ハードウェアと連携したセキュアインタプリタ技術の提案

村上啓造　岸晃司　山本隆広

日本電信電話株式会社　NTT セキュアプラットフォーム研究所
〒180-8585 東京都武蔵野市緑町 3-9-11
murakami.keizo@lab.ntt.co.jp

あらまし　近年，個人情報を扱うサービスが増加しており，個人情報を扱うプログラムを保護する重要性が増している．本研究では，インタプリタ機能をハードウェアセキュリティモジュール（HSM）に持たせ，HSM外部の暗号化したアプリケーションを実行時にHSMに逐次読み込み，復号・チェック・実行させる「ハードウェアと連携したセキュアインタプリタ技術」を提案する．

A Proposal of Secure Interpreter Working with Hardware

Keizo Murakami　Kouji Kishi　Takahiro Yamamoto

NTT Secure Platform Laboratories, NTT Corporation
3-9-11 Midori-cho, Musashino-city, Tokyo 180-8585, JAPAN
murakami.keizo@lab.net.co.jp

Abstract Today services handling personal information have been increasing. So protection of programs handling personal information has been more important. This study proposes ‘Secure Interpreter Working with Hardware’. This technology means that we construct interpreter in Hardware Security Module (HSM) and the interpreter reads in programs which are encrypted and stored in outside of HSM and the interpreter decrypts, checks and runs the programs.

1　はじめに

近年，公共分野では，電子行政推進の基本方針の一つとしてオンライン利用の拡大を推進しており[1]，パソコンだけでなく，行政キオスク端末や，携帯電話等のモバイル端末，デジタルテレビ等を活用した行政サービスに対するアクセス手段の多様化が検討されている[2]．従来は行政サービスへのアクセスには，IC カードを用いた認証が必要であった．IC カードで認証を行うためには，IC カード R/W が端末に接続できる必要があるため，パソコンに利用が限られ，アクセス手段多様化の妨げとなっていた．

また，従来の IC カードサービスでは，利用者に IC カードを配布後に IC カード内のアプリケーションを更新・追加・削除を行う際には利用者に個別に対応してもらう必要があり，運用者による一括操作は不可能という課題があった．

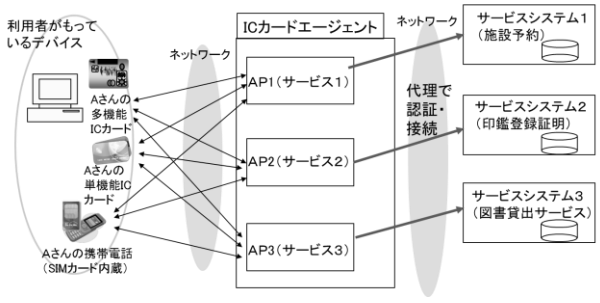

図 1　IC カードエージェント

そこで我々は，IC カードに格納されているアプリケーションをサーバ内に格納し(以下，IC カードエージェントとする），代理で実行する方式を検討してきた（図 1）．利用者の持つデバイスで本人確認を行い IC カードエージェントにログインし，IC カードエージェントが従来のカードサービスを代行する，という方式である．

これにより，IC カード R/W を持たない利用者も従来の IC カードと同等のサービスを受けることが可能となり，かつ，従来 IC カード内に格納されていたアプリケーションはサーバ内に格納されるため，運用者による一括操作が行えるようになる．

このような IC カードエージェントを考える際，IC カードエージェント内に格納されるアプリケーションは個人情報を扱うアプリケーションであり，従来は IC カードという耐タンパハードウェア内に格納されていたものであるため，同等のセキュリティを確保する必要がある[3][4]. セキュリティを確保するには多層防御を行うことが重要であり，アプリケーションの保護も行う必要がある[5].

2　ソフトウェア保護の課題

サーバ上で動作する，個人情報を扱うアプリケーションを保護する方法として，ソフトウェアで守る方法とハードウェアで守る方法がある．

ソフトウェアで守る方法として，アプリケーションの難読化・暗号化がある．しかし，アプリケーションの難読化は時間をかければ解析可能であり，暗号化は動的解析に弱いという欠点がある[6].

ハードウェアで守る方法として，耐タンパ性を持ったハードウェアである，ハードウェアセキュリティモジュール（以下，HSM）内にアプリケーションを格納し，HSM 内で実行する，という方法がある．HSM 内でアプリケーションを実行した場合，動的解析の脅威をなくすことができる．ただし，運用開始後のアプリケーションは随時アップデートを行う必要があるが，HSM に格納したアプリケーションは，セキュリティ確保のため，厳重な管理化でシステム開発者が専用のツールにより操作する必要があり，開発完了後の運用時の HSM へのアプリケーション更新・追加・削除は想定されていない．そのため，従来の技術では，運用開始後のアプリケーションの柔軟な運用は難しかった．

このように，従来のアプリケーションを保護する技術には，

- アプリケーションの難読化は時間をかければ攻撃可能である．
- アプリケーションの暗号化は動的解析に弱い．
- HSM 内でアプリケーションを実行する方法は，運用開始後のアプリケーションの柔軟な運用が困難

という課題があった．

こうした現状を踏まえ，我々は，ソフトウェア・ハードウェア両方でアプリケーションを保護する「ハードウェアと連携したセキュアインタプリタ技術」を提案する．

3　提案技術の概要

提案技術は，HSM 内部にインタプリタを構築し，HSM 外部に格納した暗号化したアプリケーションを実行時に HSM に逐次読み込み，復号・チェック・実行させる技術である．構成は図 2 の通りであり，

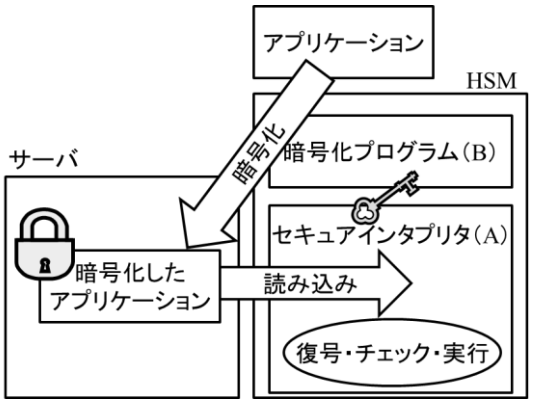

図 2　提案技術の構成例

A) HSM 内部で動作し, B が暗号化したアプリケーションを復号・チェック・実行する機能をもったインタプリタ（セキュアインタプリタ）
B) A のインタプリタが復号・チェック・実行できる形式で暗号化を行うプログラム（暗号化プログラム）

からなる．具体的な動作は,

- セキュアインタプリタを HSM 内部に構築する．
- HSM 内部で暗号鍵を生成し, その暗号鍵を用いて，暗号化プログラムでアプリケーションを HSM 内で暗号化する．
- 暗号化したアプリケーションを HSM 外部に格納する．
- 実行時には，実行したいアプリケーションを HSM 内に逐次読み込み，HSM 内のセキュアインタプリタが復号・チェック・実行する．

となる．

この技術により 1 章で述べた課題を以下のように解決できる．

- 暗号化したアプリケーションを HSM にデータとして読み込み，セキュアインタプリタが解釈・実行するため，動的解析を防ぐことができる．
- アプリケーションを暗号化した暗号鍵を HSM で保護するため, 時間をかけても攻撃は不可能である．仮に改ざんを受けてもアプリケーション実行時に命令を復号した後，セキュアインタプリタが実行前に命令の安全性をチェックすることにより，アプリケーションに混入されたマルウェア等, 悪意のあるルーチンを検知し, 実行を止めることができる．
- アプリケーションは HSM 外部に格納するため，更新・追加・削除が容易にできる．

4 提案技術の特徴

4.1 セキュアインタプリタによる実行を容易にする暗号化方法

インタプリタは字句解析，構文解析，意味解析，実行，の四つのフェーズで動作する．

インタプリタがソースコードを解析し、実行を行う必要があるが，本稿提案の方式では字句解析の前に逐次復号が行われるため、ヒント情報がないと，ソースコードの先読みができないため効率が悪いという課題がある．

そこで本方式のプロトタイプでは，容易にインタプリタの字句解析，構文解析が実装でき，実行時に効率化できるように，暗号時には①トークン単位で暗号化し，②インタプリタがソースコードを構文解析するときに，特に注意すべき要素（LISP における”(“, ”)”, ”.” など）はその他の要素と区別するラベル情報をつけて暗号化した．

今回の方式のトークンの暗号化は，AES で暗号化している．さらに同じ文字列でも異なった文字列に暗号化されるように，暗号化するときには，乱数でパディングを付けた上で暗号化している．また，パディングを付けた上で暗号化することにより，暗号結果の長さを一定にすることができるため，セキュアインタプリタは一定の長さで文字列を切り出して復号すれば直ちに字句解析をすることができる．

さらに，アプリケーションを暗号化する際には，セキュアインタプリタが字句解析を行うヒント情報として，各要素が何のラベルであるかを示すラベル情報のタグを付加して暗号化している．これにより，セキュアインタプリタが字句解析時にラベルを推論する時間を短縮している．

セキュアインタプリタは暗号化されたプログラムを以下の手順で実行する．

1. 暗号化プログラムをバッファに読み込む.
2. バッファに読み込んだ暗号化プログラム

のうち，制御構造に関する部分のみ（LISP における”(“，”)”，”.” など）を，ラベル情報を元にデコードする．こうすることで，セキュアインタプリタが字句解析，構文解析を行う時間を短縮している．

3. 実行する最小要素(一つの文)を切り出して，デコードする．
4. デコードした最小要素を実行する．
5. 以下 3.を繰り返し．

4.2 起動時間を短縮するためのアプリケーション分割暗号化

暗号化されたアプリケーションを HSM に読み込む際に，アプリケーション全体を一度に読み込み，復号・チェック・実行すると，アプリケーションのサイズによっては，起動までのオーバーヘッドが大きくなり，待ち時間が発生することが考えられる．そのため，アプリケーションを暗号化する際に，分割して暗号化し（分割したアプリケーションをスクリプトと呼ぶこととする），セキュアインタプリタへは実行する部分のスクリプトのみを逐次読み込ませることを考えた．

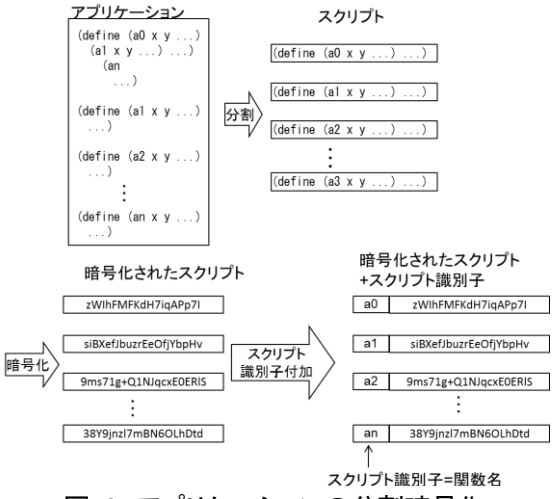

図 3 アプリケーションの分割暗号化

行番号があるプログラミング言語の場合は行単位で，行番号がないプログラミング言語の場合は関数単位でスクリプトに分割する．分割されて暗号化されたスクリプトがどの行・関数であるか識別できるよう，スクリプトを暗号化後に行番号・関数名を表すスクリプト識別子を付加する．セキュアインタプリタで実行中のスクリプトに行・関数の呼び出しがあった際には，タグを見て必要な行・関数をセキュアインタプリタに読み込む．

5 提案技術の課題

提案技術を実装するには以下が課題となる．

A) 暗号化したアプリケーションをインタプリタが逐次復号して実行することで性能が劣化する可能性がある点．
B) HSM と連携した際のオーバーヘッドにより性能が劣化する可能性がある点
C) HSM による暗号鍵管理の方法を検討する必要がある点．

このうち，A について，プロトタイプを作成して検証を行った．

6 プロトタイプによる性能測定

6.1 プロトタイプの実装

HSM 上で動作させる前段階として, セキュアインタプリタと暗号化プログラムのプロトタイプを作成した．

インタプリタの規模が小さく，容易にインタプリタが作成可能である点，暗号ライブラリが存在する点から，lisp 系プログラミング言語 Scheme の実装の一つである Racket を選択し，Racket インタプリタ上で動作する，最小限の命令セットを備えたセキュアインタプリタのプロトタイプ「microScheme」と microScheme が復号して実行できるようにアプリケーションを暗号化するための暗号化プログラムを作成した．暗号化プログラムは Scheme で書かれたアプリケーションを字句単位で AES128 を用いて暗号化する．

microScheme は，暗号化プログラムで暗号化されたアプリケーションを読み込み，逐次

復号して実行する．復号した関数の定義はメモリに読み込まれ，再帰呼び出しされた際には再び復号する必要はない．今回は，実際に暗号化されたアプリケーションを，実行時にセキュアインタプリタが復号して実行できることを確認すること，アプリケーションを暗号化することによるオーバーヘッドが処理性能へ与える影響の傾向を確認すること，が目的である．

暗号ライブラリは OpenSSL を用いており，鍵の生成や暗号演算は通常のサーバ上で行っており，HSM で保護することは行っていない．また，変数の暗号化は行っていない．

6.2 測定方法

5 章の課題 A を評価するため，アプリケーションを逐次復号し、実行することによるオーバーヘッドの傾向を測定した。アプリケーションを逐次復号することによるオーバーヘッドは復号するスクリプトのバイト数と処理内容に依存するため，一度の復号あたりの処理の重さが異なる複数のアプリケーションを用意し，オーバーヘッドの傾向を調べた．具体的には、フィボナッチ数列の 1 番目，5 番目，10 番目，15 番目を計算する 4 つのアプリケーションを microScheme(アプリケーションの暗号化あり)，microScheme(アプリケーションの暗号化なし)で記述し，処理にかかる時間を比較した．それぞれ 10 回測定し，平均値を比較した．フィボナッチ数列を一度計算する際に復号されるスクリプトのバイト数は 4 つのアプリケーションで一定であるため，復号にかかる時間は一定である．また，どのアプリケーションも一度復号されると関数の定義がメモリに読み込まれるため，再帰呼び出しされた際に再び復号処理が実行されることはない．

6.3 結果

表 1 に示す通り，フィボナッチ数列の 1 番目，5 番目，10 番目，15 番目の計算にかかる処理時間の，暗号化ありと暗号化なしの差は，それぞれ，4 ミリ秒，4 ミリ秒，3 ミリ秒，5 ミリ秒であり，処理時間の，暗号化ありと暗号化なしの倍率はそれぞれ約 3.0 倍，2.0 倍，1.2 倍，1.0 倍であった．また，フィボナッチ数列の 1 番目，5 番目，10 番目，15 番目の計算時間に占める復号にかかる時間の割合はそれぞれ，67%，50%，16%，3%であった．

表 1 測定結果

n番目	1	5	10	15
差(暗号化あり-暗号化なし)(ミリ秒)	4	4	3	5
倍率(暗号化あり/暗号化なし)	3.0	2.0	1.2	1.0
実行時間に占める復号の時間(%)	67	50	16	3

6.4 考察

結果を見ると，暗号化されたアプリケーションを実行時に復号することによるオーバーヘッドは，再帰呼び出しが少ない場合，3 倍程度になるが，フィボナッチ数列の 1 番目，5 番目，10 番目，15 番目と，再帰呼び出しの回数が増えるほど，実行時間に占める復号の時間の比率が低くなる．したがって，スクリプト中の再帰呼び出しが入れ子になり深くなっている場合や，スクリプトの命令の実行に時間がかかる場合など，復号するスクリプトのバイト数当たりの処理内容が大きい場合は復号によるオーバーヘッドは無視できるほどであるが，スクリプトのバイト数当たりの処理内容が小さい場合は無視できなくなる．

したがって，暗号化されたアプリケーションを逐次復号するオーバーヘッドによる性能劣化が実用上問題ないかどうかは，適用するアプリケーションの特性（等）を考慮して判断する必要がある．例えば，再帰呼び出しを多用しているアプリケーションならば性能劣化は抑えられるので適用可能だが，多種の命令を次々呼び出すようなアプリケーションならばオーバーヘッドが大きいと考えられるので，注意する必要がある．

7 おわりに

本稿では，IC カードエージェントのようなサーバ上で動作し，機密情報を扱うソフトウェアを保護する技術として，ハードウェアと連携したセキュアインタプリタ技術を提案した．また，提案した技術のプロトタイプを作成して，性能測定を行った．

今後の課題として，今回のプロトタイプでは, HSM を使用していないため, 実際に HSM と連携した際の性能面での影響・暗号鍵管理の方法を検討する必要がある．

また，プロトタイプでは，アプリケーション暗号化の際の分割の粒度は字句単位としているが，今後，HSM への逐次読み込みにおける処理速度を考慮し，粒度の検討を行う必要がある．

さらに，処理速度やプログラム作成の難易度を考慮し，インタプリタのプログラミング言語やプログラム構造，実装するコマンド種類の検討を行う必要がある．

参考文献

[1] 高度情報通信ネットワーク社会推進戦略本部, “行政キオスク端末のサービス拡大のためのロードマップ,” 2011.

[2] 高度情報通信ネットワーク社会推進戦略本部, “新たなオンライン利用に関する計画,” 2011.

[3] 情報セキュリティ政策会議, “政府機関の情報セキュリティ対策のための統一技術基準,” 2011.

[4] 情報セキュリティ政策会議, “政府機関の情報セキュリティ対策のための統一管理基準,” 2011.

[5] 小野寺匠, “セキュリティ対策の要点解説,” http://technet.microsoft.com/ja-jp/librali/dd362910.aspx.

[6] 山内寛己, 門田暁人, 松本健一, “実行系列差分攻撃によるプログラム耐タンパー性評価,” 2009.