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

M2005MM025 大矢 真実
指導教員　河野 浩之

1　はじめに

近年コンピュータにおける情報保護技術が重要となってきている。そのプロテクション技術はデータやソフトウェアなどの保護する対象によってさまざまな方式があり、その一つとして暗号化がある。暗号化は秘密の鍵を持つものだけがその保護された情報を利用することができる方式である。しかしソフトウェアのみでの暗号化は実行時にメモリ上で復号するため、メモリを解析することにより保護されていた情報が漏洩する問題がある。またソフトウェアのみで行われるプロテクション技術はプログラム中に実行情報を全て持つため有限時間内に解析されてしまうことが証明され [3]、より強固なプロテクション技術が必要となってきている。さらに解析技術の進歩により、プログラムの知識がある者が容易にプログラムを解析できるツールなども増えてきている。

これらの問題を解決するためにハードウェアを利用することでさらに強固なプロテクション技術の提案がされてきている [4]。そこでソフトウェアのプロテクションを行うために、プログラムの復号を CPU で行うことでメモリ解析に耐性を持ち、実行効率の低下が少ない on-the-fly 暗号方式の一種を提案する。

2　関連研究

一般的な暗号方式の弱点として暗号の強度を上げるためにその処理内容を複雑にすればするほど実行効率の低下と実装コストが増大する問題がある。これはハードウェアを利用する方法でも同様である。よって暗号の強度とそれに伴う実行効率と実装コストのバランスを考え必要とするレベルの暗号化を行うべきである。

2.1　ブラックボックス

隠したい情報がソフトウェア、隠すべき箱がハードウェアとするブラックボックスの考えを利用する方法では、ハードウェアをブラックボックス化する範囲に応じてコストが変わる。例を挙げると、CPU 以外の他のハードウェア部品からアクセスが不可能となるメモリ (execute-only メモリ) を用意し、プログラム全体をそのメモリ上で実行することで攻撃者からの解析に耐性を持つ方式や、プログラム自体をハードウェアの回路として構成することで、プログラムの情報を攻撃者に与えにくくする方法がある。

ブラックボックス化する方式は一般的にハードウェアへの攻撃に対して強い耐性を持つが、製造にかかるコストが高くなる問題がある。

2.2　on-the-fly 暗号

on-the-fly 暗号は暗号化されたプログラムを実行時に一命令単位で復号する方式であり、単純な構造で実行できコストも低い利点がある。単純な例として鍵を一つ用意し実行時にその鍵を利用し CPU で復号することで、メモリ上には復号されたプログラムがなくメモリ解析に耐性を持つようになる。しかし暗号のブロック長が短いことと鍵が一つであるため暗号強度の面で問題が生じる。これらの問題を解決するために次の暗号方式が提案されている。

プログラムカウンタ (以下 PC と略) の値とあるレジスタの値を種値としたものを利用しハードウェア内部で鍵を生成する方式がある [1]。これにより命令毎に異なる鍵が生成されるため暗号の強度が強くなる。しかし PC を使うためアドレスが固定にならざるを得ず、利用がファームウェアに限定される問題が生じる。

プログラムの適切な一部を限定して暗号化しプログラムが正常に動作しなくなることでプロテクトを行う方式がある [2]。このプログラムを実行するには正規の鍵を持つ者だけで、不正なコピーを行った鍵を持たない者は正しい実行ができない。実行時に CPU がこの暗号部分を復号するため、メモリ上にはその情報がなく暗号化された箇所の特定が難しい。しかし暗号が一箇所なためその他の部分では安全であると言えない。

CPU のデコーダに有限オートマトンを利用し、そのオートマトンでの各状態によって命令の解釈が異なってくる暗号方式がある [5]。デコーダに暗号化された命令が入ると現在のオートマトンの状態により正しい命令に解釈される。そして命令が解釈されるとオートマトンはその命令に応じた別の状態に遷移し、一つ前の命令とは異なった命令の解釈が行われるようになり暗号の連鎖が生じる。しかし分岐の合流点ではオートマトンの状態を特定の状態にしないと誤動作を起こす可能性があるため、ダミー命令を挿入することでそれを回避する。この方法は暗号の強度は高くなるが、復号処理の増加とダミー命令の挿入分の実行効率の低下が生じる。

3　PC 差分値を利用した暗号方式

ハードウェアによるソフトウェアプロテクションはどの方式を行うかにより暗号の強度とコストが変化する。そこでメモリ解析に耐性を持つことを最低条件とし、実行効率の低下が少なくなる暗号方式を考える。

3.1　提案方式概要

本稿では実行効率の低下が少ない on-the-fly 暗号を利用し、その弱点である鍵の変化がない部分を改良するた

め、CPU 内部で鍵を生成する暗号方式を提案する。

鍵生成に利用するパラメータとして [1] のように PC を利用するが、そのままの PC を利用するとアドレスが固定になるという問題が生じるため、特定の 2 点の PC の差分を取ることで解決する。特定の 2 点のアドレスの距離が一定であれば、差分も一定となるためメモリ上の位置に影響されない鍵の生成が行えるようになる。

ここでどの 2 点の PC を利用するかという点と鍵生成に伴う時間コストの二つの問題が生じる。前者の問題はハードウェアがその情報を取得可能なことが条件となる。後者では一命令毎に鍵を生成すると実行効率が低くなるため、特定の単位毎に行う方法を考える。

これらの問題を解決する方法として分岐命令を考えると、その命令は実行時の PC 情報と分岐先の PC の情報の二つの PC 情報を持つため、これらの情報を利用して鍵生成を行う。

またパイプライン方式の CPU で分岐命令を実行する場合には制御ハザードが生じる。制御ハザードでは分岐先の PC が書き込まれるまで待ち状態となるため、その間に鍵生成を終われば鍵生成による実行効率の低下の問題がなくなる。鍵を利用した復号はメモリから命令がフェッチされた時点で行われるため、鍵生成はこの処理までに行われればよい。この復号処理は実行効率の低下が少ない方法で行う。

鍵を変更することの問題点として [5] と同様に分岐合流点での誤動作を回避する必要がある。分岐合流点では鍵生成に利用する差分値が異なるため、その地点の命令の復号に利用する鍵が一意に決定できない。よって分岐合流点での命令に対し一意に鍵が決定できるように差分値の変換を行うことで解決する。これをソフトウェアで解決することでハードウェアへの負担を軽減できる。

3.2 CPU の実行動作詳細

CPU はメモリから暗号化されたプログラムを一命令単位で受け取り、CPU 内部で鍵を利用し復号する。復号された命令は通常の CPU と同様にデコードされる。このときデコードされた命令が分岐命令であるかの判断を行う。分岐命令でない場合は通常の実行をし、分岐命令であった場合はその分岐条件の成否を調べる。分岐が不成立の場合は通常の実行をし、分岐が成立の場合はその分岐命令の実行時の PC と分岐先の PC の値を取得する。この二つの PC の値を利用し鍵を生成する。生成された鍵は分岐命令の分岐先の命令からの次の分岐命令までの復号に利用されることになる。この動作をプログラムが終了するまで繰り返す。

3.3 満たすべきプログラムの条件

この暗合方式では各命令に対し復号に利用する鍵は一意に決まっている必要がある。分岐の状況により鍵生成時にこの条件を満たせない場合が生じるため、その解決をソフトウェアで行う方法を 5 章で述べる。

4 追加される CPU 機能

4.1 要求動作

提案する暗号方式で CPU が行う動作は、命令復号、分岐実行検出、分岐実行時の PC 値取得、鍵生成である。

CPU は暗号化された命令の復号をメモリからのフェッチ時に一命令毎に行う。このときデコードした命令が分岐命令であった場合はその分岐の成否を判断し、分岐が成立した場合は鍵生成に利用する二つの PC 値を取得する。この値から鍵を生成する。

また補助的な機能として CPU で復号をするかしないかを切り替えるスイッチを設けることで、暗号化されたプログラムだけでなく通常のプログラムも動作できる。

4.2 追加機能

この方式で必要とする追加機能は次のようになる。

復号器 暗号化されたプログラムを復号する回路が必要となる。復号器はメモリからの暗号化された命令を入力とし、鍵を利用し復号した命令を出力する。復号には入力と鍵の排他的論理和を取ることで 1 ステップで可能となり、実行時間への影響を最小限に抑えることができる。復号された命令はデコーダの入力となる。

分岐検出 復号された命令がデコードされるときにその値が分岐命令であるかの判断を行う。また分岐実行を検出するには分岐命令を判断した後にその分岐命令の条件に対応するフラグレジスタを参照することで分岐の成否を判断する。

PC 値取得 鍵生成に利用する値として、分岐実行を検出したときの分岐命令と分岐先の二つの PC の値を取得する。これは分岐実行検出の際に PC レジスタを参照することで分岐命令の PC の値を、分岐命令のオペランドの値を取得することで分岐先の PC の値を取得することができる。ただし PC レジスタを参照する際にはすでに PC の値が変化している可能性もあるため、必要な場合には値の調整を行う。

鍵生成器 鍵生成器は分岐点での二つの PC の値を入力とし、結果として鍵値を出力する。その演算例として、鍵を一意に決定可能とするため 2 つの PC の差分を鍵の個数で割った余りを求め、鍵を決定する値にする。その値を鍵集合に渡すことで鍵値を決める。この回路は鍵値を決定する重要な部分のため外部からの攻撃に対して耐性を持つ必要がある。また効率を上げるために鍵の個数を 2^k とすることで、差分や余りの演算は下 k ビットの演算にすることができる。

復号切り替えスイッチ メモリから入力された命令が暗号化されたものか否かにより復号の有無の切り替えが必要となる。CPU がこの状態を切り替えることで復号の有無を決定する。この切り替えの制御は OS によるシステムコールや CPU で特定の命令列を読み込むことにより切り替える方法などがある。

図 1 は本来の CPU 機能に、本稿で提案する暗号方式の復号を可能とする機能を追加した構成図で、図 2 はその動作のフローチャートを示している。

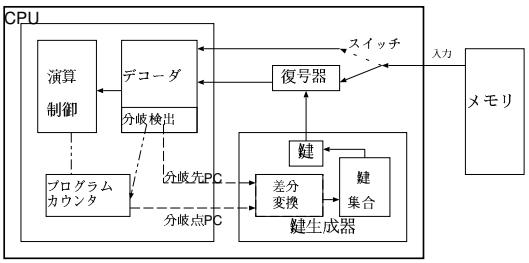

図 1　CPU への機能追加

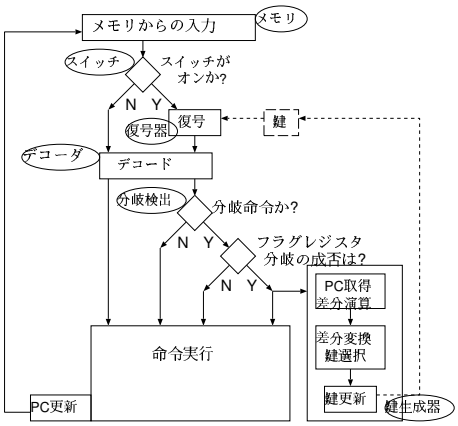

図 2　CPU の動作フローチャート

5　ソフトウェアによる解決法

5.1　プログラムの構造変換

機能を追加した CPU で正しい動作を可能にするには、各命令に対し一意の鍵で復号するという条件を満たす必要があり、分岐で発生する次の二つの状況を解決しなければならない。一つ目は分岐が発生せずに他の分岐命令と合流した場合に片方の鍵が生成されない点である。二つ目は二つ以上の分岐の合流点では差分が異なるため一意に鍵が決定できない点である。

一つ目の状況を解決する方法として、プログラムの制御フローを解析し分岐が発生する地点とその分岐先でプログラムを分割し、その分割したまとまりの間の移動を全て分岐命令で行うように分岐命令を追加する。結果としてこのプログラムを分割するまとまりは基本ブロックであると分かる。この方法により基本ブロック間の移動は全て分岐命令で行われるようになり一つ目の状況が解決される。ただし分岐命令を追加するため制御ハザードが増え実行効率の低下の原因ともなる。

二つ目の状況を解決する方法として、プログラムの仕様に影響を与えないダミー命令を挿入することで差分の値を調整し、各命令に対し復号に利用する鍵を一意に決定できるようにする。またダミー命令の挿入により実行効率の低下が起こるため、その挿入数をできるだけ少なくする方法を行う必要がある。

5.2　実行効率の改善

ダミー命令の挿入による実行効率の低下を防ぐため、その挿入数を少なくする方法を考える。まずプログラムをアセンブルすることによりこの問題を解くのに必要な PC の情報などを取得する。ここで問題を単純化するために一つの基本ブロックには条件分岐命令 (以下 JC と略) と無条件分岐命令 (以下 JP と略) がそれぞれ一つとする。このときダミー命令が挿入可能な位置として図 3 のように基本ブロックの先頭、JC の前後がある。このダミー命令を挿入することで JC や JP の位置がずれ差分の調整が可能となる。基本ブロックの先頭に挿入されるダミー命令は実行される位置にあるため実行効率に影響を与えない。

ここで K を鍵の個数、差分から鍵を決定する値に変換する式を$F(\mathrm{Diff}) = \mathrm{Diff} \bmod K$、各基本ブロックの先頭位置を K の倍数に取ることで、基本ブロック間の PC の差分値は他の基本ブロックのダミー命令や大きさに影響されない計算式で表現できるようになる。これを行うにはプログラムに対しメモリの容量が十分であることが必要となる。

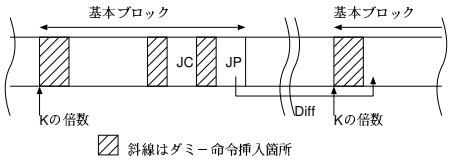

図 3　ダミー命令の挿入箇所

このとき変数として各基本ブロックの先頭に挿入するダミー命令数、各基本ブロックの鍵を決定する値、基本ブロックの順列があり、これらを変化させることで、実行効率に影響が生じる位置のダミー命令の挿入数を少なくすることができると考えた。

実験として基本ブロックの大きさと 2 つの分岐命令の分岐先を乱数で発生させたものを利用する。このとき入力データが実行が終わらなくなる制御フローとなった場合は入力をやり直す。変数の各基本ブロックの鍵を決定する値は 0 から $K-1$ まで、基本ブロックの先頭に挿入するダミー命令数も 0 から $K-1$ まで、基本ブロックの順列は全てのパターンを行う。この三つの変数の組み合わせ全ての場合を調べ、実行効率に影響が生じる位置に挿入されるダミー命令数が最小となる値を取る。

この実験を基本ブロック数と鍵の個数を変化させ 100 回から 10000 回行った平均値を取り、どのような傾向があるかを調べた。その結果が表 1 のようになり、基本ブロック数と鍵の個数に対しダミー命令の平均挿入数は比例的に増加することが分かる。

5.3　考察と今後の課題

前節の実験によってプログラムを正しく実行するために挿入されるダミー命令数の平均を求めることができたが、実行効率に影響を与えるのはダミー命令の挿入数ではなく、実行数である。よってプログラムのループなど

表 1　最小ダミー命令数の平均値

$K \diagdown N$	3	4	5	6	7
4	1.44	1.86	2.31	2.91	3.40
8	2.42	3.44	4.93	-	-
16	6.58	8.65	-	-	-
32	9.79	-	-	-	-

の構造に応じてその実行数を最小となるような挿入法を求めることが今後の課題となる。

6　その他の検討すべき点

6.1　解決すべき問題点

この暗号方式を実際に実装する場合には、複雑な処理をする命令や、OS やハードウェアの機能におけるいくつかの問題を解決しなければならない。

複雑な処理をする命令　プログラム中には関数呼出を行う call-return 命令や、Index レジスタを利用した複雑な処理をする命令がある。このとき問題となるのは一つの分岐命令に対し複数の分岐先が存在するという点である。このため各命令で差分を利用した鍵が一意に決定されない問題が生じる。

関数呼出を解決する案としては、ハードウェアに鍵値を保存することが可能な鍵用スタックを用意する方法がある。これを利用することで関数が呼び出されたときに、そのときの PC がスタックにプッシュされるのと同様に、そのとき利用していた鍵値を鍵用スタックにプッシュする。そして RETRUN が行われたときにこの鍵値をポップすることで元の状態に戻すことができる。

ハードウェアの機能　OS にはハードウェアでプログラムを効率よく実行するための機能がある。例を挙げると、プログラムをメモリに読み込んだ際にメモリの断片化が生じた場合には動的再配置を行いメモリ空間を効率よく扱う機能がある。物理メモリ空間と論理メモリ空間をページと呼ばれる単位で分割し、OS がその管理をすることでメモリの効率化を図るページング方式もある。この方式によりプログラムはメモリ上で連続でなくても動作することが可能となる。

これらの機能によりメモリ上の分岐命令と分岐先の上下位置が反転してしまう可能性もあり、その場合は鍵を決定する値が変わる問題が生じる。よってメモリの動的再配置が行われたとしても復号に利用する鍵が一意に決まる条件を考える。

メモリの動的再配置が行われる単位としてページフレームがある。この単位が K の倍数である場合、メモリ上の上下に変化がなければプログラムの途中で分割されたとしても分岐命令とその分岐先の PC の差分は変化しない。またメモリ上の上下の変化があり鍵を決定する値が変わったとしても鍵値が変化しない方法を行えばこの問題を解決することができる。

例を挙げると、メモリ上の上下の変化により鍵を決定する値 Key_j は $K - \mathrm{Key}_j$ となる。よって Key_j による鍵値と $K - \mathrm{Key}_j$ による鍵値が等しくなるように鍵集合の鍵値を設定すれば解決ができる。これは鍵値を鍵集合の中心から対称に決定することで条件を満たすことが可能となる。

6.2　その他の追加機能

暗号方式は機能により暗号の強度、実行効率、実装コストが変化する。そこで必要とする暗号強度に対し、その実行効率や実装コストが選択できるようにいくつかの機能を検討する。

部分的な暗号化　プログラム全体を暗号化するよりも、プログラムのアルゴリズムや鍵などの重要な一部のみを暗号化することで、その実行効率をよくする方法がある。これは単純に復号切り替えスイッチを利用することにより実行することができる。

しかしスイッチを切り替えたときに鍵を正しく決定できるための機能が必要になり、また一部のみを暗号化するためその暗号箇所の推測が容易になる問題が生じる。

暗号強化　提案した暗号方式では基本ブロック毎には鍵が変化しているが、基本ブロック内では鍵の変化がない。そこで基本ブロック内でも鍵が毎回変化するような構造を作ることでこの方式を改良する方法もある。単純な例としては基本ブロックの先頭の鍵値を命令がフェッチされる度に単純増加することで暗号に利用する命令毎の鍵の変化が可能となる。これは PC レジスタが命令毎に増加するのと同様な方法で実装でき、その実行時間もプログラムに影響がない。

7　おわりに

本稿では PC 差分を利用することで実行効率への影響が少ない暗号方式の提案をした。そしてそのプログラムを正しく実行するために必要なダミー命令の挿入数を求めることができた。しかし実行効率への影響はダミー命令の実行数であるため今後の課題としてその実行数を少なくする方法の検討が必要である。

参考文献

[1] G. Yearsley et al. : “Firmware Encryption for Microprocessor / Microcomputer”, US Patent 5,386,469 (1995)

[2] 西垣 正勝ら : “データのスクラッチングと動的復元によるバイナリープログラムの不正コピー防止方式”, 電子学会論文誌 A, Vol.J83-A, No.11 (2000).

[3] B. Barak et al. : “On the (Im)possibility of Obfuscating Programs”, LNCS 2139 (2001).

[4] 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.4,5 (2005).

[5] A. Monden et al. : “Tamper-resistant Software System based on a Finite State Machine”, IEICE Trans. Vol.E88-A, No.1 (2005).