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

2007－ARC－172 (41)
2007－HPC－109 (41)
2007／3／2

耐ソフトウェアタンパ・プロセッサ

清水一人[†] 入江英嗣[††]
五島正裕[†] 坂井修一[†]

プログラムを解析や改ざんから守るセキュア・プロセッサが研究されている．セキュア・プロセッサは，プログラムを暗号化して配布し，プロセッサ内部で復号化を行い，実行するもので，ソフトウェアによるタンパだけでなく，ハードウェアによるタンパも防ぐことができる．しかし，メモリアクセスに暗号処理が必要となるため，プログラムの実行速度は低下する．

そこで，現在行われているタンパの多くはソフトウェアによるタンパであり，これを防ぐだけで大半のタンパに対応できるという考えから，本論文では，ソフトウェアによるタンパを防ぐモードを持った耐ソフトウェアタンパ・プロセッサを提案する．耐ソフトウェアタンパ・プロセッサはメモリアクセスに暗号処理を必要としないため，高速な実行ができる．

さらに，シミュレータを用いて，セキュア・プロセッサよりも速度低下を軽減した実行が可能であることを確認した．

Software-Tamper-Resistant Processor

KAZUTO SHIMIZU,[†] HIDETSUGU IRIE,[††] MASAHIRO GOSHIMA[†]
and SHUICHI SAKAI [†]

Secure Processor is researched as a technique for protecting the program from Tampering. It is considering Hardware Tampering, so the memory access speed decreases because the memory is encrypted.

Many of Tampering is Software Tampering, and the greater part of Tampering can be prevented by resistance to Software Tampering. Then, we propose Software-Tamper-Resistant Processsor, it had the mode that prevents Software Tampering. This doesn't need the code encryption/decryption for the memory access, and a high-speed access is possible.

In addition, it was confirmed with a simulator to be able to execute Software-Tamper-Resistant Processor more high-speed than Secure Processor.

1. はじめに

近年，社会の情報化が進むに伴って，あらゆる場面でコンピュータが用いられるようになってきており，これらのコンピュータ上で動くプログラムの中には，保護されるべき重要なものがある．

外部からの攻撃や，不正な複製／利用から保護することも重要であるが，**タンパ**からの保護に着目した研究が盛んになっている．タンパとは，プログラムをリバースエンジニアリングによって解読し，プログラムに含まれるデータや特殊なアルゴリズムを解析したり，改ざんしたりする行為である．著作権保護技術を用いたプログラムや特殊なアルゴリズムを用いたプログラムにおいて，プログラムの内部動作や実行の手順を知られ，改ざんされてしまうと，それらの技術を無効化されてしまう．また，アルゴリズムそのものを秘密にする必要がある場合は，解析自体を防がなければならない．このタンパからプログラムを守る性質のことを，**耐タンパ性**という．

耐タンパ性に関する研究の一つに，プロセッサで耐タンパ性を確保する**セキュア・プロセッサ**がある[7),8),10)]．セキュア・プロセッサは，暗号化されたプログラムをプロセッサ内部で復号化して実行する．このため，プロセッサ外部のメモリ上などでは平文のプログラムが存在せず，タンパに対する高い耐性を持つ．しかし，メモリアクセスに暗号化が必要になるため，オーバヘッドが大きい．

タンパは，手法によって二つにわけることができる．一つは，ハードウェアを利用して行う**ハードウェアタンパ**である．本論文ではハードタンパと呼ぶ．ハードタンパには，基盤上のデータバスにプローブを取り付け，ロジックアナライザなどで直接データを読み取る手法などがある．ハードウェアタンパを用いると，よ

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 科学技術振興機構
Japan Science and Technology Agency

り多くの情報を得られるが，対象のハードウェアを直接触ることができる必要があり，また，特殊な機器を用いることや，技術的な問題から敷居が高い．そのため，多くのユーザは実行できず，大きな脅威ではない．

もう一つは，ソフトウェアを利用して行う**ソフトウェアタンパ**である．本論文ではソフトタンパと呼ぶ．ソフトタンパには，オペレーティング・システム (OS) を改造して行う手法や，目的のプログラムと同時に何らかのプログラムを動作させる方法などがある．

OS を改造すると，特権を利用して，実行中の状態を知ることができたり，キャッシュにアクセスできたりするため，タンパしやすい．特にオープンソースの OS は，OS のプログラムコードが配布されているため改造が容易であり，ソフトタンパを行いやすい．

ソフトタンパを行うプログラムは多数製作され，Web 等で簡単に入手できてしまうのが現状であり，一般ユーザでも比較的容易に行うことができる．例えば，あるソフトウェアの認証回避をするタンパ用プログラムが出回ってしまうと，そのプログラムさえ入手できれば，だれでも認証回避ができてしまう．また，ハードタンパと異なり，対象の前にいなくても実行可能である．

このように，ソフトタンパに比べてハードタンパは実行できる場合が少ないため，ソフトタンパを防ぐだけでタンパの大部分を防ぐことが可能である．そこで，本論文では，耐ソフトタンパ実行を設けた**耐ソフトウェアタンパ・プロセッサ**を提案する．耐ソフトタンパ実行とは，耐ソフトタンパのみを考慮したモードで，メモリアクセスに暗号処理を必要としない．そのため，ハードタンパに対応したセキュア・プロセッサよりも高速な実行を可能とする．

以下，本論文は次のように構成される．2 節で，耐ソフトタンパ性を確保する既存の手法について，いくつか説明する．3 節で，セキュア・プロセッサについての解説を行う．4 節では，本論文で提案する，耐ソフトウェアタンパ・プロセッサについての解説を行う．5 節で，提案手法の評価を行い，6 節でまとめと今後の課題について述べる．

2. 耐ソフトタンパ性を確保する既存手法

ソフトタンパへの耐性を高める手法は，第 1 節で触れたセキュア・プロセッサの他に，ソフトウェアによる手法と，OS の完全性を守ることによる手法がある．セキュア・プロセッサについては第 3 節で詳細を説明し，ここでは残りの手法について述べる．

2.1 ソフトウェアによる手法

ソフトウェアでソフトタンパへの耐性を確保する手法として**難読化**と**自己暗号化**がある．

難読化は，プログラムのコードを複雑化し，解読を困難にする技術である[5),11)]．コンパイラやトランスレータでコードを解読されにくいものに変換する．しかし，プログラムの機能を維持しなければならないため，完全な難読化は不可能とする文献[3)] もある．

自己暗号化は，プログラム自体を暗号化しておき，実行時に復号化を施すことで耐タンパ性を持たせる技術である[2)]．この技術を用いると，実行される前の状態ではプログラムが暗号化されているため，その内容を知ることは難しい．

これらのソフトウェアによる手法が実現する耐タンパ性には限界がある．難読化手法はアドホックな対応であり，公開アルゴリズムを用いた暗号化に比べると強度は劣る．一方で自己暗号化は，ソフトウェアで復号化を行う以上，復号化した後のデータが得られてしまう．

2.2 OS の完全性を保証する手法

Trusted Platform Module(TPM)[9)] は，プロセッサとは別のチップとして用意された，耐タンパ性を確保するハードウェアの一種である．インストールされた OS の完全性を保証することができ，OS が改ざんされていないことから OS を信頼できるとして TPM と OS で耐タンパ性を確保する．

しかし，改ざんされてないことと信頼性を持っていることは同一であるとは言えない．巨大な OS の全体を本当に信頼できるのかはソフトウェアベンダからはわからないといった問題があり，OS の信頼性に依存して耐タンパ性を確保することは難しい．

また，オープンソースの OS は自由に改造することが可能であり，その信頼性を確かめることも非常に難しい．

3. セキュア・プロセッサ

3.1 セキュア・プロセッサの概要

セキュア・プロセッサでは，プロセッサ内部を信頼できるもの，それ以外のソフトウェアやハードウェアを信頼できないものとする．主記憶や二次記憶，OS や他のプログラムは信頼できず，攻撃者によって改ざんされている可能性があるものと考える．

そこで，セキュア・プロセッサではプログラムを解析から保護するために

- プログラムコードの秘匿
- プログラムの挙動の秘匿

の 2 つを行わなければならない．

秘匿を行うと，それらを読み取ることはできなくなるが，改ざんをすることで正常な動作を妨げることはできる．正常動作を保証するためには，以下の機能も必要となる．

- プログラムの挙動の完全性検証

この 3 つについて，以下で説明する．

3.2 プログラムコードの秘匿

プログラムのコードを秘匿するために，セキュア・プロセッサでは暗号化を行う．プログラム作成者は，

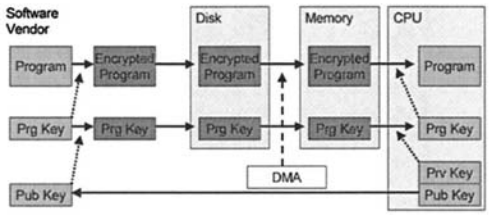

図 1　暗号化済みプログラムの実行

共通鍵暗号方式でプログラムを暗号化する．このときに用いる暗号鍵を，ここでは**プログラム鍵**と呼ぶ．共通鍵暗号方式は高速な処理が可能だが，暗号化と復号化に同じ鍵を用いる方式であるため，鍵を秘匿したまま通信先に配送する必要がある．

この鍵の配送には公開鍵暗号方式を用いる．公開鍵暗号方式は，公開鍵と秘密鍵の２つの対になる鍵を用意し，公開鍵で暗号化したものは秘密鍵でしか復号化できないという方式である．

セキュア・プロセッサは，公開鍵暗号方式の秘密鍵(**プロセッサ秘密鍵**)をプロセッサ内部に持ち，公開鍵(**プロセッサ公開鍵**)は公開されている．プログラム作成者がプロセッサ公開鍵を用いてプログラム鍵を暗号化し，暗号化されたプログラムと共に配布すると，プロセッサではプログラム鍵をプロセッサ秘密鍵によって復号化し，さらにそのプログラム鍵を用いてプログラムを復号化する (**図 1**)．

これはプロセッサ内部で行われるため，外部にプログラム鍵やプログラムの平文が漏れることはない．

3.3　プログラムの挙動の秘匿

プログラムの挙動を秘匿するためには，プログラム中で用いるレジスタ／キャッシュ／メモリを秘匿する必要がある．

レジスタとキャッシュはプロセッサ内部にあるため，アクセス保護をすることで秘匿ができるが，メモリはプロセッサ外部にあるため，そのままではハードタンパを防ぐことができない．そこでメモリの秘匿には暗号化を用いる．

以下で，レジスタ／キャッシュのアクセス保護と，メモリの暗号化について述べる．

3.3.1　レジスタ／キャッシュのアクセス保護

Lie らの提案する XOM[4),7)] では，キャッシュやレジスタにプロセス識別用のタグを追加し，これが一致しないプロセスからの読み込みができないようにしている．このタグにより，特権モードを利用してのキャッシュやレジスタの値の読み取りを防ぐことができる．

3.3.2　メモリの暗号化

メモリを OS や他のプロセス，さらにはハードウェアを用いたアクセスから秘匿するために，暗号化を行う．この暗号化には共通鍵暗号方式を用い，このとき用いる鍵を**データ鍵**と呼ぶ．

データ鍵は，プロセスごとにランダムに生成され，プロセスに関連づけてプロセッサが管理する．他のプロセスからはデータ鍵を利用することができないため，暗号化されたデータは盗み見られることはない．また，プログラム鍵とは異なる鍵をランダムに生成して利用するので，同一プログラムを複数プロセス実行した場合にも，プロセス間でタンパされることはない．

データの暗号化や復号化を行うタイミングは，内部キャッシュと外部メモリの間でデータが通信されるときである．この暗号化や復号化により，メモリアクセス時間が最大で２倍程度になる．ただし，メモリ書き込み時の暗号化によるオーバヘッドは，バッファを設けることで隠蔽することができる．読み込み時の復号化に関しては，隠蔽することはできない．

3.4　プログラムの挙動の完全性検証

プログラムの正常動作を保証するために，一部のセキュア・プロセッサではメモリの完全性検証を行う．

Suh らの提案する AEGIS[6),8)] では，キャッシュラインごとにハッシュ値を計算し，これをプロセッサで管理して完全性を保証しているが，ハッシュ値をすべてプロセッサ内部に保管することが難しいため，ハッシュ値をいくつかまとめたデータのハッシュ値をさらに計算することを繰り返し，ハッシュ値のツリーを構築する．このようにすることで，ツリーのルートの完全性が保証できていればメモリ全体としても完全性が保証できていると言えるため，少なくともルートをプロセッサで管理すればよいことになる．

4.　耐ソフトウェアタンパ・プロセッサ

セキュア・プロセッサは，ハードタンパを考慮してメモリでの暗号化を行ったために，メモリのアクセス速度が低下した．

しかし，１節で述べたように，ソフトタンパのみを防ぐだけでも大多数のタンパを防ぐことができると考え，本論文では，耐ソフトタンパ実行を設けた耐ソフトウェアタンパ・プロセッサを提案する．耐ソフトタンパ実行とは，耐ソフトウェアタンパのみを考慮した実行モードで，メモリ上ではプログラムやデータは暗号化されていない．そのため，メモリアクセスにオーバヘッドを生じない高速な実行が可能である．

4.1　OS との連携

プログラムを実行するにあたって，OS の果たす役割は大きい．プログラムを OS によるタンパから守るためには，OS との関係を考える必要がある．その例として，以下の２つが考えられる．

- OS の機能をハードウェアで実装する
- OS の機能をソフトウェアで実装し，それを信頼する

OS の機能をハードウェアで実装する方法は，柔軟性に乏しく，実用的ではない．

OS はソフトウェアで実装するが，信頼できるもの

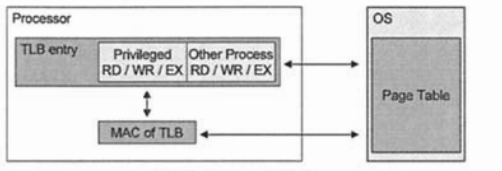

図 2 Secure TLB

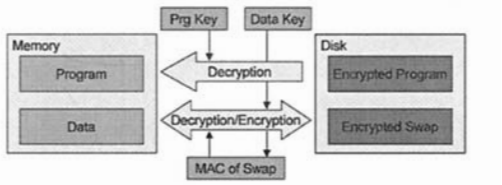

図 3 Secure DMA

に限ることで OS の信頼性に依存する方法は，2.2 で述べたように信頼性の確立が難しい．

そこで，本研究では，OS やプロセッサに手を加えることで，様々な機能をソフトウェアの OS で行いながらも，その信頼性に依存せずに耐タンパ性を確立する．

4.2 Secure DMA と Secure TLB

信頼していない OS 上で耐タンパ性を確立するためには，下の 2 つの機能が必要になる．

- アクセス制限によって OS からのアクセスを防ぐ
- アクセス制限の外に出す物には，暗号化と認証をする

本研究ではこれを実現するためのプロセッサの追加機能として **Secure TLB** と **Secure DMA** を用意し，この 2 つの協調によって耐タンパ性を確立する．

Secure TLB

Secure TLB は，拡張されたアクセス権限と，MAC による認証機能を持った TLB である．アクセス権限の拡張によってキャッシュ／メモリを秘匿し，MAC による認証でページテーブルの改ざんを防止する (**図 2**)．

Secure DMA

Secure DMA は，暗号処理機能と **Message Authentication Code**(MAC) による認証機能を持った DMA 転送で，二次記憶とメモリの間で暗号化や復号化を行いながら転送をすることができる (**図 3**)．

4.3 OS の変更

本研究では，OS に手を加えることで，ファイル管理／プロセス管理／メモリ管理の 3 つを OS で行いながらも OS の信頼性には依存せずに耐タンパ性を確立する．

以下でこの 3 点における変更点を説明する．

4.3.1 ファイル管理

耐ソフトウェアタンパ・プロセッサでは，暗号化されたプログラムをメモリ上で復号化して実行するため，暗号化されたプログラムファイルの読み込み時に復号化が必要となる．このときプログラムの復号化に用いる鍵 (プログラム鍵) はあらかじめプロセッサ内部に用意する必要があり，この鍵は 3.2 で述べたセキュア・プロセッサの場合と同様にして配送される．配送された鍵はプロセッサが管理する．管理方法については 4.3.2 で述べる．

プログラムの読み込みには，先に述べた Secure DMA を用いる．Secure DMA は，転送と同時に暗号処理を行う DMA 転送で，プログラムの読み込みと，データの読み込み／書き込みの 3 つの機能を持つ．ここでは，プログラムの読み込みについて説明する．

プログラムの読み込み

耐ソフトウェアタンパ・プロセッサでは，プログラムを実行するとき，Secure DMA でメモリ上に展開する．DMA コントローラは転送のコマンドとともにプログラム鍵を受け取り，これを使って復号化しながらメモリ上に転送する．

また，このときの転送先アドレスは OS が決定するが，このアドレスを OS が偽ると，プログラムが正常に動作しない．これを防ぐため，ページテーブルが改ざんされていないことを確認する必要があるが，この方法については 4.3.3 で述べる．

4.3.2 プロセス管理

耐ソフトウェアタンパ・プロセッサにおいて，OS のプロセス管理に関して変更を要するのは

- プロセスの起動と終了
- プロセスの切り替え

の 2 点である．

プロセスの起動と終了

耐ソフトタンパ実行をするプロセスを起動するとき，3.3.2 で述べたセキュアプロセッサの場合と同様に，プロセスごとにランダムな**データ鍵**を用意する．

プロセッサは現在実行中のコンテキスト番号を持っているが，先に述べたプログラム鍵やデータ鍵などはこのコンテキスト番号と関連づけをして，プロセッサ内部のテーブルに記憶する．他のプロセスの鍵を利用することはできず，またプロセッサの外部からこれにアクセスすることもできない．

プロセスの終了時には，そのプロセスに関連づけられた鍵をテーブルから削除する．

プロセスの切り替え

プロセスの切り替え時，割り込みが発生すると，プロセッサはすべてのレジスタについて，暗号化と MAC(Message Authentication Code) の生成を行う．暗号化は，割り込み直前のコンテキスト番号に対応するデータ鍵で行い，MAC はレジスタの値とデータ鍵から生成する．

MAC は，メッセージにパスワードを付加したデータのハッシュを取ったもので，メッセージとともに通信相手に送ると，相手側でメッセージが改ざんされていないことを確認できるという技術である．ここでは，メッセージとしてレジスタを，パスワードとしてデータ鍵を用いて MAC を生成する．

暗号化されたレジスタとMACは，プロセッサからOSが読み込んで管理し，再びこのプロセスに実行が切り替わるとき，OSからプロセッサに戻される．このとき，復号化を行ってハッシュを再計算し，MACの一致を確認することで改ざんの検出を行う．

4.3.3 メモリ管理

耐ソフトウェアタンパ・プロセッサでは，メモリ上のデータなどは復号化されており，そのままではOSなどの他のプロセスからタンパすることができてしまう．これを防ぐために，メモリ保護の強化を行う．

メモリ保護の強化

通常のメモリ保護機能では，ページやセグメントの所有者が決まっており，それ以外のプロセスがアクセスしようとするとメモリ保護違反となるが，特権を持ったOSなどのプロセスは，すべてのメモリにアクセスすることができてしまう．そこで，特権によってアクセスされないように，従来のメモリ保護の拡張が必要である．

ここでは，ページテーブルエントリに含まれる所有者によるアクセス権限の読み込み (RD) ／書き込み (WR) ／実行 (EX) に加えて，他のプロセスによるアクセス権限のRD ／ WR ／ EXビットと，特権プロセスによるアクセス権限のRD ／ WR ／ EXビットを用意する．

新しくページが作成されたとき，これらのアクセス権限はOSが決めるが，このビットがプログラムの指定通りになっているかどうかは信頼できないため，先述のSecure TLBにてプログラムが指定するビットを書き込む．Secure TLBでもアクセス権限のビットを拡張しており，メモリアクセス時にこれをチェックし，権限がなければアクセス違反となる．

キャッシュのアクセスに関しても，同様にSecure TLBで権限をチェックすることでアクセス違反を検出できる．

この拡張によって，他のプロセスからのメモリアクセスを任意に変更することができるが，ページテーブルはOSが管理しており，OSが書き換えることも可能なため，ページテーブルの改ざん防止が必要となる．

ページテーブルの改ざん防止

ページテーブルはOSが管理するが，これが書き換えられてしまうとアクセス権限が無効化されたり，異なるページを読ませたりすることができてしまう．

そこで，ページテーブルエントリにもMACを用いた認証を行う．Secure TLBからページテーブルエントリが追い出されるとき，ページテーブルエントリとデータ鍵からMACを生成する．これをOSが読み込んで管理し，再びこのページテーブルエントリが必要になったときに，ページテーブルエントリと共にSecure TLBに渡される．Secure TLBではこのMACを確認し，改ざんされていなければ有効とする．

以上でメモリに関してはアクセス保護をすることが可能となったが，メモリはスワップ処理で二次記憶に書き出されることもある．二次記憶上ではこのようなメモリ保護機能は働かないため，スワップされたメモリの保護には別の方法が必要となる．次で，スワップの保護について説明する．

スワップの保護

スワップによって二次記憶上に書き出されたメモリを保護するため，Secure DMAによって暗号化とMACによる認証を行う．スワップアウト時，ページのデータは先に述べたデータ鍵を用いて転送時に暗号化され，二次記憶に書き込まれる．また，データにデータ鍵を付加した上でハッシュを計算し，MACとする．このMACはOSが読み込み，スワップインまで保存しておく．

ページがスワップインするとき，OSがSecure DMAに対してMACを渡し，プロセッサ内でハッシュを再計算してMACとの一致を確認する．同時にページはSecure DMAで復号化されて物理メモリに読み込まれる．

これでスワップにおけるページの盗み見と改ざんを防ぐことができる．

5. 提案手法の評価

本論文では，メモリアクセスの高速化が実行速度に与える影響を調べるため，Simplescalar[1)]を用いて評価を行った．

評価パラメータを**表1**に示す．ベースラインは，耐タンパ性を持たない，通常のプロセッサである．セキュア・プロセッサはメモリアクセスに暗号処理を行うため，アクセスレイテンシを2倍とした．また，耐ソフトウェアタンパ・プロセッサは，Secure TLBのページテーブルエントリ認証機能があるため，TLBのアクセスレイテンシを45サイクルとした．

SPEC2000intベンチマークの11種類プログラムを用いて計測した結果が**図4**である．ベースラインのIPCを1としたときのIPCの相対値を表している．提案手法は，通常のプロセッサに対して平均で1.1%の速度低下にとどまり，セキュア・プロセッサからは6.7%の速度向上があった．

6. おわりに

6.1 まとめ

本論文では，耐ソフトタンパの重要性について述べ，耐ソフトタンパに特化した耐ソフトウェアタンパ・プロセッサを提案した．耐ソフトウェアタンパ・プロセッサで耐タンパ性を確保するために行う，メモリ保護の拡張と，レジスタ／スワップの暗号化，MACによるレジスタ／ページテーブル／スワップの認証について説明した．

メモリ保護の拡張では，特権プロセスや他のプロセ

表 1 評価パラメータ

モデル	ベースライン	セキュア・プロセッサ	耐ソフトウェアタンパ・プロセッサ
Simulator	SimpleScalar(sim-outorder) ベース		
L1 ICache	64KB, LRU, 4way, 32 Bytes line 3cycles access latency		
L1 DCache	64KB, LRU, 4way, 32 Bytes line 3cycles access latency		
L2 Cache	2MB, LRU, 4way, 64 Bytes line 12cycles access latency		
Memory Access Latency	150cycles	300cycles	150cycles
Instruction TLB	16sets, LRU, 4way 30cycles access latency		16sets, LRU, 4way 45cycles access latency
Data TLB	32sets, LRU, 4way 30cycles access latency		32sets, LRU, 4way 45cycles access latency
命令セット	Alpha 21264 ISA		
ベンチマーク	SPEC2000int (入力セット:test)		
実行命令数	全命令		

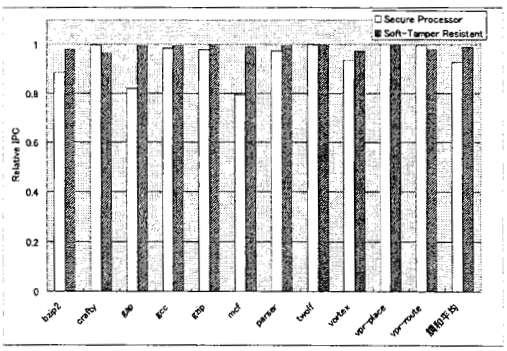

図 4 相対 IPC の比較

ス用のアクセス権限ビットを別に用意し，OS からであっても読むことのできないページを用意することを可能にした．

退避されるレジスタやスワップは，アクセス権限の変更では保護することができないため，暗号化を行うことで内容が読み取られることを防いだ．

また，レジスタ，ページテーブル，スワップについて MAC による認証を行うことで改ざんを防ぎ，正常な動作を保証した．

さらに，この手法を用いることで起こる実行速度の低下は 1.1%程度にとどまり，セキュア・プロセッサに対しては 6.7%の速度向上が見込めることを確認し，この手法の優位性を示した．

6.2 今後の課題

本論文ではコンテキストスイッチとスワップに関するオーバヘッドのシミュレーションを行っていないが，今後，オーバヘッドの正確な見積もりとともに行っていく予定である．

謝辞 本論文の研究は，一部 21 世紀 COE 「情報技術戦略コア」、及び科学技術振興機構 CREST「ディペンダブル情報基盤」による。

参 考 文 献

1) Simplescalar. http://www.simplescalar.com/.
2) David Aucsmith. Tamper Resistant Software: An Implementation. In *Information Hiding*, pp. 317–333, 1996.
3) Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (Im)possibility of Obfuscating Programs. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pp. 1–18, London, UK, 2001. Springer-Verlag.
4) Dan Boneh, David Lie, Pat Lincoln, Lohn Mitchell, and Mark Mitchell. Hardware support for tamper-resistatnt and copy-resistant software. Technical report, Stanford University Computer Science, 1999.
5) Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical report, July 1997.
6) Blaise Gassend, G. Edward Suh, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Caches and hash trees for efficient memory integrity verification. In *Proceedings of International Symposium on High-Performance Computer Architecture*, 2003.
7) David Lie, Chandramohan A. Thekkath, and Mark Horowitz. Implementing an untrusted operating system on trusted hardware. In *Proceedings of ACM Symposium on Operating Systems Principles*, 2003.
8) G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *International Conference on Supercomputing*, 2003.
9) Trusted Computing Group. *TPM Main Part 1 Design Principles*. Trusted Computing Group, 1.2 revision 85 edition, 2005.
10) 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 コンピューティングシステム, Vol. 45, No. 3, March 2004.
11) 佐藤弘紹, 門田暁人, 松本健一. データの符号化と演算子の変換によるプログラムの難読化手法. 電子情報通信学会技術報告情報理論研究会 (情報通信サブソサイエティ合同研究会), p. 13.