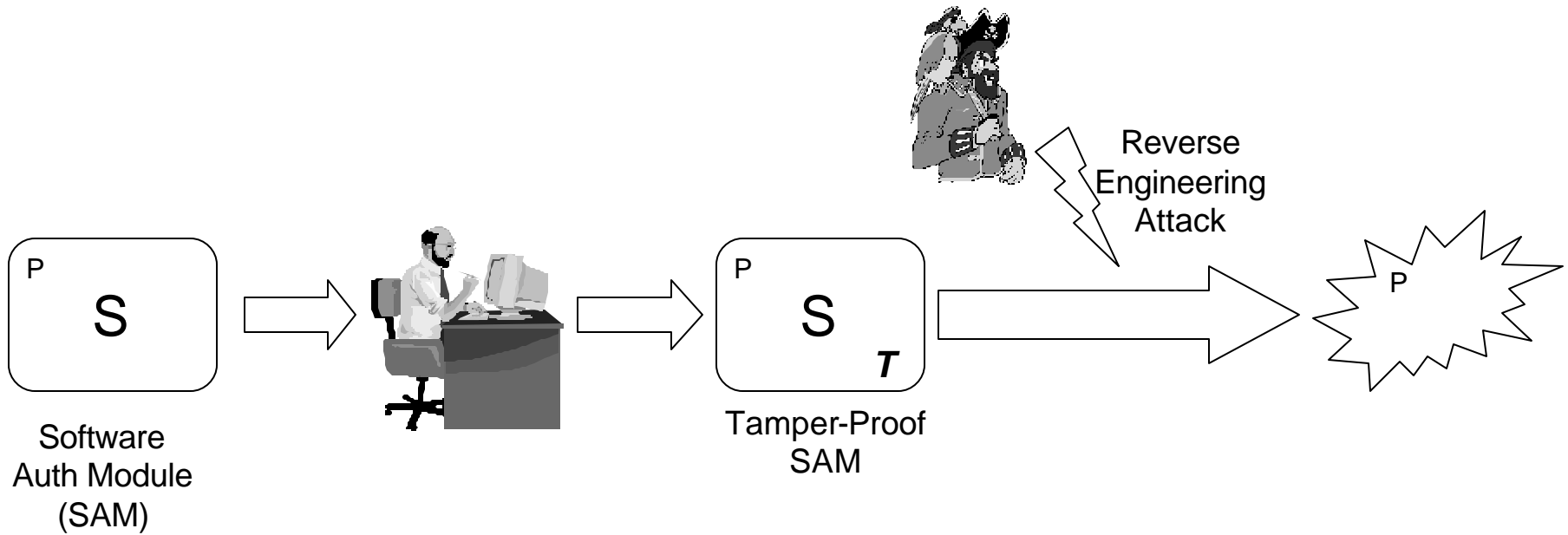
Watermarking



Tamper - Proofing

- ◆ Key component of client-side security
- ◆ User is assumed to have sufficient time, resources and inclination to subvert protection
- ◆ www.cracks.am
- ◆ Demonstrates need for stronger protection against reverse engineering

Tamper-Proofing



Commercial Applications

- ◆ Code Obfuscators
 - Zelix Klassmaster
 - Semantic Design
- ◆ Digital Rights Management
 - Windows RMS Lockbox
 - Apple's iTunes
- ◆ Tamper Proofing
 - Cloakware
- ◆ Watermarking
 - Digimarc

Conclusion

- ◆ Malicious host presents new security threats
- ◆ Code obfuscation makes reverse engineering difficult
- ◆ Watermarking and tamper-proofing address issues of software piracy and tampering respectively
- ◆ Still an inchoative field

References

- [1] Collberg, C., Thomberson, C., Low, D., *A Taxonomy of Obfuscating Transformation*, Technical Report # 148, 1997
- [2] Wang, C., *A Security Architecture of Survivability Mechanisms*, PhD thesis, University of Virginia, October 2000
- [3] Wroblewski, G., *General Method of Program Code Obfuscation*, Wrowclaw, 2002
- [4] Collberg, C., Thomberson, C., *Watermarking, Tamperproofing, and Obfuscation – Tools for Software protection*, 2002

