INFORMATION
SCIENCE
TECHNICAL
REPORT

NAIST-IS-TR2009007
ISSN 0919-9527

実行系列差分攻撃による プログラムの耐タンパー性評価

山内 寛己, 門田 暁人, 松本 健一

December 2009

NAIST

〒 630-0192
奈良県生駒市高山町 8916-5
奈良先端科学技術大学院大学
情報科学研究科

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan

実行系列差分攻撃による プログラムの耐タンパー性評価

山 内 寛 己[†]　門 田 暁 人[†]　松 本 健 一[†]

本レポートでは，プログラム内部に秘密情報を持つソフトウェアを対象にして，耐タンパー性を評価する手法を提案する．提案手法は，プログラムに異なる入力を与え，それぞれの入力から実行時の実行系列を抽出し，得られた実行系列を比較するという実行系列差分攻撃を行う．ここで得られる実行系列とは，変数の参照や代入，またメソッドの開始や終了，そして演算のようなアトミックな命令を指す．実験では，暗号のアルゴリズムにも注目して，実行系列差分攻撃を行うことで，DES, C2, AES の 3 つの暗号ソフトウェアからラウンド鍵を特定することができた．また，最後に，この攻撃からソフトウェアに含まれる秘密データを保護する方法を述べる．

The Instruction Sequence Differential Attack for Evaluation of Software Tamper-Resistance

Hiroki Yamauchi,[†] Akito Monden[†] and Ken-ichi Matsumoto[†]

This thesis proposes a method to evaluate tamper-resistance of software products which includes secret information. For this evaluation, this thesis proposes a method to extract instruction sequences at runtime using different inputs, and to compare those sequences. This method is called an instruction sequence differential attack.

AddTracer is a tool for injecting tracers (monitoring code) into Java class files for extracting instruction sequence. This thesis exploits AddTracer for extracting instruction sequence including assignment and reference of variables, entering and exiting method, and calculations.

Through experimental evaluations, round keys in DES, C2 and AES cryptographic programs was revealed based on the proposed attack. Finally, this thesis shows techniques for preventing the instruction sequence differential attack.

1. はじめに

近年，不正行為を目的としたソフトウェアの解析・改ざんが問題なっており，不正行為者の存在は，ソフトウェア業界における脅威となっている．例えば，ソフトウェアのシリアルナンバーやプロダクト ID の正当性をチェックするルーチンを解析し，そのルーチンが無効になるように改ざんして，ソフトウェアを不正使用する問題は後を絶たず，開発者側に大きな不利益を生じさせている．

また，システムの安全性に関わるソフトウェアが解析され，その安全性が破られる事例が報告されている．例えば，DVD のコピー防止機能である CSS(Content Scrambling System) の解読が挙げられる[26]．

† 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

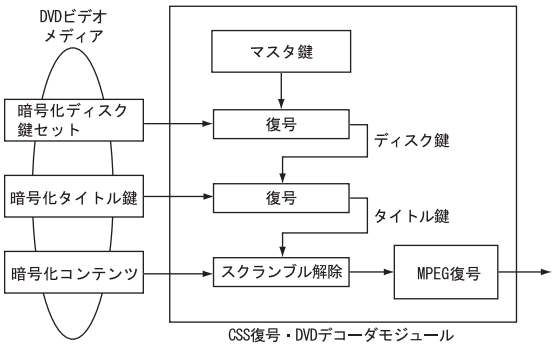


図 1　DVD のスクランブル (暗号) 解除の手順

図 1 は DVD の CSS のスクランブル (暗号) 解除の流れを示したものである[9]．DVD メディアの中には，暗号化されたディスク鍵セット，タイトル鍵とコンテンツが含まれている．コンテンツのスクランブルを解除するには，まず暗号化されたディスク鍵セットをプレイヤーに内蔵しているマスタ鍵にて

復号し，ディスク鍵を得る．そして，ディスク鍵から，暗号化されたタイトル鍵を復号し，タイトル鍵を得る．さらに復号して得たタイトル鍵から，暗号化されたコンテンツを復号し，これを MPEG でデコードを行えば，オリジナルのコンテンツが得られる．しかし，DVD のライセンスを受けていたある会社が，発売していた DVD 再生ソフトウェアに内蔵しているマスタ鍵を隠蔽 (暗号化，難読化) することを怠り，ノルウェイのハッカー集団にその DVD 再生ソフトウェアをリバースエンジニアによって解析され，その鍵を発見された．また，そのハッカー集団は CSS の暗号を解除するソフトウェア DeCSS を開発し，公開した (現在は，公開されていない)．

現在は，CSS を改良した CPPM(Content Protection for Prerecorded Media) / CPRM(Content Protection for Recordable Media)[9] というコンテンツ保護方式が策定され，市場に普及しつつあるが，プレイヤーに内蔵される秘密鍵を隠蔽しなければならないことに変わりはない．

以上のような，システムの安全性に関わる秘密データを含んだクライアントソフトウェアは，近年急激に増加している．その理由の一つは，有線放送，インターネット，携帯電話などのネットワークと介してコンテンツを購入することが可能となったためである．クライアント側では，コンテンツを復号する必要があるため，復号鍵をシステム内蔵に含むことになる．

以上のような背景から，近年，ソフトウェアの内部解析や改ざんを防止するための技術の必要性が著しく高まっている．それに呼応して，プログラムの解析を困難にする技術として，プログラムの難読化や暗号化など，いくつかの手法が提案され，製品化されている (2 章で説明する)．

プログラムの難読化とは，与えられたプログラムを理解しにくい (読みにくい) プログラムに変換する技術であり，難読化ツールにかけるだけで容易にプログラムを保護できるため，よく用いられている．プログラムの暗号化は，あらかじめ暗号化しておいたプログラムをユーザに配布し，実行時に復号する技術である．こちらも暗号化／実行時復号ツールが数多く普及している．

しかし，これらの技術は，その有効性がこれまで十分に評価されてきたとはいえない．プログラムに対して一定の保護を与えることは確かであるが，前述のような鍵内蔵プログラムに対して鍵を隠蔽するのに有効とは限らない．松岡ら[15] は，ソフトウェア技術者を被験者として，難読化された DES 復号プログラムを解析して復号鍵を導出する実験を試みたが，鍵を導出することができたという結果を報告している．

本レポートでは，共通鍵ブロック暗号の復号プログラムを対象とし，プログラム内部の復号鍵の耐タンパー性 (鍵の導出の困難さ) を評価する方法を提案する．共通鍵ブロック暗号は，多くのコンテンツプレーヤで用いらており，耐タンパー性を評価することの必要性は高い．例えば，前述の CPPM/CPRM では，C2 暗号[1] と呼ばれる共通鍵ブロック暗号方式が採用されている．

耐タンパー性を評価するに当たっては，何らかの攻撃モデルを定義し，そのモデルに対して鍵導出がどのくらい困難かを評価する必要がある．提案方法では，多くのハッカー (クラッカー) が行う攻撃である，(1) デバッガやメモリダンプツールによるメモリの覗き見，(2) メモリの差分の分析，(3) プログラムの仕様やアルゴリズムを考慮した探索範囲の絞込み，に基づいた攻撃モデルを想定する．そして，モデルに基づいて容易に評価を行うための手段として，プログラムの実行系列生成ツール Addtracer を用いた攻撃方法を提案する．提案方法を用いてプログラムから鍵の導出を試みることで，耐タンパー性が評価できる．

以降，本レポートでは，次の 2 章では，プログラムの解析を困難にする技術について説明後，これら技術の評価の難しさについて述べる．3 章では、どのような方法でプログラムが解析されるかについて述べる．4 章では，提案方式を適用する具体例を述べる．5 章で実験を行い，考察を行う．最後に 6 章で，まとめと今後の課題について述べる．

2. ソフトウェア耐タンパー技術とその評価

2.1 ソフトウェア耐タンパー技術

プログラムの解析を困難にする技術をソフトウェア耐タンパー技術といい，プログラムの解析の困難さの性質を耐タンパー性という．ソフトウェア耐タンパー技術は，従来数多くされており，それらの技術はプログラムの難読化，プログラムの暗号化などがある．

プログラムの難読化 (program obfuscation) は，与えられたプログラムを読みにくい (複雑な) プログラムに変換することで，解析に要するコストを増大させる技術である．難読化したプログラムは，その表現や計算手順が複雑化しており，人間にとっ

て解析が困難となっているが，難読化していないプログラムと同様，計算機上で実行が可能である．開発したプログラムを難読化してからプログラムの使用者 (ユーザ) へ配布することで，ユーザや第三者によってプログラムが解析される危険性を減らすことができる．プログラムの難読化の具体的な方式としては，プログラムの制御構造を複雑にする方式[18)19)]，高レベル命令を，複数の低レベル命令に置き換える方式[17)20)]，プログラムの実行結果に影響を与えない与えない (無意味な) プログラムコードを挿入する方式[5)6)]，データ構造を変形する方式[7)]，プログラム中の手続きの名前 (メソッド名) を変換する方式[29)]，配列やポインタの参照，代入を利用してプログラムを複雑化する方式[25)32)]，プログラムの自己書き換えを利用した方式[12)]，マルチバージョンソフトウェアを応用した方式[30)] などがある．

プログラムの暗号化 (program encryption) は，プログラムの全体または一部を暗号化することによって，解析を困難にする技術である．暗号化された部分のプログラムは，人間が読んでもその内容を理解することは不可能である．ただし，暗号化された部分のプログラムはそのままでは計算機で実行できないため，プログラムの実行直前，もしくは，実行中に必ず復号されることになる．したがって，暗号化された命令コードを復号するための機構をあらかじめ追加しておく必要がある．プログラムの暗号化の具体的な方式は，文献[10)11)] などにおいて提案されている．

2.2 耐タンパー性評価の問題点

ソフトウェア保護を行う者は，攻撃者から隠蔽したい具体的な秘密情報 (定数，アルゴリズム，サブルーチンなど) に見合った耐タンパー技術を採用する必要がある．しかし，これまで提案されてきた難読化法の多くは，「ソフトウェアの解析を困難にする」という漠然とした目的しか述べられておらず，その評価もごく限定的であった．そのため，防御者が隠蔽したい情報がどの程度保護できるのか不明確であった．ソフトウェアに含まれる秘密データを整理・分類し，従来の保護方式が各秘密情報をどの程度隠蔽できるのかを評価する必要がある．

また，従来のソフトウェア耐タンパー技術の研究において想定されている攻撃モデルは，現実的な攻撃 (特に動的解析を含むもの) に即したものとはいえない．例えば，ポインタが指すオブジェクトを特定する points-to analysis という静的解析が一般に NP-Hard となることを根拠として，解析が困難であること (保護の効果が高いこと) を主張している研究[24)] があるが，実際に攻撃者が points-to analysis を行うかどうかは疑わしい．例えば，鍵データを含むソフトウェアを解析する者は，ポインタの解析ではなく，プログラムに含まれる定数やプログラム実行中のメモリに現れる数値に着目して鍵データの候補を探そうとするであろう．

プログラムの複雑度メトリクスにより解析の困難さを評価する研究[5)] もあるが，解析の困難さとプログラムの複雑度は必ずしも相関がないことが別の研究によって指摘されている[21)]．例えば，ループや変数の数が多いからといって，プログラムから鍵データを発見しにくいとは限らない．

従来，多くの攻撃モデルでは，攻撃者は静的解析のみを行うものと想定されていたため，動的解析が行われることを前提に解析の困難さを評価した耐タンパー技術はほとんど存在しなかった．しかし，最近では，デバッガやメモリダンプツールなどの動的解析ツールが広く普及しており，動的解析による攻撃がむしろ主流となっている．現実的な攻撃方法を整理・分類し，各攻撃に対する耐性を評価することが求められる．

動的解析に着目したソフトウェア耐タンパー技術の評価手法として，赤井ら[2)] は，プログラムのランタイムデータ (実行時の変数スタックの状態の系列) を用いて，スタックメモリ中に秘密データが表れるかどうかを評価し，プログラムの実行中に秘密データが見つからないように実装をすべきと提案している．この評価方法は，静的解析のみを対象とした評価と比べてより現実的であると言える．

しかし，実際のプログラム開発現場では，実行時のメモリ上の秘密データが現れないようなプログラムを作成することは容易でない．例えば，Chow ら[3)] は，DES の復号プログラムに含まれる復号鍵を隠蔽する方法を提案しているが，実行効率の低下，及び，プログラムサイズの増加が著しいため，実用システムに採用することは難しい．また，プログラム中の定数値を隠蔽する難読化方法として，隠蔽したい定数 x を特定の準同型写像によって別のドメインの値に変換し，以降の x に対する演算を変換後のドメインで行うようにプログラムを変換する方法が提案されている[4)]．しかし，準同型写像として線形変換を用いる方法 (文献[4)] の linear encoding) は，ビット演算 (AND, OR, XOR など) を含む復号プログラムには適用が難しい．また，XOR 演算

を用いる方法 (文献[4] の bit-exploded coding，及び，custom base coding) は，和や積の演算を含む復号プログラム (C2 暗号など) には適用が難しい．従って,「メモリ上に秘密データが現れないこと」は，ソフトウェアの耐タンパー性を評価するための有力な基準ではあるが，満たすことが難しい基準であるといえる．

2.3 本レポートにおける耐タンパー性評価のアプローチ

本レポートでは，秘密データは，プログラム実行中のメモリ上に現れるものと仮定する．この仮定の下で，メモリ上から秘密データを特定することの困難さを評価することを目的とする．

一般に，メモリ上に秘密データが現れたとしても，攻撃者が即座にそれを特定できるとは限らない．多くの共通鍵ブロック暗号の復号プログラムでは，隠蔽すべき秘密データとして，ラウンド鍵と呼ばれる復号鍵が複数個 (十数個) 存在する．たとえこれらのラウンド鍵がプログラム実行中のメモリ上に現れたとしても，全てのラウンド鍵を正しく特定することは攻撃者にとって必ずしも容易でない．

例えば，実行時にメモリ上に現れる数値の数を n とすると，k 個のラウンド鍵を全数探索により求めようとすると，n^k 回の試行 (探索) が必要となる．n が大きい場合，現実的な時間で正しいラウンド鍵の集合を見つけることは不可能となる．ただし，現実の攻撃者は，単純な全数探索は行わないであろう．攻撃対象のプログラムで採用されている暗号方式の知識に基づいて，ラウンド鍵が現れそうなメモリ上の位置を特定し，各ラウンド鍵の候補を絞り込むと想定される．

防御者の立場からすると，攻撃者の行いそうな行動を予め予測し，各ラウンド鍵の候補を絞り込むことが困難なようにプログラムを作成すればよいことになる．そのためには，プログラムリストを難読化するのではなく，プログラムの実行系列 (メモリ上の値の変化) を難読化する必要がある．

以上の議論に基づき，本レポートでは，プログラム実行中のメモリ上に現れる数値から秘密データを特定するための攻撃モデル，及び，モデルに基づく攻撃方法を提案する．提案方法を用いてプログラムから鍵の探索を試みることで，鍵探索の困難さが評価できる．

3. ソフトウェアの解析手法

ソフトウェアの解析手法は大別すると 2 つに分類ができる．ひとつは解析対象となるプログラムを動作させずに行う静的解析であり，もうひとつは実際にプログラムを動作させながら行う動的解析である．

3.1 静的解析によるソフトウェアの解析

3.1.1 逆アセンブラを用いた解析

プログラムは，CPU に対する命令とデータを表したビット列の並びといえる．これらは，マシンコードあるいは，マシン語などと呼ばれている．通常マシンコードは 16 進数の並びとして表現されるが，それぞれの数字がどのような命令を意味しているのかを判読することは非常に困難な作業である．そこで，そのマシンコードを可読可能なアセンブリコードに変換するツールがディスアセンブラ (disassembler) である．アセンブリコードはマシンコードと 1 対 1 に対応しており，16 進数の羅列に比べてはるかに可読性が高い．また，プログラムが OS に対して呼び出している API(Application Programming Interface) や，システムコールの名前や，静的に保持しているデータを表示させることもできる．これらのディスアセンブラの機能を利用し，ソフトウェア内部構造を把握し命令列を詳細に調べることができる．しかし，プログラムに対して逆アセンブルを行うとアセンブラコードは膨大な量になるため，その全てを解析するには根気と時間を要することになる．

3.1.2 逆コンパイラを用いた解析

デコンパイラ (decompiler) は，マシンコードやバイトコードなどをソースコードに復元するツールである．マシンコードをソースコードに復元すること困難な課題であり，実用レベルの効率的な解析に役立つレベルのものはない．なぜなら，一般ユーザにリリースされるソフトウェアは通常はシンボル情報が削除されており，関数名や変数名などの手がかりを得ることができない．また，通常コンパイル時の最適化処理により，よく複雑なマシンコードに変換されており，デコンパイルできても複雑で読みにくいソースコードになってしまう．ただし，Java や．NET などの中間言語については，いくつかの有力な逆コンパイラが存在する．

3.2 動的解析によるソフトウェアの解析

3.2.1 メモリダンプを用いた解析

メモリダンプは，対象のプログラムが実行中にメモリの使用している内容を記録，あるいは表示することである．プログラムを開発するときに動作を追跡するために利用することが多く，ダンプされた

内容はデバッガに読み込ませてプログラムの問題を分析するために用いられる．ソフトウェアの解析においては，例えば暗号化されたプログラムを実行して，メモリ上に復号されたオリジナルのプログラムをメモリダンプにて入手したり，メモリ上に書き出されたデータを時間毎，あるいは，データの変更毎にダンプを行い，それぞれの差分をとることでデータの内容，または対応関係を把握したりする事が可能となる．

UNIX 系の OS では，プログラムが不正終了したときに自動的にその時点でプログラムが使用していたメモリの内容をダンプするようになっている．

3.2.2 デバッガを用いた解析

デバッガは，ソフトウェア開発過程において実装上の不具合 (バグ) を発見するためのツールである．開発者はソースコードを持っているので，ソースレベルで解析 (デバック) を行うことができる．一方，解析者はソースコードを持っていないので，アセンブリ言語レベルの解析を行うことになる．ソースコードレベルに比べれば，困難な作業になるが，解析対象となるソフトウェア内の注目する箇所にブレークポイントを設定したりステップ実行させたりしながらメモリやレジスタの内容を調べたり，制御フローを追跡したりすることができる．通常のデバッガはアプリケーションレベルで動作するが，デバイスドライバなどのハードウェアと通信を行うソフトウェアをデバッグする際には，カーネルモードデバッガを使用する．カーネルモードとは，カーネル領域で OS のカーネルプログラムを実行するためのもので特権命令 (入出力命令，割込み禁止命令，記憶保護設定命令など) を行使することができる．そのため，OS レベルでの細かいデバッグが行える．

その他には，インサーキットエミュレータ (ICE:In-Circuit Emulator) と呼ばれる解析ツールは CPU とロジックボードの間に入ってすべての命令やデータを観察できるツールである．OS やデバイストライバなどハードウェアに密着したプログラムのデバッグや解析に用いられるが，OS 上で動作する一般のアプリケーションを解析する用途には向かない．また ICE は一般に高価であり，誰もが利用できるツールではない．そのため，アプリケーションの解析ツールとしての優位性は低い．PC のハードウェアをソフトウェアでエミュレーションするといわゆる PC エミュレータというソフトウェアがある．エミュレータ自身は 1 つのアプリケーションであるため，デバッガ上でエミュレータを動作させることも原理上は可能だが，解析対象のアプリケーションから見ると，ICE と同じレイヤーで解析されることになるため，やはり有効な解析手段とはいえない．

3.3 解析の流れ

ソフトウェアの解析は人為的な作業であり，その作業の概略をモデル化して記すと図 2 のようになる．解析者は，まずはじめに最初に解析対象のプログラムを静的状態，つまりプログラムを実行しない状態で解析することを試みる．もしくは，実行状態で解析する．前者の場合，攻撃者は逆アセンブラを用い，得られたアセンブリプログラムを解析する．その結果，プログラムの制御フローを理解し，改変を施して秘密情報の取り出しを試みようとする．それが達成できれば解析は成功となる．静的解析で解析が十分にできないと判断した場合は，解析者は次にプログラムを実行させながら解析を試みようとする．仮にプログラムコードの一部が暗号化されていて実行時に復号される解析防御手法が施されている場合，暗号化部分はそのままではプログラムコードとしては意味を持たないので，静的解析は不可能である．しかし，プログラムを実行させながら動的解析を行えば，暗号化が施されている部分を復号された状態で観測できる．また，実行中に計算されるデータの値やその制御フローを，静的解析に比べて効率よく観察することもできる．解析者はデバッガを用いてコードをトレースし，条件分岐命令を無視させたり，計算結果の値を記録したり，また値を書き換えて実行させたりするであろう．その結果，解析が成功するかもしれない．解析が不十分な場合であっても，解析者はそれまでのコードの実行履歴を記録できる．この情報を用いて，再び静的解析に戻り，解析を進めることも可能となる．このように解析者は，静的解析と動的解析を組み合わせ繰り返しながら解析を進めていくと考えられる．

4. 実行系列差分による耐タンパー性評価手法

4.1 評 価 対 象

提案する実行系列差分による耐タンパー性評価手法は，プログラム内に存在する秘密データの守秘性を評価する．

解析の対象となるプログラムは，アルゴリズムが公開されている共通鍵ブロック暗号の復号ルーチンで，Java で実装しているものとする．そして，そ

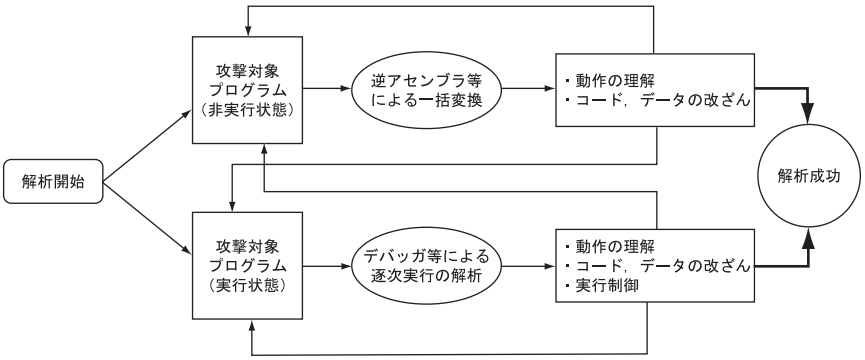


図 2 解析手順のフローチャート

のソースコードをコンパイルし，Java クラスファイルに変換したものを使用する．

解析者は復号鍵を求めるために，入力にあたる暗号文と入力より得られた出力の平文のみによる入出力解析に加え，評価対象プログラムに対して，静的解析と動的解析を行うことができる．

また，解析によって探索する秘密データは復号鍵以外に，次のものが考えられる．

4.1.1 ラウンド鍵

共通鍵ブロック暗号において，復号鍵がなくても，r 段のラウンド関数と r 個のラウンド鍵の系列 $\{S_i\}$ があれば，暗号化，あるいは復号の処理が可能となる．

4.1.2 S-box

S-box は入出力サイズの小さい非線形変換である．通常，入出力対応がテーブルで与えられるため，Substitution box の省略形からこのように呼ばれる．複数の並列の S-box によって S-box 層を構成する．暗号アルゴリズムによって，S-Box を使用する数が変化するが，この S-Box は非公開な場合もあるため，プログラムの解析によって S-Box が知られるべきではない．

4.2 提案方法の概略

提案方法では，デバッガやメモリダンプツールを用いてハッカー (クラッカー) が行う攻撃をモデル化する．ただし，クラッカーの行う攻撃は，デバッガ，アセンブリ言語，バイナリプログラムの構造などについての詳細な知識が要求され，また，多大な時間を要するため，攻撃をそのままモデル化しても評価に用いるのは難しい．そこで，提案方法では，プログラムの実行系列 (プログラム実行時に各変数に代入・参照される数値の系列) を容易に得ることができるツール (Addtracer：詳しくは 4.3 節で説明する) を用いることを前提とし，得られた実行系列から系統的に数値の絞込みを行うことで秘密データの候補を得ることにする．この提案方法は，デバッガやアセンブリ言語などの知識を必要とせず，Java プログラムが理解できる人間であれば容易に使うことができる．また，デバッガやメモリダンプツールを用いた攻撃よりも効率よく鍵候補を探せるため，提案方法により秘密データが得られないならば，デバッガやメモリダンプツールによる攻撃に対しても耐タンパー性が確保できていると判断できる．

提案方法では，次の 3 つのステップから成る．

(1) プログラム中の実行系列の収集 (4.3 節)
　このステップでは，デバッガやメモリダンプツールを用いてメモリののぞき見を行い，秘密データの候補となる数値の集合を得るという攻撃を，最も効率よく行った場合をモデル化する．

(2) 複数の実行系列間の差分の取得 (4.4 節)
　一般に，クラッカーは，プログラムに複数の互いに異なる入力を与え，実行時のメモリの差分をとることで，入力値とメモリ上の値との関連を調べるという攻撃を行う場合がある[14]．このステップは，このような攻撃をモデル化する．

(3) 秘密情報の存在箇所の絞込み (4.5 節)
　攻撃対象となるプログラムの仕様やアルゴリズムが既知である場合，クラッカーはそれらの知識を用いてデバッガにブレークポイントを設定したり，探索すべきメモリの範囲を特定したりする[31]．このステップは，このような攻撃をモデル化する．

本章の以降では，各手順の詳細を述べる．

4.3 実行系列出力ツール：AddTracer

従来，プログラムの実行系列を取得するためには

デバッガを用いなくてはならなかった．しかし，デバッガを用いる方法は，実行系列をバッチ処理的に抽出することが難しいという問題がある．そこで，本研究において，実行系列を取得する方法として，AddTracer[27] を用いることにする．

AddTracer は Java プログラムの動的解析を容易にするため，開発されたトレーサ埋め込みツールである[28]．動的解析は非常に高度な知識を必要とする作業であり，初心者が行うことは難しく，多くの場合，プログラムのソースコードも合わせて必要とされる．しかし，AddTracer は Java クラスファイルを入力として，変数に値が代入されたとき，変数の値を参照したとき，演算が行われたとき，メソッドの開始時，終了時にプログラムをモニタリングするための情報を出力するコードを埋め込み，クラスファイルを出力するコンバータとして実装されている．入力をクラスファイルとしているため，ソースコードを必要とせず，また，AddTracer が出力するクラスファイルを実行させることで，動的解析に必要な多くの情報が出力されるため，初心者であっても容易に動的解析を行うことができる．

実行系列を取得することができる他のツールとして，DataExtractor が挙げられる[16]．DataExtractor は AddTracer と同じく，プログラムの実行途中のデータを捕捉することを可能にするツールである．しかし，AddTracer とはアプローチが異なっており，DataExtractor は Java Platform Debugger Archtecture を拡張してデータの捕捉を実現させている．しかし，この方法では Java 仮想マシンに手を入れる必要があり，また，HotSpot と呼ばれる動的コンパイラには未対応であり，この点が DataExtractor の弱点となっている．

表 1 AddTracer が出力する情報

	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

4.3.1 AddTracer が出力する情報

AddTracer は表 1 で表われる出力を行う．ローカル変数の参照，代入時には変数の ID，値とその時の行数を出力する．入力で与えられたクラスファイルがデバッグ情報を持っていれば変数の ID は変数名となる．また，フィールド変数の参照，代入のときには，ローカル変数の参照，代入の情報に加えて，どのクラスの変数なのかを示すため，クラス名も出力する．加えて，これらの変数がもし配列ならば，配列のどの要素にアクセスしたのかを表すインデックスも出力する．

演算が行われたときには演算子と演算の型，そして，演算結果とその処理の行数を出力する．メソッドの開始，終了においてはそのメソッドが定義されているクラス名とメソッド名，そして，その処理の行数を出力する．

出力のサンプルとして，図 3 に示す Fibonacci 数列を計算するプログラムを AddTracer にかけ，fibonacci メソッドに 4 を渡して実行した結果，得られる出力を図 4 に示す．

```
1: public class Fibonacci{
2:     public int fibonacci(int n){
3:         if(n <= 0)
4:             throw new IllegalArgumentExcep
                              tion(Integer.toString(n));
5:         if(n <= 2)
6:             return 1;
7:         return fibonacci(n - 1) + fibonacci(n - 2);
8:     }
9: }
```

図 3 サンプルプログラム (Fibonacci 数列)

4.4 複数の実行系列差分の取得

解析の対象のプログラムに互いに異なる入力を与えると，入力に依存する部分と依存しない部分とに分かれる．図 5 の Feistel 型構造において，入力データの左ブロック L，右ブロックを R，ラウンド鍵を K とすると，本レポートの対象モデルでは，鍵は固定なため，ラウンド鍵 K はどんな入力にも依存せず，常に値は変化しない．しかし F 関数は，入力データの右ブロック R とラウンド鍵 K を引数とした関数なので，$F(R, K)$ は入力によって値が異なる．

また，ブロック暗号はラウンド数が固定のものが多く，互いに異なる入力を与えても常に実行系列の長さがほとんど変化しないため，共通鍵ブロック暗号の実行系列の差分を直接とることが可能である．

4.5 秘密データの絞り込み

共通鍵ブロック暗号のほとんどは，図 6 のように入力された平文をランダム置換して暗号文として出力するデータ攪拌部と，暗号化鍵から拡大鍵を生成しデータ攪拌部に供給する鍵スケジュール部から

```
start Fibonacci#fibonacci at line 3
n       assignment (line 3):    4
n       reference (line 3):     4
n       reference (line 5):     4
this    reference (line 7):     Fibonacci@121cc40
n       reference (line 7):     4
- <int> operation (line 7):     3
start Fibonacci#fibonacci at line 3
n       assignment (line 3):    3
n       reference (line 3):     3
n       reference (line 5):     3
this    reference (line 7):     Fibonacci@121cc40
n       reference (line 7):     3
- <int> operation (line 7):     2
start Fibonacci#fibonacci at line 3
n       assignment (line 3):    2
n       reference (line 3):     2
n       reference (line 5):     2
end Fibonacci#fibonacci at line 6
this    reference (line 7):     Fibonacci@121cc40
n       reference (line 7):     3
- <int> operation (line 7):     1
start Fibonacci#fibonacci at line 3
n       assignment (line 3):    1
n       reference (line 3):     1
n       reference (line 5):     1
end Fibonacci#fibonacci at line 6
+ <int> operation (line 7):     2
end Fibonacci#fibonacci at line 7
this    reference (line 7):     Fibonacci@121cc40
n       reference (line 7):     4
- <int> operation (line 7):     2
start Fibonacci#fibonacci at line 3
n       assignment (line 3):    2
n       reference (line 3):     2
n       reference (line 5):     2
end Fibonacci#fibonacci at line 6
+ <int> operation (line 7):     3
end Fibonacci#fibonacci at line 7
```

図 4 図 3 の fibonacci メソッドに 4 を渡して得られた結果

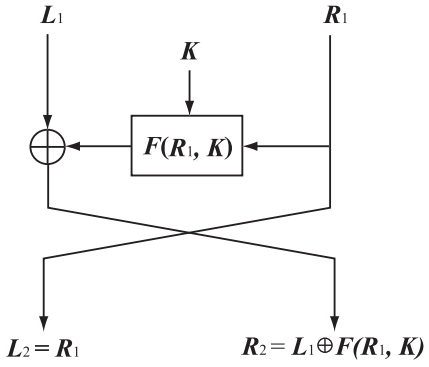


図 5 Feistel 型構造

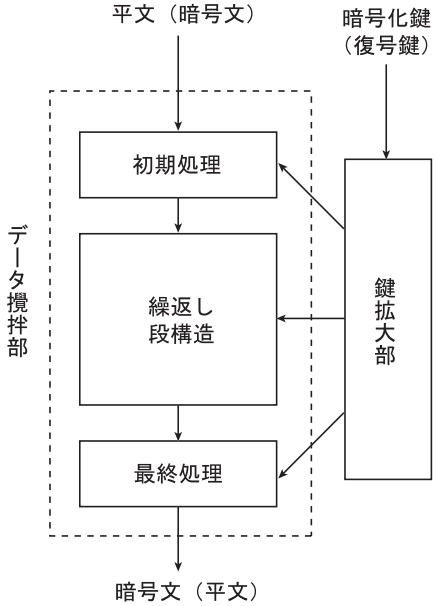


図 6 共通鍵ブロック暗号の基本構成

構成される．

ブロック暗号では，通常，実装コストを下げるため，データ攪拌部は同じ関数の繰り返しで構成する．繰り返しの単位をラウンド，1 ラウンドの処理をラウンド関数と呼ぶ．ラウンド関数において，入力は前ラウンドの出力と鍵拡大部が生成する拡大鍵の一部であるラウンド鍵，出力は次のラウンドの入力となる．

提案手法では，特にラウンド関数に着目して秘密情報の探索を試みる．データ攪拌部のラウンド関数は，繰り返しによって構成されているため，ラウンド関数の実行系列をとると，一段のラウンド関数の実行系列を元にパターンになると予想される．ラウンド鍵は入力データがないと，次のラウンドの計算が行えない．また，ラウンド関数の前後に S-Box が存在する．

これらの仮定を検証するため，次の 5 章で実験を行う．

5. 提案手法に対する評価実験

5.1 実 験 概 要

この章では，提案する手法によって，評価対象のプログラムから秘密データが取り出せるか実験を行った．具体的には，4.1.1 や 4.1.2 のところで述べたラウンド鍵や S-Box を探索し，発見することである．本実験では，DES[23]，C2 暗号 (Cryptomeria Cipher)，AES[22](Rijndael[8]) を実装し，これらを評価対象のプログラムとした．また，評価対象のプログラムは Java のクラスファイルとした．なお，この実験は，CPU が Intel Pentium4 Extreme

Edition 3.20 GHz，メモリ容量が 1024 MB，OS が Windows XP SP2 のマシンを用い，Java 言語を使用し，J2SDK 1.4.2_06 を実行環境とした．

5.2 実験手順

以下に示す手順に従って実験を行った．

(1) アルゴリズムが公開されている共通鍵ブロック暗号アルゴリズムを実装した Java のソースプログラムをコンパイルし，Java クラスファイルを作成する．

(2) 1. で作成した Java クラスファイルに AddTracer を用いて，実行系列がトレースできるように変換を行う．

(3) 2. で得た Java クラスファイルに対して，復号鍵を固定して，入力の暗号文を複数用意し，各入力に対する対象プログラムの実行系列 (トレース) を得る．

(4) 3. で得た実行系列を入力の異なる実行系列と差分をとる．この差分を元に，共通鍵ブロック暗号プログラム内の秘密データを探索していく．

5.3 実験結果と考察

5.3.1 DES

DES は入力ブロック長 64bit，鍵長 56bit，ラウンド数 16 である (図 7)．1977 年に正式採用されてから，現在では，計算機の発達により鍵の総当り攻撃が容易にできるようになってしまったため，安全とはいえなくなった．

AddTracer による実行系列は，全体で 53,810 行になった．ラウンド部 (16 段) とラウンド鍵の候補は約 1,668 行まで絞り込むことができた．プログラムの実装上，鍵が int 型により実装されていたため，1 段のラウンドに対して，2 つの鍵があった．ラウンド鍵を得るためには，まず入力の暗号文がはじめに代入されているところを探し，ラウンド鍵の候補が絞ることができた (図 10)．S-box は配列のサイズが 64 の配列が 8 個あるため，容易に探し当てることができた．

5.3.2 C2 暗号

C2 暗号は DES と同じく，ラウンド部に Feistel 構造 (図 5) をもつ．入力ブロック長は 64bit，鍵長 56bit であるが，ラウンド数は 10 で，ラウンド鍵を生成する鍵拡大部が DES のように対称ではなく非対称であり，ラウンド鍵の鍵長も DES が 48bit なのに比べ，32bit の鍵長と短い (図 8)．

AddTracer による実行系列は全体で 2,412 行であり，ラウンド部 (10 段) と鍵候補がおよそ 1,570 行ほどに絞れた (図 11)．S-Box☆は一つしかなく，プログラム実行時に配列に値が代入されるため，特定することが容易である．

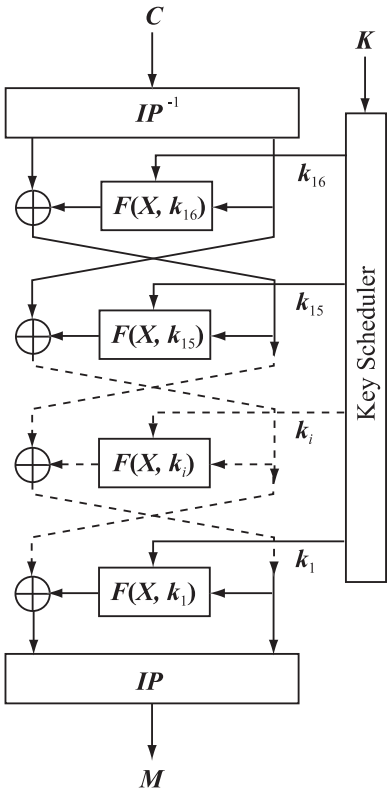


図 7 DES の復号アルゴリズムの構成

5.3.3 AES(Rijndael)

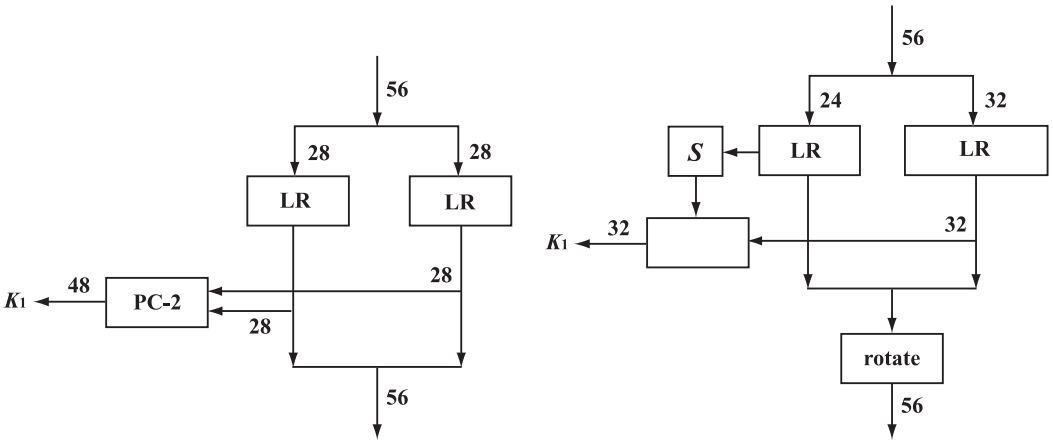
AES(図 9) は，DES に替わる次世代の暗号として 2001 年に Rijndael が AES として採用されたアルゴリズムである．入力ブロック長は 128bit，鍵長は 128, 192, 256bit の 3 種類がある．ラウンド数は鍵長によって異なり，10, 12, 14 となっている．実験は鍵長を 128bit で行ったので，復号に必要な鍵の数はラウンド数 +1 の計 11 個となる．

AddTracer による実行系列は，全体 728,197 行に及んだ．そのうち，2,767 行に鍵の系列があると特定できるパターンが 11 個が現れた (図 12)．

5.4 実行系列差分攻撃秘密データを守る方法

提案手法を用いた攻撃から，プログラム内の秘密データを守るには，データの存在を特徴づける差分を解析者がとる事ができなければ，秘密データの保護が可能だと考える．以下のようなものが考えら

☆ 今回の S-box は，ライセンス元の 4C Entity が公開しているサンプルの S-box を使用．正規の S-box はベンダーのみに公開している．



DES の鍵拡大部　　C2 暗号の鍵拡大部

図 8　DES と C2 暗号の鍵拡大部の比較

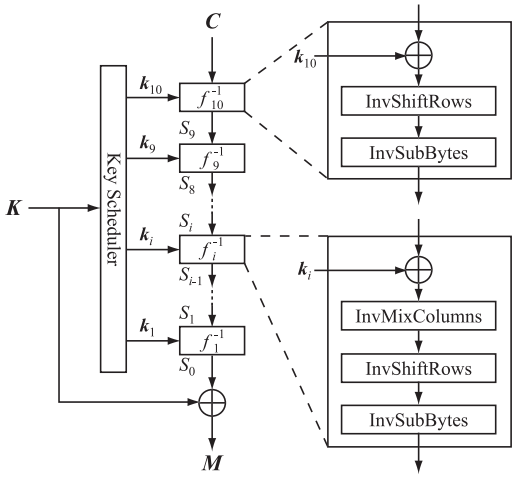


図 9　AES の復号アルゴリズムの構成

れる．

5.4.1　ダミーコードの挿入

対象のプログラムの機能とは関係ないダミーコードを挿入する．これにより，特徴のある差分が出る部分にダミーコードの挿入を行うことで，解析範囲を広げることとなり，解析にかかるコストが増えることが期待される．

5.4.2　乱数を用いた実行順序の入れ替え

順序依存のない処理の部分を乱数を用いて，その部分を実行するたびに順序を入れ替えるという手法である．これを用いることで，特にラウンド関数などの繰り返して処理を行うところを，差分をとれなくすることができると考えられる．

6. おわりに

本レポートでは，メモリ上から秘密データを特定することの困難さ評価をすることを目的として，プログラム実行中のメモリ上に現れる数値から秘密データを特定するための攻撃モデル，及び，モデルに基づく攻撃方法を提案した．提案方式を用いて，共通鍵ブロック暗号プログラムから鍵の探索を試みることで，鍵探索の困難さが評価できる．

最後に，今後の課題について述べる．まず，今回実験対象として扱わなかった他の共通鍵暗号アルゴリズムに対して，評価することが挙げられる．この評価手法が今回評価に用いた暗号アルゴリズムに対して効果的だったのか，評価手法として妥当なのかを調べる必要がある．また，データの変換を行う難読化について，評価ができるのではないかと考えている．提案手法によってデータの変換する難読化を施した値を見つけ，その変換はどのように変換しているのかを静的解析で見つけることができるのではないかと考える．

参 考 文 献

1) 4C Entity, “Content protection for recordable media specification - Introduction and common cryptographic elements,” rev. 1.0, 31 pp.1.0, Jan. 2003.
2) 赤井 健一郎, 三澤 学, 松本 勉, “ランタイムデータ探索型耐タンパー性評価法,” 情報処理学科論文誌, vol.43, no.8, pp.2447-2557, Aug. 2002.
3) 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.
4) S. Chow, H. Johnson, and Y. Gu, “Tamper resistant soft-ware encoding,” United States Patent 6,594,761, Filed 9 June 1999, Issued 15 July 2003.
5) C. Collberg, C. Thomborson, and D. Low, “A taxonomy of obfuscation transformations,” Technical report 148, Department of Computer Science, the University of Aukland, Aukland, New Zealand, 1997.
6) C. Collberg, C. Thomborson, and D. Low, “Manufacturing Cheap, Resilient, and Stealthy Opaque Construcs,” ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages(POPL98), San Diego, California, 1998.
7) C. Collberg, C. Thomborson, and D. Low, “Breaking Abstractions and Unstructuring Data Structures,” IEEE International Conference on Computer Languages(ICCL’98), Chicago, IL, May, 1998.
8) J. Daemen, and V. Rijmen, “AES Proposal: Rijndael,” AES Algorithm Submission, http://csrc.nist.gov/CryptoToolkit/aes/rijndael/, Feb. 2001.
9) 石原 淳, “DVD コンテンンツ保護,” 東芝レビュー, Vol.58, No.6, 2003.
10) 石間 宏之, 斉藤 和雄, 亀井 光久, 申 吉浩, “ソフトウェアの耐タンパー化技術,” 富士ゼロックステクニカルレポート, No.13, pp.20-28, 2000.
11) 鴨志田 昭輝, 松本 勉, 井上 信吾, “耐タンパーソフトウェアの構成手法に関する考察,” 信学技報, ISEC97-59, pp.69-78, 1997.
12) 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.
13) 片方 善治 (監修), “2 章 暗号技術”, IT セキュリティソリューション大系 下巻, (株) フジ・テクノシステム, 東京, 2004.
14) Kracker’s & BEAMS, クラッカー・プログラム大全, (株) データハウス, 東京, 2004.
15) 松岡賢, 赤井健一郎, 松本勉, 竹脇和也, “鍵内蔵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.
16) 松本 勉, 赤井 健一郎, 中川 豪一, 大内 功, 竹脇 和也, 村瀬 一郎, “Java 対応ランタイムデータ捕捉ソフトウェア”, 情報処理学会論文誌,Vol.44 No.8 pp.1947-1954, Aug. 2003.
17) M. Manbo, T. Murayama, and E. Okamoto, “A tentative approach to constructing tamper-resistant software,” In Proc. New Security Paradigm Work-shop, Cumbia, UK, 1997.
18) 門田 暁人, 高田 義弘, 鳥居 宏次, “プログラムの難読化法の提案,” 情報処理学会第 51 回全国大会講演論文集, 5G-7, pp.4-263–4-264, 1995.
19) 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,” 電子情報通信学科論文誌 D-I, Vol.J80-D-I, No.7, pp.644-652, July 1997.
20) 村山 隆徳, 満保 雅浩, 岡本 栄司, 植松 友彦, “ソフトウェアの難読化について,” 電子情報通信学会技術研究報告, ISEC95-25, Nov. 1995.
21) M. Nakamura, A. Monden, T. Itoh, and K. Matsumoto, Yuichiro Kanzaki, and Hirotsugu Satoh, “Queue-based cost evaluation of mental simulation process in program comprehension,” Proc. 9th IEEE International Software Metrics Symposium (METRICS2003), pp. 351-360, Sydney, Australia, Sep. 2003.
22) National Institute of Standards and Technology, “Announcing the Advanced Encryption Standard (AES),” FIPS PUB 197, Nov. 2001.
23) National Institute of Standards and Technology, “Data Encryption Standard (DES),” FIPS PUB 46-3, Oct. 1999.
24) T. Ogiso, Yusuke Sakabe, Masakazu Soshi, and Atsuko Miyaji, “Software obfuscation on a theoretical basis and its implementation,” IEICE Trans., Fundamentals, vol. E86-A, No. 1(2003),pp. 176-186.
25) T. Ogiso, Y. Sakabe, M. Soshi, and A. Miyaji, “Software tamper resistance based on the difficulty of interprocedural analysis,” In Proc. International Workshop on Information Secturity Applications(WISA2002), pp. 437-452, August 2002.
26) A. Patrizio, Why the DVD Hack Was a Cinch, Wired News, http://www.wired.com/news/technology/0,1282,32263,00.html, Nov. 1999.
27) Haruaki TAMADA, “AddTracer: Injecting Tracers into Java Class Files for Dynamic Analysis,” http://se.aist-nara.ac.jp/addtracer/, Dec, 2004
28) 玉田 春昭, 門田 暁人, 中村 匡秀, 松本 健一, “Java プログラムの動的解析のためのトレーサ埋め込みツール”, 第 46 回プログラミング・シンポジウム報告集, pp. 51-62, Jan 2005.

29) P. M. Tyma, “Method for renaming identifies of a computer program,” United States Patent, No. 6,102,966, Assignee: PreEmptive Solutions, Inc., Aug. 2000.
30) 山内 寛己, 神崎 雄一郎, 門田 暁人, 中村 匡秀, 松本 健一, “マルチバージョン生成によるプログラムの解析防止,” ソフトウェア工学の基礎 XI, レクチャーノート/ソフトウェア学 30, pp.157-160, 近代科学社, 2004.
31) やねう解析チーム, 解析魔法少女美咲ちゃんマジカル・オープン！, (株) 秀和システム, 東京, 2004.
32) C. Wang, J. Hill, J. Knight, and J. Davidson, “Software tamper resistance: Obfuscationg static analysis of programs,” ‘Technical Report SC-2000-12, Department of Computer Science, University of Virginia, Dec. 2000.

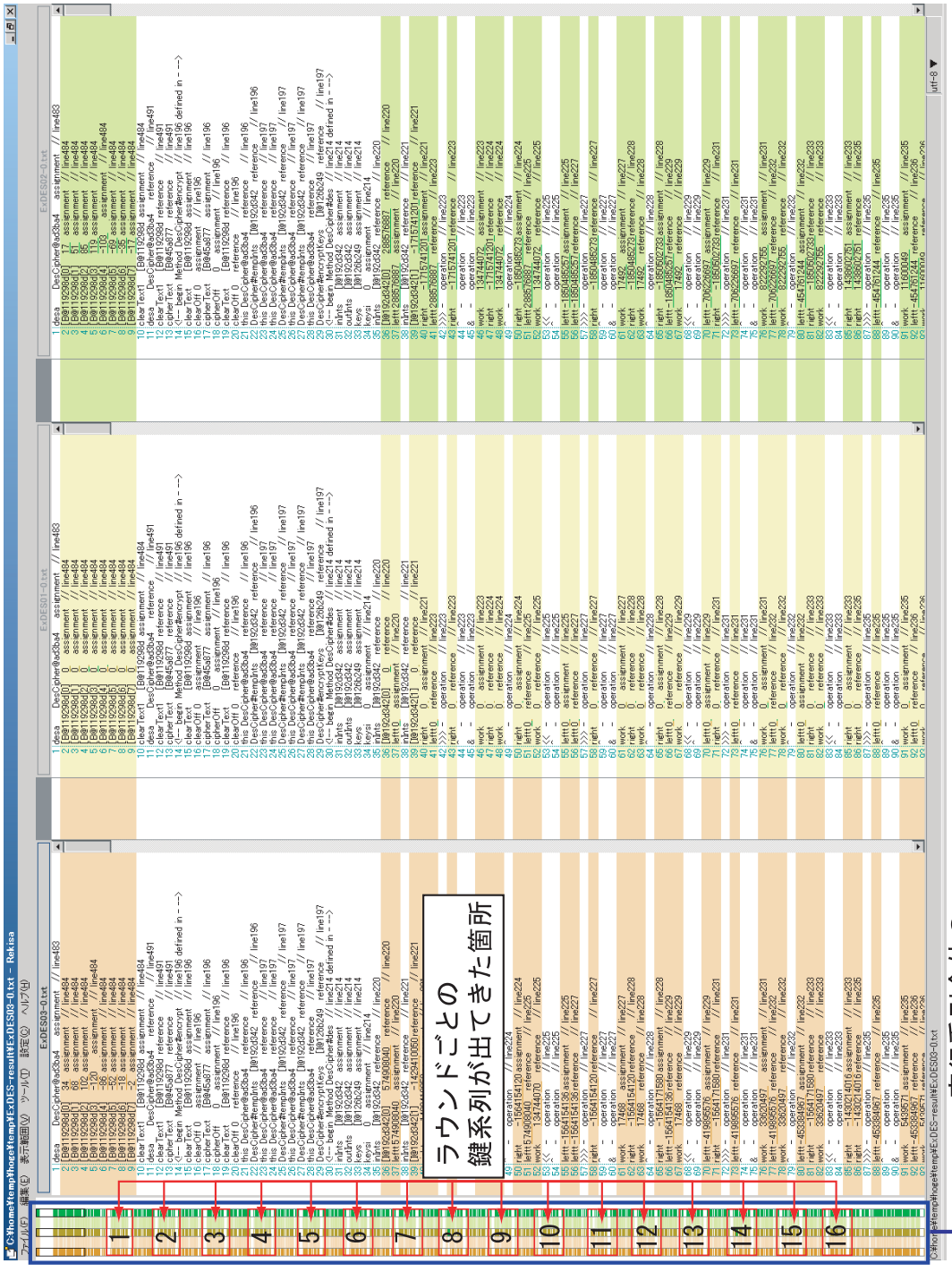


図 10　DES の実行系列差分 (ラウンド部絞込み後)

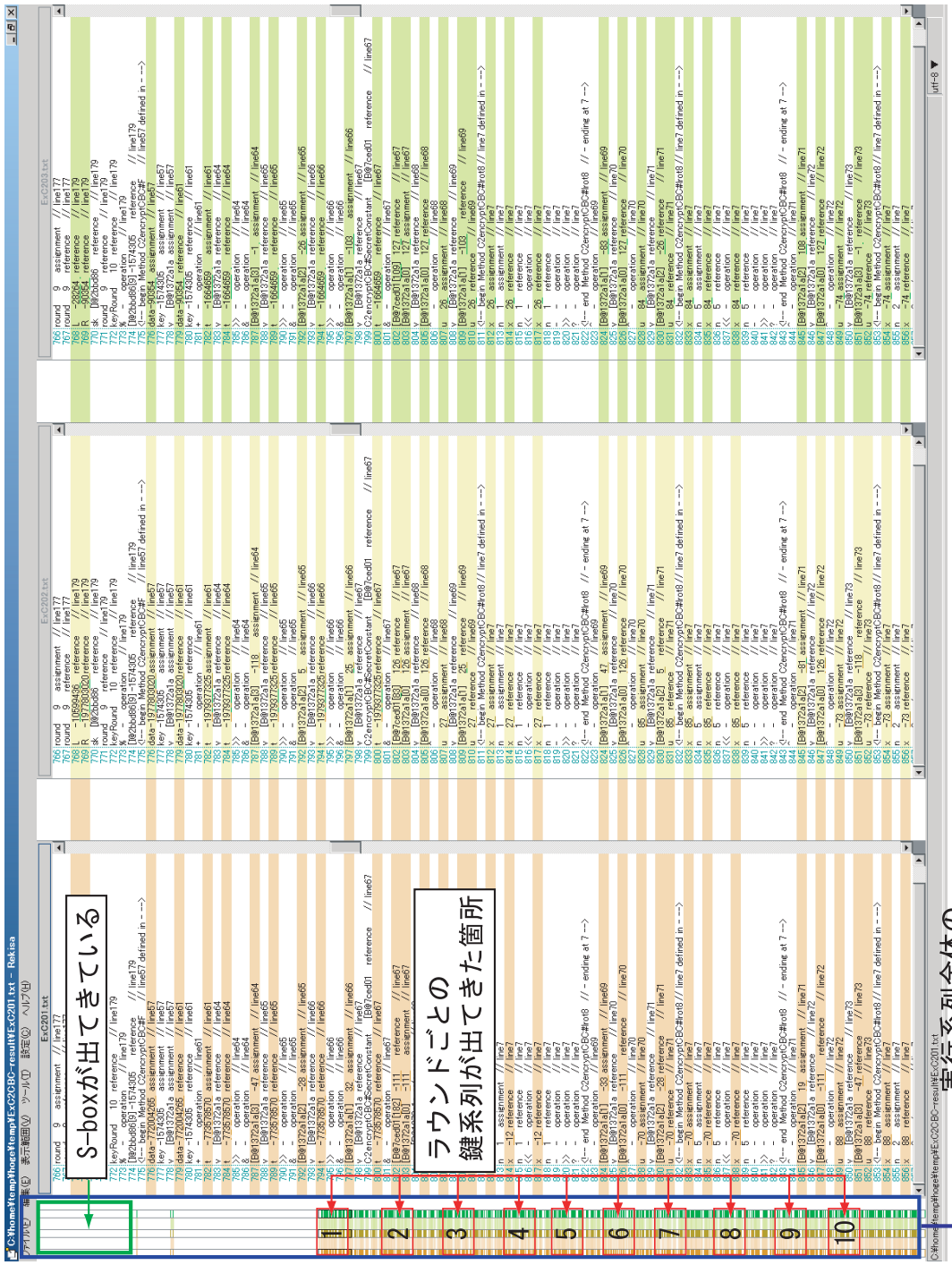


図 11　C2 暗号の実行系列差分 (全体)

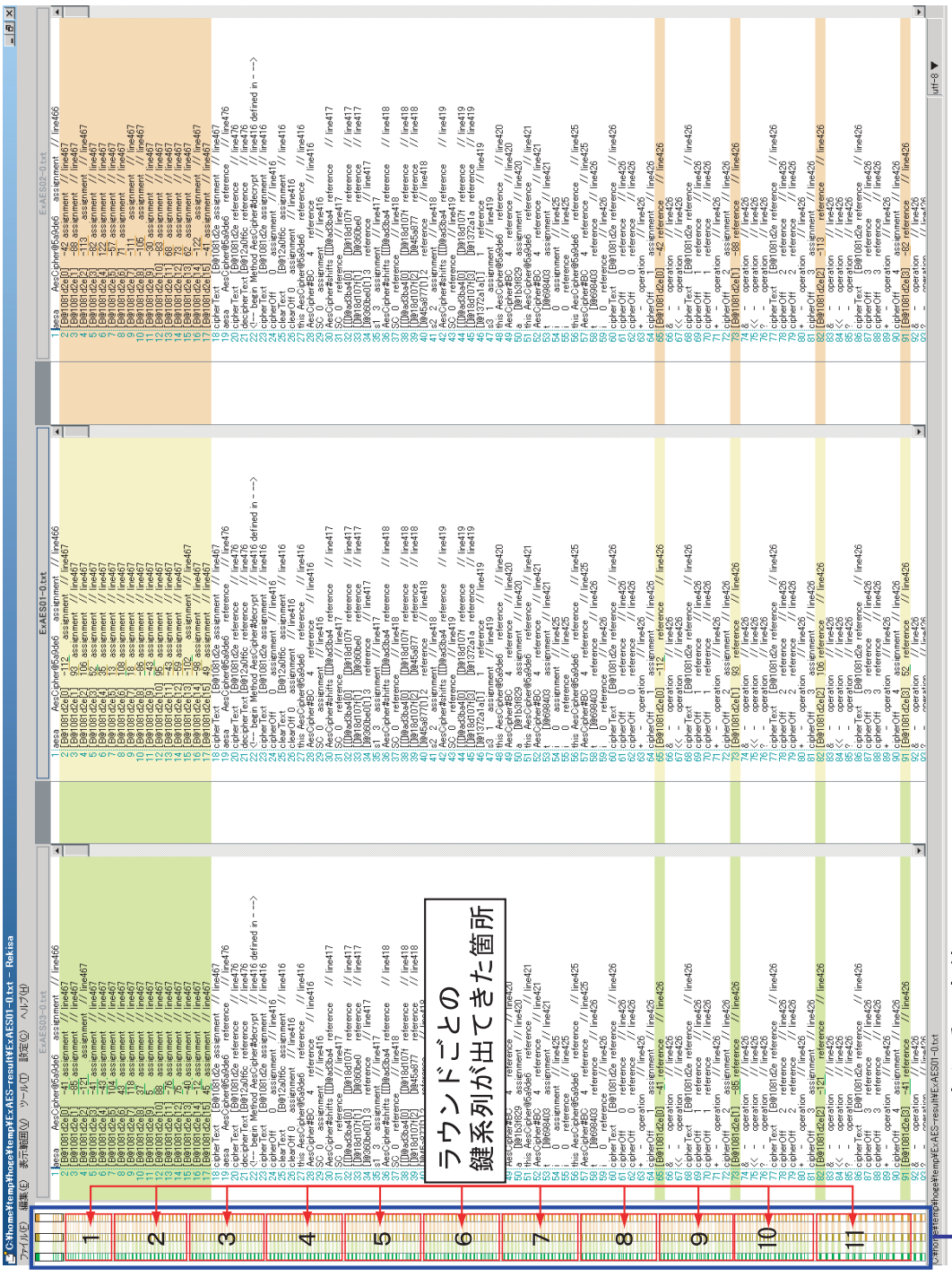


図 12　AES の実行系列差分 (データ範囲を限定後)