社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2005-59(2005-12)

攻撃タスクを考慮した難読化による暗号プログラムの保護

山内　寛己† 　神崎雄一郎† 　門田　暁人† 　中村　匡秀† 　松本　健一†

† 奈良先端科学技術大学院大学 情報科学研究科
〒 630-0192 奈良県生駒市高山町 8916-5
E-mail: †{hiroki-y,yuichi-k,akito-m,masa-n,matumoto}@is.naist.jp

あらまし　本稿では，暗号プログラムに対する攻撃タスクを整理し，攻撃タスクからプログラムを保護するための難読化適用手法の検討を行う．具体的には，まず暗号プログラムに対する典型的な攻撃を挙げ，各攻撃に必要な基本タスク (操作，観測等) を整理する．次に，これらの基本タスクを攻撃者が実行困難となるように，暗号プログラムに難読化を適用する．ケーススタディとして，Java で実装された Cryptomeria Cipher(C2) 暗号を用いたソフトウェアの解析を困難にする方法について検討する．

キーワード　ソフトウェア保護，耐タンパー技術，動的解析，静的解析，共通鍵暗号

Obfuscating Cipher Programs Against Primary Attacking Tasks

Hiroki YAMAUCHI†, Yuichiro KANZAKI†, Akito MONDEN†, Masahide NAKAMURA†, and Ken-ichi MATSUMOTO†

† Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara, 630-0192, Japan
E-mail: †{hiroki-y,yuichi-k,akito-m,masa-n,matumoto}@is.naist.jp

Abstract　In this paper, we propose a cracker-centric approach to give a realistic guideline for applying software obfuscation methods to disrupt crackers' actions. As a case study, we define a security goal and a threat model for a Java implementation of a cryptomeria cipher (C2) program, and then, based on the model, we describe a guideline to protect secret keys included in the program.

Key words　Software Protection, Tamper-resistance, Static Analysis, Dynamic Analysis, Symmetric-key Cryptography

1. はじめに

最近のソフトウェアシステムの多くは，何らかのセキュリティゴールを達成するために暗号メカニズムが含まれている．典型的な例として，DVD プレイヤーや映像ストリーム再生プレイヤーなどのデジタル著作権管理 (DRM) ソフトウェアがある．これらはメディアコンテンツの不正使用を防止するために対称鍵暗号が使われている [1]．このようなソフトウェア (本稿では，暗号ソフトウェアと呼ぶ) は，通常，その内部に秘密情報を含んでいる．例えば，デバイス鍵，ラウンド鍵，S-box テーブルなどであり，これらはソフトウェアユーザから隠ぺいしなければならない [1] [3]．このような秘密情報を含むソフトウェアが増加しているなか，攻撃者によってソフトウェア内部が解析されるのを防ぐための保護技術が求められる．

これまで，ソフトウェア内部の秘密情報を守るために様々なソフトウェア難読化手法提案されてきた．その中でも，暗号ソフトウェアを保護する強力な手法として，white-box cryptography がある．この手法は，Chow らによって提案され [3] [4]，今もなお改良が続けられている．この手法では，秘密鍵を用いる演算に新たな暗号化/復号処理を追加し，それを数式ではなく，テーブル参照により実現する．秘密鍵はテーブル内に隠ぺいされ，攻撃者は秘密鍵を発見できなくなる．しかしながら，この手法は巨大なメモリ領域を必要とし，深刻なパフォーマンス低下をもたらすため，その適用範囲は著しく限定される．そのため，特に計算資源の限られたモバイル環境においては，より効率の良い保護方法が必要となる．

一方，より手軽な (light-weight な) 難読化手法も数多く提案されている．制御フローの難読化 [5] [14]，内部モジュール呼び出し関係の難読化，高レベル命令から低レベル命令セットへの置き換え，データ構造の変換 [7]，準同型写像を用いたデータ変

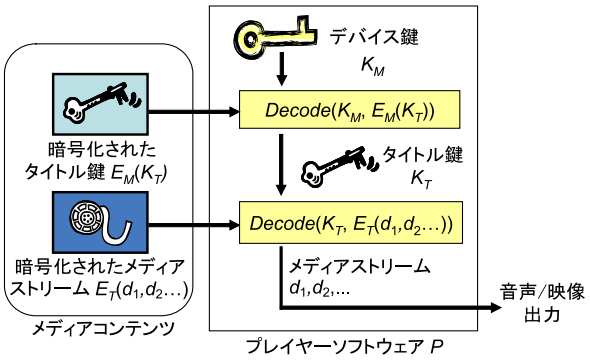

図 1　典型的なメディアプレイヤーの仕組み

換 [6] [16]，自己書き換えコードの挿入 [9] などがある．これらの多くは，white-box 方式と比べて必要とするメモリ領域やパフォーマンスの低下が小さく，計算資源が限られている環境でも適用可能である．しかし，これらの手法は暗号ソフトウェア内部の秘密情報を保護するのにどの程度有効なのか不明確である．さらに，ほとんどの難読化手法は，明確な攻撃モデル (攻撃者がセキュリティを破壊しようとする際に取りうる行動のモデル) を定義していない．これらの難読化の有効性を評価するためには，明確なセキュリティゴールと，攻撃モデルを定義する必要がある．

本稿の目的は，秘密情報を抽出しようとする熟練した攻撃者から暗号プログラムを保護するために，light-weight な難読化手法を適用するためのガイドラインを確立することである。従来の難読化法が，防御者の視点から “複雑な” プログラムを作成することを主眼に置いていたのに対し，本稿では攻撃者の視点から攻撃方法を整理し，攻撃を妨げることに焦点を当てる．

2. セキュリティゴール

暗号プログラムを保護するためのセキュリティゴールは，プログラム P 中に含まれる暗号鍵 K をできるだけパフォーマンス (計算資源，実行速度) を損なわずに隠ぺいすることにある．このセキュリティゴールを必要とするアプリケーションとしては，マルチメディアプレイヤーが挙げられる．マルチメディアプレイヤーのほとんどは，DES, AES, C2 などの共通鍵暗号が採用されており，暗号化されたコンテンツを復号するためにソフトウェア内に暗号 (復号) 鍵が内蔵されている．このようなプレイヤーソフトウェアは，高速なパソコン向けに開発されているのみならず，PDA，携帯電話，携帯ゲーム機などの低速な機器向けにも開発されている．そのため，暗号鍵 K を保護するためには，深刻にパフォーマンスが低下しない light-weight なセキュリティ技術を用いる必要がある．

図 1 に，典型的なメディアプレイヤーソフトウェアの仕組みを表す．プレイヤーソフトウェア P 内に，デバイス鍵 (マスター鍵) と呼ばれる K_M が含まれており，これを (攻撃者になりうる) ソフトウェアユーザから隠す必要がある．図 1 に示すように，デバイス鍵 K_M は暗号化されたタイトル鍵 $E_M(K_T)$ の復号に使用される．そして，復号したタイトル鍵 K_T は暗号化されたコンテンツ $E_T(d_1, d_2, \ldots)$ の復号に使われる．通常，デバイス鍵など秘密情報は規格化団体からライセンスを受けたメーカーに提供され，メーカは秘密情報を含んだプレイヤーを開発する．万が一，秘密情報である K_M がユーザによって発見されてしまうと，暗号化されたメディアコンテンツから，オリジナルのコンテンツを取り出すプログラムを作成されるおそれがある．そのようなプログラムがインターネット上で流通すると，復号後のコンテンツが (P2P ソフトなどを介して) 不正に流通しやすくなり，大きな問題となる．

SS2005-36 (2005-12)

3. 攻撃者の能力モデル

3.1 暗号プログラム一般における能力モデル

前節で述べたセキュリティゴールに対する攻撃モデルを定義するためには，まず，攻撃者の知識や能力を予めモデル化しておく必要がある．攻撃者の取りうる行動は，攻撃者の知識や能力に依存するためである．例えば，攻撃者は，実行可能な (バイナリ) プログラムを所持し，プログラムに使用されている暗号アルゴリズムの知識を持ち，逆アセンブラや逆コンパイラのような静的解析ツールを，および，ブレークポイント機能を持つ動的解析ツール (デバッガ) を持つと仮定するのが現実的であろう．

文献 [13] において，Monden らは，攻撃者の能力を，(A) 採用されている保護方式（難読化法）に関する知識，(B) 攻撃者がシステムから観測できる情報，(C) 攻撃者がシステムに対して行うことのできる行動，の 3 つのカテゴリによって特徴づけている．ただし，本稿では，特定の難読化法を未だ想定していないため，(A) のカテゴリを,「攻撃対象のソフトウェアで採用されているアルゴリズムに関する知識」とする．

上記の 3 つのカテゴリそれぞれについて，様々なレベルの攻撃者が考えられる．非常に限られた技術と能力しか持たない攻撃者もいれば，デバッガを所持し，それを使いこなせる攻撃者もいる．本稿では，あらゆるクラッキングツールやコンピュータシステムを自由自在に扱うことができ，攻撃対象のプログラムで採用されている暗号アルゴリズムについても深い知識を持つ，最高レベルの攻撃者を想定する．具体的には，各カテゴリについて，次のように特徴づける．

(A)　アルゴリズムの理解
攻撃者は，暗号プログラム P で使用されている暗号アルゴリズムの仕様についての十分な知識を持つ．一方，ライセンス団体から P の開発者にのみ提供される秘密情報 (デバイス鍵，S-box など) は知らないものとする．

(B)　システムの観測
攻撃者は，P が実行される計算機システム M，P の実行ファイル，逆アセンブルされたプログラムリスト，及び，逆コンパイルされたプログラムリストを所持する．攻撃者は，M の内部状態 (例えば，M のメモリのスナップショット)，M への入出力，及び，P への入出力を観察できるブレークポイント機能を備えたデバッガを所持する．また，P の実行トレース (実行された命令コードの履歴) を観測できる．

(C) システムの制御
攻撃者は，P に含まれる任意の命令を変更することができ，任意の入力を与えて変更した P を実行できる．また，P に含まれるレジスタや変数の値，M のメモリの内容を（デバッガを使って）実行前および実行中に変更できる．

上記で定義したモデルにおいて，攻撃者は様々な攻撃手段を持っている．例えば，P 実行中のスタックメモリを観察して，メモリ内に秘密鍵の候補があるかを調べたり，異なる入力を与えて複数の実行トレース収集し，それらを比較することで，定数オペランドからデバイス鍵の候補を見つけたりすることができる [20].

3.2 C2 暗号プログラムにおける能力モデル

3.2.1 C2 暗号プログラム

C2(Cryptomeria Cipher) 暗号は CPPM(Content Protection for Prerecorded Media) / CPRM(Content Protection for Recordable Media) に採用されており，保護された音声や映像コンテンツを DVD-R/RW/RAM，SD メモリーカードなどのリムーバブルな記憶媒体で提供するために 4C companies(Intel，IBM，松下，東芝) によって開発された．CSS 暗号の失敗の経験に基づき，セキュリティが強化された．しかしながら，デバイス鍵は未だプレイヤーに埋め込まれているため，ユーザから隠ぺいする事が要求されている．従って，攻撃者から C2 暗号プログラムを保護することは CPPM/CPRM に準拠したシステムを開発する者にとって必須である．

本稿では，Java で実装された C2 暗号 (ECB モード) を保護対象とする．Java の中間言語コードはネイティブコードに比べて非常に攻撃に脆弱であり，もし，Java での実装への攻撃を防ぐことができれば，他の実装 (例えば，x86 バイナリコード等) での攻撃を防御することも容易となると考えられる．

3.2.2 暗号アルゴリズムの理解

C2 暗号のアルゴリズムの概要を図 2 に示す．C2 暗号は DES と同様，Feistel 構造をもつ共通鍵ブロック暗号である．図中 “F” は，Feistel 関数 (F 関数) である．DES との違いは C2 は演算の一部に算術加算および減算を含む点である．

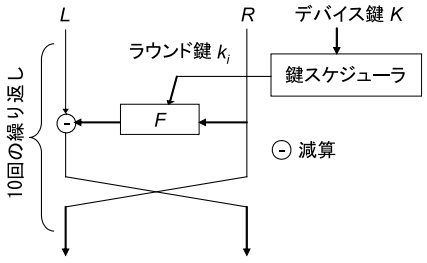

図 2 復号時の C2 暗号における Feistel 構造

C2 暗号の仕様書は公開されているため，攻撃者は次のような知識を持っていると思われる．

- ラウンド鍵 k_i は鍵スケジュールルーチンから提供されるか，またはに定数として P の内部直接書かれている．前者の場合，P の中にデバイス鍵 K_M が存在し，k_i は鍵スケジュールルーチンによって供給される．後者の場合は，K および鍵スケジュールルーチンは P には存在しないかもしれない．この場合，攻撃者の攻撃目標はラウンド鍵を含む P 内の鍵を見つけることである．
- ラウンド (繰り返し) の回数は 10 回であり，10 個のラウンド鍵 $k_1, \ldots, k_{10}$ が存在する．
- ラウンド鍵の鍵長は，それぞれ 32 ビットである．
- 入力のブロック長は 64 ビットである．また，それらは L(32 ビット) と R(32 ビット) に分割して処理される．
- F 関数が P 内にある．L と R は交互に F 関数の入力になる．
- F 関数直後に算術減算がある．

同様に，F 関数において，攻撃者は次のような知識を持っていると思われる．

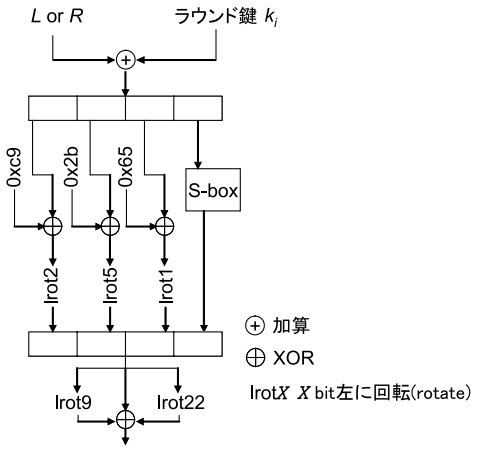

図 3 C2 暗号における F 関数

- L または R はラウンド鍵と加算される．
- 加算の結果の X(32 ビット) は，4 つのブロック $x_1, \ldots, x_4$ に分割される．分割は次の記述で行える ($x_1 = (X \gg 24)\&0\text{xff}, x_2 = (X \gg 16)\&0\text{xff}, x_3 = (X \gg 8)\&0\text{xff}, x_4 = X\&0\text{xff}$).
- 最下位の 8 ビットブロック x_4 は S-box によって変換される．S-box は要素 256 個，8 ビットのテーブルである．このことは，P の中に要素数 256 個の配列がある事が示唆される．変換は配列の参照 (例えば，S-box_array[x_4]) をすることで行える．S-box の値は，CPPM/CPRM の場合，公開されていない．従って，デバイス鍵やラウンド鍵と同様に S-box も隠す必要がある．
- 残りの 3 ブロック $x_1, \ldots, x_3$ は，それぞれ 0xc9，0x2b と 0x65 と XOR を行う．これは，XOR による式と，0x9c，0x2b，0x65 という定数が P 内に存在することを示唆している．その後，3 つのブロックはそれぞれ 2 ビット，5 ビット，1 ビットずつ左回転 (rotate) される．
- 4 つのブロック $x_1, \ldots, x_4$ は結合され，32 ビットに戻される．さらにこの値は，この値を 9 ビット左回転させた値と 22 ビット左回転された値で XOR される．これは，シフト演算と，9 と 22 という定数値が P に存在することを示唆している．また，結合は，$x_1 \ll 24 \mid x_2 \ll 16 \mid x_3 \ll 8 \mid x_4$ という式による実現できるため，この式が存在すると推測される可能性もある．

表 1　AddTracer が出力する情報

トレースの種類	出力する情報
ローカル変数参照時	変数の ID, 行数, 変数の値
ローカル変数代入時	変数の ID, 行数, 代入された値
フィールド参照時	クラス名, 変数名, 行数, 変数の値
フィールド代入時	クラス名, 変数名, 行数, 代入された値
変数が配列の場合	アクセスする要素のインデックス
演算	演算の種類, 型, 演算結果, 行数
メソッド開始時	クラス名, メソッド名, 行数
メソッド終了時	クラス名, メソッド名, 行数

3.2.3　システム観察および制御のためのツール

攻撃者は，P を静的解析するために逆アセンブラと逆コンパイラを使用する．Sun Microsystems が Java 逆アセンブラ (javap -c で知られている) を提供している．さらに，Java 逆アセンブラ D-java が Meyer [11] によって提供され，Jasmin [12] 形式の逆アセンブリコードを生成する．これによって逆アセンブルされたコードは，Jasmin アセンブラ [12] [17] によって，クラスファイルに再度構築できる．また，様々な Java 逆コンパイラが利用可能であるが，ほとんどの場合，その逆コンパイルされたソースコードは不完全なものである [19]．P を動的解析

```
public static byte lrot8(int x, int n) {
        return (byte)( (x << n) | ( x >>> (8-n)) );
}
```

lrot8関数のソースコード

```
...
x       0x47 assignment          // line 7
n       0x1 assignment           // line 7
x       0x47 reference           // line 7
n       0x1 reference            // line 7
<<      0x8e{int} operation      // line 7
x       0x47 reference           // line 7
-       0x8{byte} constant       // line 7
n       0x1 reference            // line 7
-       0x7{int} operation       // line 7
>>>     0x0{int} operation       // line 7
|       0x8e{int} operation      // line 7
...
```

lrot8関数の実行トレース

図 4　AddTracer を用いた実行トレース出力例

するために，攻撃者は Sun Microsystems が提供している jdb などを使用する．さらに AddTracer [18] と呼ばれる強力なツールがある．AddTracer は，Java クラスファイルの至る所にトレーサ (モニタリングコード) を挿入する．トレーサが挿入された Java クラスファイル P を実行することによって，容易に P 全体の実行トレースを得ることができる．表 1 は，トレーサが実行の間に出力する情報を示す．図 4 は，AddTracer を用いて生成された実行トレースの例を示す．変数への参照および，代入とそれらの値が出力されるので P が単純な実装である場合，秘密鍵がトレース中に現れてしまう．

4.　攻撃モデル

前章で述べた攻撃者の能力モデルに基づいて，本章では，攻撃者の取りうる行動のモデル (攻撃モデル) の定義を試みる．攻撃者は，攻撃対象 (ラウンド鍵など) を発見するために，様々な手がかりを探す行動を取ると考えられる．

例えば，ラウンド鍵は F 関数の内部での演算に用いられるため，攻撃者は，ラウンド鍵を発見するための手がかりとして，まず，F 関数を探す可能性がある．さらには，F 関数を探すための手がかりとして，F 関数に特有の演算 (XOR やシフト演算など) や定数 (0xc9 や 0x65) を探す可能性がある．また，F 関数を探すための別の手がかりとして，攻撃者は S-box を探す可能性もある．

このように，攻撃者は，ある情報を探すために，手がかりとなる別の情報の探索を行い，そのために必要なより小さな情報を探索していくと考えられる．本稿では，このような多数の探索行動の間のつながりを木構造で表現したものを，攻撃モデルとする．

図 5 は，C2 暗号プログラムに対する攻撃モデルの一例である．親ノードは，攻撃者の最終目的である「ラウンド鍵を探す」であり，その子ノードには，ラウンド鍵を探すための手がかり (F 関数，32 ビット値，鍵スケジューラ，入力データ) を探すノードが存在する．さらには，それらの手がかりとなる孫ノードが存在する．葉の部分は，XOR を探す，定数 24 を探す等のプリミティブな行動が示されている．これらプリミティブな行動は，攻撃者の能力モデルに基づいて，逆アセンブラやデバッガを用いて行われる．

次章では，攻撃対象となりうる情報をどのように難読化をすればいいのかを説明する．

5.　難読化適用のガイドライン

4. 章で述べた攻撃を困難にするため，対象プログラムを難読化することによって，図 5 の攻撃モデルに含まれる各ノードの探索行動を妨げることを考える．具体的には，次の情報の探索を妨げることを考える．

- ラウンド鍵
- S-box
- F 関数
- 特徴ある演算，定数オペランド
- 難読化そのもの

以下において，上に挙げた各々の情報をどのように難読化，あるいはどのように隠す必要があるかを説明していく．

5.1　ラウンド鍵の難読化

図 6 の中で示されるように，ラウンド鍵が 32 ビットの長さの大きな整数であるので，ラウンド鍵の候補は逆アセンブルされたコードから容易に見つけることが可能である．したがって，鍵をより小さな値に分割する必要がある．例えば，単純な実装によるラウンド鍵が次のような，

rkey[0] = 0x789ac6ee;

である時，鍵を隠す一つの方法として，鍵を 4 つの 8 ビットの整数 (副鍵) として，以下のような分割を行う．

b_rkey[0] = 0x78;
b_rkey[1] = 0x9a;
b_rkey[2] = 0xc6;

図 5　C2 暗号プログラムに対する攻撃モデルの一例

```
rkey[0] = 0x789ac6ee;
rkey[1] = 0x79bc3398;
rkey[2] = 0x48d15d62;
....
```

ラウンド鍵を定義する部分のソースコード

```
 17:    iconst_0
 18:    ldc        #6; //int 2023409390
 21:    lastore
 22:    aload      8
 24:    iconst_1
 25:    ldc        #7; //int 2042377112
 28:    lastore
 29:    aload      8
 31:    iconst_2
          ....
```

ラウンド鍵を定義する部分の
逆アセンブルコード (by javap -c)

図 6　ラウンド鍵定義の逆アセンブリリスト

```
static byte SecretConstant[] = { (byte)0xB6, (byte)0xAA, ....
```

S-box宣言部分のソースコード

```
static {};
  Code:
    0:    sipush    256
    3:    newarray byte
    5:    dup
    6:    iconst_0
    7:    bipush    -74
    9:    bastore
    10:   dup
    11:   iconst_1
    12:   bipush    -86
    14:   bastore
          ...
```

Byte型256個の配列の宣言
0番目の要素
1番目の要素

S-box宣言部分のアセンブリコード(by javap -c)

図 7　S-box 宣言部分の逆アセンブリコード

b_rkey[3] = 0xee;

もし，鍵で計算をする必要がある場合 (例えば，rkey[0] ^= n)，副鍵で次のように計算することが可能である．

b_rkey[0] ^= n_1;
b_rkey[1] ^= n_2;
b_rkey[2] ^= n_3;
b_rkey[3] ^= n_4;

この時，　$n = n_1 \ll 24 \mid n_2 \ll 16 \mid n_3 \ll 8 \mid n_4$

準同型写像によるデータの難読化 [6] [16] をラウンド鍵隠すため，副鍵に分割する前に適用することを強く推奨する．例えば，線形変換 (例えば，new_rkey[-] = 4 * rkey[-] + 3) を用いてラウンド鍵を以下のように定義する．

new_rkey[0] = 0x1a2b3f797L;
new_rkey[1] = 0x3784903dbL;
new_rkey[2] = 0x13579d667L;

もし，オリジナルの鍵で計算する必要がある場合 (例えば，t = rkey[n] + $data$)，新しい鍵を次のようにして計算できる (t' = new_rkey[n] + $data * 4$)．また，オリジナルの鍵で計算が必要な場合は，　$t = (t' - 3)/4$ として計算をすることで，得ることができる．

線形変換の使用する替わりに，中国剰余定理による変換，bit exploded encoding [6]，秘密共有準同形 [2]，変数の結合 [8]，および error collectiong code [10] の使用などがある．

また，配列はそれ自身が攻撃者に対して大きな手掛かりであることに注意を払う必要がある．10 個のラウンド鍵があるので、攻撃者はラウンド鍵を保持する 10 の要素の配列があると予測するかもしれない．次章中のこの問題について議論を行う．

5.2　S-box の難読化

C2 暗号は，DES や AES と異なり，S-box の配列が CPPM/CPRM では公開されてない．したがって，ラウンド鍵と同様にメーカはこれを隠ぺいする必要がある．

C2 暗号の S-box のテーブルは 256 個，8 ビットの集合であるため，単純な実装による S-box は要素数 256(図 7), 8 ビットの配列になると考えられる．

そのような実装では，S-box は容易に逆アセンブルする事によって見つけることが可能である．したがって，S-box の実装をする際は，より複雑なデータ構造を使用する必要があると考える．最も単純な方法として，配列を分割したり，結合したり，混ぜ合わしたり (interleave) する事である [8]．さらに，秘密データを隠ぺいするのにふさわしいデータ構造が最近提案されている [15]．

5.3 *F* 関数の隠ぺい

P で最も実行頻度の高い関数の 1 つであり，プロファイラや AddTracer などの動的解析ツールの使用することによって，攻撃者は容易に *F* 関数を見つけることが可能である．攻撃者が *F* 関数の場所を見つけることができたのならば，ラウンド鍵を簡単に探すことができるようになる．なぜなら，ラウンド鍵は *F* 関数の入力に使われるからである．

F 関数の実行頻度を減少させる一つの手法として，*F* 関数を異なる方法で複数実装し，それらを必要に応じて任意に実行するという手法が考えられる．

さらに，*F* 関数をメソッドあるいは，関数として *F* 関数を書かないことを推奨する．また，*F* 関数を 10 回のループで書くべきでないと考える．10 ラウンドの Feitstel 構造のループであることを示すからである．また，動的解析を妨げる方法としては，ランダム性を加えた内部，あるいは外部制御フローの難読化 [5] [14] を適用することによって，Feistel 構造を隠ぺいするのに有効ではないかと考えられる．

5.4 特徴ある演算の隠ぺい

アルゴリズムをそのまま実装した C2 暗号には，いくつかの特殊 (顕著) な演算命令，および定数オペランドがある．例えば，XOR，加算，減算，ビットシフト命令などの演算，0xc9, 0x2b, 0x65, 9, 22, 0xff などの定数オペランドが挙げられる．これらの演算命令と定数オペランドは全て他の演算命令への置き換えや数値変換を行うべきである．例えば，XOR は AND, OR および NOT に置き換えることが可能である．*P* に顕著な定数オペランド (0xc9 のような) を隠すことが不十分であることに注意する必要がある．また，同様に実行トレース (図 4) に現れないためにそれらを隠す必要がある．

5.5 難読化の隠ぺい

難読化自身が特徴を表してしまうため，攻撃者に難読化を使用していることを認識させないためにこれらを隠す必要がある．例えば，中国人剰余定理 [6] を用い，ラウンド鍵の隠ぺいを行った場合，剰余 (mod) 演算がプログラム内に数多く現れてしまう．そのため，剰余演算を難読化する必要があると考えられる．

6. おわりに

本稿では，プログラム内部に秘密情報を持つ暗号プログラムを攻撃者から保護するための難読化適用のガイドラインの作成を目的に，セキュリティゴールとそれに対する攻撃モデルを定義した．具体的には，まず，C2 暗号を攻撃対象とし，攻撃者の能力モデルを定義した．次に，攻撃者が行うであろう攻撃タスクについて整理した．また，それらの攻撃を妨げるための難読化適用の検討を行った．

文　献

[1] 4C Entity, “Content protection for recordable media specification - Introduction and common cryptographic elements,” rev. 1.0, 31 pp., Jan. 2003.

[2] J. C. Benaloh, “Secret sharing homomorphisms: keeping shares of a secret,” Proc. Advanced in Cryptology, pp.251-260, 1986.

[3] S. Chow, P. Eisen, H. Johnson, P. van Oorschot, “A white-box DES implementation for DRM applications,” Proc. 2nd ACM Workshop on Digital Rights Management (DRM2002), Lecture Notes in Computer Science, Vol. 2696, pp. 1-15, 2003.

[4] S. Chow, P. Eisen, H. Johnson and P.C. van Oorschot, “White-box cryptography and an AES implementation,” Proc. 9th International Workshop on Selected Areas in Cryptography (SAC2002), Lecture Notes in Computer Science, Vol. 2595, pp. 250-270, 2003.

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

[6] S. Chow, H. Johnson, and Y. Gu, “Tamper resistant software encoding,” United States Patent 6,594,761, Filed 9 June 1999, Issued 15 July 2003.

[7] C. Collberg, C. Thomborson, D. Low, “Breaking abstractions and unstructuring data structures”, Proc. IEEE International Conference on Computer Languages (ICCL98), pp. 28-38, May 1998.

[8] 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.

[9] 神崎雄一郎, 門田暁人, 中村匡秀, 松本健一,“命令のカムフラージュによるソフトウェア保護方法,” 電子情報通信学会論文誌, Vol.J87-A, No.6, pp.755-767, June, 2004.

[10] S. Loureiro and R. Molva, “Function hiding based on error correcting codes,” Proc. International Workshop on Cryptographic Techniques and Electronic Commerce (CRYPTEC99), pp. 92-98, July 1999.

[11] J. Meyer, “D-Java,” http://mrl.nyu.edu/ meyer/jvm/djava/

[12] J. Meyer, “Jasmin Home Page,” http://jasmin.sourceforge.net/

[13] A. Monden, A. Monsifrot, and C. Thomborson, “Tamper-resistant software system based on a finite state machine,” IEICE Trans. on Fundamentals, Vol.E88-A, No.1, pp.112-122, Jan. 2005.

[14] 門田暁人, 高田義弘, 鳥居宏次, “ループを含むプログラムを難読化する方法の提案,” 電子情報通信学会論文誌 D-I, Vol.J80-D-I, No.7, pp.644-652, July 1997.

[15] 小黒博昭, 飯野徹, 平井康雅, 箱守聰, “White-box 攻撃に対する耐性を向上させた多倍長演算の実装,” 2005 年 暗号と情報セキュリティシンポジウム (SCIS2005), 予稿集 pp.145-150, 2005.

[16] A. Patrizio, “Why the DVD hack was a cinch,” Wired News, Nov. 1999, http://www.wired.com/news/technology/0,1282,32263,00.html

[17] Soot: A Java optimization framework, http://www.sable.mcgill.ca/soot/

[18] H. Tamada, “AddTracer, Injecting tracers into Java class files for dynamic analysis,” http://se.aist-nara.ac.jp/addtracer/

[19] The decompilation Wiki of Program-Transformation.Org, http://www.program-transformation.org/Transform/De-Compilation

[20] 山内寛己, “プログラムの実行系列差分による耐タンパー性能評価手法の提案,” 修士論文, 奈良先端科学技術大学院大学 情報科学研究科, NAIST-IS-MT0351135, Feb. 2005.