RC-007

# プログラム保護を行なうプロセッサの保護能力評価と改良

# Protection Ability Evaluation and Improvement of Processor with Program Protection Feature

松田 稔彦 *1　山下 純一 *1 *2　北村 俊明 *1
Toshihiko Matsuda　Juniti Yamashita　Toshiaki Kitamura

我々はプログラムを暗号化し，プロセッサ内でのみ復号するセキュアプロセッサを提案している．しかし，本セキュアプロセッサは，プログラムのみを暗号化対象としており，コンテキスト等のデータを暗号化していない．そのため，本研究では，暗号化されていないデータの実行前後の変化からプログラムの同定が可能であるか，本セキュアプロセッサの保護能力評価を行なった．その結果，プログラムの同定が可能であることが判明したため，改良案を提案し，その有効性を評価した．本改良案は，外部割込みによるプログラムの逐次実行処理を防ぐために，割込みチェックの間隔を広げるという考えを基礎としている．評価の結果，本提案の改良により，命令同定難易度が上昇し，この改良案の有効性を確認できた．これは，複数命令実行で積算された実行結果のみを観測しているため，データ変化の区別が困難であることが原因であると考えられる．

We have been proposing the secure processor which decrypts the encrypted program on only the instruction cache. This Secure Processer encrypted only the program, data, such as the contexts was not encrypted. In this research, the protection ability of this secure processor was evaluated. The key idea to break protection is observing the changes of the data between before and after an instruction execution. As a result, it is possible that the instructions within the encrypted programs are identified. So, we proposed an improvement of protection and evaluated it. The idea to prevent the individual execution by the external interrupt, which can be easily achieved by enlarging the interrupt check interval. By this improvement, the instruction identification becomes difficult, and we can confirm the effectiveness of this improvement. The reason is that we cannot separate the effect of each instruction from the sum-upped changes within the sequence of instruction execution.

## 1　はじめに

近年，様々な製品に組込みプロセッサが利用され，プログラムによる制御が実現されている．それらは専用 LSI 等を用いたハードウェアによる直接的な機能と異なり，機能の変更や修正，追加が容易である．このように，プログラムによる制御は多くのメリットに繋がる．制御に用いられるプログラムには，製品の仕様から簡単に導き出せる論理・手順だけでなく，より良い制御を行なうためのノウハウが含まれている．このようなノウハウは特許で保護することが難しく，公表すること自体が権利の消滅につながる．しかし，製品が市場に出回ることで，悪意のあるリバースエンジニアリングによってプログラムが不正利用される危険性が生じる．そこで，我々はプログラムを暗号化し，プロセッサチップ内でのみ復号を行なうセキュアプロセッサを提案している [1].

本セキュアプロセッサは，プログラムの保護に重点を置いて設計が行なわれている．そのため，プログラムのみが暗号化の対象であり，コンテキスト等のデータは暗号化されていない．これまでの設計及び評価では，保護するプログラムのアルゴリズムが複雑であれば，データ等からのプログラム同定は困難だと考えていたが，実際に調査はしていなかった．本研究では，データ変化の解析を用いた命令同定アルゴリズムを考案し，実際に解析することで，本セキュアプロセッサの保護能力を評価した．そして，1 命令実行前後のデータ変化を解析することで，プログラムの同定が可能であることが判明した [2]．そこで，本セキュアプロセッサでプログラムを保護するための改良案を提案する．そして，その改良案の有効性を解析を試みることで評価した．

*1 広島市立大学大学院 情報科学研究科
*2 現在，ローム株式会社

## 2　研究の背景と目的

### 2.1　セキュアプロセッサ

プログラム保護の手段として，プログラム難読化による耐タンパー性の向上が考えられる [3][4]．耐タンパー性とは，第三者からの不正利用に対する耐性のことである．また，プログラムを暗号化することで逆アセンブルを不可能にする手法も考えられる．しかし，実行時にソフトウェアを用いた復号では，デバッガによる動的解析によって，復号ルーチンを解析される危険性がある．

我々が提案しているセキュアプロセッサとは，プログラムコードを暗号化し，復号処理をプロセッサチップ内部のみで行なうことによって，平文のプログラムコードがチップ外部に流出するのを防ぐプロセッサである．暗号方式には，共通鍵暗号を用いてプログラム本体を暗号化し，その暗号化に用いた共通鍵 (プログラム鍵) を公開鍵暗号で暗号化してプロセッサに渡すハイブリッド方式を採用している．

#### 2.1.1　本セキュアプロセッサの特徴

セキュアプロセッサ上で実行されるプログラムは，それぞれ異なるプログラム鍵によって暗号化されている．そして，復号時はプログラムに対応したプログラム鍵を用意しなければならない．通常，プロセスごとに ID を設定し，その ID とプログラム鍵を対応付ける方法が多く用いられている．しかし，この方法では共有ライブラリや OS プログラムの一部といった，複数のプログラムを同一プロセスで内包する場合，それぞれのプログラムに対応したプログラム鍵を対応付けることができない．そこで，本研究室による提案プロセッサではプログラム鍵の割付け単位として，仮想記憶システムを用いたページ単位を採用した．これにより，1 つのプロセスに内包される各ページにそれぞれ異なるプログラム鍵を対応付けることを可能である．したがって，共有ライブラリの

暗号化も行なうことができる．

本プロセッサではプログラムを保護することに重点を置き，実行プログラム本体のみを暗号化の対象としている．2.3 節で紹介する関連研究では，コンテキスト等のデータを暗号化することも検討されている．しかし，我々の提案プロセッサでは，データの暗号化がセキュリティホールに成り得ると考え，敢えてデータの暗号化を行なっていない．

### 2.2 ARM アーキテクチャ

本研究で評価を行なうセキュアプロセッサは，ARM アーキテクチャ [5] を採用して検討を進めている．ARM アーキテクチャは，組込み機器に広く使用されている 32 ビットの RISC プロセッサのアーキテクチャである．ユーザレベルのプログラムでは，15 本の 32 ビット長の汎用レジスタ（r0〜r14），32 ビット長のプログラムカウンタ（PC），条件コードフラグを含むカレント・プログラム・ステータス・レジスタ (CPSR) が使用される．条件コードフラグは CPSR の最上位 4 ビットで示される．

ARM アーキテクチャの特徴に，全命令に対する条件付き実行が可能という点が挙げられる．条件フィールドは 32 ビット命令フィールドの最上位 4 ビットで表現され，15 種類の条件指定が可能である．条件コードフラグの状態が条件を満たす場合に限り命令が実行される．条件を満たしていなければ命令は実行されずにスキップされ，NOP 命令と同等となる．

### 2.3 関連研究

プログラムだけでなくデータの暗号化も検討されており，Lie らの提案する XOM(eXecute-Only Memory) では，データを主記憶に格納する際に暗号化を行ない，データのハッシュ値とともに格納する [6]．ハッシュ値によって，データの改竄を検出する機能も備えている．

また，割込み発生時に主記憶に退避されるコンテキストも改竄を防ぐために暗号化されるが，橋本らの提案する L-MSP（License-controlling Multi-vendor Secure Processor）では，XOM がレジスタ個別に暗号化を行なっているのに対し，コンテキストを主記憶にまとめて暗号化することで，コンテキスト退避時の遅延を緩和している [7]．

XOM や L-MSP のように，データを暗号化する場合，主記憶からデータを取り出す際に復号を行ない，処理後に暗号化して主記憶に書き戻す経路が必要になる．それを悪用し，プログラムをデータとして読み込むことができれば，プログラムを容易に解析できる危険性がある．プログラムのみを暗号化する場合，プログラム自体を主記憶に書き戻す経路は存在せず，前述の危険性とそれに対する処理は必要ない．そのため，本研究ではデータの暗号化を行なわずに保護できるかという命題を考慮している．

## 3 命令同定アルゴリズムの前提条件

暗号化されていないデータ等の変化からプログラムを同定した命令同定アルゴリズムの前提条件について説明する．命令同定アルゴリズムの前提条件は，セキュアプロセッサにとって非常に不利な環境を想定している．その理由は，攻撃者にとって有利な環境でも充分にプログラムを保護できれば，保護能力は充分だと示せると考えたからである．命令同定アルゴリズムの前提条件を以下に示す．

- **プロセッサ以外の周辺回路の構成が自由**
  セキュアプロセッサ以外の周辺回路は自由に構成でき，メモリバス等のプロセッサ外部に存在するバスも観測できる．
- **OS の特権機能を使用可能**
  攻撃者の思い通りに動作する OS を用意して解析するため，攻撃者は OS 特権機能を自由に利用できる．
- **主記憶装置の閲覧・改竄が可能**
  主記憶の値は任意の時点で自由に閲覧及び改竄できる．本研究では，ワード転送のロード命令を実行する際に，レジスタにロードアドレスが格納されるように初期化した．
- **解析対象プログラムの逐次実行処理**
  外部割込みを用いて，解析対象プログラムを 1 命令処理するごとに中断し，OS に制御を戻す．
- **データキャッシュの無効化**
  解析対象プログラムと無関係なデータをロードすることで，データキャッシュに存在するデータを全て入れ替える．そして，キャッシュミスやデータの書き戻しを発生させ，主記憶にアクセスさせる．

これらの条件が成り立つ環境において，1 命令実行前後でデータ等の変化を解析することで，プログラムの同定が可能であるということが判明した．

## 4 改良案

本セキュアプロセッサのプログラム保護能力を改善するために，どのような改良が必要か考える．本研究では，命令同定アルゴリズムの要である前提条件を成立させないことで解析難易度が上昇すると考え「解析対象プログラムの逐次実行処理」をハードウェアとして制限することを考えた．命令同定アルゴリズムは，1 命令の実行前後で外部割込みを発生させ，他の前提条件で構成される環境において，1 命令実行前後のデータを取得・改竄して解析する．プロセッサの割込みチェック間隔が 1 命令間隔なので，外部割り込みが 1 命令ごとに発生し，プログラムは逐次実行する．それならば，割込みチェック間隔を 1 命令間隔でなく，N 命令間隔 (以降，N は 1 より大きい整数) に変更すると，N 命令ごとでしかデータの観測ができなくなる．それにより，実行前後のデータから推測できる命令の組み合わせは激増し，解析は困難になると考えられる．N は 10 命令以下と考えているので，割込みチェック間隔が広がる事による割込みの応答速度の低下はほとんど無視できるレベルである．また，この案に基づいた改良は，プロセッサへの改良が容易に実装可能なので実現コストが低いことも大きな利点である．以上より，本研究では，解析対象プログラムの逐次実行処理をハードウェアとして制限するために，「割込みチェック間隔を広げる」という改良案を提案する．

この改良によって，命令同定アルゴリズムの前提条件

が変更される．「解析対象プログラムの逐次実行処理」が削除され，代わりに「割込みチェック間隔の把握」という前提条件を加える．この条件は，実際に外部割込みをペンディングして，命令を実行し，その前後での PC を比較すれば成立する．その際に分岐命令が実行された場合，PC が大きく変動するが，何度か異なる命令アドレスから開始して検証すれば成立すると考えた．

## 5 命令同定アルゴリズム

### 5.1 解析対象命令

解析対象となる ARM 命令は，プログラム内においてユーザモードで使用される主な命令を想定した．それを以下に記す．

- 分岐命令
- 乗算命令 (32 ビット，64 ビット)
- データ処理 (数値演算)
- データ転送 (シングル/マルチ，ロード/ストア)

ARM アーキテクチャのデータ転送命令は，単一レジスタに対して転送を実行するシングル系転送命令に加えて，一括で複数のレジスタに対して転送を実行するマルチ系転送命令が存在する．データ処理命令は，数値演算命令と数値比較命令の 2 種類に分けられる．数値演算命令は，算術や論理演算の命令であることを指す．数値比較命令は，結果をレジスタに反映させずに条件コードフラグのみを更新する命令であることを指す．数値比較命令は，条件コードフラグにしか変化を観測できないために，同定が困難である．本研究では有効な解析方法を導出できなかったので，数値比較命令は解析対象外命令とする．

### 5.2 解析フロー

本節では，解析対象プログラムに対しての解析フローを記す．同定アルゴリズム内でレジスタやフラグを参照・変更する動作が含まれているが，これは主記憶に退避されたコンテキストに含まれるデータに対して行なうものであり，プロセッサ内部に存在するレジスタに直接操作を行なう意味ではない．以降，単に「レジスタを変更する」といった表現を使用する．

命令同定を行なう解析処理全体のフローチャートを図 1 に示す．図 1 に記した論理を進めていき，論理に応じてコンテキストを変更して，命令を実行，新たな情報を得て論理を進めることで最終的な同定まで行なう．

### 5.3 差分解析

改良案によって複数命令を同時に解析することになるので，その改良案に対抗するためにデータの差分解析を行なう．同定アルゴリズムの解析フローでは，ある命令アドレスに対して解析が終了すると解析対象アドレスを 1 命令分進める．この操作によって，解析ごとの変化の差分から解析全体の効率を向上させる．例として図 2 を用いて説明する．図 2 は，割込みチェック間隔を 3 命令間隔として，図中の命令列を解析する様子を示している．解析 1 では命令 1，2，3 に対して解析を実行し，解析 2 では命令 2，3，4 に対して解析している．そのため，解析 2 は解析 1 で観測していた命令 1 の変化を観測

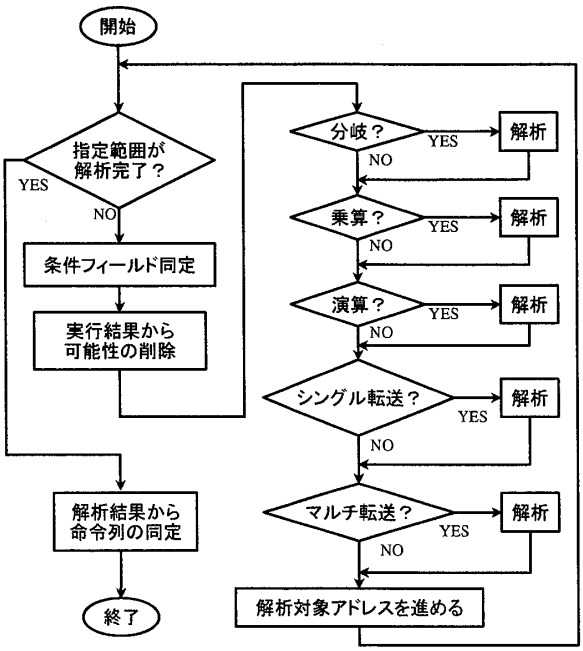


図 1 フローチャート

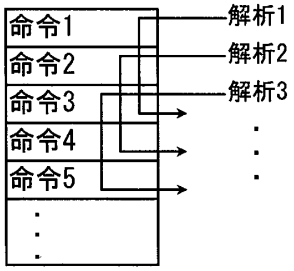


図 2 差分解析例

せず，新たに命令 4 の変化を観測している．その結果，解析できなくなった変化を起こす命令が命令 1 であり，新たな変化を起こす命令が命令 4 である可能性が高いと判断できる．

また，命令 3 に対して，解析 1，2，3 が順に解析を実行する．解析時の解析対象命令は 3 命令中の 1 つ目であり，解析 3 では命令 3 を同定したい．その際，解析したい対象アドレスの 2 つ前の対象アドレスからの実行変化，つまり解析 1，2 の実行後の変化も解析に利用することができる．このように複数の命令が起こす変化の差分を観測し，利用することで解析全体の効率が向上する．

### 5.4 条件フィールドの同定

実行後の変化を観測しやすくするために，条件フィールドの同定開始前に演算に使用されるレジスタ（PC 以外）を乱数で書き換えて実行する．解析対象プログラムが逐次実行処理できる場合，実行前に条件コードフラグを変更し，命令が実行されるかを作成した論理木に従って解析した．それによって，最小で 3 回から最大で 6 回の割込み解析で条件フィールドを同定できた．

しかし，改良後は 1 度の割込み解析で，1 命令でなく N 命令実行されるため，N 命令中で条件の組み合わせは様々である．そこで，条件コードフラグ 4 つの組み合わせで考えうる 16 通りの組み合わせにフラグを変更し，各組み合わせで実行して解析した．16 通りの組み合わ

せの内，各組み合わせのデータ変化の違いを解析することで複数の条件フィールドを同定した．条件が同定されると以降の命令個別解析において，条件つき実行の命令が確実に実行できることになる．

### 5.5 命令種類別の可能性削除

条件フィールド同定時における実行前後のデータ変化を基に，その変化から候補命令の可能性を種類別に削除する．命令が発生させる変化には様々なパターンが存在するが，当然それはアーキテクチャで決定され，実行中には変動しない．以降の解析から候補命令を削減することで解析の効率が向上する．本研究では，ARM アーキテクチャを採用しているのでその仕様に準拠して命令の可能性を削除する．これは命令の種類ごとに限定されるので，割込みチェック間隔が 1 命令ならば，1 回の解析で可能性を絞ることができた．

しかし，割込みチェック間隔が N 命令間隔となると，複数命令の変化から各命令の可能性を絞ることになり，解析が困難になる．そのため，5.3 節で説明した差分解析を用いて，候補を可能な限り絞る．

### 5.6 命令の種類別解析

可能性の残っている命令を全て命令の種類別に解析し，命令アドレスと命令の種類，解析結果を保存する．以下で命令の種類別に解析方法を説明する．

#### 5.6.1 分岐命令

ARM 命令セットの分岐命令には，通常の分岐命令とリンク付き分岐命令の 2 種類が存在する．分岐命令が実行されると，目的アドレスに PC(r15) が変更されて分岐が発生する．リンク付き分岐の場合には，分岐命令の次の命令アドレスが r14 へ格納される．ちなみに条件付き分岐であったとしても，条件フィールドの同定によって，必ず分岐するようにフラグが変更されているので，解析方法は分岐命令で全て共通である．以下に解析方法を簡単に示す．

1. 汎用レジスタ（PC 以外）を乱数で書き換えて実行
2. 実行後に PC が同じ分岐先になる事を確認，PC が異なれば分岐命令ではないとして終了
3. 実行後の PC になる分岐命令であると同定
4. 実行後の r14 が分岐前の PC+4 の値ならば，リンク付き分岐であると同定

#### 5.6.2 データ転送命令マルチ系

ARM のデータ転送命令マルチ系では，1 命令で一括して複数のレジスタで転送を実行できる．同定したい命令がデータ転送命令マルチ系の可能性がある場合，

- マルチ系転送ではベースレジスタは転送レジスタ数に依存して増減し，後述のシングル系転送のようなオフセット増減は行われない点
- 1 度に転送されるレジスタの最大数は 16 であるため，ベースレジスタで示されるアドレスの上下 0x40 付近までのアドレスにしかアクセスできない点

以上を利用して同定を行なう．

ロードマルチ命令の同定を行なう際は，汎用レジスタ（PC 以外）を一定間隔（上下 0x40）以上あけた値に変更して命令を実行する．その結果，ロードされた値からベースレジスタと転送対象レジスタ全てが判明する．ベースレジスタが更新されていても，汎用レジスタを一定間隔以上あけた値に変更しているので，ベースレジスタは特定できる．以上でロードマルチ命令が同定可能である．

ストアマルチ命令を同定する場合，同定手順はロードマルチ命令とほぼ等しい．

#### 5.6.3 データ転送命令シングル系

ARM 命令セットのデータ転送命令シングル系として本研究では，ワード転送，符号なしバイト転送，ダブルワード転送，ハーフワード転送を解析対象とした．

ARM 命令には，シフト命令が存在せず，オフセット指定時やオペランド指定時にシフト演算したものを指定可能である．それによって，エンコーディング方式のパターンが多いので，全数探査を用いて同定を行なう．全数探査では，全ての可能性について検証を行なう．エンコーディング方式，オペランドレジスタ等から考えうる組み合わせを全て検討し，実行後と同じ結果になる命令のエンコーディングを求める．

ストア命令では，実行後に主に変化が生じるのは主記憶である．主記憶では同じアドレスにストアしない限り，変化は上書きされないので変化の解析が容易である．これにより，候補を絞ることができ，全数探査の効率が改善される．

#### 5.6.4 データ処理命令

ARM 命令セットのデータ処理命令は，全部で 16 命令存在し，12 命令がデスティネーションレジスタ（Rd）に結果を反映させる数値演算命令，4 命令がレジスタに結果を反映させず，条件コードフラグの更新のみ行なう数値比較命令となっている．本研究では，数値演算命令の 12 命令のみを解析対象命令としている．

ARM 命令セットでは，データ転送命令シングル系で説明したように，シフト演算に関連してオペランドの指定方式が多様である．そのため，命令を実現する形式のパターンが非常に多いので，全数探査を用いて同定を行なう．全数探査での効率向上のため，同等な結果を出力する命令の組合せについて，事前にその組合せを全数探査の候補から削除しておく．例えば，$Rd = r1 + r2$ と $Rd = r2 + r1$ は同一の結果となる．よって，片方を候補からあらかじめ除外しておく．

#### 5.6.5 乗算命令

ARM 命令セットで対象となっている乗算は，結果を 32 ビットで出力する短乗算と 64 ビットで出力する長乗算で合わせて 6 種類である．乗算命令を同定する上で，

- 演算ソースが全て-2 だった場合，6 種類の乗算命令で出力される結果が全て異なる点
- 演算ソースが全て異なる素数だった場合，結果を素因数分解すると利用されたソースが判明する点

以上の 2 点が重要な項目である．

乗算命令の同定を行なう際の手順を簡単に示す．まず，全てのレジスタ値を-2 に変更して実行し，命令の種類と Rd を判明させる．そして，全てのレジスタ値を異なる素数に変更して実行し，演算ソースが判明させ，乗算命令を同定する．命令の種類が積和算の場合は，演算

ソースレジスタ値と Rd の値の関係から，全ての組合せを考える全数探査を行なう．すべての演算ソースの組合せにおいて結果が異なる値を使用することで，全数探査も一回の命令実行で可能である．

### 5.7 解析結果からの命令同定

指定範囲まで解析対象アドレスが進み解析が完了すると，解析時に保存しておいた解析結果等から，候補命令を取捨選択する．解析対象プログラムの逐次実行処理が可能ならば，命令種類別の可能性削除で候補を絞っているので，この処理は必要ない．しかし，改良案によって，複数命令の実行結果しか観測できない場合，あり得る命令の可能性に対して全て解析を行なうので，解析結果からさらに候補を絞る必要がある．

例として，割込みチェック間隔が 1 命令より大きく，データ転送命令マルチ系が解析結果に含まれている場合，同じ対象に対してデータ転送命令シングル系をいくつも組み合わせて同等とする結果になる．この場合，データ転送シングル系の命令を削除する．このように解析結果から更に候補を絞り，最後に残っていた命令を最終的な同定結果とする．

## 6 評価

改良した命令同定アルゴリズムに基づいて，本セキュアプロセッサのソフトシミュレータを内包する検証システムを PCLinux 上に構築し，改良案の有効性を評価した．

### 6.1 評価方法

ソフトシミュレータで対象プログラムを実行することで，対象プログラム解析に必要な情報を得て解析を行なう．必要な情報は，命令実行前後のコンテキスト (レジスタ，条件コードフラグ)，主記憶へのアクセス内容である．

対象プログラムには Stanford ベンチマークの Bubble，Intmm の一部を採用した．それらの特徴として，Bubble は再帰的な処理をするプログラムである点，Intmm は数値演算処理をするプログラムである点が挙げられる．解析対象プログラムにおいて，Bubble は 43 命令中 37 命令，Intmm は 27 命令中 25 命令が解析対象命令である．割込みチェック間隔は，1，3，5，10 命令間隔として，それぞれの間隔で同定結果の正解命令数，割込み回数（割込み解析の実行回数），検証システムにおけるシミュレート時間を評価した．

本来のクラックシステムは，セキュアプロセッサ上で動作することになるが，本研究の検証システムでは，PCLinux 上にシミュレータを作成して検証する．そのため，本来のクラックシステムを用いた同定までの解析処理時間を得る事が出来ない．実際にセキュアプロセッサ上のプログラムを解析する場合，メモリの初期化など処理時間を多大に要する処理が何度も必要となり，シミュレート時間と実際の解析処理時間は大きく異なる．しかし，このシステムは割込み解析を何度もループ処理することでプログラムを解析している．割込み解析の実行回数が解析処理のコストと大きく関係しているので，割込み回数を評価項目としている．シミュレート時間も命令の同定難易度の変化を考える目安になると考え，評価項目としている．検証システムを動作させた PCLinux のマシン環境は，OS が「CentOS 5.3」，CPU が「Intel®Pentium4 3.00GHz」，メモリが「2GB」である．

### 6.2 評価結果

図 3 と図 4 に各割込みチェック間隔での正解命令数，シミュレータによる割込み回数と解析時間を示す．割込みチェック間隔が 1 命令間隔である場合，解析対象命令は全て同定可能で，他の割込みチェック間隔と比べると解析時間は短く，割込み解析の実行回数も少ない．しかし，セキュアプロセッサの割込みチェック間隔が広くなることで，プログラムの逐次実行処理が不可能になり，1 命令ごとの同定に必要なデータ観測は困難になる．その結果，割込み回数と解析時間が増加しただけでなく，プログラムの同定結果の正解命令数が減少した．

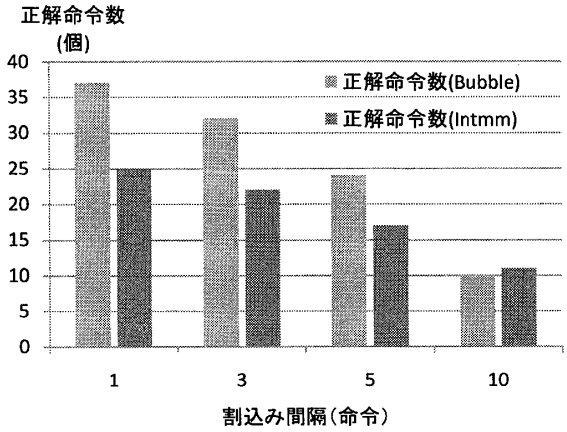


図 3 正解命令数

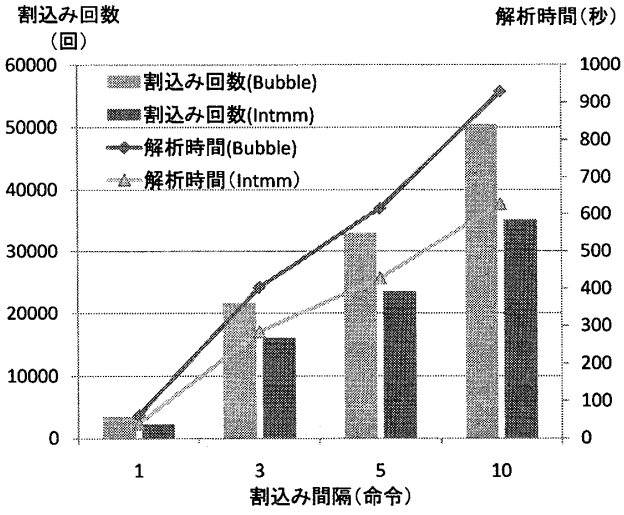


図 4 割込み回数と解析時間

割込みチェック間隔が 1，3 命令間隔のように狭い場合ならば，同定結果の正解命令数も多く，時間をかければプログラムを完全に同定できる可能性もある．しかし，割込み間隔が 10 命令のように広い場合，実行後の変化が異なる命令で更新され，正しく観測できない場合が非常に多く，正解命令数が減少する．この場合，時間をかけても完全に同定するのは困難である．以上の結果

より，改良によって解析難易度が上昇しているので，プログラム保護能力を改善できていると判断できる．

本研究の評価は定量的な評価ではない．そのため，本研究のみで本セキュアプロセッサのプログラム保護能力が充分に改善できたか確認できない．しかし，割込みチェック間隔が広くなることで，命令の同定に必要なデータの観測は困難になり，命令の同定難易度が上昇した．よって，この改良案の有効性を確認できた．

#### 6.2.1 解析が困難な命令

図 3 より，割込みチェック間隔が 10 命令間隔ならば Intmm の方が正解命令数が多いことから対象プログラム間で同定難易度に差があることが分かる．Bubble の方が対象命令数が多いにもかかわらず，この結果になった理由には，Bubble の特徴である再帰的な処理が多いこと．つまり，分岐命令が関連している．実際に命令ごとの解析結果では，割込みチェック間隔が広くなることで，分岐命令や分岐前後の命令の同定が困難になっている．それらの命令が解析困難になった理由について説明する．

- データ比較系命令が解析対象外
  割込みチェック間隔が広くなり，割込み解析内でデータ比較系命令と同時に実行された場合，正しい条件が同定できず，条件付き分岐の同定が困難になる．
- 分岐先が観測できない
  図 5 は，分岐先が解析困難な場合の例である．図中の解析可能範囲が対象となっているリンク付き分岐のデータ変化の差分解析が可能なアドレスの範囲である．この例で割込みチェック間隔を 10 命令間隔として，解析可能範囲から解析を始めると，10 命令後にはサブルーチンからリターンしており，分岐先が観測できない．

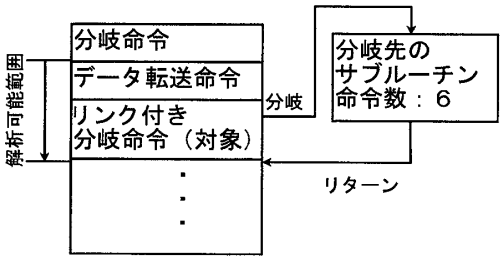


図 5 分岐先が解析困難な例

- 分岐前後の命令の区別が困難
  解析時に各命令を逐次実行できず，複数命令実行によって積算されたデータ変化しか観測できないため，分岐先の命令と分岐前の命令による変化の区別が困難であることが原因である．

#### 6.2.2 解析が容易な命令

同定があまり困難にならなかった命令には，実行後の変化が正しく観測できる命令というのは当然だが，主にデータ転送命令が多く含まれる．これはデータ転送命令がコンテキスト変化だけでなく，主記憶へのアクセス情報も同定に利用する解析方法だからである．通常，同じメモリアドレスに連続でアクセスすることは稀である．このため変化が正確に観測でき，同定が容易だったと考えられる．

## 7 おわりに

我々が提案しているセキュアプロセッサのプログラム保護能力が充分ではなかった．そのため，プログラム保護能力を改善することを目的に，「割込みチェック間隔を広げる」という改良案を提案した．この改良案によって，解析対象プログラムの逐次実行が不可能となり，解析に必要なデータの観測が困難になると考えた．改良案の有効性を評価するために，データ変化の解析を用いた命令同定アルゴリズムによる検証システムを作成した．そして，検証システムで解析することによって改良案の有効性を評価した．結果として，本改良により，命令の同定難易度は上昇し，改良案の有効性を確認できた．

本研究の命令同定アルゴリズムでは，「割込みチェック間隔の把握」が重要な前提条件となっている．そのため，割込みチェック間隔の把握が不可能になると，プログラム同定のコストが大きく増大する．具体的な方法としては，割込みチェック間隔をランダム値生成器を用いることで，実行中にランダムに変更する方法が考えられる．この方法を適用した場合，本研究で同定が困難だった分岐命令，分岐前後の命令の解析が更に困難となる．そして，分岐命令はプログラムのフロー制御に直結しているので，プログラム全体の解析の難易度が上昇する．このように更なる改良案を検討することで命令の同定難易度は上昇し，プログラム保護能力の改善につながると考える．

## 参考文献

[1] 城本正尋ほか. 公開鍵暗号を用いてプログラムの保護を行うプロセッサの提案. 情報処理学会論文誌：コンピューティングシステム, Vol. 47, No. SIG 18(ACS 16), p.55-64, Nov. 2006.

[2] Junichi Yamasita; et al. Reverse Engineering on the Processor with Program Protection Feature. COOL Chips X, p.143, April 2007.

[3] 門田暁人ほか. 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, Vol. J80-D-1, No.7, p.1-11, July 1997.

[4] 石間宏之, 亀井光久, 齊藤和雄. ソフトウェアの耐タンパー化技術. IPSJ Magazine, Vol. 44, No. 6, pp. 622.627, June 2003.

[5] ARM Limited. ARM アーキテクチャリファレンスマニュアル, 第 ARM DDI 0100HJ-00 版, March 2005.

[6] David J. Lie. ARCHITECTURAL SUPPORT FOR COPY AND TAMPER-RESISTANT SOFTWARE. Doctoral dissertation of Stanford University, Dec. 2007.

[7] 橋本幹生ほか. 敵対的な OS からソフトウェアを保護するプロセッサアーキテクチャ. 情報処理学会論文誌,vol.45, No.SIG3(ACS 5), p.1-10, March 2004.