社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2005－ARC－165 (5)
2005／11／30

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開発

城本正尋† 田端猛一† 酒井智也†
島田貴史† 北村俊明†

近年，プログラムの不正利用が問題となっており，プログラムの著作権を保護するための研究が各種行われている．そのなかで，プログラムを暗号化し，復号をプロセッサチップ内で行うことで，プログラムをリバースエンジニアリングから保護する機能を持ったセキュアプロセッサが提案されている．本稿ではセキュアプロセッサにおいて，暗号化されたプログラムとそれを復号する鍵との対応付けに，仮想記憶の枠組を利用したプログラム保護システムを提案する．本システムでは，プログラムはページ単位で復号する鍵と対応付けられるため，1つのプロセスの中に，異なる鍵で暗号化されたプログラムを存在させることができる．また，提案システムのフィージビリティ・スタディを行った．

Development of Processor that has Program Protection Feature by use of Public Key Cryptosystem

Masahiro Shiromoto,† Takekazu Tabata,† Tomoya Sakai,†
Takashi Shimada† and Toshiaki Kitamura†

Recently, the program abuse is becoming an issue, therefore there are various researches to copyright protection of programs. Among those researches, secure processor achieves program protection feature, by the encrypted programs are stored on main memory and programs are decrypted inside processor. In this report, we propose the program protection system which has the mechanism of correspondence between the program and the decryption key using virtual memory system in secure processor. In this system, the decryption key is specied by page frame basis, and one process can have the programs that encrypted by different keys. And we conduct a feasibility study on proposed program protection system.

1. はじめに

近年，家電製品や通信機器などにプロセッサが搭載され，プログラムによって装置制御が行われる場面が増えている．制御をハードウェアでの実現する場合に比べて，プロセッサを内蔵しプログラムによって実現することで，機能の変更・追加・修正は容易になり，製品の開発サイクルを大幅に短縮できるメリットがある．しかし，システム開発者にとってプログラムによる制御を行うことは，リバースエンジニアリングによって重要なプログラムを不正に使用される危険性を含んでいる．

一部の特殊なアルゴリズムについては，特許などの手段で保護することも可能である．しかし，組み込みシステムで使用されるプログラムには，制御対象となる装置の仕様から簡単に導き出せる論理・手順だけでなく，より良い制御を行うために各種のノウハウが使われている．たとえば，プリンタの発色管理では，単に出力する画素の色情報そのもので制御するのではなく，印刷する紙の種類になどに応じて微妙にカラーバランスを変化させるなどの工夫がなされている．このようなノウハウは，公表すること自体が権利の消滅につながり，特許で保護することが難しい．

プログラムの著作権を保護する研究は様々な形で行われており，一例としては，プログラムを難読化することによってリバースエンジニアリングにかかるコストを増大させる手法などがある．また，特に秘密にしたい重要なプログラムに対しては暗号化を行い，逆アセンブルを不可能にする手法もある．ただし，プログラムを処理するためには，暗号化されたプログラムを復号する必要があるので，復号された瞬間のプログラムを，リバースエンジニアリングをしようとするユーザに取得される可能性がある．そこで，プロセッサ内部でプログラムの復号を行うことで，プロセッサチップ内にのみ復号された命令が存在し，命令が転送されるバスをロジックアナライザなどでハードウェア的に観測するような攻撃からもプログラムを保護する機能を有したプロセッサであるセキュアプロセッサが提案されている．[1)~3)]

† 広島市立大学
Hiroshima City University

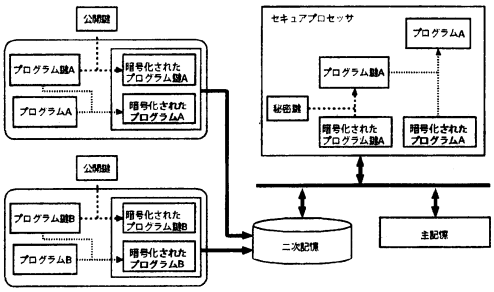

図 1 プログラムの暗号化と復号の概略図

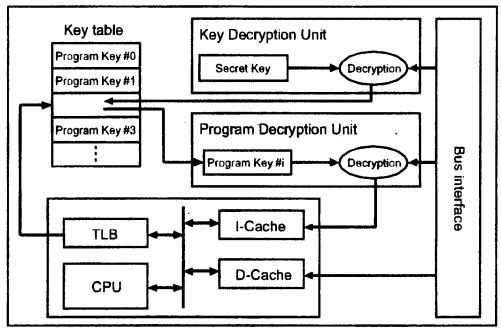

図 2 提案するプロセッサのブロック図

セキュアプロセッサでは，システム上に存在する複数の実行プログラムはそれぞれ異なる時期にコンパイルされる可能性が高く，異なる鍵で暗号化できるという自由度を持つことが望ましい．このためには，処理するプログラムに対応した鍵を用意する機構が必要となるが，本研究では，その機構として仮想記憶の枠組を利用したセキュアプロセッサを提案する．

以下本稿では，2 章で現在までに研究されている他のセキュアプロセッサについて紹介する．3 章では，本研究で提案するプロセッサを用いたプログラム保護システムについて説明する．4 章では，提案システムのフィージビリティ・スタディとしてセキュリティ機能を実現するために必要となるチップ面積や性能のトレードオフを調査し考察を行う．5 章では，まとめを述べる．

2. セキュアプロセッサ

セキュアプロセッサにおいてプログラムの暗号化には，暗号化と復号に異なる鍵を用いる非対称鍵暗号（公開鍵暗号）が用いられる．2 つの鍵のうち 1 つはプログラムを暗号化するための鍵としてソフトウェア開発者やベンダに公開する．もう 1 つの鍵は，復号のための鍵としてプロセッサ内に秘密に保持する．以上のようにして，プログラムの暗号化は誰でも行えるが，復号は秘密鍵を持ったプロセッサのみが可能となり，プログラム保護機能を有したままソフトウェア開発は自由に可能な環境が構築できる．

公開鍵暗号は，暗号化と復号に同じ鍵を用いる対称鍵暗号（共通鍵暗号）に比べて非常に計算量が多いため，暗号化と復号には膨大な処理時間を要する．したがって，多くのセキュアプロセッサでは，実行プログラムはいったん共通鍵暗号によって暗号化し，その暗号化に用いた共通鍵（プログラム鍵）を公開鍵暗号によって暗号化するハイブリッド方式を採用している．これにより，プログラム鍵を安全にプロセッサに渡すことができ，プログラム復号の処理時間を抑えることができる．セキュアプロセッサにおいてのプログラムの暗号化と復号の概略図を図 1 に示す．

セキュアプロセッサでは，図 1 に示すように，システム上に存在する実行プログラムはそれぞれ異なるプログラム鍵で暗号化されているため，復号時には処理するプログラムに対応したプログラム鍵を用意する必要がある．XOM をはじめとするセキュアプロセッサでは，プロセス毎に ID を割り当て，ID とプロセスを復号するプログラム鍵を対応付けている．この方法の場合，シェアードライブラリの実現のため複数のプロセスから共有されるライブラリプログラムは，様々なプロセス ID で処理されるので暗号化することができない．そこで本研究では，実行プログラムとプログラム鍵との対応付けに仮想記憶を用いたシステムを提案する．プログラムはページ単位でプログラム鍵を対応付けられるため，1 つのプロセスの中に，異なるプログラム鍵で暗号化されたプログラムも存在させることができる．

本研究では，プロセッサは内部にプログラム鍵を保存するテーブルを持ち，そのテーブルの管理は OS が仮想記憶の枠組の中で行う．ただし，OS が知ることができるのは，システム上に存在するプログラムのプログラム鍵がテーブルのどのインデックスに格納されているかという情報だけであり，鍵自体は知ることができないようすることで，プログラム鍵を盗まれることがないようにしている．本研究で提案するプロセッサのブロック図を図 2 に示す．

先行研究では，データに対する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は暗号化を行い，そしてデータのハッシュ値と共に格納する．[1] ハッシュ値を付加することで改ざんの検出も行っており，さらに Suh らの提案する AEGIS では，そのハッシュ値をツリー構造で効率的に管理している．[2] また，割り込み発生時に主記憶に退避されるコンテキストも改ざんを防ぐために暗号化されるが，橋本，春木らの提案する L-MSP (License-controlling Multi-vendor Secure Processor) では，XOM がレジスタ個別に暗号化を行っているのに対し，コンテキストを主記憶へ格納する際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.[3)]

本稿では，プログラム内の重要なアルゴリズムを保護するという観点に重点をおいて，実行プログラムのみの暗号化を行っており，プログラムの処理対象となるデータの暗号化は対象としていない．入力データと出力データが観測されることで内部アルゴリズムを推定される危険性はあるが，保護するアルゴリズムが十分複雑であれば，入力データと出力データの組合せから内部アルゴリズムを同定することは十分に困難であろうと考えている．

3. 本研究のプログラム保護システムの概要

3.1 プログラム鍵の復号

プロセッサが暗号化された実行プログラムを処理するには，その実行プログラムのプログラム鍵をあらかじめ復号する必要がある．そのため本研究では，暗号化されたプログラム鍵の復号を行う命令 (KEYDEC) を用意した．KEYDEC 命令は，OS が実行プログラムを二次記憶から主記憶へのローディング処理の間に発行される．暗号化されたプログラム鍵は，実行プログラム内にあるプログラム鍵の情報を示すセグメントから取り出され，いったん主記憶に保持される．その後発行された KEYDEC 命令によって，暗号化されたプログラム鍵をプロセッサ内に取り込み，復号を行う．復号されたプログラム鍵はプログラムの復号時に使用できるように，プロセッサ内部に保存しておく．

3.2 プログラム鍵の管理機構

システム上には複数の実行プログラムがローディングされるが，それぞれは，異なったプログラム鍵を用いて復号されなければならない．このためには，以下に示す点について対応する機構が必要である．

(1) 複数のプログラム鍵を蓄えておくプログラム鍵のレジスタスタック
(2) 実行プログラムを復号する時に，どのプログラム鍵を用いて復号すべきか選択する手段
(3) プログラム鍵のレジスタスタックの管理

(1) に対しては，復号されたプログラム鍵をプロセッサから取り出して扱うことはプログラム鍵を盗まれる可能性があるため，プロセッサ内にプログラム鍵を格納するレジスタスタック（鍵テーブル）を用意した．鍵テーブル内の各プログラム鍵に対しての操作は OS からインデックスでハンドリングすることとした．

(2) において，ローディングされた実行プログラムが実記憶上のどのアドレスを占めるかは OS のみが知り得る情報である．アドレス区間とプログラム鍵の対応関係表をプロセッサ内にハードウェアとして持ったとしても検索に時間がかかるので，仮想記憶の枠組みを利用することとした．すなわち，復号される可能性のある実行プログラムは，実記憶上に存在するので，OS がアドレス変換表や TLB に，各実ページごとに復号に使うプログラム鍵のインデックスを格納することとした．これにより，同一ページ内に複数の実行プログラムをローディングすることはできないが，大きな制約とはならないと考えている．

(3) は上述の鍵テーブルはその個数が有限であるので，プログラムの実行開始や終了に同期して入れ替えを行う必要がある．これは，ハードウェアでは知ることができないので，OS の助けが必要である．また，鍵テーブル実現のためのハードウェア量の削減のため，復号したプログラム鍵の鍵テーブルへの割り付け管理も OS が行うことにした．OS はプログラム鍵の復号時に格納する鍵テーブルのインデックスを KEYDEC 命令によって指定する．ここで，鍵テーブルに格納できるプログラム鍵の個数が問題となるが，同時に実行状態となるプログラムは，高々十数個であろうと想定しているが，この個数のトレードオフは，これからの課題である．

実行プログラムがローディングされ実行されるまでの動作を，ハードウェアテーブルウォーク型の仮想記憶システムを例にとって具体的に説明する．なお，ソフトウェアテーブルウォーク型の TLB の場合でもほぼ同様に実現できる．最初に二次記憶上の実行プログラム内から暗号化されたプログラム鍵を実記憶上の特定のアドレスに格納する．実行プログラムが二次記憶から実記憶にローディングされている間に，プロセッサは KEYDEC 命令によって復号回路でプログラム鍵の復号を行う．復号されたプログラム鍵は，KEYDEC 命令内で指定される鍵テーブルのインデックスに格納される．OS はアドレス変換表に，ローディングした実行プログラムの仮想アドレス空間を登録する際に，その実行プログラムを復号するプログラム鍵の鍵テーブル上のインデックスも登録する．実行プログラムが終了すると OS は，対応するプログラム鍵のインデックスを開放する．

プロセッサが最初に実行プログラムを処理する時には，TLB に変換対がまだ登録されていないので，TLB ミスが起こる．そこで新たな変換対をアドレス変換表を参照し登録する．その際に，TLB に登録する実ページの実行プログラムに対応するプログラム鍵のインデックスも共に登録される．次に TLB によるアドレス変換を行った時には，プロセッサは実行プログラムを復号するために必要なプログラム鍵のインデックスを知ることができる．その後キャッシュミスが起こった場合には，TLB で得られるプログラム鍵のインデックスをもとに鍵テーブルを参照し該当するプログラム鍵を取り出す．そのプログラム鍵を用いて主記憶から読み込まれる暗号化された命令に対して復号を行い，命令キャッシュに格納する．以上のキャッシュミス時にプロセッサ内で行われる処理を図 3 に示す．キャッシュにヒットした場合は，既に復号された命令がキャッシュ上に存在しているので，復号処理は必要ない．

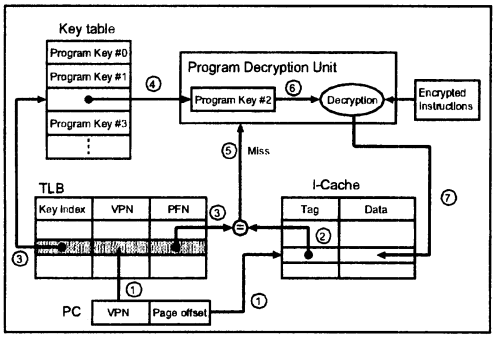

図 3 キャッシュミス時のプロセッサ内部処理

3.3 シェアードライブラリへの対応

本研究のようにプログラムが暗号化されている場合，命令の書き換えが行われると，書き換える命令の暗号化が必要となるため，対応を考えなくてはならない．命令書き換えが必要となるケースは，シェアードライブラリなどの動的リンクの場合である．すなわち，多くのアプリケーションプログラムでは，シェアードライブラリと呼ばれる，多くのプログラムで利用されるような汎用的な機能をまとめたライブラリをプログラム実行時にリンクする場合がある．シェアードライブラリの動的リンクは，プログラム中のシステムコールによって割り込みを発生させ，主記憶上のシェアードライブラリの対応エントリアドレスを得て，そのアドレスへの分岐命令に置き換えることによって実現される．本節では，命令の書き換えを行わずにシェアードライブラリの機能を用いる手法について述べる．

まず，単純な手法としては，必要となるシェアードライブラリの機能をリンクしてから，コンパイルして実行プログラムを作り，暗号化する方法が挙げられる．このように静的なリンクによる手法では，動的なリンクのメリットであった実行プログラムのサイズの縮小といった効果は得られなくなるが，簡単に実現できるため，リンクするプログラムサイズが小さい場合には有効である．

次に挙げる手法は，動的リンク時に割り込みを発生させるシステムコール命令に条件によっては分岐命令の機能を持たせた命令を実装する方法である．当命令は，デコード時にオペランドを参照する．オペランドには，命令の機能を決定するフラグを用意し，その内容によって割り込みと分岐の機能を使い分ける仕様にする．この手法では，機構を実現するためのハードウェアが必要となるが，従来通りの動的リンクが行える．

3.4 プロセッサの起動

本システム上では，高い安全性を実現するため全てのプログラムの暗号化を行っているので，プロセッサ起動時に処理される boot プログラムも暗号化する必要がある．したがって，プロセッサは起動してから最初に，KEYDEC 命令によって boot プログラムのプログラム鍵を復号する必要があるが，その KEYDEC 命令自体も暗号化しなければならない．そのため本研究のプロセッサでは，起動時に自動的に設定されるプログラム鍵 (boot_Key) を用意し，その鍵を用いて最初の KEYDEC 命令を暗号化することとした．

また本研究ではプログラムとプログラム鍵との対応付けに仮想記憶を利用しているので，OS が仮想記憶モードに移行するまでに処理される OS の初期化プログラムなどは 1 つの鍵で処理されることとした．ただし，実アドレスで処理されている時にも，実アドレスでも TLB をアクセスするようにしておき，TLB にアドレスとプログラム鍵のインデックスを登録すれば，複数の実行プログラムに対応することも可能であると考えられる．

4. フィージビリティ・スタディ

4.1 共通鍵暗号の復号回路について

4.1.1 AES

共通鍵暗号はプログラムの暗号化に用いられているため，復号のレイテンシは，性能低下に大きく影響する．しかし，復号のレイテンシが短くても DES (Data Encryption Standard) のように効果的な解読攻撃が存在するような暗号方式ではセキュリティ面の問題がある．本研究では，プログラムを暗号化する共通鍵暗号に，暗号強度と処理速度のバランスのとれた暗号方式として AES (Advanced Encryption Standard) を採用した．[4] AES は NIST (National Institute of Standard and Tecnology) によって，DES に代わる次世代の標準暗号として選定されている．

AES では，暗号化および復号を行うデータのブロックサイズを 128 [bit] としている．プログラムは 128 ビット毎に分割され，それぞれ暗号化（復号）が行われる．AES の暗号化は AddRoundKey, ShiftRows, SubBytes, MixColumns，復号は AddRoundKey, InvShiftRows, InvSubBytes, InvMixColumns のそれぞれ 4 つのステージで構成されるラウンド関数を定められた回数繰り返すことで実現される．繰り返されるラウンド数は，鍵の長さによって決まり，鍵の長さが 128 ビットの場合のラウンド数は 10 回となっており，192 ビットの場合は 12 回，256 ビットの場合は 14 回となっている．本研究では復号のレイテンシを抑えるため，プログラムの暗号化（復号）鍵を 128 ビットとしたので，暗号化（復号）の際のラウンド数は 10 回となる．

また AES では，鍵拡大処理 (Key expansion) を行い暗号化（復号）鍵から各ラウンドの AddRoundKey の処理に必要なラウンド鍵を生成する．ラウンド鍵は全部でラウンド数＋ 1 個必要であり，本研究では 11 個のラウンド鍵を生成する必要がある．

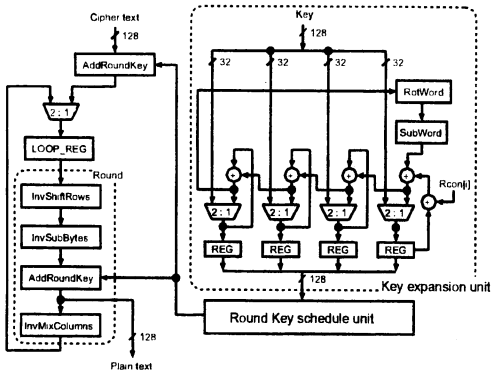

図 4　AES 復号回路の構成

4.1.2　AES 復号回路の設計

本研究で設計した AES 復号回路の構成を図 4 に示す．設計した回路は，1 サイクルで 1 ラウンドの処理を行うので，復号のレイテンシは 10 サイクルとなる．回路に入力された暗号文は，初期処理の AddRoundKey を行ってループレジスタに格納される．その後，10 回のラウンド処理が終わると復号結果が出力される．鍵拡大処理部では，キャッシュミスが発生した時点で復号に必要なプログラム鍵を入力として受け取り，メモリアクセスと並行して鍵拡大処理が行われる．鍵拡大処理を行う回路は 1 サイクルで 1 個のラウンド鍵を生成し，11 サイクルで復号に必要なラウンド鍵を用意することができる．キャッシュミス時のメモリアクセスタイムは 11 サイクルよりは長いと考えられるので，命令がプロセッサに到達する前に，ラウンド鍵を用意することができる．鍵スケジュール部では，鍵拡大処理によって生成されたラウンド鍵を格納し，復号処理時に，各ラウンドの AddRoundKey に必要なラウンド鍵を用意する．

図 4 の回路を Verilog-HDL で記述し，Synopsys 社 Design Compiler で HITACHI 0.18 μm プロセス用に京都大学で作成されたスタンダードセルライブラリを用いて論理合成を行った．論理合成時の遅延制約は 5.0 [ns]（動作周波数 200 [MHz]）とした．その後，Synopsys 社 Apollo を使用して配置配線を行った．設計した回路の面積を表 1 に示す．また，参考として同じく 0.18μm プロセスで設計された ARM 社のプロセッサ ARM920T と ARM922T の面積も表 1 に示す．[5),6)] 動作周波数は共に 200 [MHz] であるが，ARM920T のキャッシュサイズは 16K/16K バイトで，ARM922T は 8K/8K バイトとなっている．

4.1.3　性能への影響

AES 復号は，キャッシュミス時に行われるため，復号のレイテンシは，キャッシュミス時のペナルティに加算される．一般的な組み込みプロセッサとして，Intel 社の PXA255 のキャッシュミスペナルティは 60 サイクルとなっている．[7)] PXA255 の動作周波数は 400 [MHz] であるので，動作周波数が 200 [MHz] の場合のキャッシュミスペナルティを 30 サイクルと仮定すると，AES の復号処理は 10 サイクルであるので，キャッシュミスペナルティは 40 サイクルに増加する．キャッシュミスペナルティは 3 割以上増加するが，充分実用の範囲であると判断した．実際のアプリケーションの実行時間の増加は命令キャッシュのヒット率にも影響するので，今後ソフトウェアシミュレータによって各種アプリケーションの実行時間を見積り，性能に対する影響を詳細に評価していきたい．

表 1　AES 復号回路各ユニットの面積

	面積 $[mm^2]$
鍵拡大処理	0.15
鍵スケジュール部	0.35
AES 復号部	0.52
ARM920T (16K/16K cache)	11.8
ARM922T (8K/8K cache)	8.1

4.1.4　面積への影響

ARM920T と ARM922T には 16K バイトのキャッシュサイズの差があり，3.7 $[mm^2]$ の面積の差がある．ここでそれを面積と性能のおおまかな指標として用いると，今回設計した回路面積の合計は 1.02 $[mm^2]$ であり，設計した AES 復号回路をプロセッサに搭載するには，4.4K バイトのキャッシュサイズの縮小と引き換えに実現することができると考えられる．また ARM920T に設計した回路を搭載したとすると，回路面積は約 9%増加し，ARM922T の場合は約 13%の増加となる．ただし，今回の設計では，鍵スケジュール部の回路面積の大部分を占めているラウンド鍵を保存するレジスタファイルが，簡単のためフリップフロップセルでの実現となっているため，今後，6T-SRAM セルなどで実現されたレジスタファイルでの適切な評価を行う必要がある．

より小面積な AES 復号回路を実現を目指すため，AES 復号部の各ステージを個別に合成し回路面積を求めた．結果を表 2 に示す．ただし，InvShiftRows は配線のつなぎかえで実現しているため評価は行っていない．表 2 から，InvSubBytes が AES 復号回路中でも回路面積の大部分を占めている．InvSubBytes は，16 個の変換テーブルから構成される．変換テーブルは 8 ビット入出力の 1 対 1 写像となっており，今回の設計では Verilog-HDL の case 文で記述し，自動合成で最適化することによって実現している．森岡，佐藤の研究によれば，変換テーブルを BDD (Binary Decision Diagram) を用いて実現すると，消費電力は増えるが，自動合成で得られる回路と同程度の遅延時間で，回路面積を削減できることが示されている．[8)] BDD による変換テーブルは，鍵拡大処理内の SubWord にも適用することができるため，回路面積縮小への効果は高

表 2 AES 復号回路の各ステージの面積

	面積 $[mm^2]$	遅延 [ns]
AddRoundKey	0.01	0.5
InvSubBytes	0.36	3.2
InvMixColumns	0.10	2.5

いと考えられる.

4.2 公開鍵暗号の復号回路について

本研究において公開鍵暗号の復号は，KEYDEC 命令処理時に実行される．したがって，1 つの実行プログラムに対して 1 回だけ行われるので，プログラム実行途中の性能には影響しない．また，公開鍵暗号の復号は，実行プログラムが二次記憶からメモリへローディングされる処理と並行して行われるので，復号にかかる処理時間をある程度までは隠蔽できると考えられる．公開鍵暗号の復号回路では，実行プログラムのローディングで十分に隠蔽できる程度の復号のレイテンシに留め，回路規模の小面積化に重点を置いた設計とすることで影響を最小化できる．

インターネットなどで広く実用化されている公開鍵暗号として RSA がある.[9] RSA の暗号化/復号で行われる処理はべき剰余演算である．現在の計算機の能力で解読されない最低限の安全性を RSA に持たせるためには，公開鍵 e と秘密鍵 d の長さを 1024 ビットとする必要がある．暗号化は 1024 ビットの整数 n を用いて，$Ctext = (Ptext)^e \, mod(n)$ のべき剰余演算を行う．暗号文 $Ctext$ は e の長さと等しくなる．復号では，$Ptext = (Ctext)^d \, mod(n)$ のべき剰余演算を行う．1 回のべき乗演算は，$A \cdot B \, mod(n)$ のように表される剰余乗算となるので，RSA 復号回路の性能は剰余乗算回路の実現方法により決まる．本稿では，RSA 復号回路は性能には大きな影響がないと判断したことで，詳細な設計を行うことはしなかったが，本システムの実現に最適な剰余乗算回路の選定，RSA 復号回路の設計と評価は今後の課題としたい．

5. おわりに

本稿では，暗号化されたプログラムと，それを復号するプログラム鍵との対応付けに仮想記憶の枠組を利用したプログラム保護システムを提案した．本システム上に存在するプログラムは，ページ単位でプログラム鍵と対応付けられているため，1 つのプロセスの中に，異なるプログラム鍵で暗号化されたプログラムが存在することができる．これにより，複数のプログラムで共有されるようなライブラリプログラムでも暗号化することができる．

また，提案システムのフィージビリティ・スタディとして共通鍵暗号 AES の復号回路を設計し，性能やチップ面積に与える影響を調査した．今後は，ソフトウェアシミュレータによる性能への影響の評価と，AES 復号回路と RSA 復号回路の詳細な設計を行いたい．

今回提案したシステムではデータの暗号化は考えていないが，リバースエンジニアリングをしようとするユーザが，割り込み発生時に退避されるコンテキストを改ざんすることで，1 命令実行毎の入出力データを解析して機械語命令を 1 命令ずつ同定していく可能性は否定できないが，プログラム全域に渡って，上記の解析行為を行うことは，暗号解読に匹敵する時間が必要になると考えている．今後，解析行為に要する時間に対して検討を行い，十分に現実的な時間内で解析行為が可能であるならば，コンテキスト保護を含むデータの保護への対応を考慮に入れたい．現在のシステムの上で，データの暗号化機構の実現を制約するものはないので，データ側のキャッシュと主記憶間に暗号化回路・復号回路を挿入し，性能やチップ面積に与える影響を検討したい．

謝辞 本研究は，東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社ならびに株式会社日立製作所の協力で行われたものである．また本研究の一部は，文部化学省科学研究費補助金基盤研究 (C) 課題番号 17500044 の支援により行った．

参 考 文 献

1) D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell and M. Horowitz: Architectural Support for Copy and Tamper Resistant Software, *ASPLOS-IX*, pp. 168–177 (2000).
2) G. E. Suh, D. Clarke, B. Gassend, M. van Dijk and S. Devadas: AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing, *ICS03* (2003).
3) 橋本幹生, 春木洋美: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, Vol.45, No.SIG 3 (ACS 5), pp.1–10 (2004).
4) National Institute of Standard and Tecnology: Announcing the ADVANCED ENCRYPTION STANDARD (AES), FIPS PUB 197 (2001).
5) ARM Ltd.: ARM920T, http://www.arm.com/products/CPUs/ARM920T.html.
6) ARM Ltd.: ARM922T, http://www.arm.com/products/CPUs/ARM922T.html.
7) *Intel PXA255 Applications Processors Developer's Manual.*
8) 森岡澄夫, 佐藤証:共通鍵暗号 AES の低消費電力論理回路構成法, 情報処理学会論文誌, Vol.44, No.5, pp.1321–1328 (2003).
9) R. L. Rivest, A. Shamir and L. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Communication of the ACM*, Vol.21, No.2, pp.120–126 (1978).