# マルウェアの暗号関数特定方式 ～実装と評価～

山本匠[†1†2]　J.D.Tygar[†2]

**概要**：昨今のマルウェアは暗号技術を活用し通信を秘匿する．そのため企業内でマルウェア感染が発生した際に，攻撃者がどのような情報を窃取したのかを追跡することが困難となっている．これに対応するためには，マルウェアが通信の秘匿に利用した暗号関数を見つけ，暗号化された通信を復号する必要がある．この作業には通常，マルウェアのバイナリを解析する必要があり手間暇を要する．そこでマルウェアのメモリイメージから，暗号関数によく見られる特徴やマルウェアが暗号関数を利用する際の特徴を探すことで，暗号関数を自動的に特定する手法を提案する．提案方式のプロトタイプシステムを実装し，実マルウェアを用いた評価の結果を報告する．

**キーワード**：マルウェア，暗号関数，バイナリ，マルウェア解析

# Identification of Cryptographic Functions in Malware Binaries ~implementation and evaluation~

Takumi Yamamoto[†1†2]　J. D. Tygar[†2]

***Abstract***: Recent malwares tend to use encryption on communication to avoid being exposed. Once an infected PC starts to communicate with unknown hosts via the Internet, its organization has to investigate which information has been leaked by the malware. Usually such an investigation involves malware binary analysis, which is vastly complicated and time-consuming as well, to figure out cryptographic functions and keys used in the fraudulent communication. To facilitate the investigative task, we propose a scheme to automatically detect cryptographic functions in memory images of malwares using characteristics found commonly in cryptographic implementations. Moreover we've carried out an experiment to check whether the proposed system identify cryptographic function in malware binaries.

***Keywords***: Malware, Cryptographic Function, Binary, Malware Analysis

## 1. はじめに

近年新しいセキュリティ脅威として，特定の組織をねらい，執拗に攻撃を行う Advanced Persistent Threat（APT）と呼ばれる“標的型攻撃”が顕在化している[1,2]. APT では，メールによって標的とする組織の端末にマルウェアを感染させ，感染したマルウェアが外部の攻撃者のサーバと通信を行い，新しい攻撃プログラムのダウンロードや組織システム内の機密情報の送信を行う[3].

このようなセキュリティインシデントが発生した場合，組織ではインシデントの原因や被害の調査, 対応策の検討, サービスの復旧，再発防止策の実施などのインシデントレスポンスが行われる．顧客やビジネスパートナーによっては，マルウェア感染によって何が漏えいし，あるいは何が漏えいしなかったのかを，明確にする必要がある．

インシデントの原因や被害の調査を行うにあたり，パソコン，サーバ，ネットワーク機器などが生成したログやネットワーク上で記録されたパケットを解析し，マルウェアの侵入経路，感染端末，アクセスした情報，攻撃者からの命令，外部に送信した情報などを調査するネットワークフォレンジックが重要な役割を担う[4,5,6,7,8].

ところが，昨今のマルウェアは暗号技術を活用し通信を秘匿する．そのためネットワークフォレンジックを行ったとしても，攻撃者から送られる命令が何であり，どのような情報が外部に送信されたのかを追跡することが困難となっている[9].

したがって，マルウェアが通信の秘匿に利用した暗号関数と鍵を見つけ, 暗号化された通信を復号する必要がある．通常この作業にはマルウェアのバイナリを解析する必要があり，膨大な手間と時間を要する．

この問題に対応するために，マルウェアを実行して得られる実行トレースを解析して，マルウェアが利用している暗号関数を特定する技術が提案されている[10,11,12,13,14,15]．しかし既存研究には，解析に時間を要する，大量に暗号関数の候補が出力される（誤検知），未知の暗号関数を特定できない（検知漏）などの課題がある．

そこで本稿では，マルウェアを実行した際のメモリ上のバイナリイメージを解析し，マルウェアの暗号関数の特徴を探すことで，実行トレースを用いずに短時間で暗号関数を特定する手法を提案する．

以下，2 章でマルウェアの暗号関数の特定に関する既存研究を紹介し，3 章で提案方式のコンセプトを説明する．4 章で提案方式のプロトタイプと評価実験について説明する．

†1 三菱電機株式会社
Mitsubishi Electric Corporation
†2 University of California, Berkeley

5 章で本稿をまとめる．

## 2. 既存研究

著者がこれまでに調査した限りでは，暗号関数と鍵の特定に関する既存技術は，シグネチャを用いるアプローチ[ 16 , 17 , 18 , 19 ] と実行トレースを用いるアプローチ[10,11,12,13,14,15]の 2 種類に大別される．以下では，両解析技術について概説する．

### 2.1 シグネチャを用いるアプローチ

シグネチャを用いるアプローチでは，あらかじめ暗号関数特有のコードやデータのパターンを DB に登録しておき, ハードディスク（HDD）上のプログラムのイメージにパターンが含まれているかを確認する．このパターンのことを一般的にシグネチャと呼ぶ．シグネチャは暗号アルゴリズムごとに用意される．

シグネチャを用いるアプローチは，静的解析を用いる手法に分類される. 静的解析とは, プログラムを実行せずに, HDD 上のプログラムのイメージだけから, プログラムの特徴を抽出し，解析を行う手法のことである．シグネチャを用いて既知の暗号関数を特定する既存技術としては，draca[16], SnD Crypto Scanner[17], Hash & Crypto ，PEiD[18]. Detector[19]，が知られている．

同アプローチのメリットは，高速な解析が可能なこと及びバイナリに関する高度な知識や技術をユーザに必要としない点である．一方，デメリットは，実行ファイルに圧縮や暗号化などの難読化が適用されていると，正しく解析することができない点である．また同アプローチは，未知の暗号関数の検知はできない．

### 2.2 実行トレースを用いるアプローチ

実行トレースを用いるアプローチとは，プログラムを実際に動かし，実行された命令，実行時のレジスタやメモリの値などの情報を逐次記録し，記録された情報を解析する手法である．プログラムを実際に動かすため，動的解析の 1 つと言える．

動的解析とは，プログラムを実際に実行し，実行された命令, 実行時のレジスタやメモリの値などの情報を活用し, プログラムの特徴を解析する手法である．実行トレースを用いて実行ファイルの中の暗号関数を特定する既存研究としては，文献[10,11,12,13,14,15]が知られている．

これらのツールは Intel の Pin[20]を使って収集した実行トレースを解析する. 実行トレースには, 実行された命令, 実行時のレジスタやメモリの値などの情報が記録される．実行トレースを解析することで暗号関数を特定する．実行トレースを収集する手法としては， Valgrind[ 21 ] や QEMU[22]を使う方法も知られている．

同アプローチのメリットは，メモリに展開された情報から平文や鍵などの暗号関数に入力された引数も抽出できる点である．また，同アプローチでは，圧縮や暗号化などの難読化が適用された実行ファイルも解析することができる．

同アプローチのデメリットは，実行トレースが膨大になるため, 解析に時間と記憶領域を大量に消費する点である．

近年のマルウェアの多くは，圧縮や暗号化などの難読化が適用されているものがほとんどである．そのため本研究では，実行トレースを用いるアプローチを採用する．以下では，実行トレースを用いる方式の代表的な既存研究として，Gröbert らの方式[10]，Calvet らの方式[11]，西川らの方式[15]を紹介する．

● **Gröbert らの方式[10]**

本手法では，プログラム内のループ構造を探し，ループを含むブロックに対して，暗号関数特有の命令列や定数の組みをパターンとしたパターンマッチング，入力と出力のエントロピの差が大きい処理の抽出，を行うことで暗号関数の特定を行う．暗号関数特有の命令列とは，算術演算やビット演算の並びをいう．

本手法は非常に多くの暗号関数の候補を出力するため，さらに手作業で候補を絞り込む必要がある．

● **Calvet らの方式[11]**

本手法では命令列や定数のパターンには注目しない．本手法では，プログラム内のループ構造を探し，ループを含むブロックに対して，入出力を抽出する．抽出した入力を既知の暗号関数の実装に入力し，同暗号関数を実行する．同暗号関数から得られた出力が，同ブロックの出力と等しければ，同ブロックは同暗号関数であると断定する．

●西川らの方式**[15]**

本手法では，暗号アルゴリズムが算術・ビット演算を頻繁に用いるという特徴を利用し，これらの演算が集中している場所を計算によって特定する．これにより，解析対象とする実行トレースを減らすことで，実行トレースを利用する他の既存研究よりも軽量な暗号ブロック特定手法を実現している．しかしながら，実行トレースを収集しなければならない課題は残ったままである．

### 2.3 課題設定

実行トレースを利用する方式は，マルウェアがデータの暗号化に利用した命令だけでなくレジスタやメモリ上のデータも解析に利用することができ，高い特定精度が期待される．しかし，実行トレースの取得には時間と記憶領域を大量に消費する．

また実行トレースを取得するためには，実際にマルウェアを実行する必要がある．安全のためにインターネットや組織 LAN から物理的または論理的に隔離した特殊な環境

でマルウェアを動作させ実行トレースを取得することが通常である．このような特殊な環境では，マルウェアはインターネット上の攻撃者から指令（コマンド）を受け取ることができないため，期待する動作（例えば，暗号関数の実行）を行わないことが多い．仮想環境やデバッガ上で自分が動作していることを検知し，動作を停止する耐解析機能を有するマルウェアも多い．そのため必ずしも暗号関数のコードを含んだ実行トレースを取得できるとは限らない．当然のことながら，暗号関数のコードを含まない実行トレースを解析しても，暗号関数を特定することはできない．マルウェアの動作を制御し暗号関数のコードを含んだ実行トレースを取得するまでの手間のかかる解析作業の中で，暗号関数が特定できてしまう可能性もある．

## 3. 提案方式

本章で提案方式について記述する．

### 3.1 コンセプト

提案方式のコンセプトは，メモリにロードされたマルウェアのバイナリイメージを解析し，そこから得られた特徴情報をもとに暗号関数の特定を行うというものである．あらかじめ解析しておいたマルウェアが持つ暗号関数の特徴情報をもとに，マルウェアが利用する暗号関数について機械学習を行い，学習した識別器を使って解析対象のマルウェアの暗号関数を特定する（図 1）．

提案方式は，メモリにロードされたマルウェアのバイナリイメージを静的解析し，実行トレースを利用せずに暗号関数の特定を行うため，実行トレースを用いた解析をする際，実行トレースを収集する際の時間や記憶領域の問題がない．またメモリにロードされたバイナリイメージ全体を解析するため，マルウェアが暗号関数を実行しなくとも，暗号関数の特定が期待できる．

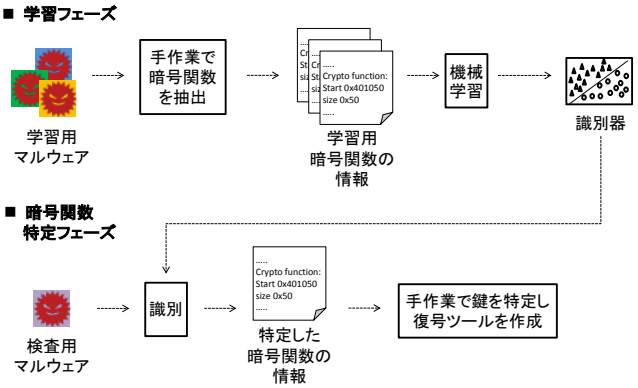


図 1　提案方式のコンセプト

### 3.2 特定対象のブロック

本稿では，暗号学でいうところの暗号関数，復号関数，ハッシュ関数に加え，圧縮関数や解凍関数，さらには，図 2 のようなマルウェア作成者が自作したと考えられる難読化用の関数も，本研究における意味での暗号関数として含め，それらを特定することを目的とする．著者らがマルウェアを解析した経験上，マルウェアの多くは，通信や命令コードを自作の関数を利用して難読化している．このような自作の難読化関数も特定することはインシデント対応作業を容易化に効果的と考える．

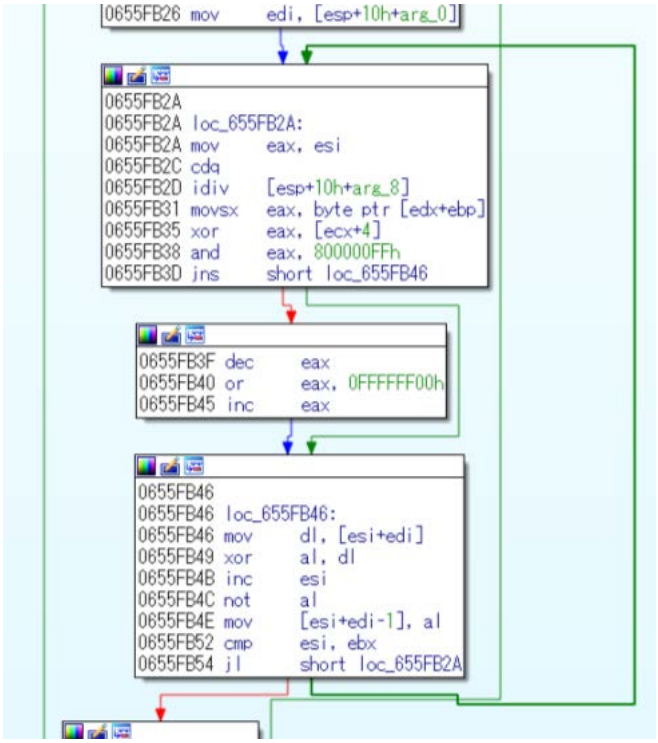


図 2　単純な暗号関数の例

### 3.3 ブロックの特定方法

CPU による基本演算は，BYTE，WORD，DWORD，QWORD などのサイズが基本である．暗号や難読化の対象となるデータのサイズは通常それらの基本サイズ以上である．すなわちデータの暗号化や難読化には基本演算処理の繰り返しが必要となる．そのため繰り返し処理を実現するループ構造をブロックとして特定する．ループの特定は以下の 2 種類の方法を採用した．

**●ループ特定方法 1**

コントロールフローグラフ（Control Flow Graph：CFG）からループを抽出する．CFG の抽出には高機能逆アセンブラである IDA Pro[23]の機能を利用する．得られた CFG のノード間で支配関係を解析することで，ループ構造を特定することができる[24,25].

IDA Pro は，関数として特定できたコードに対してのみ CFG を抽出することができる．そのため関数として特定されなかったコードに対してはループを特定できない．

表 1　機械学習に利用した特徴情報

| 特徴情報 | 説明 |
|---|---|
| ゼロ初期化以外の XOR 命令の有無 | 与えられた入力（平文）をランダムに見えるデータに変換する際に，頻繁に利用される演算であり暗号関数に特徴的である．そのため本情報を特徴情報とした．ただし xor eax, eax のように第一オペランドと第二オペランドが同一の場合，オペランドをゼロ初期化することを意味するため，このケースは除外した |
| 論理演算の数 | 数学的な演算を多く必要とする暗号関数に特徴的な演算であるため，その数を特徴情報とした |
| 算術演算の数 | 「論理演算の数」と同様 |
| 論理演算でも算術演算ない演算の数 | 暗号関数は，算術演算や論理演算の数と比べると，それら以外の演算の数の割合が少ないことが予想される．そのため，論理演算でも算術演算でもない演算の数を特徴情報とした |
| メモリ参照をする演算の数 | 暗号関数は BYTE，WORD，DWORD，QWORD などのレジスタのサイズ以上のデータを扱うため，メモリからデータの読み取り（平文や鍵の読み取り）が必ず発生する．そのためメモリの参照の数を特徴情報とした |
| メモリ書込みをする演算の数 | 「メモリ参照をする演算の数」と同様 |
| ブロックのサイズ | 暗号関数内のループのサイズに特徴があると想定し，ブロックのサイズを特徴情報とした |
| ブロックのエントロピ | 暗号関数は利用する演算に偏りがあるため，コード領域のエントロピが通常の関数とは異なる可能性がある．そのためブロックのエントロピを特徴情報とした |
| 関数呼び出しの数 | 暗号関数に利用するテーブルの作成，鍵系列の生成や取得など，サブルーチン化し呼び出す可能性があり，関数呼び出しに特徴が現れることが期待できる．そのため関数呼び出しの数を特徴情報とした |
| 循環的複雑度（ブロック内の線形的に独立した経路の数） | 暗号関数は与えられたデータの値に対し，値に応じた様々な処理を行う．そのためブロック内の制御フローに特徴が現れる可能性がある．そのため，循環的複雑度を特徴情報とした |

**●ループ特定手法 2**

命令アドレスの実行方向よりも逆，すなわち小さいアドレスへジャンプする命令を探し，ジャンプ先からジャンプ命令の場所までをループとして特定する．得られる構造としては必ずしもループを構成していないこともあるが，単純で実装が容易な手法である．

### 3.4 特定したブロックの絞り込み

既知のマルウェアから特定した暗号関数の特徴情報を機械学習し，特定したブロックを暗号関数に関係するブロックかそうでないブロックかに分類する．機械学習に利用するブロックの特徴情報には，表 1 に示す情報を利用した．

## 4. 評価

実マルウェアの中にある暗号関数を提案方式が正しく特定できるかを評価した．

### 4.1 評価用のプロトタイプ

提案方式のプロタイプの実装について表 2 に示す．

表 2　プロトタイプの実装に用いた言語とツール

| | |
|---|---|
| マルウェアのメモリイメージ抽出，ループ特定 | IDA Python （IDA Pro 6.7）<br>1434 ライン |
| 特定したブロックの分類 | Python 2.7.10<br>Scikit Learn<br>1787 ライン |

### 4.2 評価用のマルウェアのサンプル

インターネットから収集し手作業で解析できたマルウェア 15 体を用意した．15 体のマルウェアは，2009 年から 2015 年に発見されたものである．15 体のうち，2 体については提案方式では暗号関数を特定することはできないことが評価前に判明した．

この 2 体のうち 1 つは，暗号に Crypto API を利用していた．プロトタイプでは API による暗号の実装までは追跡しないため，同マルウェアの暗号関数を特定できない．もう 1 つのマルウェアは，暗号関数にループ構造を用いていなかった．提案方式はまずループ構造をもとにブロックを特定する．そのためループ構造を持たない暗号関数は特定できない．本稿ではこれら 2 つのマルウェアに関して，提案方式の現状の課題として明記し，本評価からは除外する．

### 4.3 評価環境

評価に利用したマシン環境を表 3 に示す．ホストマシンは標準的なオフィス PC である．VMWare Workstation 上のゲストマシンでマルウェアを実行しメモリ上のバイナリイメージを抽出する．

表 3　評価環境

| | | |
|---|---|---|
| ホストマシン | CPU | Intel (R) Core(TM) i7-5600U CPU @ 2.60GHz |
| | RAM | 16.0 GB |
| | OS | Windows 10 64 bit |
| | VM | VMWare Workstation 11.1.3 build-3206955 |
| ゲストマシン VM の設定 | CPU | プロセッサ数 4 |
| | RAM | 4.0 GB |
| | OS | Windows 7 Pro 32 bit |
| | その他 | IDA Pro Version 6.7 |
| | | Python 2.7.10 |

13 体のマルウェアのメモリイメージから特定したループのブロック数は 4657 件である. マルウェアを手作業で解析しこれらのブロックが暗号関数に関係するブロックかどうかラベル付けし評価用データを作成した.

なお，OS が提供している標準的な DLL のメモリ領域などはバイナリイメージの抽出対象とはしない．マルウェアが別プロセスに不正なコードを注入しそれを実行した場合，同メモリ領域はメモリイメージの抽出対象とした.

表 4 にメモリ上のバイナリイメージから特定したループ（ブロック）について記載する．なお，全ての暗号関数のブロックが通信の暗号化や難読化に利用されたわけではない. いくつかはコードの難読化や復号に利用されていた. この結果をもとに評価を行った.

表 4　ループの自動抽出と特徴抽出結果

| | |
|---|---|
| 全ブロック数 | 4657 |
| 暗号関数のブロック数 | 73 |
| 非暗号関数のブロック数 | 4584 |
| ブロックとその特徴情報の抽出に要した時間（秒/体） | 164.16 （最短 : 7.16，最長 : 1692.81, 標準偏差 : 462.06） |

ここで本評価の妥当性について記載しておく．手作業でのラベル付けは時間を要するため評価用のサンプルは少ない．またラベル付けした評価用データの質は解析者の能力に依存する．例えば，非暗号関数のブロックとしてラベル付けされたブロックが，実際はその逆，つまり暗号関数のブロックの可能性もある．また暗号関数のブロックとしてラベル付けされたブロックが, 非暗号関数の可能性もある.

そのため本評価は十分妥当性を持ったものとは断言することはできない．しかし今回のラベル付けを行った解析者の結果を，どこまで自動的に再現することができるかについては，本評価で確認することができる.

### 4.4 評価結果

交差検証（Cross Validation）を利用して提案方式を評価行った．データの分割数は 5（k=5）とした．評価結果を表 5 に示す．評価の結果，約 62%の割合で暗号関数のブロックを特定することができた．利用した機械学習のアルゴリズムは Random Forest を利用した．アルゴリズムやパラメータの設定はグリッドサーチを用いトライアンドエラーによって選択した.

表 5　暗号関数特定の精度

| | |
|---|---|
| 交差検証のスコア | 0.991 |
| 暗号関数のブロックの特定率の平均 | 0.616 (45/73) |
| 非暗号関数のブロックの特定率の平均 | 0.998 (4572/4584) |
| 交差検証（k=5）の実行時間（秒） | 256.14 |
| マルウェアのサンプル数 | 13 |
| 暗号関数のブロック数 | 73 |
| 非暗号関数のブロック数 | 4584 |

### 4.5 考察

**●検知精度**

非暗号関数のブロックとしてラベル付けされた 4572 件のブロックのうち非暗号関数に分類されたものは 4584 件であり，真陰性率（True Negative Rate）は 0.998 (4572/4584) と高い．一方，暗号関数のブロックとしてラベル付けされた 73 件のブロックのうち暗号関数に分類されたものは 45 件であり，真陽性率（True Positive Rate)）は 0.616 (45/73) と十分ではない．これは機械学習に与えた暗号関数のブロックのサンプルの数が，非暗号関数のブロックのサンプル数に比べ圧倒的に少ないことに起因すると考えられる．また今回マルウェアのサンプルが少ないこともあり，機械学習に利用した特徴情報も 10 種類と少ない. これらが真陽性率を下げた主な原因と考えられる.

**●解析時間**

特徴情報の取得に必要な時間はマルウェア 1 体につき平均約 164 秒，交差検証に必要な時間は約 256 秒である．最も時間を要したケースでも特徴情報の取得に要した時間は 30 分弱である．実行トレースを利用する既存研究では，実行トレースの取得とその解析にそれぞれ数時間以上要する. そのため提案方式は既存研究に比べ，非常に短時間でマルウェアを解析し暗号関数を特定することができると方式と言える.

## 5. 今後の課題

今後の課題は検知率の向上である．今回，提案方式の課題として除外した 2 体のマルウェアなど，Crypto API の呼び出しの特定やループ構造を使わない暗号関数の特定もで

きるよう改良が必要である．

また今回の評価実験は，実マルウェアのサンプルが少ないため十分な評価とは言えない．今後も手作業でのマルウェア解析を通じ，マルウェアが持つ暗号関数の評価サンプルを増やしていく必要がある．サンプルが増えれば，機械学習に利用できる特徴ベクトルの次元数，すなわち，特徴情報の数を増やすことができる．そのため利用可能な特徴情報についても検討していく必要がある．

最終的に，暗号通信を復号するには，マルウェアが暗号化に利用した鍵を特定する必要がある．鍵の自動特定を行うためには，提案方式によって特定したブロックをさらに解析していく必要がある．例えば，ブロックの入力情報をもとに，データフローを解析することで，利用された鍵の場所を特定することができる可能性がある．これについても，今後の課題として検討していく予定である．

## 6. おわりに

本稿では，メモリ上のマルウェアのバイナリイメージの特徴と機械学習を利用することで，マルウェアの暗号関数を特定する手法を提案した．プロトタイプを実装し，手作業で解析した 13 体の実マルウェアを使って, 提案方式の評価を行った．

交差検証の結果，約 62%の割合で暗号関数を特定することができた．特徴情報の取得に必要な時間はマルウェア 1 体につき約 164 秒，交差検証に必要な時間は約 256 秒であり，実行トレースを利用する既存研究に比べ短時間で解析を行うことが可能である．

## 参考文献


[1] シマンテック，"「標的型攻撃」に備える－サイバー攻撃: 標的型攻撃とは，APT とは", http://www.symantec.com/ja/jp/theme.jsp?themeid=apt_insight （2016 年 7 月 25 日確認）.

[2] kaspersky，"Advanced Persistent Threat（APT）攻撃：今までにない高度なマルウェア", http://www.kaspersky.co.jp/downloads/pdf/advanced-persistent-threats-not-your-average-malware_kaspersky-endpoint-control-white-paper_jp.pdf （2016 年 7 月 25 日確認）.

[3] Securelist, "The Icefog APT: A Tale of Cloak and Three Daggers", http://www.securelist.com/en/blog/208214064/The_Icefog_APT_A_Tale_of_Cloak_and_Three_Daggers（2016 年 7 月 25 日確認）.

[4] Karen Kent, Suzanne Chevalier, Tim Grance and Hung Dang, "Guide to Integrating Forensic Techniques into Incident Response", NIST, Special Publication 800-86.

[5] Paul Cichonski, Tom Milar, Tim Grance and Karen Scarfone, "Computer Security Incident Handling Guide", NIST, Special Publication 800-61 Revision 2.

[6] Sherri Davidoff and Jonathan Ham, "Network Forensics: Tracking Hackers through Cyberspace", PRENTICE HALL.

[7] Joe Fichera and Steven Bolt, "Network Intrusion Analysis: Methodologies, Tools, and Techniques for Incident Analysis and Response", FICHERA BOLT.

[8] Steven Anson, Steve Bunting, Ryan Johnson and Scott Pearson, "Mastering Windows Network Forensics and Investigation", SYBEX.

[9] Shawn Denbow, Matasano Security, "pest control: taming the rats", http://www.matasano.com/research/PEST-CONTROL.pdf （2016 年 7 月 25 日確認）.

[10] Felix Gröbert, Carsten Willems and Thorsten Holz, "Automated Identification of Cryptographic Primitives in Binary Programs", Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, 2011.

[11] Joan Calvet, Jose M. Fernandez and Jean-Yves Marion, "Aligot: Cryptographic Function Identification in Obfuscated Binary Programs", Proceedings of the 19th ACM Conference on Computer and Communications Security, 2012.

[12] 山本 匠，西川 弘毅，河内 清人，中嶋 純子，·桜井 鐘治，暗号ロジック特定手法の提案，コンピュータセキュリティシンポジウム 2014 論文集,2014(2),835-842 (2014-10-15).

[13] 山本 匠，西川 弘毅，河内 清人，中嶋 純子，桜井 鐘治，暗号ロジック特定手法の提案 その 2，SCIS2015 暗号と情報セキュリティシンポジウム.

[14] 西川弘毅，山本匠，河内清人，桜井鐘治，暗号アルゴリズムの特徴を利用した軽量な暗号ブロック特定手法，研究報告コンピュータセキュリティ（CSEC）,2015-CSEC-68(26),1-7 (2015-02-26) .

[15] 西川弘毅，山本匠，河内清人，マルウェアから暗号処理を特定する手法の実装と評価，研究報告コンピュータセキュリティ（CSEC）,2016-CSEC-72(24),1-6 (2016-02-25) , 2188-8655.

[16] Literatecode, "draft crypto analyzer", http://www.literatecode.com/draca （2016 年 7 月 25 日確認）.

[17] Collaborative RCE Tool Library, "SnD Crypto Scanner (Olly/Immunity Plugin)", http://www.woodmann.com/collaborative/tools/index.php/SnD_Crypto_Scanner_(Olly/Immunity_Plugin)（2016 年 7 月 23 日確認）.

[18] peid.info, "Peid 0.95", http://www.peid.info/ （2014 年 7 月 26 日確認）.

[19] Collaborative RCE Tool Library, Hash & Crypto Detector, http://www.woodmann.com/collaborative/tools/index.php/Hash_%26_Crypto_Detector （2014 年 7 月 26 日確認）.

[20] Intel, "Pin - A Dynamic Binary Instrumentation Tool", https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.（2016 年 7 月 25 日確認）.

[21] Valgrind Home, "Vargrind", http://valgrind.org/ （2016 年 7 月 25 日確認）.

[22] QEMU open source processor emulator, "Main Page", http://wiki.qemu.org/Main_Page （2016 年 7 月 25 日確認）.

[23] Hex Rays, IDA: About, https://www.hex-rays.com/products/ida/ （2016 年 7 月 25 日確認）.

[24] 中田育男，コンパイラの構成と最適化，ISBN4-254-12139-3 C3041，朝倉書店，1999 年 9 月.

[25] Flemming Nielson, Hanne Riis Nielson and Chris Hankin, Principles of Program Analysis, Springer, 1999.