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提案

城本正尋[†,☆] 田端猛一[†] 酒井智也[†]
島田貴史[†] 窪田昌史[††]
川端英之[††] 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．その中で，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，仮想記憶の枠組みを利用したセキュアプロセッサを提案する．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．

A Processor with Program Protection Feature by Use of Public Key Cryptosystem

Masahiro Shiromoto,[†,☆] Takekazu Tabata,[†] Tomoya Sakai,[†]
Takashi Shimada,[†] Atsushi Kubota,[††] Hideyuki Kawabata[††]
and Toshiaki Kitamura[††]

Recently, software piracy is becoming an issue, therefore there are various researches to copyright protection of programs. Among those researches, the secure processor achieves program protection feature, by encrypting programs on the main memory and decrypting programs inside the processor. In this paper, we propose the program protection system which corresponds between the program and the decryption key by using a virtual memory mechanism of the secure processor. In this system, the decryption key is specified by a page frame basis, and one process can have the programs that encrypted by different keys. And we conduct a feasibility study on the 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合が増えている．制御をハードウェアで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

† 広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University
†† 広島市立大学情報科学部
Faculty of Information Sciences, Hiroshima City University
☆ 現在，ソニーエルエスアイデザイン株式会社
Presently with Sony LSI Design Inc.

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),13)]．

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個別に開発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み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だれでも行えるが，復号は秘密鍵を持ったプロセッサのみが可能である環境が構築でき，プログラム保護機能の実現と，公開された環境でソフトウェア開発を行えるという自由度を両立させることができる．

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．

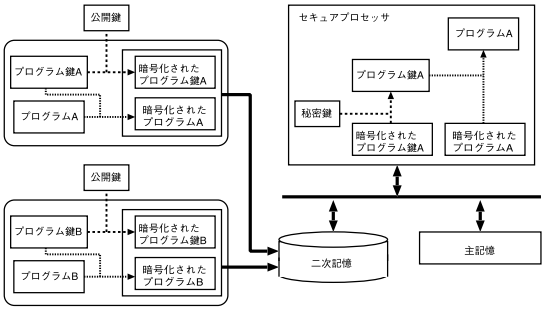

図 1 プログラムの暗号化と復号の概略図
Fig. 1 An outline of encryption and decryption of programs.

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．先行研究で提案されているセキュアプロセッサでは，プロセスごとに ID を割り当て，ID とプロセスで使用するプログラム鍵を対応付けている．この方法の場合，共有ライブラリのように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の枠組み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と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．

本提案では，プロセッサ内部に，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 が仮想記憶の枠組みの中で行う．この処理で必要となるのは，システム上に存在するプログラムを復号する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，鍵自体を知る必要はない．このため，本提案では，テーブルを直接アクセスする機能を装備せず，システム機能が乗っ取られてもプログラム鍵が盗まれないようにすることを考えた．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）

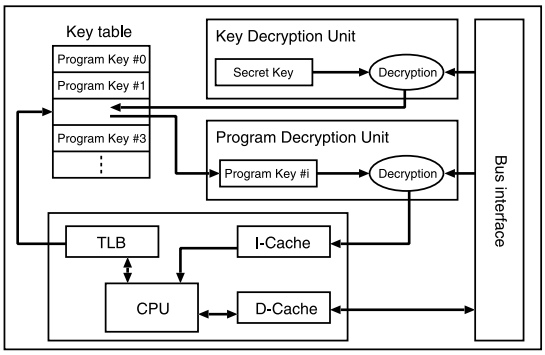

図 2 提案するプロセッサのブロック図
Fig. 2 The block diagram of the proposed processor.

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ともに格納する[1]．ハッシュ値を付加することで格納したデータの改ざん検出も行っており，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]．

本提案と，XOM をはじめとする提案との違いは，本提案では，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ことである．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．

もう 1 点，暗号化を命令に限定している理由は，データも暗号化すると，主記憶からのデータの取り出し時に復号を行い，それをまた暗号化して主記憶に書き戻す経路が存在してしまうためである．この経路を悪用すれば，解読対象のプログラムをデータとして読むことができれば，解読することができる．XOM では，コンパートメントという境界を主記憶に設けて他のプログラムの領域を読めないようにしているが，データの共有という点で，コンパートメントをまたぐロード/ストア命令も用意されている．システム機能を乗っ取られると，コンパートメントを偽ったり，これらの命令を用いたりして復号される可能性がある．このために，本提案では，あえてデータの暗号化を行わなかった．

また，Kawabata らの提案するセキュリティを強化した MeP プロセッサ[13] では，本提案と同様に，命令だけの暗号化を行っているが，プロセッサ内部に用意したローカルメモリにユーザが明示的にプログラムをローディングしてから実行することを前提としており，復号は，この転送時に行われる．このような明示的な転送指示は，ユーザの負担となると考えられる．

3. 本提案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．そのため本提案では，暗号化されたプログラム鍵をオペランドとして復号を行う命令（KEYDEC）を用意した．暗号化されたプログラム鍵は，実行プログラムの一部として保持されているとする．これは，たとえばオブジェクトファイルが ELF 形式であれば，暗号化されたプログラム鍵のセグメントを追加することで実現できる．実行プログラムのローディング処理は，通常のローディング処理に加えて，プログラム鍵のローディング処理が必要となる．これは，OS が暗号化されたプログラム鍵を，実行プログラムから主記憶に取り出し，この主記憶上の暗号化されたプログラム鍵をオペランドとする KEYDEC 命令を発行することによって，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ためのレジスタファイル
(2) 実行プログラムを復号するときに，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) プログラム鍵を蓄えておくレジスタファイルの管理

(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ファイル（以下これを鍵テーブルと呼ぶ）を用意した．また，鍵テーブル内の各プログ

ラム鍵の指定は，鍵テーブルのエントリに付けられたインデックスを用いることとした．これは，プログラム鍵の復号やプログラム鍵を用いたプログラムの復号などは，すべてハードウェアで行われるため，OS などのソフトウェアが，鍵テーブルの内容，すなわち復号されたプログラム鍵そのものを用いて何らかの処理を行う必要はないためである．したがって，プログラムからプログラム鍵を参照する必要はないので，鍵テーブルをアクセスする命令や機能は持たない．

(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 のみが知りうる情報である．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．これは，実行プログラムのサイズは，ページサイズに比べて十分大きいと考えられることと，ページ単位以下の細かい単位で実行プログラムのリロケーションを行って詰め込みたい場合は，実行プログラム自体を再リンクして暗号化することで可能となるためである．

(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．同時に実行状態となる独立した実行プログラムに対応して持つことが望まれ，十数個程度と想定している．現在，根拠となるデータを持っていないので，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．

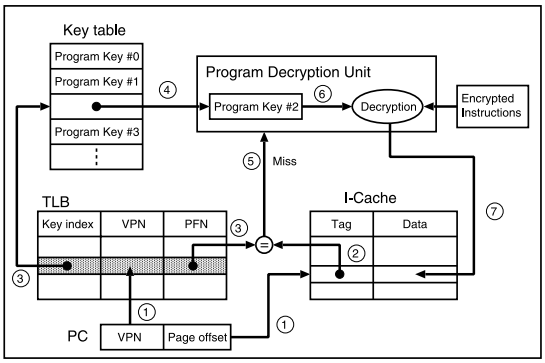

図 3 命令キャッシュミス時のプロセッサ内部処理
Fig. 3 The processing flow of an instruction cache miss.

TLB ミスが起こると，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ともに登録される．次に TLB を参照したときには，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．キャッシュミスが起こった場合には，TLB から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，すで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．

3.3 動的リンクへの対応

本提案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，共有ライブラリなどの動的リンクの場合である．動的リンクは，プログラム中のシステムコールによって OS にリンクを依

頼し，主記憶上の共有ライブラリの対応エントリアドレスを得て，システムコール命令を，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．組み込みシステムのようにプログラム構成が限定されている場合は，あえて動的リンクを使用しなくてもよいとも考えられるが，本節では，命令の書き換えを行わずに動的リンクを実現する一例を示しておく．

これは，動的リンクを依頼するシステムコール命令のオペランドを追加して，拡張されたオペランドの値に応じて，分岐命令の機能と割出しの機能を切り替える方法である．当命令の実行は，まず動作モードを決めるオペランドを参照する．このオペランドの内容によって，もう 1 つのオペランドをパラメータとしてシステムコール割込みを発生させるか，または，もう 1 つのオペランドを分岐先アドレスとして分岐を行うかを決定する仕様にする．動的リンクを依頼された OS は，このオペランドを書き換えることで，動的リンクを実現できる．この手法では，命令の機能追加のためのハードウェアが必要となるが，従来どお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すべ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，そのための KEYDEC 命令を含むプログラムも暗号化しなければならない．そのため本提案のプロセッサでは，チップ中にマスクパターンやフューズで起動時に使用されるプログラム鍵（boot_Key）を用意し，その鍵を用いて初期化プログラムを暗号化することとした．

また本提案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ときにも，TLB を索引するようにし，実アドレスとプログラム鍵の対応表を仕様化して主記憶に用意することで，TLB に実アドレスとプログラム鍵のインデックスを登録することで，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．

3.5 デバック機能

ブレークポイントやステップ実行などのデバック機能も命令の書き換えによって実現されることが多い．しかし，デバック機能はプログラム保護の弱点となることが考えられるため，本提案では，デバックはデバック機能を持った別システムで行うものと想定した．同様に，ハードウェア診断用に，内部のレジスタやキャッシュメモリなどを，直接アクセスできる命令も実装しないこととした．また，製造検査段階では，鍵テーブルなどを外部からアクセスする必要があるが，そのような回路には，フューズなどを挿入しておき，検査終了後に遮断する必要があると考える．

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 Technology）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 ビットごとに分割され，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．

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

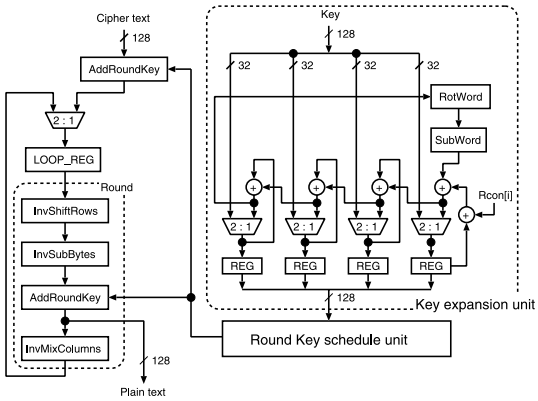

図 4 AES 復号回路の構成
Fig. 4 The block diagram of AES decryption circuit.

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，プロセッサコア面積との対比のために，同じく 0.18 μm プロセスで設計された ARM 社のプロセッサ ARM922T の面積を表 1 に示す[5]．動作周波数は 200 [MHz] であり，ARM922T のキャッシュサイズは 8 K/8 K バイトとなっている．

4.1.3 プログラムの復号処理時間

本システムでは，キャッシュミス時に，主記憶からプログラム（命令）を命令キャッシュに転送する過程で復号処理を行うので，復号処理にかかる時間は，プロセッサの通常のキャッシュミスペナルティに加算されることとなる．ここで，キャッシュミス時に主記憶から転送されるラインサイズのプログラムの復号を考えた場合に，復号処理時間はラインサイズやメモリバスの転送能力などによって変化する．たとえば，ラインサイズが 16 B（128 ビット）の場合は，AES-128 復号回路を用いて 1 回の復号処理でラインサイズのすべてのデータを復号できるが，ラインサイズが 32 B の場合は，すべてのデータを復号するためには 2 回の復号処理が必要となる．本稿では，AES-128 復号回路を搭載するプロセッサを以下に示すような仕様として仮定し，実際に設計して復号時間を評価した，

- ARMv5 命令セットに準拠し，Intel 社 XScale[9] を参考にしたパイプラインを持つプロセッサ構成．
- 主記憶として SDRAM PC-100 CL3 を想定する．プロセッサの動作周波数が 200 MHz であり，メモリバスの動作周波数は 100 MHz である．
- キャッシュラインサイズは 16 B，メモリバス幅は 64 ビットである．キャッシュミス時には，主記憶から 64 ビットごとに 2 回のバースト転送が行われる．プロセッサに到着したデータは，バスインタフェースによって 32 ビットごとにキャッシュに転送される．
- キャッシュミスが発生してから 22 サイクル後に 1 回目のデータがバスインタフェースに到着する．その 2 サイクル後（メモリサイクルでは 1 サイクル後）に 2 回目のバースト転送分のデータが到着し，キャッシュラインサイズの命令がすべて到着する．
- プロセッサは，キャッシュミスを発生させたデータがバスインタフェースからキャッシュに転送される際に，バイパスしてプロセッサに取り込む．そのため，プロセッサに見えるキャッシュミスペナルティは，キャッシュミスを発生させたデータがキャッシュラインの最初である場合は 23 サイクルとなり，キャッシュラインの最後である場合は 26 サイクルとなる．

以上の条件をもとに，AES-128 復号回路を用いた復号処理について考える．まず，キャッシュラインサイ

表 1 AES 復号回路各ユニットの面積
Table 1 The area size of AES decryption circuit.

	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
ARM922T (8 K/8 K cache)	8.1

表 2 各ベンチマークプログラムの入力データ
Table 2 The input data for each benchmark program.

	Format	Size
bitcount	array	75000
qsort	string	10000
susan	pgm image	76 × 95 pixel
dijkstra	matrix	100 × 100
sha	text	311,824 Byte

ズは 16 B となっており，AES-128 のブロックサイズ 1 個分であるので，AES-128 復号処理 1 回ですべてのデータを復号できる．キャッシュミスから 24 サイクル後にブロックサイズのデータがプロセッサに揃い，その後，10 サイクルかけて復号処理を行うので，キャッシュミスから 34 サイクル後に復号処理が完了する．復号結果である平文のデータは，通常どおり 32 ビットごとにキャッシュに転送され，プロセッサはそこからキャッシュミスを発生させたデータをバイパスするため，プロセッサに見えるキャッシュミスペナルティは 35〜38 サイクルとなる．

4.1.4 ソフトウェアシミュレータによる評価

本研究室で作成されたプロセッサの性能評価シミュレータ（asim）を用いて Stanford integer Benchmark [6),7)] と MiBench [8)] の一部のベンチマークプログラムを実行し，復号処理が加わった際の実行時間の増加を評価した．asim は，ARM アーキテクチャを対象とした性能評価シミュレータである．asim の基本構造は，Intel 社の XScale [9)] を参考にしている．XScale では，データ処理命令，ロード/ストア命令，乗算命令をそれぞれ独立したパイプラインで処理しており，asim はこれをモデル化しデータ依存によるパイプラインのストールを算出している．

本評価では，ベンチマークプログラムとして Stanford Benchmark と MiBench の bitcount，qsort，susan，dijkstra，sha の各ベンチマークプログラムを用いた．Linux 上に構築したクロスコンパイル環境は，gcc のバージョンは 3.4.1，コンパイルオプションは armv3-softfloat-linux-gcc -static -nostartfiles -msoft-float -march=armv3 -O2 である．MiBench の各ベンチマークプログラムの入力データを**表 2** に示す．

asim の命令キャッシュの構成は，XScale を参考にしているため 32 KB，ラインサイズ 32 B，32 ウェイセットアソシアティブ方式となっており，組み込みプロセッサの中では，豊富な命令キャッシュを有しているといえる．この構成では，ほとんど 99.9%の命令キャッシュヒット率となり，復号時間の影響がほとんど出ない．他の組み込みプロセッサではこれよりも小規模な命令キャッシュを搭載していることも考えられるため，より厳しい条件での評価を行うため 1 KB の構成に設定した．命令キャッシュの容量削減は，ウェイを削減することで実装しており，1 KB の構成は，ラインサイズは 32 B のダイレクトマップ構成となっている．復号処理なしの通常の命令キャッシュのミスペナルティを 24 サイクル，復号処理ありの命令キャッシュのミスペナルティを 36 サイクルとし，それぞれのキャッシュミスペナルティの場合の総実行クロック数を**表 3** に示す．

表 3 asim 実行結果
Table 3 The results of the asim simulation.

Stanford	通常時	復号処理あり	I$ hit ratio
Perm	2,760,596	2,760,878 (1.00)	99.9%
Towers	2,802,314	2,802,638 (1.00)	99.9%
Queens	33,966	34,229 (1.01)	99.8%
Intmm	1,827,057	1,980,585 (1.08)	98.6%
Puzzle	11,082,916	11,083,684 (1.00)	99.9%
Quick	1,827,791	2,068,079 (1.13)	97.4%
Bubble	1,700,228	1,712,468 (1.01)	99.9%
Trees	7,216,612	8,176,336 (1.13)	97.0%
Mm	14,098,480	14,498,976 (1.03)	99.5%
FFT	813,998	942,866 (1.16)	96.5%
MiBench			
bitcount	86,952,070	86,958,609 (1.00)	99.9%
qsort	48,250,708	48,953,410 (1.01)	99.8%
susan -e	8,513,606	10,233,964 (1.20)	95.2%
susan -c	5,219,322	6,125,693 (1.17)	95.8%
susan -s	35,001,513	36,673,096 (1.05)	99.3%
dijkstra	93,971,207	97,001,026 (1.03)	99.5%
sha	20,096,791	20,804,839 (1.04)	99.5%

表 3 より，実行時間増加の割合は最大で 1.20 倍となり，かなり小規模な命令キャッシュのサイズであっても 2 割程度の実行時間の増加で抑えることができていることが分かる．また，命令キャッシュのサイズが増加すれば，本提案の実現によるプロセッサの性能低下はさらに減少する．

4.1.5 面積への影響

今回設計した回路面積の合計は 1.02 [mm^2] であり，ARM922T に設計した回路を搭載したとすると，回路面積は約 13%増加することになる．ただし，今回の評価では，鍵スケジュール部の回路面積が全体の大部分を占めている．これは，ラウンド鍵を保存するレジスタファイルを，簡単のためフリップフロップセルで合成し評価したためであり，今後，6T-SRAM セルなどを用いることで，小面積化を図り，より現実的な評価を行う必要がある．

AES 復号回路の小面積化の実現を目指すため，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 4** に示す．ただし，InvShiftRows は配線のつ

表 4 AES 復号回路の各ステージの面積
Table 4 The area size of each AES decryption stage.

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

なぎかえで実現しているため評価は行っていない．表 4 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ことが分かる．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．森岡らの研究によれば，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[11]．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いと考えられる．

以上の評価面積以外に，鍵テーブルや TLB に追加する鍵テーブルインデックスのハードウェア量も必要となる．TLB についてはインデックスのみを格納するので，数ビットの増加であり，他のフィールド長と比較してこの評価では無視しうると考える．鍵テーブルであるが，これはエントリ数を検討したうえで評価が必要であるが，エントリ数が十数個程度であれば，本提案方式の有効性を覆すほどのハードウェア量増加にはならないと考える．

また，Suh らは，文献 10) で AEGIS における AES-128 暗号化/復号回路などの評価を行っている．AEGIS では，プログラムの暗号化だけでなく，データの暗号化も行っているため，暗号化/復号の両方を行う回路として実装されているが，ハードウェア構成は復号のみの場合と大きくは変わらない．基本ゲートの面積が異なるので，ゲート数で対応する部分を比較した結果を，**表 5** に示す．ほぼ同等のハードウェア量となっており，この比較からも，本評価結果は妥当であるといえる．

4.2 公開鍵暗号の復号回路について

本提案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，また，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かなり隠蔽でき，プログラム実行途中の性能には影響しないと考えられる．このため，公開鍵暗号の復号回路は，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．

表 5 回路面積の比較（文献 10) の Table 5 を参考）
Table 5 The comparison of the area size (referred to the Table 5 of the 10)).

AEGIS		本提案	
回路	ゲート数	回路	ゲート数
AES unit (3 AES units)	23,807 (71,426)	復号処理 +鍵拡大処理	19,253
Key SPRs	10,843	鍵スケジュール	11,331

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[12]．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．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．RSA 復号回路の設計と評価は，詳細なシステム設計の中で検討していく予定である．

また，公開鍵暗号の復号鍵をチップに実装する方式について検討した．これには，チップ製造時に内蔵フューズを切る方法で，作り込むことが現実的であると考えた．実際，現在のプロセッサには，個体ごとの識別 ID が付加されているので，同様の方式が可能と考える．ただし，書き込み後に書き込み回路自体を外部から遮断するようなフューズ回路を設けて，復号鍵を盗まれないような方策が必要である．一般のコンピュータシステムのソフトウェアライセンス保護に利用する場合は，各個体ごとに公開鍵を異なったものにする必要があるが，組み込みシステムの組み込みプログラムのアルゴリズム保護を目的とする場合，組み込みシステムベンダごとで，あるいは，製品品種ごとで，同じ公開鍵を用いることで十分であり，部品の故障時の交換が，容易に行えるなどの利点も考えられる．

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み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，ソフトウェアシミュレータによる性能への影響の評価を行った．

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，データの暗号化機構の実現を制約するものはないので，コンテキスト保護を含むデータの保護への対応も，今後，必要性も含めて検討したい．

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，文部科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) Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software, *ASPLOS 2000*, pp.168–177 (2000).
2) Suh, G.E., Clarke, D., Gassend, B., van Dijk, M. and Devadas, S.: AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing, *ICS 2003* (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，情報処理学会論文誌，Vol.45, No.SIG3, pp.1–10 (2004).
4) National Institute of Standard and Technology: Announcing the ADVANCED ENCRYPTION STANDARD (AES), FIPS PUB 197 (2001).
5) ARM Ltd.: ARM922T. http://www.arm.com/products/CPUs/ARM922T.html
6) Hennessy, J. and Nye, P.: Stanford Integer Benchmarks, Stanford University (1988).
7) Weicker, R.P.: An Overview of Common Benchmarks, *IEEE Computer*, Vol.23, No.12, pp.65–75 (1990).
8) Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T. and Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite, *IEEE 4th Annual Workshop on Workload Characterization* (2001).
9) Intel: *Intel XScale(R) Core Developer's Manual* (2004).
10) Suh, G.E., O'Donnell, C.W., Sachdev, I. and Devadas., S.: Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions, *ISCA 2005* (2005).
11) 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).
12) Rivest, R.L., Shamir, A. and Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, *Comm. ACM*, Vol.21, No.2, pp.120–126 (1978).
13) Kawabata, T., Tamai, T., Hashimoto, M. and Miyamori, T.: Security Enhanced Embedded Processor using Local Memory Protection Mechanism, *Cool Chips IX*, pp.143–157 (2006).

(平成 18 年 5 月 8 日受付)
(平成 18 年 8 月 6 日採録)

城本 正尋

1982 年生．2004 年広島市立大学情報科学部情報工学科卒業．2006 年同大学大学院情報科学研究科情報工学専攻修士課程修了．同年ソニー・エルエスアイ・デザイン株式会社入社．計算機アーキテクチャ，コンピュータセキュリティに興味を持つ．

田端 猛一（学生会員）

1983 年生．2006 年広島市立大学情報科学部情報工学科卒業．現在，同大学大学院情報科学研究科博士前期課程在学中．組み込みプロセッサやプロセッサアーキテクチャに興味を持つ．

酒井 智也

1982 年生．2005 年広島市立大学情報科学部情報工学科卒業．現在，同大学大学院情報科学研究科情報工学専攻修士課程に在籍．計算機アーキテクチャ，コンピュータセキュリティに興味を持つ．

島田　貴史

1982 年生．2005 年広島市立大学情報科学部情報工学科卒業．現在，同大学大学院情報科学研究科情報工学専攻修士課程在籍．計算機アーキテクチャ，マルチスレッドプロセッサの研究に興味を持つ．

窪田　昌史（正会員）

1993 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年日本 IBM（株）入社．1998 年京都大学大学院工学研究科情報工学専攻博士後期課程単位認定退学．同年広島市立大学情報科学部助手．2001 年から 2002 年までオーストラリア Bureau of Meteorology Research Centre 客員研究員．主として並列化コンパイラ，ハイパフォーマンスコンピューティングに関する研究に従事．IEEE-CS，ACM 各会員．

川端　英之（正会員）

1992 年京都大学工学部情報工学科卒業．1994 年同大学大学院工学研究科修士課程修了．同年より広島市立大学情報科学部助手．博士（工学）．高性能数値処理ソフトウェアおよびその開発環境に関する研究に従事．ACM，IEEE-CS，SIAM，電子情報通信学会，日本ソフトウェア科学会，日本応用数理学会各会員．

北村　俊明（正会員）

1955 年生．1978 年京都大学工学部情報工学科卒業．1983 年同大学大学院博士課程研究指導認定退学．同年富士通（株）入社．汎用コンピュータ，スーパコンピュータ VPP シリーズの VLIW 型 CPU，M アーキテクチャ・命令エミュレーション，米国 HAL 社において SPARC プロセッサなどの研究開発に従事．博士（工学）．2000 年京都大学総合情報メディアセンター助教授を経て，2002 年広島市立大学情報科学部教授．電子情報通信学会，IEEE，ACM 各会員．