ハードウェアを利用したソフトウェアプロテクションのための実験システム

2003MT038 加藤 哲郎　2003MT083 大橋 一寛　2003MT104 鳥谷 祐司

指導教員 真野芳久

1 はじめに

近年、ソフトウェアの再利用性が高まる中、ソフトウェアの盗用は益々容易になってきている。そこで盗用を防ぐために研究されているソフトウェアプロテクションの方法がいくつか考え出されているが、ソフトウェア単体の技術だけでは完全に防ぐことは不可能だと証明されている[1]。

こうした中、ハードウェアを利用したソフトウェアプロテクションに注目が集まり[2]、ハードウェアを利用した暗号化手法の提案、それに伴うハードウェアの拡張提案と様々なものが発表されている[3][4][5][6]。

しかしこれらの手法は、実装し動作させてみなければ実際の評価はすることができないが、既存の CPU に実装することは困難である。そこでソフトウェアでシミュレートすることが必要になり、実行効率や暗号化の耐性など、さまざまな評価が行える実験システムが必要だと考えた。

そこで我々は提案されている手法の一つをとりあげ(2節参照)、それがハードウェア上で実現可能であり CPU の基本機能は既に提案されているとした上で、紙上検討、および模擬実行が行える実験システムを作成し、提案された手法を評価するシステムについて検討と評価を行う。

2 実現する暗号化手法

2.1 基本ブロックを単位とする暗号化

[6]の暗号化手法では、基本ブロックを暗号化する単位としている。分岐命令とその移動先のプログラムカウンタ(以下PC)の値の差を関数 F に与え、その値で暗号化の種類が決定する。基本ブロック毎に暗号化を行うが、一つの基本ブロックに複数の分岐命令先がある場合、関数 F の値が異なり、分岐先の暗号が定まらない状態になる。その問題を解決するために[6]では、配置変更やダミー命令の追加を行っている。

2.2 CPU で復号実行するために

暗号化手法[6]は、先に挙げた基本ブロックを利用し、分岐命令とその移動先 (基本ブロックの先頭) のPC値の差に基づいて鍵を決定する。決定した鍵で移動先の基本ブロックは暗号化され、復号を CPU で行う。本来CPUには、基本ブロックの判別をする機能はもっていないため、基本ブロックを認識する手法が必要となる。そのために[6]では、基本ブロックの移動時にはすべて無条件分岐を追加し、CPU で分岐の実行を認識可能にすることにより、基本ブロック単位で鍵の復号を可能にしている。

3 システム開発のための事前調査

3.1 パイプラインとパイプラインハザード

プログラムの命令実行をいくつかの過程に分け、複数の命令を過程ごとにずらして並列実行を行い、プログラム実行を高速化する技術がパイプラインである。

上記のパイプラインを利用し、プログラムを実行させていくと起こる問題がある。連続して命令を実行しているため、命令が確定する前に次の命令処理が行われ、変化する値を反映できなくなってしまう。その場合はプログラムが正確に動作しないので、正しく実行させるためにパイプラインの動作が遅れる状態が起こる。これをパイプラインハザードと呼ぶ。パイプラインハザードには、値の変更を伴う際に、その値を利用する命令がある場合に起こるデータハザードと、分岐命令が実行されるときに起こる制御ハザードがある。

3.2 想定する CPU

我々が扱う CPU は、5 段パイプラインモデルで実行されるとする。命令の動作はフェッチ (F) 、デコード (D) 、メモリアクセス (M) 、実行 (E) 、書き込み (W) の 5 つで、各動作の内容を**図 1** に記す。

<table>
<tr><td></td><td colspan="2">Load命令</td><td>Store命令</td><td colspan="2">演算命令</td><td>分岐命令</td></tr>
<tr><td></td><td>r1,x</td><td>r1,r2</td><td></td><td>r1,x</td><td>r1,r2</td><td></td></tr>
<tr><td>F</td><td colspan="6">命令 (W1,W2) →IR</td></tr>
<tr><td>D</td><td colspan="5">命令判別→処理内容→送信</td><td>命令判別 (分岐の有無)
→処理内容→送信</td></tr>
<tr><td rowspan="2">M</td><td>EA計算</td><td></td><td>EA計算</td><td>EA計算</td><td></td><td>EA計算</td></tr>
<tr><td>OF</td><td>読込</td><td>読込</td><td>OF, 読込</td><td>読込</td><td></td></tr>
<tr><td>E</td><td></td><td></td><td></td><td colspan="2">計算処理</td><td></td></tr>
<tr><td>W</td><td colspan="5">書き込み</td><td>PC←EA</td></tr>
</table>

読込：参照するレジスタやアドレスの値を読み込む
書き込み：レジスタやアドレスに値を書き込む

図 1. 命令実行における動作内容

3.3 鍵決定に関する検討

手法[6]の鍵の変更を CPU で行う場合のパイプライン動作について検討する。鍵の変更が行われるのは、分岐命

令が実行された場合に起きる。本研究の想定する CPU では、**図1**よりDで分岐命令の判別を行う。そこで分岐が確定した場合、現在の PC 値を鍵決定装置に渡す。M で分岐先の PC 値を鍵決定装置に渡す。この 2 つの値から鍵を決定する(**図2**)。

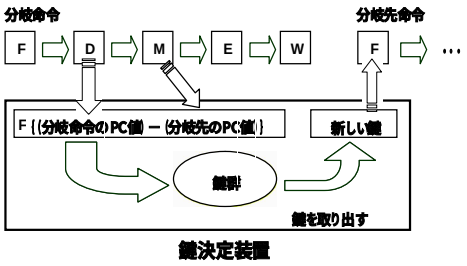

図 2. 鍵決定の流れ

4 実験システムの概要と設計

実験システムは、仮想計算機COMET Ⅱ[7]を対象に、トランスレータとCPUエミュレータの2つのモジュールからなる。トランスレータでソースプログラムをデータ分析・暗号化し、CPUエミュレータで実行する。

また、本研究では、CPUの動作を完全に制御する必要があるため、機械語と1:1に対応する記述をするアセンブリ言語で書かれたソースプログラムを対象とする。

4.1 トランスレータ

トランスレータではプログラム変換のための処理(アセンブル、基本ブロックの判別、配置)と機械語プログラムの暗号化を行い、秘密鍵情報を使用する。

ここでは[6]の配置アルゴリズムは使用せず、仮の方法で配置を行うことする。これは[6]のアルゴリズムが改善の余地があることと、我々の研究の趣旨がこのアルゴリズム部分ではなくシステム全体を評価することにあるためである。各機能を分け、配置アルゴリズム部分を独立させることにより、配置アルゴリズムの変更に対応させる。

1. アセンブル・基本ブロックの判別

ソースプログラムをアセンブルし、メモリイメージに変換する。アセンブルされたデータの基本ブロックを判別し、配置に用いるために基本ブロック情報を抽出する。抽出する基本ブロックのデータには、基本ブロックの番号とサイズ、分岐命令の情報(分岐命令の個数、分岐命令の位置、分岐命令の種類、遷移先のブロック番号)が含まれ、遷移を表すデータ構造となる。基本ブロックの判別方法は分岐命令で行い、基本ブロックの末尾は無条件分岐命令で終わるようにする。

2. 配置・ダミー命令の追加

基本ブロックのデータを入力とし、基本ブロックごとの暗号化に使用する鍵を決定するために、配置アルゴリズムを使い PC 差分を決定する。PC の差分を調整するのに必要ならばダミー命令を挿入する。鍵の関数 F により基本ブロック単位でどの鍵を使って暗号化するかを決定する。

3. 暗号化・出力

配置により決定された、ダミー命令追加済みの基本ブロックの順列を用いて、アドレスの決定を行う。アドレスの決定では配置前と配置後ではアドレスが異なるため、変更されたアドレスにそれぞれのオペランドを変更する。

配置時に決定された鍵情報によって各基本ブロック単位で暗号化し、メモリイメージを出力する。

4.2 CPU エミュレータ

トランスレータからのメモリイメージを入力し、鍵を利用して復号しつつプログラムを実行する。

4.2.1 復号機能

トランスレータからのメモリイメージを逐次復号し、実行する。また、鍵を生成するために PC 値の差分をとって鍵集合から復号鍵を決定する。分岐命令の実行で基本ブロックの移動が起こる場合には鍵が変わる。

4.2.2 パイプライン処理の設計

CPU 動作を正確にエミュレートするために、CPU のパイプラインのステージを数えるステージカウンタ(以下 SC)機能を実装する。

データハザードや制御ハザードが起こるとその分ステージ数が多くかかり遅れが生じる。以下は、データハザードと制御ハザードが起こる場合を記す。

4.2.3 パイプラインハザード状況の検討

データハザードが起こる場合は、前の結果を参照するときに起こる。その結果がわかるのは W(書き込み)が終わるときなので、それまで M(メモリアクセス)できない。よって、最大 2 ステージ遅れることになる。

制御ハザードは、無条件分岐の場合もしくは、条件分岐で分岐する場合のとき起こる。W(書き込み)するまでは、どこに分岐するかわからないため、分岐が起こった後でしか F(フェッチ)できない。よって 4 ステージ遅れることになる。

検討の詳細は卒業論文にあるが、その結果の一部を**表1**に示す。

表 1. パイプラインハザード

ハザードの種類	状態	ステージの遅れ
データハザード	状態①: 1命令前を参照	2ステージ
	状態②: 2命令前を参照	1ステージ
	状態③: 1命令前と2命令前を参照	2ステージ
制御ハザード	状態④: 分岐命令の実行	4ステージ

これらの結果を元に、パイプライン処理の実現を行う。また、総ステージ数は次式で計算することができる。

（初期値 4） ＋ （命令数） ＋ （ハザード分）

なお、データハザードは算術・論理演算命令、比較演算命令で起き、制御ハザードは分岐命令で実際に分岐が実行された場合に起きる。

5 実験システムの実現

5.1 トランスレータの実現

CASL2 シミュレータ(*1)のアセンブル機能を拡張して実現した。拡張部分は 1013 行であった。**図 3** に実装したトランスレータの実行の流れと機能を示す。

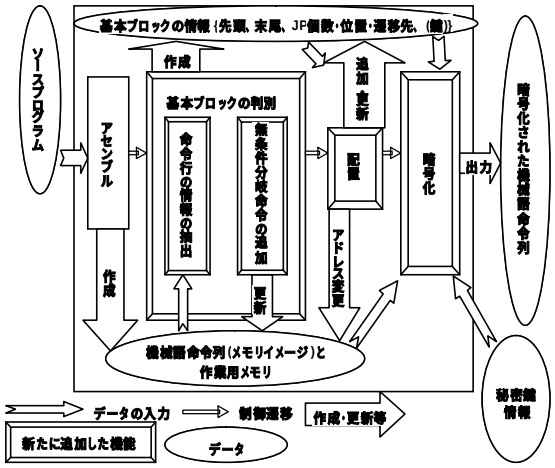

図 3. トランスレータ

1. アセンブル

CASL2 シミュレータのアセンブル機能を利用してソースプログラムから機械語メモリイメージを出力した。追加機能として、その後の基本ブロックの判別や配置処理の中でアドレスが変更されることがあるが、その場合にも容易に対応できるよう、メモリイメージを作業用メモリに格納した。

2. 基本ブロックの判別

アセンブルし、メモリ上に配置された機械語命令列(メモリイメージ)を入力データとし、基本ブロックの判別を行った。

基本ブロックの判別アルゴリズム：

1. 各命令を読み込む。
   条件分岐命令→2 へ、無条件分岐命令→4 へ、それ以外→1 へ、終了→5 へ。
2. 分岐先を基本ブロックの先頭に設定し、次の命令も分岐命令が続いていないか判別する。
   次は条件分岐命令→2 へ、次は分岐命令以外→3 へ
   次は無条件分岐命令→4 へ。
3. 無条件分岐命令の追加が必要なため、印をつける。
   この命令を仮の基本ブロックの末尾とする。1 へ。
4. 分岐先をブロックの先頭に設定し、この命令を基本ブロックの末尾に設定する。1 へ。
5. ブロック末尾に無条件分岐命令を追加する。

3. 配置・追加ダミー命令の決定

基本ブロックごとの鍵を決定するための配置が行われる。今回の配置アルゴリズムは仮のアルゴリズムである。この配置によって PC 差分が決まり、鍵の関数 F でその基本ブロックに対応する鍵が決定される。これにより、基本ブロックの情報の追加とメモリイメージが更新される。基本ブロックの情報には鍵の情報と新たな基本ブロックの先頭と末尾の情報が追加され、メモリイメージは新たな基本ブロックの先頭と末尾のアドレスの変更とダミー命令の追加が行われる。

ここで得られるデータは、配置によりアドレスが変更され、ダミー命令が追加されたメモリイメージと、情報追加された基本ブロックの情報である。

4. 暗号化・出力

配置されたメモリイメージと基本ブロックの情報を入力として、メモリに分岐命令以外の機械語を格納する。

機械語の格納では、配置によって分岐命令とダミー命令が追加されたメモリイメージに、新たな基本ブロックの先頭から配置前の基本ブロック情報を参照して分岐命令以外の機械語を格納する。今回ここで行った暗号化はそれぞれの鍵により値を変えた XOR 暗号であり、暗号鍵は秘密情報として格納されているものとした。

5.2 CPU エミュレータの実現

CASL2 シミュレータの命令取り出し、命令の実行、実行結果の出力部分を利用し、命令の復号機能、初期設定、秘密鍵情報を新たに追加、命令の実行機能に鍵の決定、実行情報の出力機能、パイプライン処理機能を追加して実現した。拡張部分は 700 行であった。**図 4** に実装した CPU エミュレータの実行の流れと機能を示す。

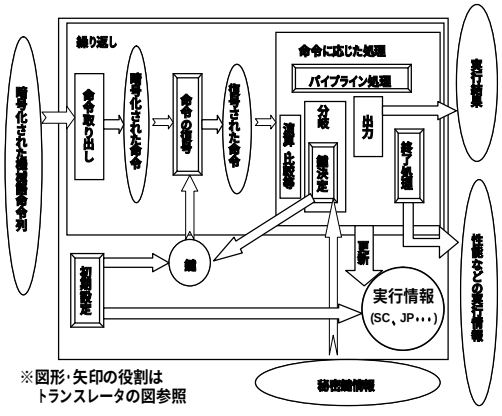

図 4. CPU エミュレータ

(*1)フリーウェア（http://bluepixy.nobody.jp/）

5.2.1 **復号機能の実現**

トランスレータから送られてきた暗号化済みメモリイメージを読む。始めの鍵と命令位置は決まっており、それ以降は一命令ごとに復号と実行を行う。分岐が実際に実行されると、前後の PC 値の差と秘密鍵集合のデータとから新しい鍵を求め、鍵を更新する。

5.2.2 **パイプライン処理の実現**

SC の初期値を 4 に設定し、1 命令読むごとに SC 値を 1 増加する。

データハザード状況を発見するために、直前の 2 命令が使用するレジスタと番地を記憶させる。**表 1** に合わせ、ハザードが起きた場合の遅れ分を SC 値の増加分とする。

制御ハザードが起きる場合では、条件分岐で分岐したとき、無条件分岐のときに 4 ステージ分の遅れが生じるため、SC 値に 4 増加する。分岐しない場合は次の命令に進む。

6 実験システムの評価

6.1 実験システムの評価

暗号化前後のプログラムをそれぞれ実行し評価を行う。比較する値として、実行する命令数、SC 値、分岐数、ループ数、ダミー命令の NOP 数を用いる。**表 2** は一部の実験データを抜粋したものである。

表 2 実験データの一部

ファイル名	命令数	暗号化前の実行する命令数	暗号化後の実行する命令数	暗号化前のSC値	暗号化後のSC値	暗号化前のジャンプ数	暗号化後のジャンプ数	ループ数	NOP数
adsb	11	11	11	21	21	0	0	0	0
divl	22	15	39	28	60	3	5	0	22
ov	11	24	122	63	167	12	14	5	96

表 2 より、暗号化前のプログラムの基本ブロック数に比例し、配置によるダミー命令数や分岐命令の挿入が増加する。そのため、暗号化後の実行命令数や SC 値が増え、実行時間が増加する。また逆に、暗号化前のプログラムに基本ブロックが一つだけ（分岐がない）の場合は、NOP 数も増えず、実行命令数も変更されない。

以上より、暗号化前後の値を算出比較することで、手法を評価するシステムを実現することができた。しかし、最適配置アルゴリズムは今回実装していないので、NOP 数がかなり増加してしまい、実行時間が増加した。

6.2 トランスレータの評価

[6]で提案された手法を実行するために、基本ブロックの判別、無条件分岐命令の追加、配置、暗号化を行った。配置アルゴリズムは今後最適あるいは準最適のものを実装していく必要があるが、基本ブロックの判別部分、無条件分岐命令の追加の部分では使用したサンプルプログラムに対し、期待していた結果を残すことができた。また、基本ブロックごとのデータ抽出部分では、配置アルゴリズムの変更を見据えたインタフェースを実装できた。暗号化部分では、PC 差分により鍵の関数を使い鍵を決定する機能を実装したため、暗号化の追加や変更に対応できると考えられる。

6.3 CPU エミュレータの評価

トランスレータで暗号化された機械語を、1 命令ずつ復号を行い、実行する機能を実装した。また、分岐命令で分岐した場合には、鍵を変更して復号する機能を実装した。これは、手法[6]のプログラム実行の正しさを評価できるシステムと言える。

実行時間の評価を行うためのシステムとして、パイプライン処理を行う SC を実装し、パイプラインハザードを含む実際のパイプラインと同様のステージ数を測る機能を実装した。また、暗号化前後のプログラムの変化を記すために、命令実行数、分岐数、ループ数、NOP 数を実装し、暗号化後の増加量を測ることが可能となった。

7 まとめと今後の課題

手法[6]を紙上およびソフトウェアで評価するため、紙上検討結果、それに基づく実験システムを作成した。トランスレータでは基本ブロックの判別、配置、暗号化を行い、CPU エミュレータでは、復号、実行、データの出力を行った。これらを実行することにより、プログラム実行の静的･動的の評価を行うことができた。

しかし、本研究で使用した配置アルゴリズムは仮のものであり、最適あるいは準最適な配置アルゴリズムを実装することが今後の課題となる。また、本研究で定めた CPU の動作設定は COMET II 専用のものであり、パイプラインや機械語も様々な種類がある。今後は、他の CPU や機械語に容易に対応できるよう実現することが課題である。

参考文献

[1] B.Bara k ら: On the (Im)possibility of Obfuscating Programs, LNCS Vol.2139, pp.1-18 (2001).

[2] 門田暁人ら: ソフトウェアプロテクションの技術動向, 情報処理 Vol.46 No.4, No.5 (2005).

[3] B.Chen.ら: Certifying Program Execution with Secure Processors, Proc. 9th WHTOS, USENIX, pp.133-138 (2003).

[4] D.Lie ら: Implemernting an Untrusted Operating System on Trusted Hardware, Proc. 19th ACM SOSP, pp.178-192 (2003).

[5] G.E.Suh ら: AEGIS: Architecture for Tamper-evident and Tamper-Resistant Processing , ICS2003, pp.178-192 (2003).

[6]大矢真実: 基本ブロックを利用した実行時復号方式によるソフトウェアプロテクション, 南山大学修論 (2007.2 予定).

[7]COMET II の仕様書 http://www.jitec.jp/