B-036

実行時間に依存したプログラムの暗号化
Program Encryption Based on the Execution Time

坂口 英司 † Hideshi Sakaguchi
神崎 雄一郎 ‡ Yuichiro Kanzaki
門田 暁人 § Akito Monden

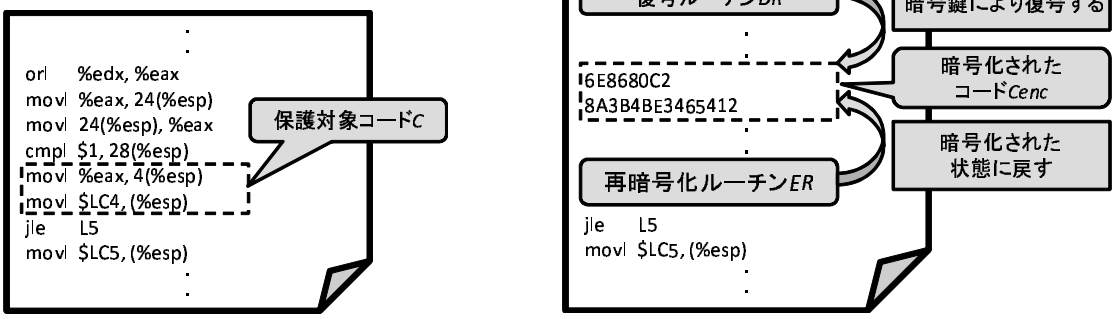
1. はじめに

ソフトウェアの内部には，ソフトウェアベンダにとって重要な情報が含まれている場合がある．重要な情報の例としては，類似製品と比較して優位性を持つ機能や，シリアルナンバーの認証ルーチンなどが挙げられる．このような情報は，ソフトウェア内部で重要な役割を持つとともに，悪意のある人物 (攻撃者) にとって有益な情報である場合が多く，不正な解析行為のターゲットになりやすい．そのため，ソフトウェアが持つ情報を攻撃者から保護するための技術 (ソフトウェア保護技術) が求められている．ソフトウェア保護技術には様々な方法が提案されているが，それらの 1 つとしてプログラムの暗号化があり，例えば，Aucsmith[1] や Cappaert[2] などによって提案されている．プログラムの暗号化とは，保護の対象となるコードまたはデータ (保護対象) をあらかじめ暗号化しておき，実行時に復号するというものである．この方法の長所としては，静的解析 (プログラムを実行せずに行う解析) に対して有効であることが挙げられる．保護対象を暗号化することによって，保護対象は意味を持たないコードやデータに変換される．そのため，攻撃者は，パターンマッチングを用いた検索等によって解析の手がかりや保護対象そのものを発見することが困難になり，静的解析に要するコストが増大する．一方で，プログラムの暗号化は，動的解析 (プログラムを実行しながら行う解析) に対して必ずしも有効ではない．例えば，攻撃者がデバッガを用いて保護対象が復号されたタイミングで実行を停止した場合，元来のデータが容易に知られる危険性が大きい．

そこで本研究では，動的解析に対して有効なプログラムの暗号化方法を提案する．提案方法では，プログラムを通常実行させた場合と動的解析を行いながら実行させた場合の実行時間の違いを用いて，動的解析が行われた場合には保護対象が正しく復号されないようにする．保護対象は，あらかじめ対象鍵暗号によって暗号化されており，プログラムの実行時に，一部のコードの実行時間を基にして生成される暗号鍵によって，復号・再暗号化される．攻撃者がデバッガなどを用いて動的解析を行い，実行の一時停止や命令のスキップなどによって実行時間が通常実行時よりも長くまたは短くなった場合には，正しい鍵が生成されず，保護対象は元来のコードに復号されなくなる．

2. 基本アイデア

提案方法では，プログラムの実行時間が通常実行時と同程度であった場合にのみ暗号化されたコードが正しく復号されるような機構をプログラムの多数の箇所に設けることで，プログラムを保護する．図 1(a) および (b) は，それぞれ元来のプログラム P および提案方法によって保護されたプログラム P_p のコードの構成の例を簡潔に表したものである．ここ



(a) 元来のプログラム P (b) 提案方法によって保護されたプログラム P_p

図 1: 基本アイデア

†熊本高等専門学校 専攻科 電子情報システム工学専攻, Advanced Course of Electronics and Information Systems Engineering, Kumamoto National College of Technology
‡熊本高等専門学校 人間情報システム工学科, Dept. of Human-Oriented Information Systems Engineering, Kumamoto National College of Technology
§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

では，プログラムを AT&T のアセンブリコードを用いて示している．P の一部は，保護対象コード C として選択され，C は対象鍵暗号によって暗号化されたコード C_{enc} として上書きされる．P_p が実行されたとき，C_{enc} は，復号ルーチン DR によって復号され，再暗号化ルーチン ER によって再暗号化される．すなわち，C_{enc} は，DR によって復号されてから ER によって再暗号化される期間だけ C となる．DR および ER は，P_p に含まれる実行時間測定対象コード B の実行時間 $T(B)$ を基に生成される暗号鍵を用いて復号および再暗号化を行う．提案方法は，実行時に以下の順番で実行される．

1. P_p の実行を開始する．
2. B に到達したとき，B の実行時間 $T(B)$ を測定する．$T(B)$ は一方向ハッシュ関数によってハッシュ値 $hash(T(B))$ に変換し，メモリ領域に保存しておく．
3. DR に到達したとき，$hash(T(B))$ を鍵として，対象鍵暗号により C_{enc} の復号を行う．
4. C_{enc} に到達したとき，復号された C_{enc}(C と同一の処理内容) を実行する．
5. ER に到達したとき，$hash(T(B))$ を暗号鍵として，対象鍵暗号により C_{enc} を再暗号化を行う．

$T(B)$ が通常実行時における B の実行時間 $T_0(B)$ よりも長いまたは短い場合には，暗号鍵 $hash(T(B))$ は $hash(T_0(B))$ と一致しなくなるので，DR の実行時に C_{enc} は元来のコードに復号されなくなる．C_{enc} が元来のコードに復号されなかった場合，アクセスバイオレーションが生じてプログラムの実行が停止するなど，元来のコードを実行したときとは異なる結果が生じる．このような仕組みをプログラムの多くの箇所に設けることで，攻撃者が動的解析によって C を取得することを困難にする．

3. 提案方法の適用手順

提案方法によって保護されたプログラム P_p を構成する手順を以下に示す．保護したいコードの数だけ，(Step 1) から (Step 6) を繰り返し適用する．各ステップでは，i 番目の適用が行われているものとして考え，C，C_{enc}，B，DR，ER にそれぞれ i という添え字をつけて表している．

(Step 1) 保護対象コード C_i の決定

はじめに，提案方法によって保護される保護対象コード C_i を決定する．C_i は元来のプログラム P から任意に選択することができる．例えば，条件判断文，デジタルコンテンツの暗号化・復号に用いる暗号鍵，ソフトウェア内で重要な役割を持つ機能などが挙げられる．

(Step 2) 実行時間測定対象コード B_i の決定と，復号ルーチン DR_i および再暗号化ルーチン ER_i の挿入位置の決定

実行時間測定対象コード B_i と，暗号化されたコード C_{enc_i} の復号を行うルーチン DR_i および再暗号化を行うルーチン ER_i の挿入位置を決定する．B_i と，DR_i および ER_i の挿入位置は，以下の条件を満たすように決定する．

1. B_i は，P_p の実行の開始位置から C_{enc_i} に至る経路の間に存在する．B_i は基本ブロックとする．
2. DR_i は，B_i から C_{enc_i} に至る経路の間に存在する．
3. ER_i は，C_{enc_i} から P_p の実行の終了位置に至る経路の間に存在する．

(Step 3) 実行時間測定命令および一方向ハッシュ関数の挿入

B_i の実行時間 $T(B_i)$ を測定するための命令および一方向ハッシュ関数を挿入する．実行時間測定命令は，B_i の直前および直後に挿入する．$T(B_i)$ は，取得した値の差分を計算することで導出する．具体的な方法としては，プロセッサの実行クロック数を測定する命令 (Intel A-32 アーキテクチャにおける RDTSC 命令 [3]) を用いて，B_i の実行に要した実行時間を測定する方法が挙げられる．この場合，コードの実行時間は実行クロック数で表すことになる．また，一方向ハッシュ関数は，$T(B_i)$ の計算を行うルーチンの直後に挿入する．一方向ハッシュ関数により，$T(B_i)$ のハッシュ値 $hash(T(B_i))$ を取得する．

(Step 4) 復号ルーチン DR_i および再暗号化ルーチン ER_i の生成と挿入

復号ルーチン DR_i および再暗号化ルーチン ER_i を生成し，P_p に挿入する．DR_i では，$hash(T(B_i))$ を鍵として，対象鍵暗号により C_{enc_i} の復号を行う．ER_i では，$hash(T(B_i))$ を暗号鍵として，対象鍵暗号により (復号された状態の)C_{enc_i} の再暗号化を行う．

(Step 5) 実行時間 $T(B_i)$ のしきい値の決定

B_i の実行時間 $T(B_i)$ のしきい値を決定する．プログラムの動的解析を行うと，実行時間が通常実行時よりも長くまたは短くなる場合があることを想定し，通常実行時における B_i の実行に必要な最小の実行時間 $T_{0_{min}}(B_i)$ と最大の実行時間 $T_{0_{max}}(B_i)$ をしきい値として決定する．$T(B_i)$ がこれらの値の間に収まれば通常実行されていると判断し，そうでなければ動的解析が行われていると判断する．$T_{0_{min}}(B_i)$ および $T_{0_{max}}(B_i)$ は，あらかじめ B_i を実行させることで決定するか，もしくは B を構成するコードと P_p の実行環境から概算する．

(Step 6) 対象鍵暗号を用いた C_i の暗号化および上書き

対象鍵暗号を用いて，C_i の暗号化および上書きを行う．暗号鍵には，B_i の通常実行時の実行時間 $T_0(B_i)$ のハッシュ値 $hash(T_0(B_i))$ を用いる．$T_0(B_i)$ は，$T_{0_{min}}(B_i)$ から $T_{0_{max}}(B_i)$ までの範囲に対応するような値に設定する．例えば，$T_{0_{min}}(B_i)$ および $T_{0_{max}}(B_i)$ に共通している上位ビットを $T_0(B_i)$ として設定する．この場合，プログラムの実行時には $T(B_i)$ の上位ビットから暗号鍵を生成する．C_i を対象鍵暗号によって暗号化されたコード C_{enc_i} に暗号化した後，C_i を C_{enc_i} で上書きする．

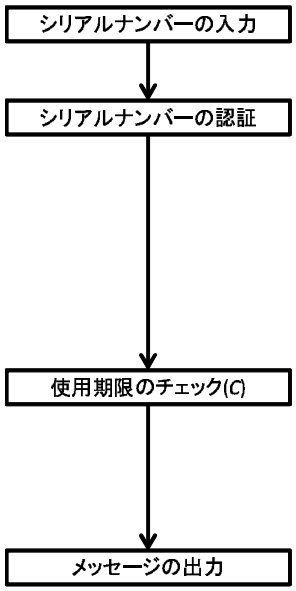
4. ケーススタディ

提案方法の動作を検証するために，簡単な実験を行った．今回の実験において提案方法を適用するプログラムは，ユーザが入力したシリアルナンバーの認証を行った後にプログラムの使用期限のチェックを行い，最後にメッセージを出力するというものである．図 2(a) および (b) は，それぞれ元来のプログラム P の実行の流れと保護されたプログラム P_p の実行の流れを表したものである．今回は，プログラムの使用期限のチェックを行うルーチンを保護対象コード C とし，シリアルナンバーの認証を行うルーチンを実行時間測定対象コード B とした．また，対象鍵暗号として 128 ビットの AES 暗号を ECB モードで使用し，一方向ハッシュ関数として MD5 を使用した．なお，動作環境は表 1 に示す通りである． 説明

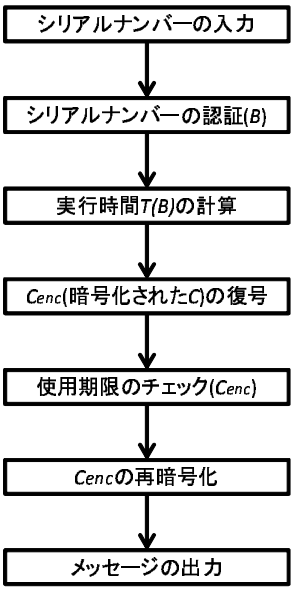
表 1: 実験の動作環境

OS	Windows 7 Home Premium 64bit
CPU	Intel(R) Core(TM) i7 CPU @ 2.80GHz
メモリ	4.00GB

のため，プログラムの実行時間は実行クロック数で表す．今回の実験では，$T(B)$ の最小実行時間 $T_{0_{min}}(B)$ は 1,048,576



(a) 元来のプログラム P の実行の流れ



(b) 保護されたプログラム P_p の実行の流れ

図 2: 実験用プログラムの実行の流れ

表 2: 各実行方法での動作結果

実行方法	実行時間 [クロックサイクル]	通常実行時の実行時間との比率	動作結果
通常実行	1,369,098	1.00 倍	正しい動作
ブレークポイントによる一時停止	9,679,263,929	約 7,070 倍	誤った動作
ログの出力 (B の全命令)	49,434,046	約 36.1 倍	誤った動作
ログの出力 (B の一部)	28,688,669	約 21.0 倍	誤った動作
命令のスキップ (B の一部)	944	約 6.90×10^{-4} 倍	誤った動作

クロックサイクル，最大実行時間 $T_{0_{max}}(B)$ は 2,097,151 クロックサイクルとした．

今回は，以下の実行方法で実験用プログラムを実行した．

- 通常実行を行う．
- デバッガのブレークポイント機能を用いて，B の実行を約 3 秒間停止する．
- B を構成するすべての命令をログファイルに出力する．
- B を構成するアセンブリ命令のうち，一部 (全命令数の約 10%分) の命令をログファイルに出力する．
- B を構成するアセンブリ命令のうち，一部 (全命令数の約 10%分) の命令をスキップする．

以上の実行を行った結果を表 2 に示す．表 2 の動作結果において，正しい動作とは C_{enc} が元来のコードに復号されたことを表す．また，誤った動作とは C_{enc} が元来とは異なるコードに復号されたことを表す．通常実行を行った場合は，C_{enc} は元来のコードである C に復号されたが，ブレークポイントによる実行の一時停止，ログファイルの出力，命令のスキップを行った場合は，C_{enc} は元来のコードに復号されなかった．C_{enc} が元来のコードに復号されなかった場合，存在しない命令を実行しようとして例外が発生したり，アクセスバイオレーションエラーが発生することを確認した．また，通常実行時の実行時間と比較すると，ブレークポイントによる実行の一時停止を行った場合は通常実行時の約 7,070 倍，B のすべての命令をログファイルに出力した場合は通常実行時の約 36.1 倍，B の一部の命令をログファイルに出力した場合は通常実行時の約 21.0 倍，B の一部の命令をスキップした場合は通常実行時の約 6.90×10^{-4} 倍の実行時間であった．ここで，B の一部の命令をスキップしながら実行したときに実行時間が通常実行時と比べて非常に短くなっているのは，処理に時間のかかる命令 (C 言語における `printf()` 関数の実行) を，スキップする命令の対象に選択したためだと考えられる．以上より，通常実行を行った場合は C_{enc} が元来のコードに復号されること，また，解析行為によって実行時間がしきい値に収まらなかった場合には C_{enc} が元来のコードに復号されず，正しく実行されないことを確認できた．

5. 議論

提案方法の特徴と課題に関する議論を行う．提案方法は，保護対象コードを暗号化することで静的解析を困難にし，実行時間を基にして暗号鍵を生成することで動的解析も困難にすることを目的としたものである．4 章のケーススタディにおいて，プログラムの実行の一時停止，ログファイルの出力，命令のスキップを行いながらプログラムを実行した場合には，実行時間が通常実行時よりも長くまたは短くなり，保護対象コードは元来のコードに復号されないことを確認した．このことから，提案方法は動的解析を用いた攻撃に対して有効であると考えられる．一方で，以下のような課題が挙げられる．

第 1 に，実行時間測定対象コード B を攻撃者から解析さ

れにくくすることである．提案方法によって，保護対象コード C を直接解析することは困難となった．しかし，攻撃者が提案方法のアルゴリズムについて理解がある場合，B の位置を探して B だけを通常実行することで，暗号化されたコード C_{enc} を元来のコードに復号することができると考えられる．例えば 4 章で用いた実験用プログラムでは，RDTSC 命令を用いて B の実行時間 $T(B)$ を測定した．しかし，RDTSC 命令は，一般に MOV 命令や CMP 命令と比べて使用頻度の少ない命令である．そのため，デバッガを用いて RDTSC 命令が使用されている位置を特定することで，B の位置を探すことは不可能ではないと考えれられる．この問題を解決するためのアプローチとしては，実行時間測定命令を保護対象コードとして提案方法を適用することが挙げられる．RDTSC 命令のような実行時間を測定するコードを保護対象として暗号化することで，攻撃者は B の位置を特定することが困難になると考えられる．また，提案方法を適用する際に使用するコード (復号ルーチン DR，再暗号化ルーチン ER など) に対して提案方法を適用することは，保護対象コードを二重に保護することになるので，この点からも解析行為に対して有効であると考えられる．その他のアプローチとしては，プログラムの実行結果に影響を与えないような位置に，実行時間を測定するコードをダミーとして挿入することが挙げられる．

第 2 に，実行時間が大きく異なる複数の実行環境にも対応できるようにすることである．4 章では，表 1 に示す実行環境で実行時間の測定を行った．実際には，プログラムの実行環境はユーザによって異なる．実行環境が異なるとプログラムの実行時間も変動するので，B の最小実行時間 $T_{0_{min}}(B)$ と最大実行時間 $T_{0_{max}}(B)$ は実行環境に応じて設定する必要がある．ただし，幅を広げすぎると動的解析を検出することが困難になる場合があるため注意する必要がある．

第 3 に，提案方法の実用性を上げるために，保護されたプログラム P_p のオーバヘッドを小さくすることである．P_p では P の一部を復号および再暗号化しているため，実行時間が P よりも増大する．また，P に暗号アルゴリズムを追加した分だけファイルサイズも大きくなる．4 章で使用したプログラムでは，P_p の実行時間 (クロックサイクル数) は P と比べて約 5.81 倍であった．また，P_p の実行ファイルのサイズは，P と比べて約 1.23 倍であった．P_p のオーバヘッドを小さくするためのアプローチとして，一方向ハッシュ関数や暗号アルゴリズムはより少ない命令数で実装できるものを採用することが挙げられる．ファイルサイズの差を小さくする場合についても，同様のアプローチを挙げることができる．

6. 関連研究

コードの暗号化に関する研究として，Cappaert らのコード暗号化 [2] と Aucsmith らのコード暗号化 [1] がある．Cappaert らが提案した方法は，ある関数をコールする関数 (caller; コーラ) とコールされた関数 (callee; コーリ) が，互いに暗号化および復号を行うというものである．コーラは，コーリを呼び出す前後で，コーリをそれぞれ復号および再暗号化する．その際に用いる暗号鍵は，コーラのコードのハッシュ値である．一方，コーリは，コーラの暗号化および復号を行う．その際に用いる暗号鍵は，コーリのコードのハッシュ値である．また，Aucsmith らによって提案された方法は，保護対象プログラムを 6 つの断片 (セル) に分割し，セル同士で排他的論理和をとり，さらに暗号化を行うというものである．それぞれのセルは 2 つのグループに分けられ，他のグループのセルと排他的論理和をとり，さらに暗号化される．ここで，セルを暗号化する際に用いる暗号鍵は，ユーザが任意に決定する．ただし，あるセルを構成しているコードが実行される際には，当該セルが元来のコードに復号されるように暗号鍵を決定しなければならない．Aucsmith らの提案方法は，セル同士での排他的論理和の演算とセルの暗号化を繰り返すというものである．提案方法は，実行時間を基にして暗号鍵を生成しているという点で，これらの方法と異なる．実行時間を基にして暗号鍵を生成することで，攻撃者に対して動的解析時の実行時間に関する制約を負わせることができる．

プログラムの実行時間を利用したソフトウェア保護方法としては，神崎らが提案した方法 [5] と Collberg らが提案した方法 [4] がある．神崎らが提案した方法は，あらかじめ保護対象コードを別のコードで隠ぺいしておき，実行時間に応じて隠ぺいされたコードを書き換えるというものである．保護対象プログラムの初期状態では，保護対象コードは別のコードで隠ぺいされている．プログラムの実行時に，プログラムの一部の実行時間を測定し，実行時間が通常実行時と同程度であった場合のみ，隠ぺいされたコードは元来のコードに自己書き換えされる．神崎らの方法では保護対象を任意のコードに書き換えているが，本研究の提案方法は，暗号アルゴリズムを利用して保護対象を書き換えているという点で異なる．また，Collberg らが提案した方法では，RDTSC 命令を用いて実行時間を測定し，測定した実行時間とあらかじめ設定されたしきい値を比較することで動的解析が行われているかどうかを判断している．しかし，条件判断文による判別を行った場合は実行時間のしきい値がプログラムに直接現れ，静的解析によってしきい値を解析されてしまう恐れがある．本研究の提案方法では，条件判断文ではなく暗号鍵の算出を行うことで，このリスクを回避している．

7. おわりに

本研究では，動的解析を用いた攻撃に対して有効なソフトウェア保護技術として，実行時間に応じて暗号鍵を生成するプログラム暗号化方法を提案した．提案方法は，保護対象コードを対象鍵暗号によってあらかじめ暗号化しておき，実行時に一部のコードの実行時間を基にして暗号鍵を生成し，復号および再暗号化を行うというものである．また，実際にプログラムに対して提案方法を適用し，いくつかの解析行為に対して有効であることを確認した．今後の課題としては，提案方法を適用する際に追加するコード (復号ルーチンや再暗号化ルーチン) を攻撃者から解析されにくくすること，様々な実行環境でも正しく動作するように実行時間のしきい値を設定すること，提案方法を適用したプログラムのオーバヘッドを小さくすることなどが挙げられる．

参考文献

[1] D. W. Aucsmith. Tamper resistant software: An implementation. In *Information Hiding*, Vol. 1174 of *Lecture Notes in Computer Science*, pp. 317–333. Springer-Verlag, 1996.

[2] J. Cappaert, N. Kisserli, D. Schellekens, and B. Preneel. Self-encrypting code to protect against analysis and tampering. *Proc. Benelux Workshop on Information and System Security*, 2006.

[3] Intel Co. *IA-32 Intel Architecture software developer's manual vol.2 : Instruction Set Reference.* http://www.intel.co.jp.

[4] C. Collberg and J. Nagra. *Surreptitious Software.* Addison-Wesley, 2009.

[5] 神崎雄一郎, 門田暁人. 実行時間差に着目したコードの隠ぺい方法. 第 8 回情報科学技術フォーラム (FIT2009) 講演論文集 第 1 分冊, pp. 361–364, September 2009.