ソフトウェア工学III

ソフトウェアプロテクション（3）

～ソフトウェアの耐タンパー化技術～

ソフトウェア工学講座
門田暁人
akito-m@is.naist.jp
B303室，内線5311

ソフトウェアプロテクションの目的

- ソフトウェアの再利用（盗用）を防ぐ．
- ソフトウェア内部の秘密（データ，アルゴリズム）を隠す．
- ソフトウェアの改ざんを防ぐ．
- 要素技術
  - 暗号化，難読化，多様化(Software Diversity)
  - アクセス制御
  - アンチデバッガ，アンチ逆アセンブラ
  - 電子透かし，バースマーク
- 法律
  - 著作権，特許，不正競争防止法

プロテクションの三つの側面

- Prevention（攻撃の防止）
  - Tamper proofing（耐タンパー化）
    - TRS (Tamper Resistant Software): 改ざんしにくいソフトウェア，改ざんすると動作しなくなるソフトウェア．要素技術として，暗号化，難読化が用いられる．
    - 理解（解析）できなければ意味のある改ざんもできない．
- Detection（攻撃の検出）
  - Integrity verification（改ざんの有無の検証）
    - Tamper detectionとも呼ばれる．
  - デバッガの検出
- Response（攻撃に対する応答）
  - ソフトウェアの実行停止（遅延）

ソフトウェアに含まれる秘密

- アルゴリズムやモジュール: デジタルコンテンツの著作権管理（DRM)システムのアルゴリズムなど
- データ（定数）: 暗号鍵や携帯電話のデバイスIDなど
- 条件判定式: ライセンスチェックのための分岐文
  “if (licensed) goto L”.
- 外部インタフェース: メンテナンス用の隠しモード

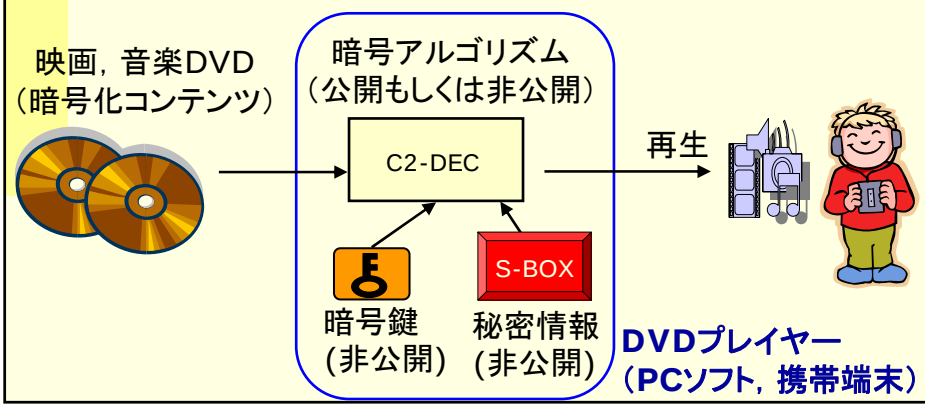
ソフトウェア内部の秘密漏洩の事例

- DVDの復号鍵の漏洩[1]
- 携帯電話のデバイスID（IMEI）の書き換えによるクローン携帯の流通[2]
- Microsoft Windows/Office XPのProduct Activationを無効化するパッチの流通[3]
- Borland Interbase SQLのバックドアの漏洩[4]

[1] A.Patrizio, “Why the DVD hack was a cinch,” Wired News, Nov. 1999.
[2] “Mobile phone crime blitz launched,” BBC News, 17Dec. 2003.
[3] A.Patrizio, “XP cracks appear before product,” Wired News, Sep. 2001.
[4] J.Havrilla, “Boland/Inprise Interbase SQL database server contains backdoor superuser account with known password,” US-CERT, Vuln. Note, VU#24731, 2001.

保護すべきソフトウェアの例(1)

- DVDプレイヤーソフト
  - 鍵や秘密情報がプレイヤーソフトに埋め込まれている.
  - 暗号が破られると生コンテンツがインターネットに流出する恐れがある.（P2Pソフトによる流通）



保護すべきソフトウェアの例(2)

- 携帯電話，セットトップボックスなどの組み込みソフトウェア
  - バグ修正のためのプログラムのアップデートが可能
  - 不正なユーザによるプログラムの書き換えを防ぐのは難しい．
  - プログラムを解析困難にしたり，プログラムの改ざんを検出する技術が必要
- パスワードやシリアルナンバーを入力させるソフトウェア
  - パスワードチェックやシリアルナンバーチェックを行う部分が改ざんされる．
- 価値のあるアルゴリズムを採用しているソフトウェア
  - 将棋，囲碁
- 価値のあるデータを含むソフトウェア
  - プリンタドライバ（カラーマップ）
- 無断で再利用されたくないソフトウェア（自社開発ライブラリ等）

ソフトウェア保護技術

- ソフトウェア単体での保護
  - 難読化
  - 暗号化
  - 自己書き換え
- ハードウェア支援による保護
  - プログラムの論理回路化
  - ワンチップ化（CPU+ROM）
  - ドングル方式
  - バス暗号方式
  - IBM, Intel, Microsoft等によるTrusted Computing構想[1]

[1] The Trusted Computing Group, “TPM Main, Part 1: Design Principles,” Specification Version 1.2, Oct. 2003.

プログラムの難読化（obfuscation）

```
int n = 52;
int i, k, p=1;

for(i=1;i<=31;i++)
{
   k = n – i + 1;
   p = p * k / i;
}
return p;
```

等価変換

```
int n=105,k,i=1,p=1;
L1: if(i <= 31){ for(;;){
k=n–2*i+2;p=(p*k-p)/2/i;
if(++i>31){k=n–2*i+2;
p=(p*k-p)/2/i++; }else
break;
p=p*(n–2*i+1)/2/i++;}
goto L1;}
return p;
```

元のプログラム　　難読化したプログラム

・プログラムの制御構造やデータ構造を複雑にする.
- 仕様を変えない.
- 実行効率を落とさない.

プログラムの難読化の技術動向（特許・商品）

・変数名（識別子）の難読化
  - 数多くの製品（Java, .NET用）PreEmptive社が有名
・フローチャートの難読化
  - 数多くの論文，Cloakware社の特許[1]，商品化されていない.
・データの難読化
  - NAISTの特許[2]，Cloakware社の特許＆サービス[3],
・その他:InterTrust[4]，PreEmptive[5]，Apple Computer[6]など

データの難読化は今後有望である.
プログラム中の暗号鍵や重要なデータを隠すのに役立つ.

[1] *United States Patent* 6,779,114, Cloakware, Aug. 2004.
[2] 特願2003-202795, 29July 2003.
[3] *United States Patent* 6,594,761, Cloakware, July 2003.
[4] *United Sates Patent* 6,668,325, InterTrust, Dec. 2003.
[5] *United States Patent* 6,102,966, PreEmptive Solutions, Aug. 2000.
[6] *United States Patent* 6,694,435, Apple Computer, 17 Feb. 2004.

データ構造の難読化 ---単純な方式(1)

```
(1) int A[10]
(2) A[i] = …;

(3) int B[10], C[20];
(4) B[i] = …;
(5) C[i] = …;
```

難読化

```
(1) int A1[5], A2[5];
(2) if((i%2)==0) A1[i/2]=…;
      else A2[i/2]=…;
(3) int BC[30];
(4) BC[3*i] = …;
(5) BC[i/2*3+1+i%2] = …;
```

	0	1	2	3	4	5	..	9
A:	a_0	a_1	a_2	a_3	a_4	a_5	..	a_9
B:	b_0	b_1	b_2	b_3	b_4	b_5	..	b_9

	0	1	2	3	4	5	..	19
C:	c_0	c_1	c_2	c_3	c_4	c_5	..	c_{19}

	0	1	2	3	4
A1:	a_0	a_2	a_4	a_6	a_8
A2:	a_1	a_3	a_5	a_7	a_9

	0	1	2	3	4	5	..	29
BC:	b_0	c_0	c_1	b_1	c_2	c_3	..	c_{19}

C. Collberg, C. Thomborson, and D. Low, “Obfuscation techniques for enhancing software security,” United Sates Patent 6,668,325, Assignee: InterTrust Inc., Filed 9 June 1998, Issued 23 Dec. 2003.

データ構造の難読化 ---単純な方式(2)

```
(1) int D[10];
(2) for(i=0; i<=9; i++)
      D[i] = 2*D[i+1];

(3) int E[3][3];
(4) for(i=0;i<=2;i++)
      for(j=0;i<=2;j++)
        swap(E[i][j],E[j][i]);
```

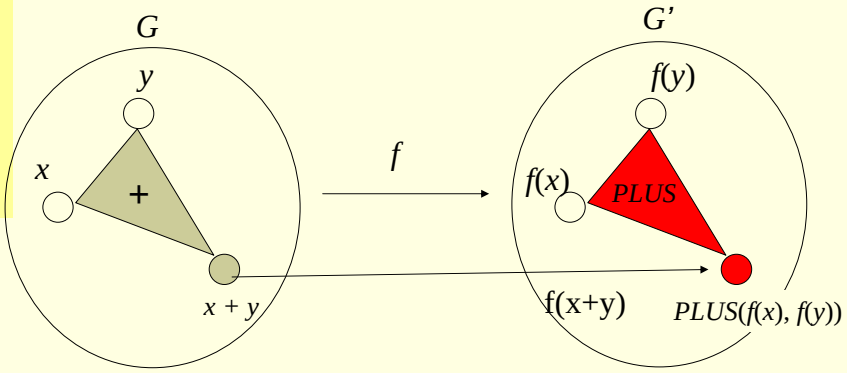
難読化

```
(1) int D1[2][5];
(2) for(j=0; j<=1; j++)
      for(k=0; k<=4; k++)
        if(k==4) D1[j][k]=2*D1[j+1][0];
        else D1[j][k]=2*D1[j][k+1];

(3) int E1[9];
(4) for(i=0;i<=8;i++)
      swap(E1[i], E1[3*(i%3)+i/3]);
```

C. Collberg, C. Thomborson, and D. Low, “Obfuscation techniques for enhancing software security,” United Sates Patent 6,668,325, Assignee: InterTrust Inc., Filed 9 June 1998, Issued 23 Dec. 2003.

データ難読化 ---準同型写像の利用(1)

2つの群G, G'と写像 $f: G \rightarrow G'$与えられた時，任意の$x,y \in G$に対して $f(x + y) = PLUS(f(x), f(y))$ を満たす演算子$PLUS$が存在する（つまり$PLUS$を効率よく計算するアルゴリズムが存在する）場合，写像 f は準同型であるという．

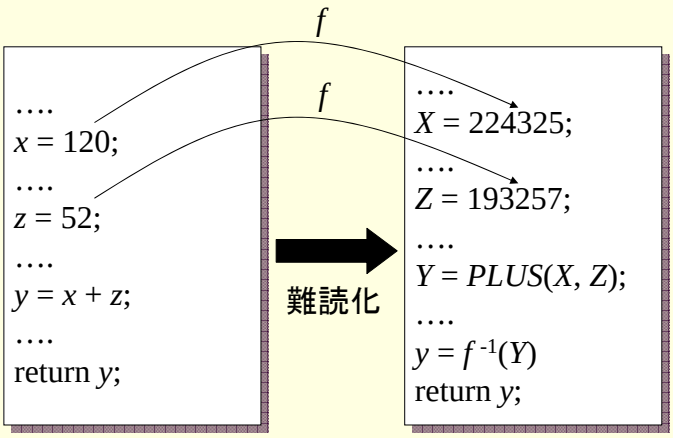


データ難読化 ---準同型写像の利用(2)

準同型写像 f を利用したプログラムの難読化

難読化の手順

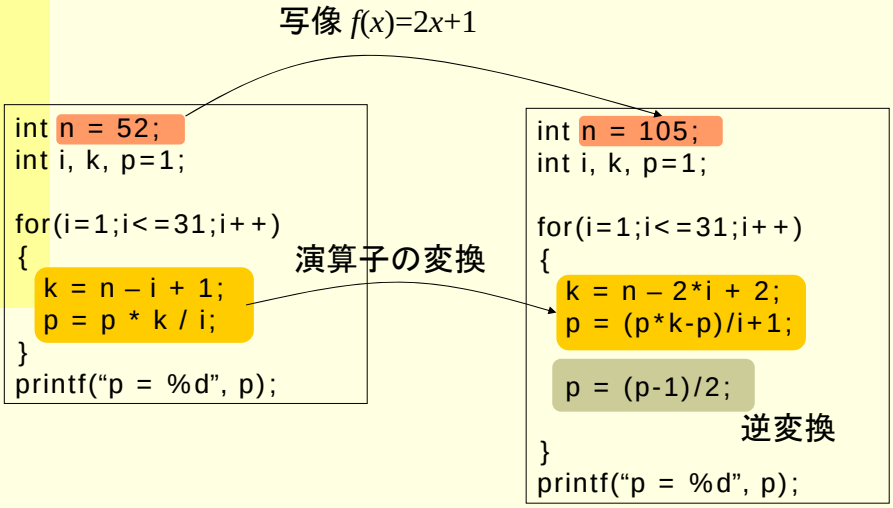
① 定数の変換
② 演算子の変換
③ 逆変換の追加



一次式 $f(x)=ax+b$ （$x \in \mathbf{Z}$, $a,b \in \mathbf{N}$）は，加減乗除に関して準同型

データ難読化 ---準同型写像の利用(3)

難読化の例



データ難読化 ---準同型写像の利用(4)

Let $f(x) = ax+b$ ($x \in \mathbf{Z}$, $a,b \in \mathbf{N}$), then

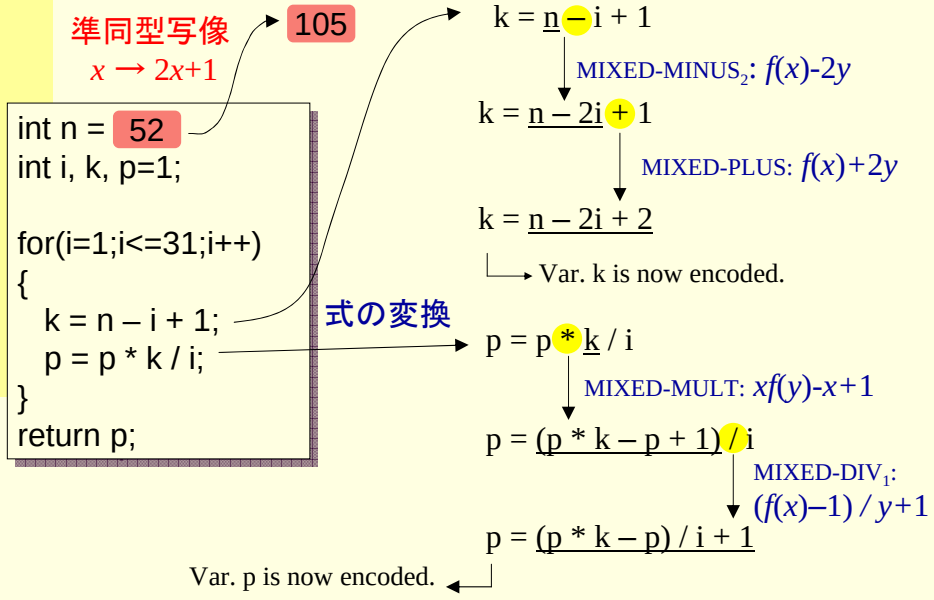
演算子変換規則

$f(x + y) = \mathrm{PLUS}(f(x), f(y)) = f(x) + f(y) - b$
$f(x - y) = \mathrm{MINUS}(f(x), f(y)) = f(x) - f(y) + b$
$f(xy) = \mathrm{MULT}(f(x), f(y)) = (f(x)f(y) - b(f(x) + f(y) - b - a)) / a$
$f(x / y) = \mathrm{DIV}(f(x), f(y)) = (af(x) + bf(y) - b(b + a)) / (f(y) - b)$

補足的な規則

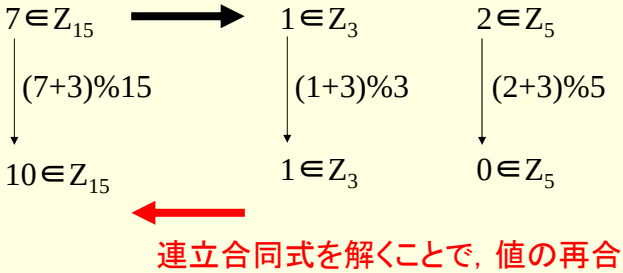
$f(xy) = \mathrm{MIXED\text{-}MULT}(x, f(y)) = xf(y) - bx + b$
$f(x + y) = \mathrm{MIXED\text{-}PLUS}(f(x), y) = f(x) + ay$
$f(x - y) = \mathrm{MIXED\text{-}MINUS}_1(x, f(y)) = ax - f(y) + 2b$
$f(x - y) = \mathrm{MIXED\text{-}MINUS}_2(f(x), y) = f(x) - ay$
$f(x / y) = \mathrm{MIXED\text{-}DIV}_1(f(x), y) = (f(x) + b) / y + b$
$f(x / y) = \mathrm{MIXED\text{-}DIV}_2(x, f(y)) = aax / (f(y) - b) + b$

データ難読化 ---準同型写像の利用(5)



データ難読化 ---中国人剰余定理の利用(1)

- 互いに素である自然数$m_1,\ldots, m_n$の積$M = m_1 \times m_2 \times \ldots m_n$を法とする剰余類$Z_M$={0,1,2,…, M-1}の各要素が，$m_1,\ldots,m_n$をそれぞれ法とする剰余類Z_{m1}={0,1,2,…,m_1-1},…, Z_{mn}={0,1,2,…,m_n-1}の要素の組に一意的に対応付けられる.
- それぞれの剰余類内で演算（和または積）を行った場合にもこの対応付けが保存される.



連立合同式を解くことで，値の再合成が可能.

データ難読化 ---中国人剰余定理の利用(2)

難読化の例

```
int func(int c){
    int x = 32;
    return c + x;
}
```

難読化

```
int func(int c){
    int c1, c2, x1 = 6, x2 = 13;
    c1 = (c + x1) % 13;
    c2 = (c + x2) % 19;
    return decode(c1, c2);
}

int decode(int a, int b){
    int x = (a+13*(b-a)*3)%247;
    if(x < 0) x+= 247;
    return x;
}
```

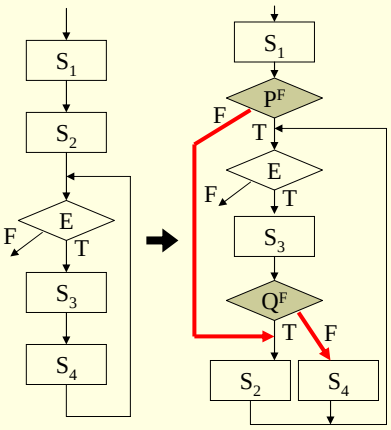
制御フローの難読化 ---opaque predicateの利用(1)

Opaque predicate（不明瞭な述語）とは

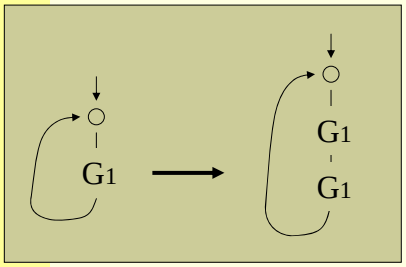
- 整数xとyに対する $7*y*y - 1 == x*x$ のような，恒偽（または恒真）となる式のこと．（常に同じ値をとる式）
- 一見したところでは，真となるのか偽となるのか不明．
  すなわち，”opaque”である．
- 条件分岐 if $(7*y*y - 1 == x*x)$ … else … は必ずelse節が実行される．
- このような条件分岐をダミーでたくさん入れることで，制御フローを難読化できる．

制御フローの難読化 ---opaque predicateの利用(2)

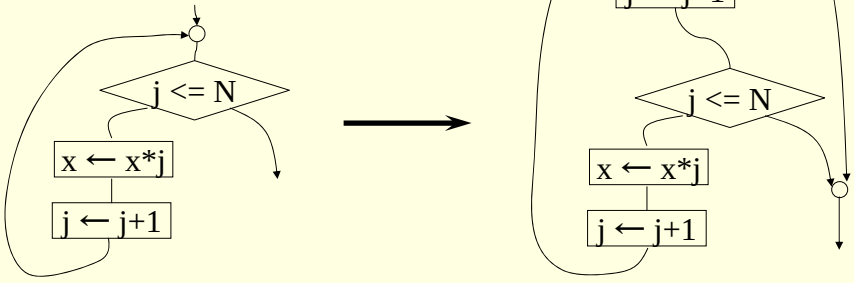
難読化の例



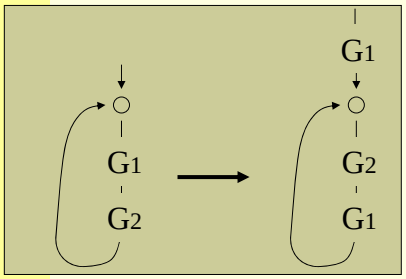
制御フローの難読化 ---ノードの複製(1)

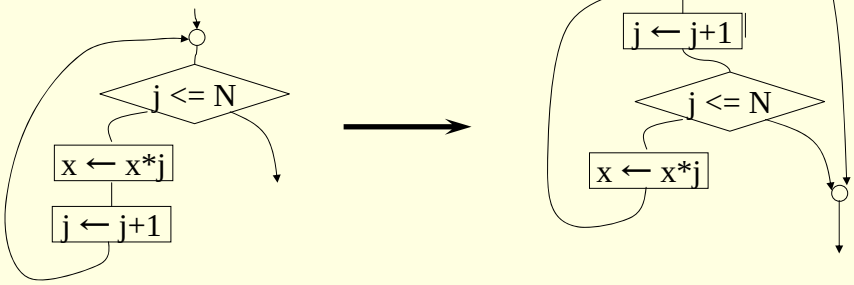


・G1から出ている辺は複製後に合流させる.
・G1へ入ってる辺は，複製したノードのどちらか一方へ入れる.



制御フローの難読化 ---ノードの複製(2)





制御フローの難読化 ---ノードの複製(3)

難読化の例

```
x = 1;
for(j = 1; j <= N; j++)
{
   x = x ＊ j;
}
```

```
x = 1;
j = 1;
L1: if(j <= N){
   x = x ＊ j;
   j++:
   for(;;){
      if(j <= N){
         x = x ＊ j;
         j++;
      }else break;
      if(!(j <= N)) break;
      x = x ＊ j;
      j++;
      goto L1;
   }
}
```

制御フローの難読化 ---処理の順序の複雑化(1)

- 処理の順序を変える.
- 実行される演算回数は変えない.

```
x = 1;
for(j = 1; j <= N; j++)
{
   x = x ＊ j;
}
```

x = 1＊2＊3＊4＊5

(N = 5のとき)

```
x = 1;
j = 1;
for(;;){
   if(j > N) break;
   x = x ＊ j;
   j++;
   if(j > N) break;
   x = x ＊ N;
   N--;
}
```

x = 1＊5＊2＊4＊3

制御フローの難読化 ---処理の順序の複雑化(2)

```
x = 1;
for(j = 1; j <= N; j++)
{
   x = x ＊ j;
}
```

x=1
j=1
j<=N
x=x*j
j++
j<=N
x=x*j
j++
j<=N
x=x*j
j++
j<=N
x=x*j
j++
j<=N
x=x*j
j++
j<=N
end

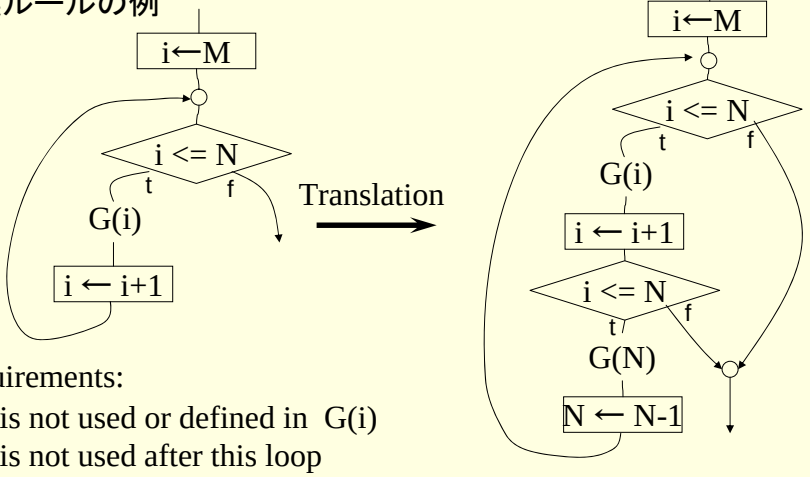
```
x = 1;
j = 1;
for(;;){
   if(j > N) break;
   x = x ＊ j;
   j++;
   if(j > N) break;
   x = x ＊ N;
   N--;
}
```

x=1
j=1
j > N
x=x*j
j++
j > N
x=x*N
N--
j > N
x=x*j
j++
j > N
x=x*j
j++
j > N
x=x*N
N--
j > N
end

(N = 5のとき)

制御フローの難読化 ---処理の順序の複雑化(3)

変換ルールの例



Requirements:

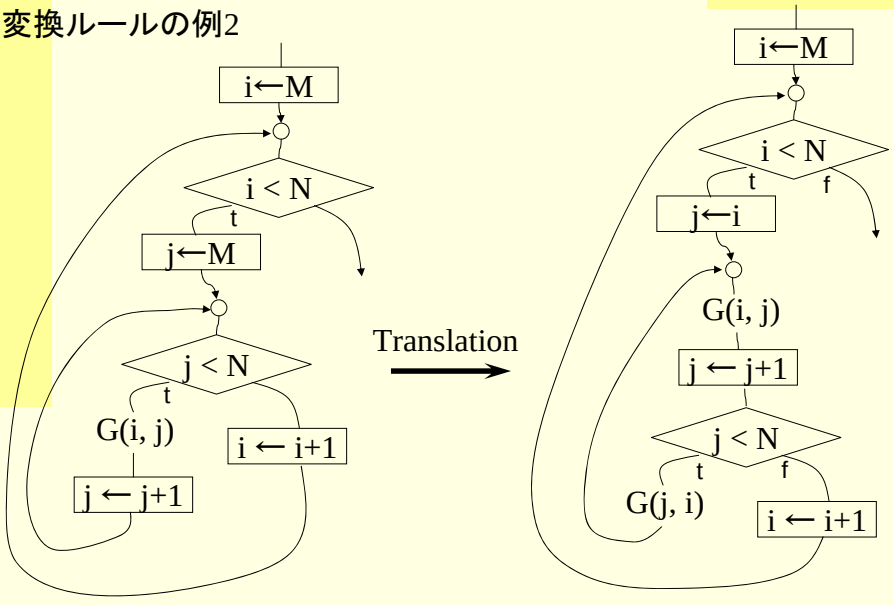
- N is not used or defined in G(i)
- N is not used after this loop
- Execution order of G(i) is changeable

  e.g. - There is no node between G(i) and outside of this loop

  - No variable is both used and defined in G(i)

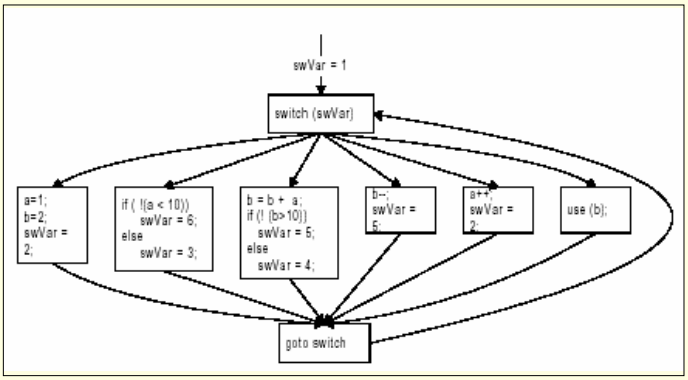
制御フローの難読化 ---処理の順序の複雑化(4)

変換ルールの例2



制御フローの難読化 ---平坦化(1)

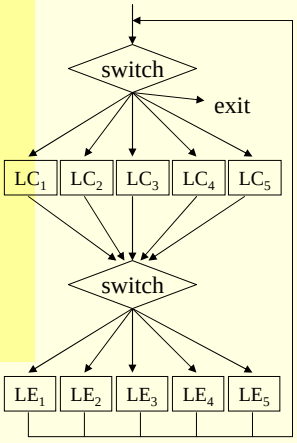
```
int a, b;
a=1;
b=2;
while(a<10){
    b=a+b;
    if(b>10)
        b--;
    a++;
}
use(b);

(a)
```



難読化

C. Wang, J. Hill, J. Knight, and J. Davidson, “Protection of software-based survivability mechanisms,” *Proc. International Conference on Dependable Systems and Networks*, pp. 193-202, July 2001.

制御フローの難読化 ---平坦化(2)



複数の基本ブロックを融合させ，ダミーの文を混在させた基本ブロック（LE_1～LE_5）を生成し，さらに，変数変換テーブル（LC_1～LC_5）を導入することで，各基本ブロックの動作を解析困難する．

S. Chow, H. Johnson, and Y. Gu, “Tamper resistant con-trol-flow encoding,” United States Patent 6,779,114, Filed 19 Aug. 1999, Issued 17 Aug. 2004.

制御フローの難読化 ---大域的な方法 (1)

関数の分割

```
func(){
  int a, b;
  …
  if (a > b)
    a = b;
  …
  b = a + 1;
}
```


```
int a, b;
…
func1(){ a = b; }
func2(){ b = a + 1; }
func(){
  …
  if (a > b)
    func();
  …
  func2();
  …
}
```

T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji, “Software obfuscation on a theoretical basis and its implementation,” IEICE Trans. on Fundamentals, Vol.E86-A, No.1, pp. 176-186, 2003.

制御フローの難読化 ---大域的な方法 (2)

関数の合成

```
func1(){…}
func2(){…}

func(){
  …
  func1();
  …
  func2();
  …
}
```


```
int sw;
func1(){…}
func2(){…}
func3(){ …
  switch(sw){
  case 0: func1(); break;
  case 1: func2(); break;
  …
  }
}
func(){ …
  sw = (sw-1)*sw%2;
  func3(); …
  sw = sw*sw*(sw+1)*(sw+1)%4+1;
  func3(); …
}
```

T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji, “Software obfuscation on a theoretical basis and its implementation,” IEICE Trans. on Fundamentals, Vol.E86-A, No.1, pp. 176-186, 2003.

オブジェクト指向言語の難読化 ---オーバーロード誘導

```
private void CalcPayroll(SpecialList employeeGroup) {
  while (employeeGroup.HasMore()) {
    employee = employeeGroup.GetNext(true);
    employee.UpdateSalary();
    DistributeCheck(employee);
  }
}
```

```
private void a(a b) {
  while (b.a()) {
    a = b.a(true);
    a.a();
    a(a);
  }
}
```

クラス名，メソッド名を出来る限り
同じ名前にする（オーバーロードする）

全メソッドの約1/3が a() という名前になる.

P. M. Tyma, “Method for renaming identifiers of a computer program,” United States Patent 6,102,966, Assignee: PreEmptive Solutions, Inc., Aug. 2000.

マシン語の難読化 ---プログラムコードの均一化

MOVSB →

高レベル命令

```
MOV AL, [SI]
MOV [DI], AL
INC SI
INC DI
```

低レベル命令

・高レベル命令を低レベル命令へと置き換える.
・可能な限り命令を入れ替えて混ぜる.

マシン語の難読化 ---プログラムの自己書き換え(1)

```
int n = 52;
int i, k, p=1;

for(i=1;i<=31;i++)
{
   k = n – i + 1;
   p = p * k / i;
}
return p;
```

コンパイル，
自己書き換え
処理の追加

```
cmpl    $123,-4(%ebp)
addl    %edx, %eax
movb    0x03, L15
ret
pushl   %ebp
addl    (%esp), %eax
xorl    $12, %esp
movb    0x06, L6
shrl    $8, %eax
subl    $2
andl    $1
addl    (%
inc     %eax
pushl   %ebp
cmpl    $123,-4(%ebp)
addl    %edx, %eax
movl    %eax, %ebx
movb    0x03, L15
pushl   %ebp
addl    (%esp), %eax
movb    0x03, L13
```

書き換え
書き換え
書き換え
書き換え

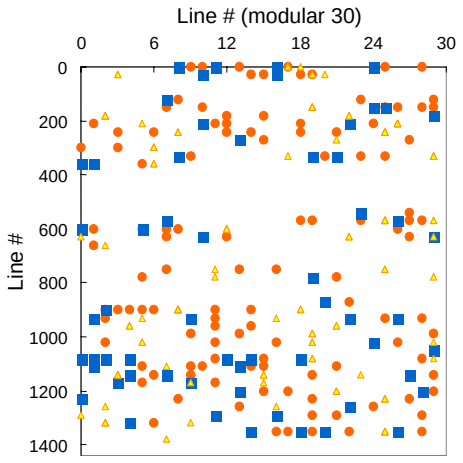
元のプログラム　　自己書き換えプログラム

・自分自身を常に書き換えながら動作するプログラム[1]
 - 命令文，データを隠蔽できる

[1] 特願2002-355881, Dec. 2002.

マシン語の難読化 ---プログラムの自己書き換え(2)

偽の命令で上書きしておき，実行時に元来の命令に戻す．



対象アセンブリプログラム

- 全体の命令数 : 1000
- 偽の命令数: 130

- 偽の命令
- 自己書き換えルーチン（復帰）
- 自己書き換えルーチン（隠蔽）

攻撃は極めて困難である．

試作版のシステムを公開中： http://se.naist.jp/rinrun/

マシン語の難読化 ---アンチ逆アセンブル(1)

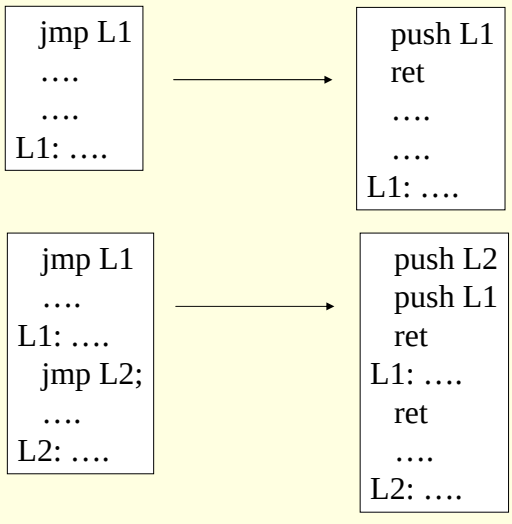
命令の真ん中へとジャンプするプログラム

```
000000
00000B      60                  ; push AD
00000C      E8 03 00 00 00      ; call 00000014
000011      D2 EB               ; shr BL, CL
000013      0B 58 EB            ; or EBX, [EAX-15h]
000016      01 48 40            ; add [EAX+40h], ECX
```

```
000014      58                  ; pop EAX
000015      EB 01               ; jmp 00000018
000017      48                  ; dec EAX
000018      40                  ; inc EAX
```

P. Červeň, “Crackproof your software – The best ways to protect your software against crackers,” No Starch Press, San Francisco, ISBN 1-886411-79-4, 2002.

マシン語の難読化 ---アンチ逆アセンブル(2)



P. Červeň, “Crackproof your software – The best ways to protect your software against crackers,” No Starch Press, San Francisco, ISBN 1-886411-79-4, 2002.

究極の難読化 ---一方向性関数

一方向性関数encode()が存在するとき,

```
if(password == 324)
   return 1;
else
   return 0;
```

難読化 →

```
if(encode(password) == 92133)
   return 1;
else
   return 0;
```

ただし, encode(324) = 92133 とする.

- 92133から324を推測することは不可能.
- 適当な入力を与えて324を見つけることも困難.
- この難読化を破る方法は？

プログラムの暗号化

```
int n = 52;
int i, k, p=1;

for(i=1;i<=31;i++)
{
   k = n – i + 1;
   p = p * k / i;
}
return p;
```

元のプログラム

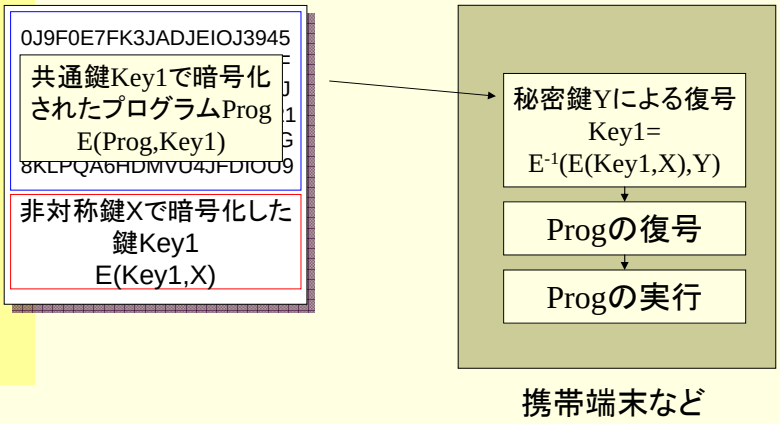
暗号化 →

```
0J9F0E7FK3JADJEIOJ3945
7ASJSI438JCI4AWIDJO45F
OLPO9HI7FF8LE43D2LIJ5J
SIO2119JASHKBMSJDU921
AKNNS33KMNZOF7H0IJAG
8KLPQA6HDMVU4JFDIOU9

[復号ルーチン]
```

暗号化後のプログラム

・プログラムの全体または一部を暗号化する.
・復号ルーチンを追加する.
・プログラム実行時に復号処理を行う.

非対称鍵暗号を用いたプログラム保護



・ライセンスされた開発者にのみXを渡す.
・Yは非公開とする（端末中に埋め込まれている）
・弱点は？

プログラムの暗号化の技術動向

・PC向けに数多くの商品がある[1][2] .
・ゲーム機でも必須の技術.
・攻撃に弱い.
  - 復号ルーチンが解析される.
  - デバッガによるメモリの覗き見，メモリダンプ
・攻撃方法が知られている.
  - 攻撃方法を解説する本 [3]
  - 攻撃方法を解説するWebページ [4]

商品化もされており，よく用いられているが，攻撃方法もまたよく知られている．難読化と併用するのがよい.

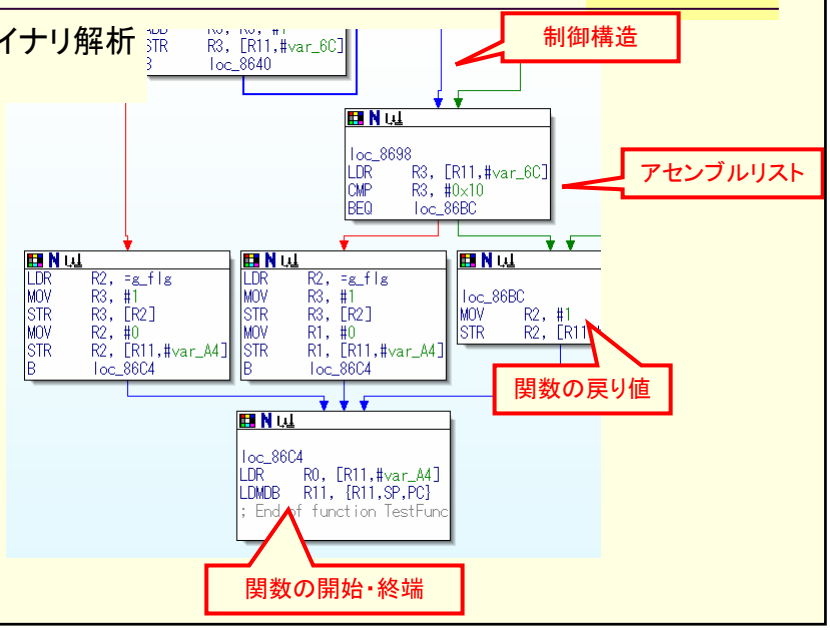
[1] ASPack Software, “ASProtect,” http://www.aspack.com/
[2] Obsidium Software, “Obsidium” http://www.obsidium.de/
[3] Kracker’s & BEAMZ, “クラッカー・プログラム大全,” データハウス，2003.
[4] “セキュリティアカデメイア – セキュリティ講座 – Krack”
http://akademeia.info/main/security_lecture.htm#krack

プログラムに対する攻撃方法

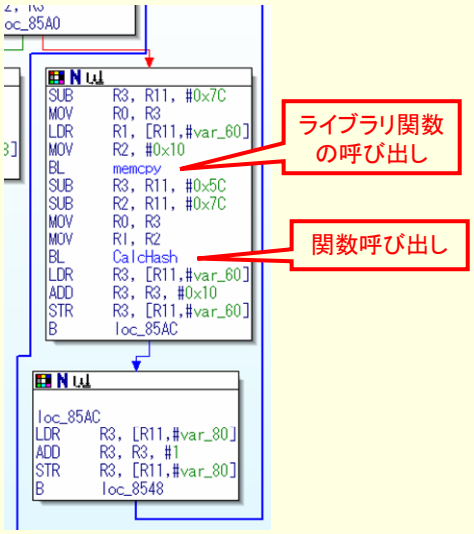
- デバッガを用いた攻撃
  - 逆アセンブラ，メモリダンパー，バイナリエディタを備える．
  - シリアルクラックによく用いられる．
  - GetWindowText等のAPI呼出しにブレークポイントを置く．
  - 使用期限チェックならばGetLocalTimeなどのAPI
  - 正常動作，異常動作の差を見る．
- 逆コンパイラを用いた攻撃（Java, .NET）
- トレーサを用いた攻撃

デバッガIDA Proの実行例(1)

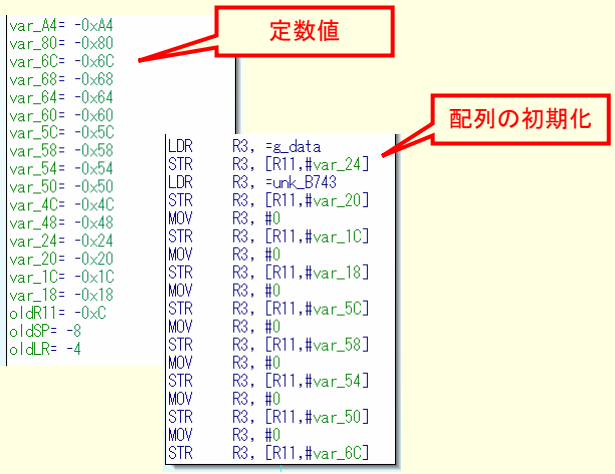
ARMバイナリ解析の例



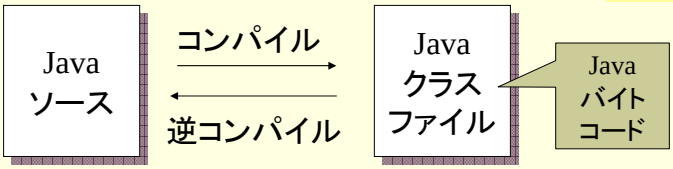
デバッガIDA Proの実行例(2)



デバッガIDA Proの実行例(3)



Java逆コンパイラ(1)



- Mocha … 古典的ツール (1996).
- Dava … 現在最も有力な逆コンパイラの一つ
  - McGill UniversityのSableグループが開発しているsootフレームワークに含まれる.
- DJ Java Decompiler … 使いやすい(Jadがエンジン)
- その他:以下のサイトから入手可能
  - http://catamaran.labs.cs.uu.nl/twiki/pt/bin/view/Transform/JavaDecompilers

Java逆コンパイラ(2)

```
private int member = 10;

public Foo() {
   int local = returnInteger();
   System.out.println( "foo constructor" );
   priv( local );
}
```

Foo.javaのオリジナル・ソースの抜粋

コンパイル

Java
class file

逆コンパイル

```
private int member;

public Foo() {
   member = 10;
   int local = returnInteger();
   System.out.println("foo constructor");
   priv(local);
}
```

Mochaで生成されたFoo.javaのソース

Greg Travis, “Javaコードを守る方法 (あるいは他人のJavaコードを参照する方法),”
http://www-6.ibm.com/jp/developerworks/java/011026/j_j-obfus.html

Java逆コンパイラ(3)

逆コンパイルを防ぐ単純な方法

- バイトコード中にダミーのgoto文を入れる.
  - Java言語(ソースコード)にはgoto文がないが, Javaバイトコードにはgoto文がある.
- 制御文字をメソッド名やクラス名に使う.
  - バイトコードでは許されるが, ソースコードとして表示できない.
- バイトコード中のスタック操作を複雑化する.
  - 演算式への変換を困難にする.

それなりの効果はあるが, 逆コンパイルを完全に防ぐことはできない.(不完全ながらソースコードは得られる.)

実行系列(トレース)出力ツール

Addtracer
http://se.naist.jp/addtracer/

- あらゆるJavaクラスファイルにトレーサを埋め込める.
- 埋め込み対象のクラスファイルのソースコードは必要なし.

```
0: public class Fibonacci{
1:   public int fibonacci(int n){
2:     if(n == 1 || n == 2){
3:       return 1;
4:     }
5:     return fibonacci(n - 1) + fibonacci(n - 2);
7: }
```

```
start Fibonacci#fibonacci at line 3
n assignment(line 3): 4
n reference(line 3): 4
n reference(line 3): 4
this reference(line 6): Fibonacci@1e893df
n reference(line 6): 4
- operation(line 6): 3
start Fibonacci#fibonacci at line 3
n assignment(line 3): 3
n reference(line 3): 3
n reference(line 3): 3
this reference(line 6): Fibonacci@1e893df
n reference(line 6): 3
- operation(line 6): 2
start Fibonacci#fibonacci at line 3
n assignment(line 3): 3
n reference(line 3): 3
n reference(line 3): 3
end Fibonacci#fibonacci at line 4
this reference(line 6): Fibonacci@1e893df
n reference(line 6): 3
- operation(line 6): 1
start Fibonacci#fibonacci at line 3
n assignment(line 3): 1
n reference(line 3): 1
end Fibonacci#fibonacci at line 4
+ operation(line 6): 2
end Fibonacci#fibonacci at line 4
this reference(line 6): Fibonacci@1e893df
n reference(line 6): 4
- operation(line 6): 2
start Fibonacci#fibonacci at line 3
n assignment(line 3): 2
n reference(line 3): 2
n reference(line 3): 2
end Fibonacci#fibonacci at line 4
+ operation(line 6): 3
end Fibonacci#fibonacci at line 7
```

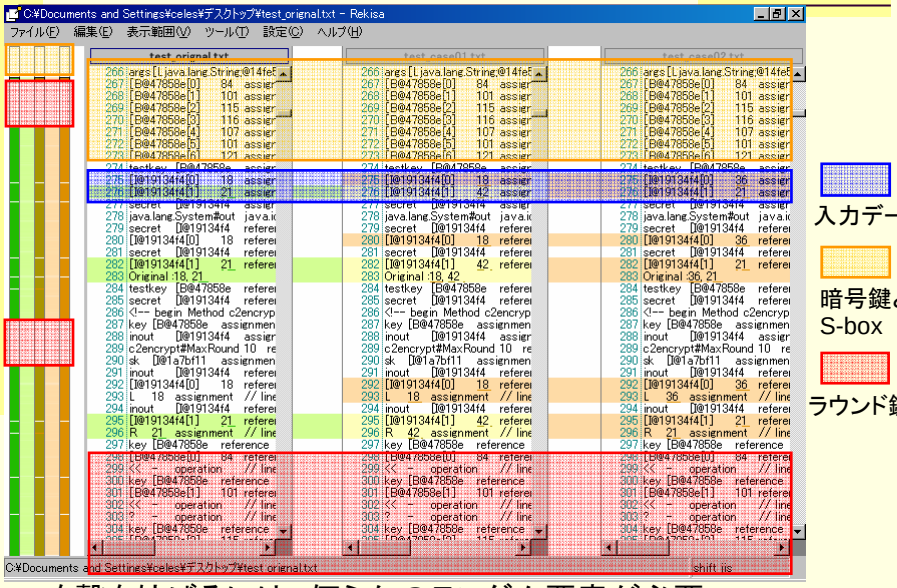
適用例

- 実行系列差分攻撃
  - 一つのプログラムに対して異なる入力を与え，得られた実行系列から差分攻撃を行い，暗号鍵やパスワードの探索を行う
  - デバッガでは，実行系列を自動的に取得するのが困難
  - デバッガを用いた攻撃に比べ，専門知識を必要としない

実験

- 目的
  - C2暗号のプログラム内部の秘匿情報を導出する
- 手順
  - C2暗号を実現するJavaクラスファイルにAddTracerを適用する
  - 入力をランダムに与え，その出力結果の差分をとる

実験結果



攻撃を妨げるには，何らかのランダム要素が必要

ソフトウェアプロテクションの課題

- 目的が明確でない．
  - プログラムを読みにくくすれば，それでよいのか？
  - どの程度役立つのか？
- より具体的な目的が必要
  - 1. 攻撃モデル
    - 攻撃者が行うことの出来る行動，行えない行動
  - 2. 攻撃者の目的
    - 例：動的攻撃により，メモリ上に現れる暗号鍵を見つける．
    - 例：静的解析により，特定のAPIの呼び出し箇所を見つける．
  - 3. 評価
    - 攻撃者が2.を達成できないことを示す．

難読化の定義

- Barakらの定義
  - プログラムへのオラクルアクセス（任意の入力を与えて出力を観測すること）により得られる以上の（入出力に関する）情報をプログラムコードが漏らさないとき，そのプログラムはvirtual black boxである（難読化されている）
  - 暗号化の定義に近い．（実現不可能）
- 現状では，誰もが納得できる難読化の定義はない．
- プログラムの種類によって保護したい物は異なるし，攻撃者の目標も異なるため，個別に攻撃モデルを考えて評価することが望ましい．

B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang, “On the (im)possibility of obfuscating programs,” *Lecture Notes in Computer Science*, Vol. 2139, pp. 1-18, 2001.

人間による評価

- エキスパートプログラマに解析を試みてもらう．
  - 難読化されたDES復号プログラムから復号鍵を導出する試み[1]
  - クラックできた者に賞金を出す試み
- メンタルシミュレーションによる評価[2]
  - 頭の中でプログラムの実行を追ってもらう．
  - 入力Xを与えたときに出力Yが得られることを，導出してもらう．
  - 導出に要した時間を計測する．

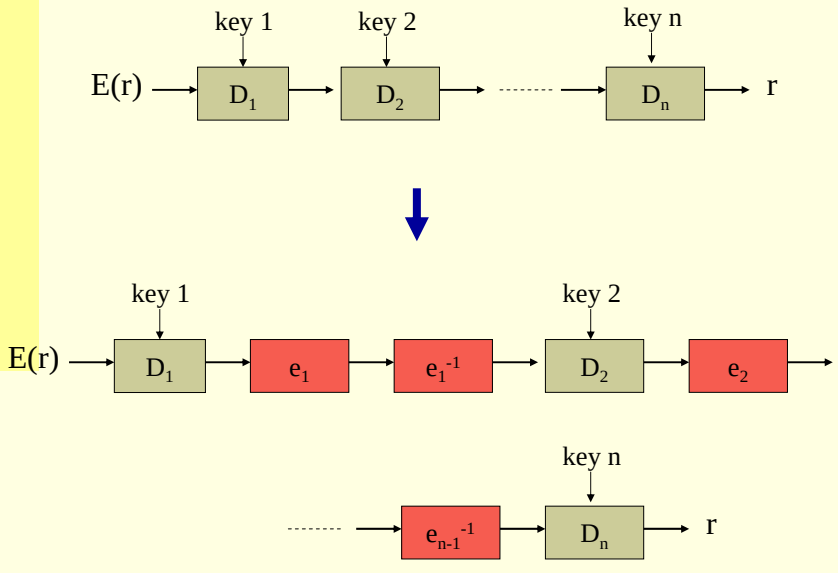
[1] 松岡賢, 赤井健一郎, 松本勉, 竹脇和也, “鍵内臓型暗号ソフトウェアの人手による耐タンパー性評価,” 2002年暗号と情報セキュリティシンポジウム(SCIS2002), Jan.-Feb. 2002.
[2] M. Nakamura, A. Monden, T. Itoh, K. Matsumoto, Y. Kanzaki, and H. Satoh, "Queue-based cost evaluation of mental simulation process in program comprehension," Proc. 9th IEEE International Software Metrics Symposium (METRICS2003), pp.351-360, Sep. 2003.

最近の動向

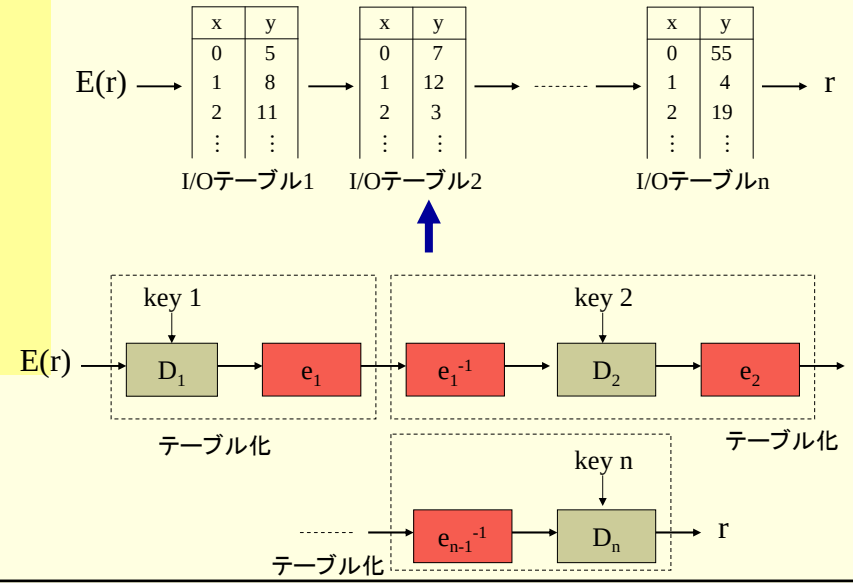
- アルゴリズムを限定した難読化方式
  - DES復号プログラム[1][2]
  - AES復号プログラム[3]
  - べき乗計算[4]
- 隠蔽すべき暗号鍵とその他の定数を混ぜる（分離不能にする）.
- 課題
  - サブルーチンごと再利用されたら？？

[1] S. Chow, P. Eisen, H. Johnson, and P. van Oorschot, “A white-box DES implementation for DRM applications,” ACM Workshop on Digital Rights Management (DRM2002), Lecture Notes in Com-puter Science, Vol. 2696, pp. 1-15, Springer-Verlag, 2003.
[2] H. E. Link and W. D. Neumann, “Clarifying obfuscation: im-proving the security of white-box encoding,” Cryptology ePrint Archive, Report 2004/025, International Association for Cryp-tologic Research, 2004
[3] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, “White-box cryptography and an AES implementation,” 9th International Workshop on Selected Areas in Cryptography (SAC 2002), Lec-ture Notes in Computer Science, Vol. 2595, pp. 250-270, 2003.
[4] 松本，赤井，”耐タンパー・ベキ乗プログラムの構成法，” 信学技報(ISEC2000)，Vol.100, No.76, pp.13-14, May 2000.

Whitebox Cryptography (1)



Whitebox Cryptography (2)

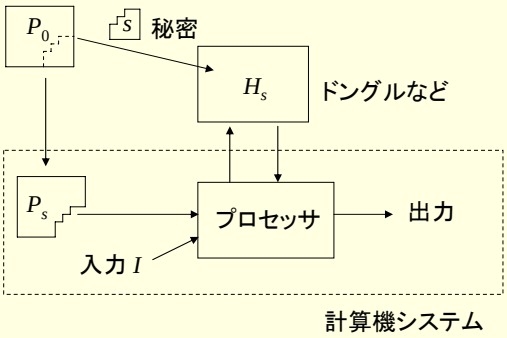


ハードウェアによるソフトウェアプロテクション

- ブラックボックス実行
  - プログラムコード，及び，実行状態の隠蔽
  - CPUと同一チップ上のROMにプログラムを焼く.
  - 課題
    - I／Oインタフェースからの攻撃，電力解析攻撃
    - プログラムのアップデートができない.

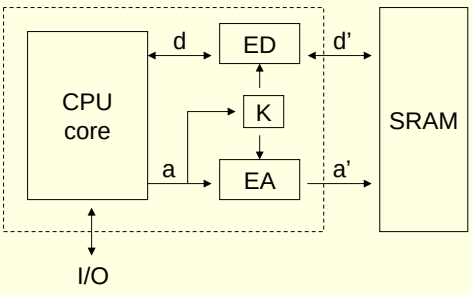
部分的なブラックボックス

- プログラムの一部分のみをブラックボックスとして実装する.
  - ドングル(シリアルインタフェースやUSBインタフェースに差し込む小さなデバイス)
  - 課題:
    - 十分な安全性を確保するための実装コストが大きい.



On-the-fly実行

- メインメモリ上の暗号化されたプログラム＆データを, CPU内部において1バイトずつ復号しながら実行する.
- 復号したプログラムを保存するためのブラックボックスを必要としない.
- データバスだけでなくアドレスバスも暗号化



- 課題: SRAM上の値の変化から命令が推測できる.

CPUキャッシュメモリによる仮想ブラックボックス

- 一度に復号する暗号文のバイト数（暗号ブロック長）を大きくする.
- 復号後の命令系列を記憶するためのキャッシュメモリをCPU内部に設ける.
- いくつかの商用プロセッサが存在する.
- 課題：パフォーマンスの低下，命令の先読みに支障が出る.

アクセス制御による仮想ブラックボックス

- プロセッサ外部のRAMに強力なアクセス制御を行い，攻撃者がRAMの中身を読めないようにする.
- 現在のWindows
  - Buffer overflowを利用した攻撃
  - OS自身が改ざんされる（アクセス制御機構が改ざんされる）問題
  - OSのバグによるセキュリティホールの問題
  - デバッガの存在，DMA（Direct Memory Access）デバイスの存在
- 近年の動向:
  - 次世代Windowsでの採用（？）

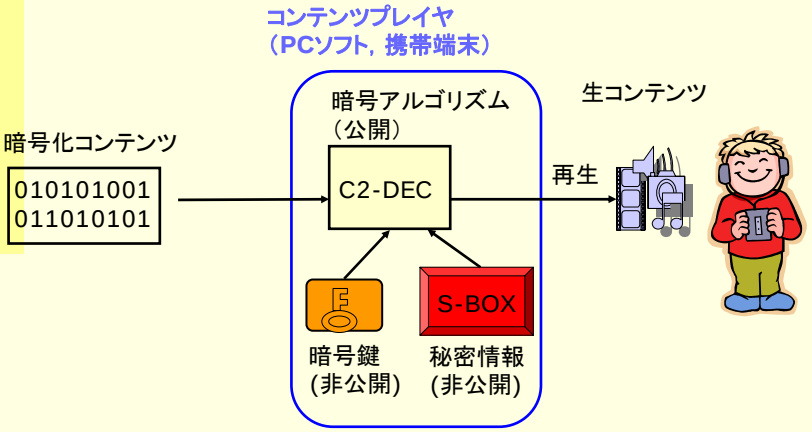
アクセス制御による仮想ブラックボックス

- NGSCBにおける要素技術
  - ブラックボックス実行のための専用のCPUモード
    - 他のプロセスやDMAデバイスからの隠蔽
  - メモリへのアクセス制御機構をnexusと呼ばれる小さなカーネルに封じ込め，仕様・実装の検証を徹底する．
  - OSが正当なものであることを保障するブートストラップ方式
    - BIOS，ブートローダ，OSその他の認証
  - 特定のハードウェア，OS，プログラムにアクセス権を与える．
    - 人間にアクセス権を与えない．
    - 信頼できない計算機にプログラムやデータを送る場合であっても，その計算機上で動作するハードウェアやOSが信頼できるならば，アクセス権を与える．
  - CPUレベルでのプログラム領域とデータ領域の分離

その他のプロテクション技術

・秘密分割によるソフトウェア保護法式

特願2005-89941, Mar. 2005.

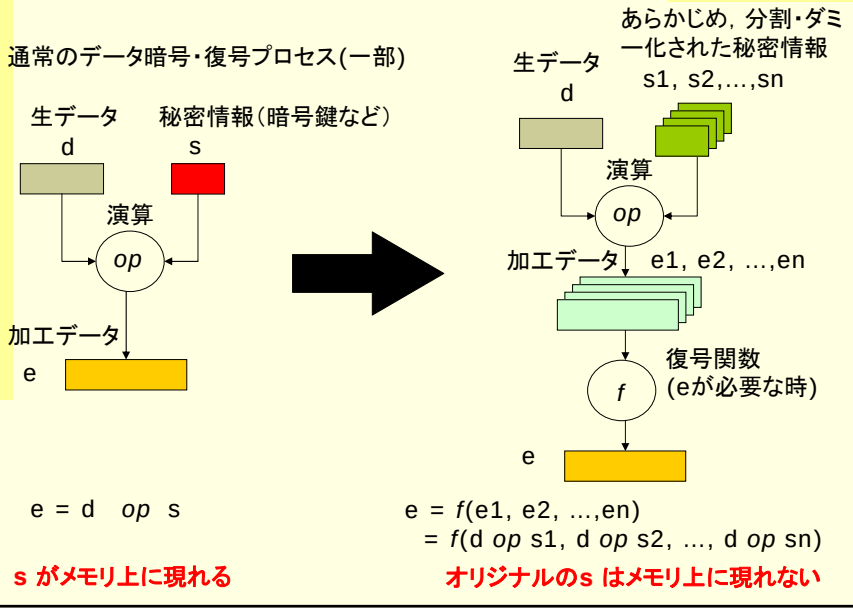


目的とアプローチ

目的:
プログラムにおける秘密情報(暗号鍵，S-BOXなど)がメモリに現れない情報隠蔽方式を提案する.
→**電力解析攻撃，メモリ覗き見攻撃**を回避する.

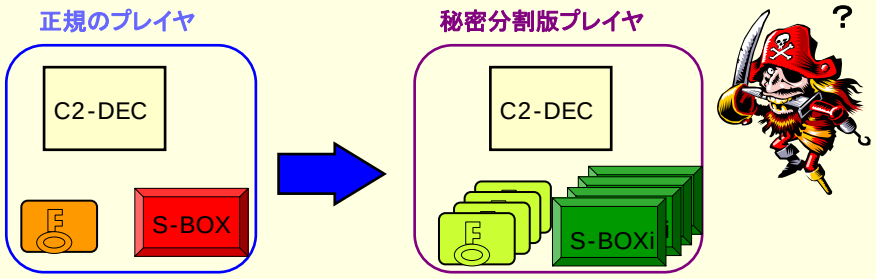
アプローチ:
秘密情報を分割し，分割したまま演算を行うことで，生の秘密情報がメモリ上に現れるのを防ぐ.
- オリジナルプログラムの動作を保障するための，うまいデータ分割・復号法が必要となる.

キーアイデア



メモリ覗き見攻撃への対処

- 秘密情報を分割してプレイヤに搭載することで，秘密をメモリ上で観測不能にする．
  - 分割情報を部分的に見てもオリジナル情報は予測できない．



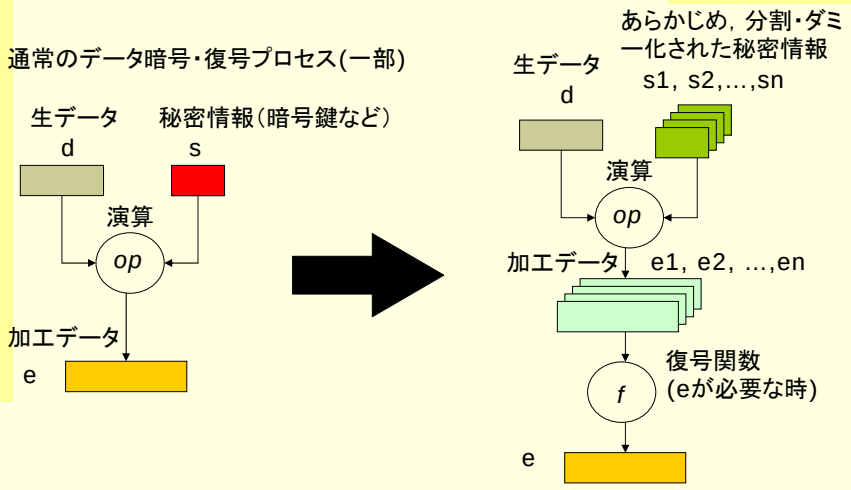
技術的課題

課題1：実行効率劣化の最小化
- 秘密を分割したまま演算を行う際，元来の動作を完全に保証しつつ，なるべく効率を落とさない．

課題2：利用可能な演算の種類の増大
- 分割した情報に対して利用可能な演算の種類をなるべく多くして，汎用性を上げる．

課題3：分割方法の自由度の向上
- 秘密の分割方法を推定されないよう，十分に自由度を取る．

キーアイデア



e = d *op* s

s がメモリ上に現れる

e = *f*(e1, e2, …,en)
= *f*(d *op* s1, d *op* s2, …, d *op* sn)

オリジナルのs はメモリ上に現れない

復号関数 *f* の選び方

d *op* s = *f*(d *op* s1, d *op* s2, …, d *op* sn)

演算opを，ビット演算に限定して考える．

- **AND**

d & s = *f*(d *&* s1, d *&* s2, …, d *&* sn)

d=0の場合，0 = *f*(0, 0, …,0) ・・・(a)

d=1の場合，s = *f*(s1, s2, …,sn) ・・・(d)

- **OR**

d | s = *f*(d | s1, d | s2, …, d | sn)

d=0の場合，s = *f*(s1, s2, …,sn)

d=1の場合，1 = *f*(1, 1, …,1) ・・・(b)

- **XOR**

d ^ s = *f*(d ^ s1, d ^ s2, …, d ^ sn)

d=0の場合，s = *f*(s1, s2, …,s3)

d=1の場合，~s = *f*(~s1,~s2, …,~sn) ・・・(c)

以上により，fは以下の条件を満たすことが望ましい．

(a) f(0,0,…,0) = 0，(b) f(1,1,….,1) = 1, (c) 自己双対である.

(d) f(s1,s2,…,sn)=s

復号関数 *f* の性質

*n*変数論理関数 *f* を以下を満たすように決定する.

f(0,…,0) = 0かつf(1,…,1)=1かつ自己双対

- 例
  - f(x,y,z) = ~x&~y&z|~x&y&~z|x&~y&z|x&y&z
  - f(x,y,z) = x&y | y&z | z&x (多数決関数)

- fによる情報復号はシンプルかつ高速
  - 2段の組み合わせ回路でHW化も容易

➡ **課題1:実行効率劣化の最小化**

- そのようなfの決め方は, $2^{2^{n-1}-1}$ とおりある.

→ nを大きくとれば, 総当りは不可能になる.

f に基づく秘密情報の分割

fを一つ決めて, 秘密情報 *s* をn個に分割する

sをmビットの情報b1b2…bmと見て, 各ビットbi(i=1,…,m)毎に

bi = f(bi1, bi2,…,bin) (i=1,…,m)

となるように, 適当なbi1,bi2,…,bin を決める.

分割例

n=3, f(x,y,z) =x&y | y&z | z&x (多数決関数)

s= 9 (=1001) とした場合

x	y	z	f	
0	0	0	0	
0	0	1	0	$f^{-1}(0)$
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	1	$f^{-1}(1)$
1	1	1	1	

s		s1	s2	s3
1	→	0	1	1
0	→	0	0	1
0	→	1	0	0
1	→	1	0	1

fを一つ決めたときの分割の方法は, $2^{m(n-1)}$ とおり, fの決め方は $2^{2^{n-1}-1}$ とおり

➡ **課題3:分割方法の自由度の向上**

分割情報に対する演算

- ビット演算　ー 分割したまま演算を継続

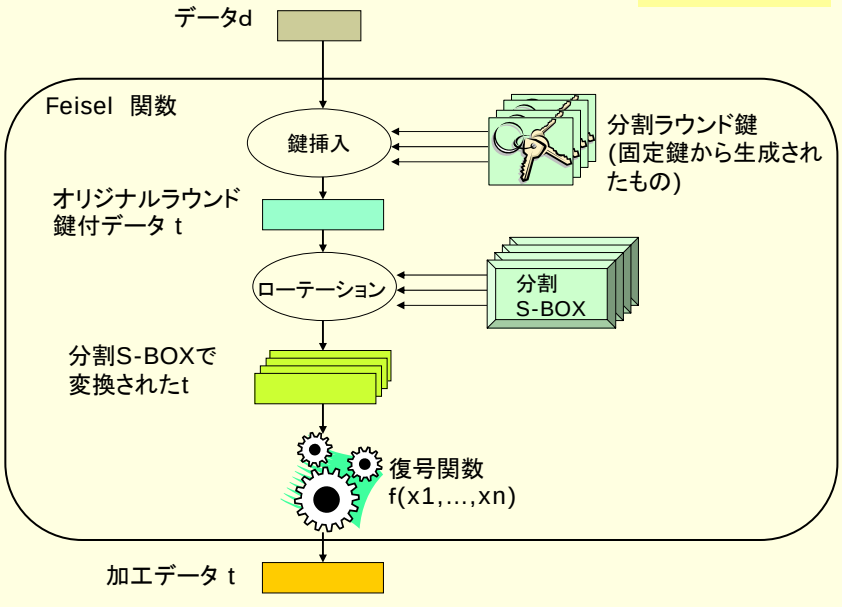
  (AND) d & s →　d&s1, d&s2, …, d&sn
  (OR)　d | s →　d|s1, d|s2, …, d|sn
  (XOR) d ^ s →　d^s1, d^s2, …, d^sn
  (NOT)　~e　→　~e1, ~e2, …, ~en
  (Shift) e<<c →　e1<<c, e2<<c,…, en <<c
  　　e>>c →　e1>>c, e2>>c,…, en >>c

- 算術演算　ー　ビット毎の全加算器として実現

（ADD)　t = d + s →
t1j=dj^s1j[]^c1j[], …, tnj = dj^snj[]^cnj[] (和)
c1j=dj&s1j[]|s1j[]&c1j-1[]|c1j-1[]&dj,…, cnj (桁上がり)

課題2：利用可能な演算種類の増大

分割版C2実装



セキュリティ評価

- オリジナルの秘密情報
  - メモリ上に現れることはない.
- 分割後の情報が抜き取られた場合
  - f が全く露見しない場合
    - 元情報を得るには, $2^{2^{n-1}-1}$ 回の試行が必要.
    - 一つの元情報に対し, 2^{n} 通りの分割方法があるため, 同一箇所からの類推攻撃に強い.
  - f の入出力が観測された場合
    - 2^{n} 個の入力に対する出力からfの真理値表が露見
  - f の内容が露見した場合
    - f(s1,s2,…,sn) = s から秘密情報が露見する.

本日の演習課題

- 課題
  - 適当なプログラムを用意し, 難読化せよ.
- レポート提出期限
  - 2006年2月2日(金)2限
  - 希望者は課題発表を行うこと
    - レポートの内容について説明する.
    - レポート用紙をスクリーンに映す, もしくは, パワーポイント等のプレゼン資料を用いる.
- 連絡先
  - 門田暁人 akito-m@is.naist.jp
  - B303室, 内線5311

本日の演習課題

- 難読化の対象プログラムの仕様を説明せよ.
- どのように難読化したかを説明せよ.
- 難読化によって達成できたことを述べよ.
  - 例:プログラム中の〇〇が隠蔽できた.
  - 例:特定の演算式やアルゴリズムの推測が困難になった.
  - 例:入出力仕様の推測が困難になった.
- 難読化したプログラムに対する攻撃手段を考えよ.
  - 例:人間が読む. エディタのサーチ機能, 実行する, 逆アセンブル, 逆コンパイル, デバッガ, 実行トレース. . .
- 攻撃に対する耐性, 攻撃に対する対策を考察せよ.
- 難読化の逆変換が可能かどうかも考察せよ.