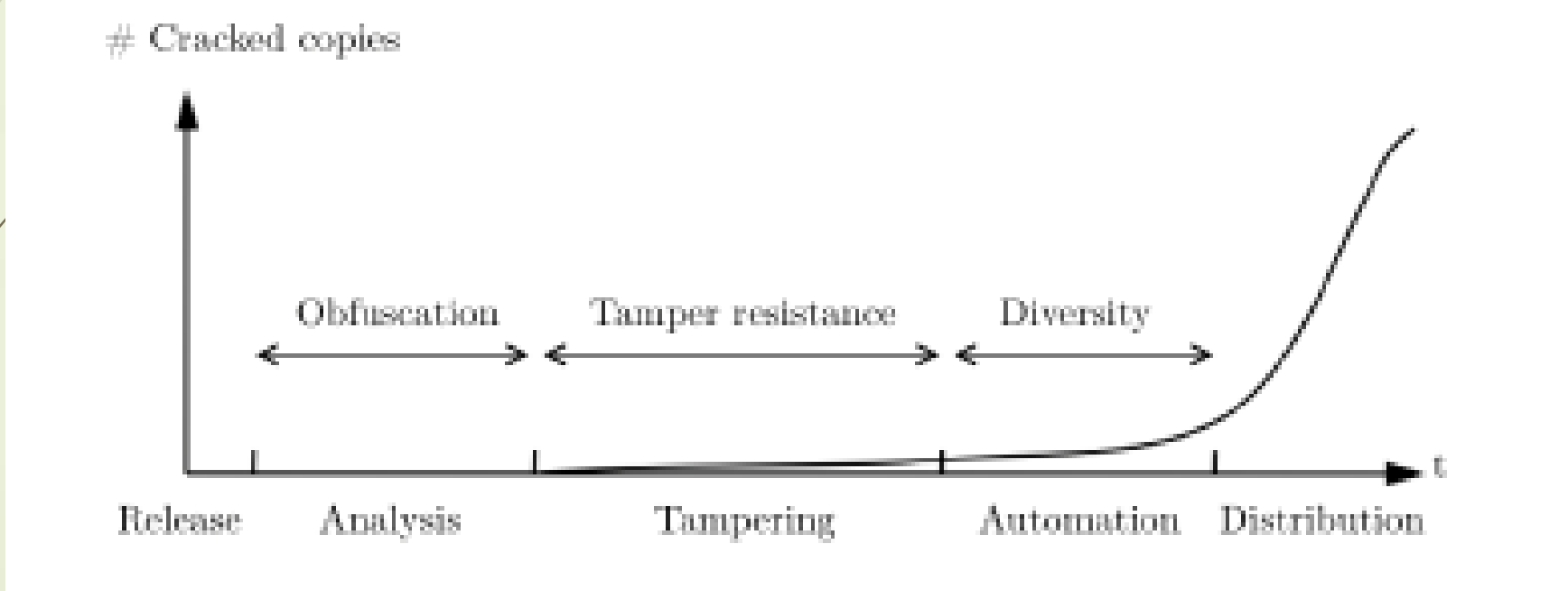
Методы обфускации для защиты от взлома

4 этапа атаки:



Черный и белый ящик

- Черный ящик – система, имеющая некий «вход» информации и некий «выход» для отображения результатов работы, при этом происходящее в ходе работы пользователю неизвестно
- Белый ящик – в этой системе указываются все ее элементы, связи между элементами внутри системы

Оценка потребности в защите:

- Уязвимость
- Значение содержания
- Время жизни содержимого
- Срок службы безопасности

Методы защиты:

- Устаревание ПО
- Водяные знаки и отпечатки пальцев
- Разнообразие
- Сервер-клиент

Методы защиты от анализа

- Метод Колберга
- Криптография белого ящика
- Шифрование кода

Методы защиты от фальсификации

- Подпись
- Стражники
- Забывчивое хеширование
- Виртуализация
- Устойчивость к фальсификации

МЕТОД КОЛБЕРГА

НА ВХОД

- Программа A
- Библиотеки L1, L2…
- Набор трансформирующих процессов T1, T2…
- Функции, определяющие эффективность трансформирующих процессов E1, E2…
- Функции, определяющие важность фрагмента I1, I2
- Определенный фрагмент S
- 2 числовые константы acceptcost>0, requireobfuscation>0

ВИДЫ ПРЕОБРАЗОВАНИЙ

- Лексическое преобразование
- Обфускация управления
- Обфускация данных
- Профилактическое преобразование

КРИПТОГРАФИЯ БЕЛОГО ЯЩИКА

- Идея: преобразование ключа
- Замена L на L':
- Рассмотрим серию таблиц поиска: L1, L2, L3. Мы введем некоторые случайные преобразования M1, M2:

$L1 \longrightarrow M1 \longrightarrow M1^{-1} \longrightarrow L2 \longrightarrow M2 \longrightarrow M2^{-1} \longrightarrow L3$

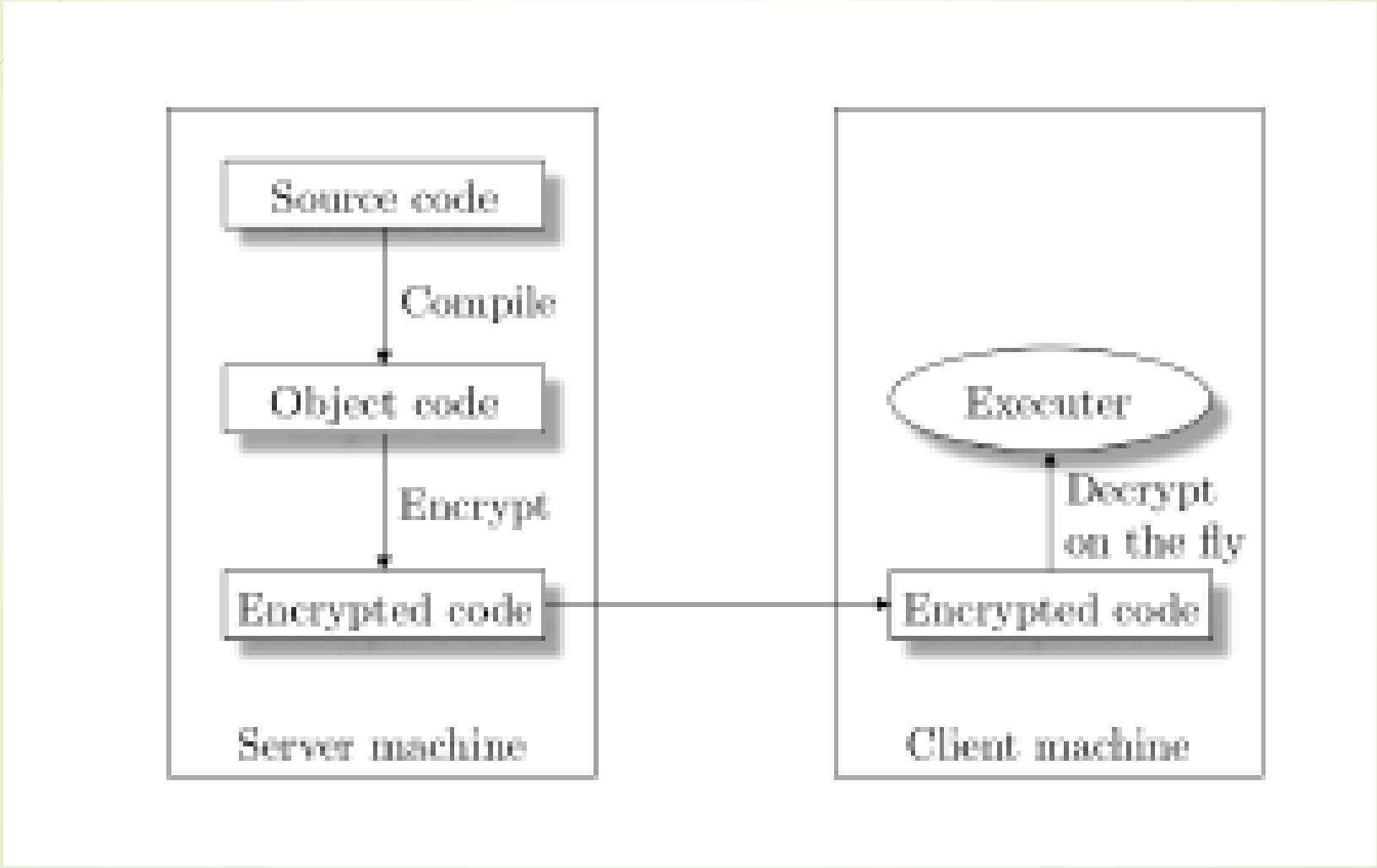
Будем хранить:

$L1' = L1 * M1$

$L2' = M1^{-1} * L2 * M2$

$L3' = L3 * M2^{-1}$

ШИФРОВАНИЕ


Source code
Compile
Object code
Encrypt
Encrypted code
Server machine
Executer
Decrypt
on the fly
Encrypted code
Client machine

ПОДПИСИ

- Владелец может подписать продукт, а пользователь, проверив подпись, добавить этот продукт
- Недостаток – подделка подписи

СТРАЖНИКИ

Стражники- небольшие подпрограммы, которые проверяют целостность кода, при этом они могут восстанавливать измененные участки кода

Тестер – подпрограмма, которая вычисляет хеш-функцию, начиная с некоторого участка кода и сравнивает ее с оригиналом

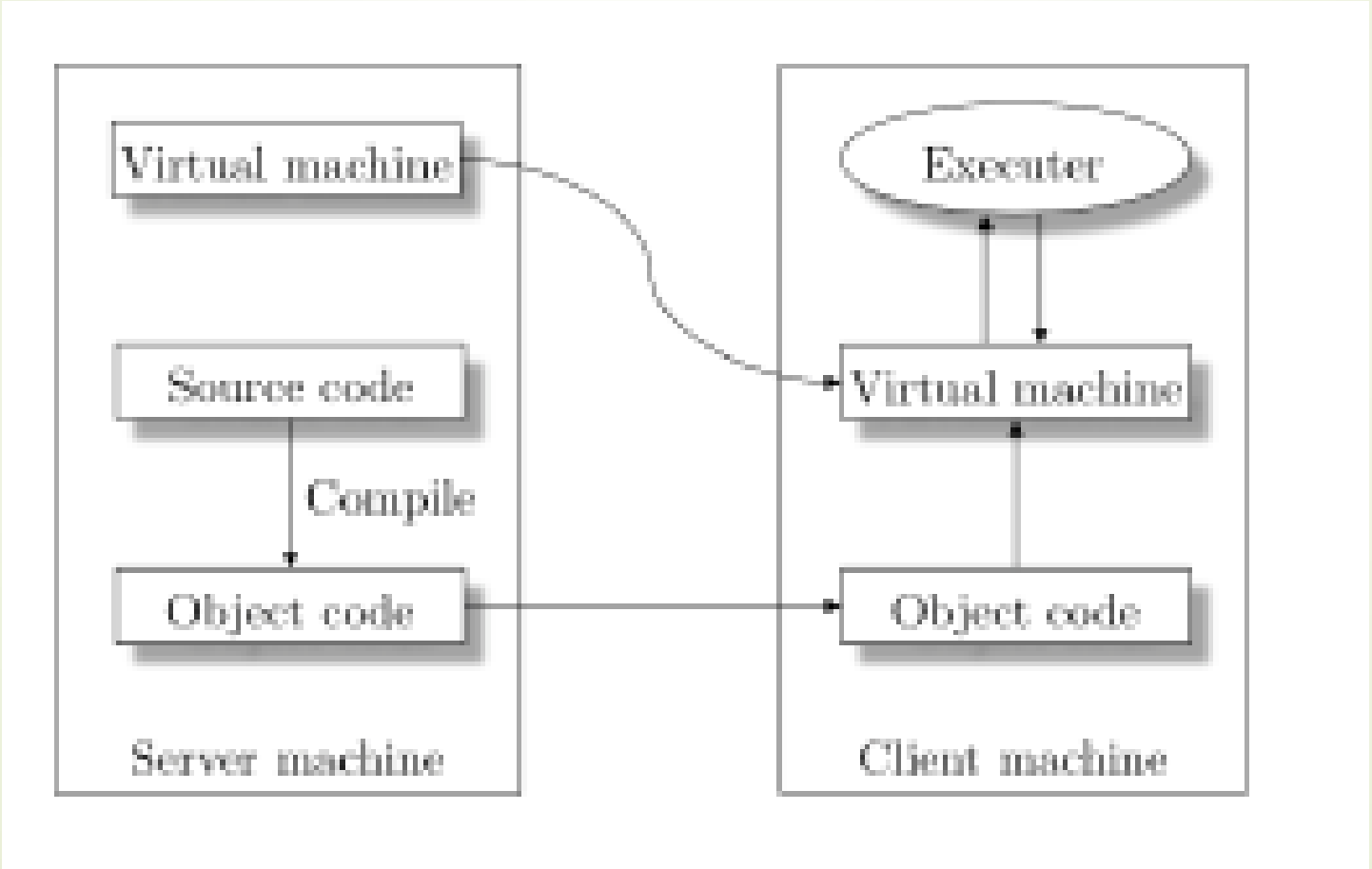
Корректоры необходимы для безопасного хранения оригинального значения хеш-функции

Следят за целостностью лексики, а не семантики

ЗАБЫВЧИВОЕ ХЕШИРОВАНИЕ

Программа представляет собой последовательность инструкций I, которые обращаются для чтения и записи к участкам памяти M , начального состояния памяти m, счетчика инструкций C, и его начального состояния c. Следом некоторого участка программы называется пятерка {I, M, C, m, P}, где P - входной параметр, который влияет на выполнение участка программы. След отражает семантику. Таким образом вычисляется хеш-функция следа

ВИРТУАЛИЗАЦИЯ



УСТОЙЧИВОСТЬ К ФАЛЬСИФИКАЦИИ

- Индивидуализированная модульная избыточность с голосованием
- Индивидуализированная модульная избыточность со случайным выполнением

Оценка методов

	Protection against			
	analysis		tampering	
Technique	static	dynamic	static	dynamic
Collberg's obfuscation transformations	P	P	P	P
White-box cryptography	F	P	F	F
Code encryption	F	P	F	P
Code signing	N	N	F	N
Aucsmith's tamper resistant software	F	P	F	F
Software guards	N	N	P	P
Oblivious hashing	N	N	P	P
Virtualization	P	P	P	P
Tamper tolerant software	N	N	P	P

Защита программного обеспечения от статических нападений

Статический анализ кода - анализ программного обеспечения, производимый (в отличие от динамического анализа) без реального выполнения исследуемых программ. Термин обычно применяют к анализу, производимому специальным ПО.

Реверс-инжиниринг (обратная разработка, обратный инжиниринг) – исследование некоторого устройства или программы, а также документации на него с целью понять принцип его работы.

Обратный инжиниринг ПО производится с помощью

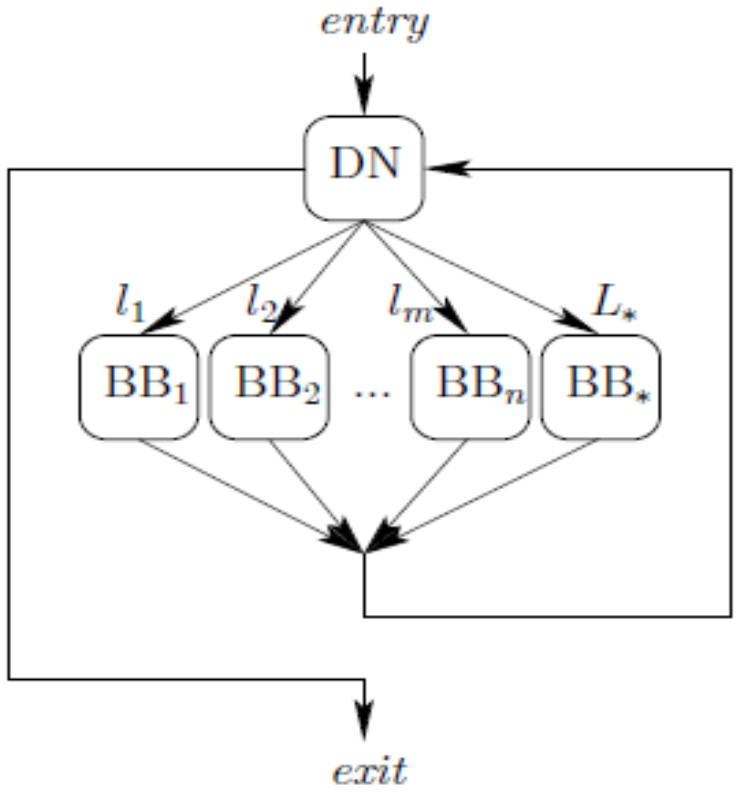
дизассемблирование

декомпиляция

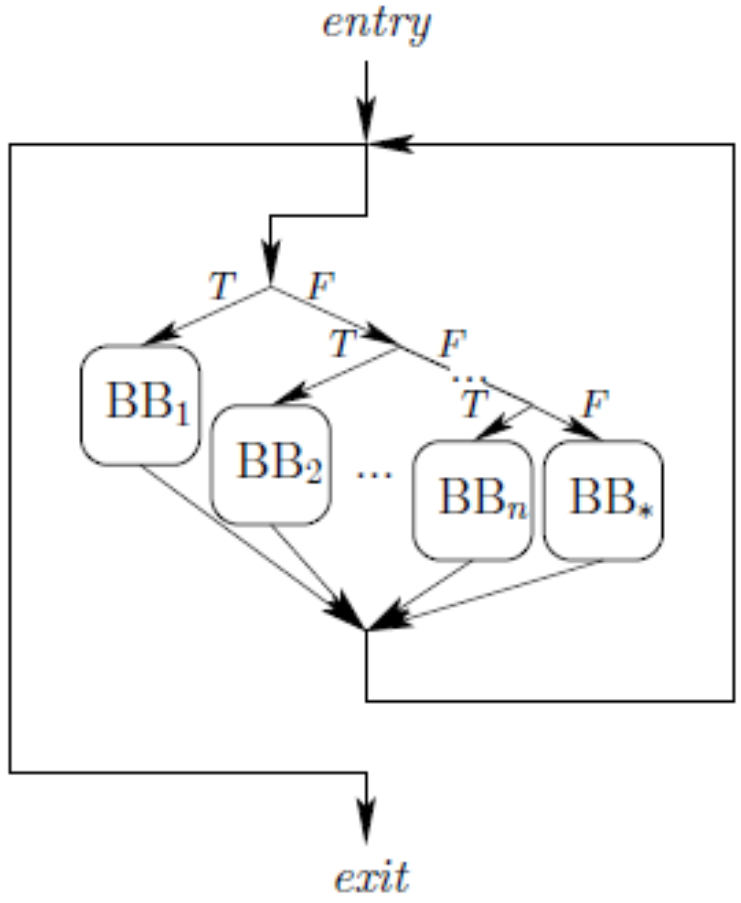
Дизассемблирование - переводит двоичный код в инструкции по сборке, которые соответствуют определенной архитектуре процессора. Например в ассемблер.

Декомпилятор в основном ищет модели, которые могут быть переведены из ассемблера в исходный код. Так как код высокого уровня богаче и более компактный, часто легче понять.

Результаты, приведенные далее направлены на защиту потока управления. Тем не менее, мы расширяем технику, называемую сплющенным графом потока управления, которая по сути переводит проблему потока управления в проблему потока данных. Таким образом, наша техника сосредоточена на данных, а именно защиты статически внедренных данных потока управления.



(a)



(b)

Figure 4.1: (a) A flattened control flow graph. (b) An `if-else if-else if-...` structure, which is equivalent to the flattened program in (a).

```
case 2:
  if (c <= 100)
       swVar = 3;
  else
       swVar = 0;
```

swVar - " переменная переключения "

Определение 1. Локальный анализ - анализ небольшой части кода, расположенного в или вокруг определенного места.

Наша цель, чтобы заставить злоумышленника сделать глобальный анализ, даже если он хочет выполнить локальную атаку.

Шаг 1. Изменение значения жестко запрограммированного потока управления.

```
case 2:
if (c <= 100)
swVar = swVar + 1;
else
swVar = swVar - 2;
```

Простая локальная атака требует глобального анализа со сложностью в худшем случае O (nk) при k число предшествующих блоков, n- число базовых блоков.

Шаг 2: Использование единого унифицированного оператора

```
case 2:
swVar = swVar + 1 - (c <= 100) * 3;
```

$B(x) = x + a + y \times b$ - *функция ветки B()*

Для любых l_s, l_{t1} , l_{t2} , сущ. a, b : $l_{t1} = l_s + a$ и $l_{t2} = l_s + a + b$

Шаг 3: Использование нелокальных значений

Вместо $X_{i+1} = X_i + a + Y \times b$ где X_i локально доступны, мы получим $X_{i+1} = X_{i-1} + a + Y \times b$.

Шаг 4: Сделать значения трудно вычисляемыми

Используем одностороннюю функцию F()

Свойства функции F():

- **Односторонность.**
- **Биективность.**
- **Распространенность.**

Примеры функции F():

- **Дискретный логарифм.**
  Дискретный логарифм над конечным полем GF (p). В GF (p) мы можем использовать F (x) = g^x(mod p), g генератор поля и p большое простое.
- **Криптографические хэш-функции.**
  F: {0, 1}^n -> {0, 1}^n: F (x) = hash(x)
  «Прообраз» - является прообразом хэш значения Y = F (x): F^-1 (y) = x.

Подводя итог, наша схема выглядит следующим образом:

• «проваливающийся» основной блок BB* содержит:

- Новую переменную z, которая хранит значения времени выполнения переменной переключателя (которые приводят к BB* и которая не жестко запрограммированы в коде): z = x, и

- Вызов к нашей переходной функции: x = F (z).

• все другие основные блоки BBi содержат:

- Конкретизация B () с помощью предварительно вычисляемых a и b. Это B () работает на "метку времени", хранящейся в z, которая является со статической точки зрения секретным значением: x = z + ai + yi ×bi. Помните, что значение метки в z есть прообраз метки, которая используется для передачи управления BBi.