修士論文

耐ソフトウェアタンパ・プロセッサ

指導教員　　坂井 修一 教授

東京大学大学院 情報理工学系研究科
電子情報学専攻

清水 一人

概要

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多数存在する．

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．

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，より多くの情報を得られるが，これを行うためには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，実行前のバイナリに対して静的に行うものと，実行中に動的に行うものがある．前者には逆アセンブラなどを用いる方法，後者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，非常にタンパしやすい．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，対象のソフトウェアが動作しているハードウェア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実行方法で，メモリ上のデータを暗号化せず，メモリアクセスに暗号処理を必要としない．よって，実行速度を落とさずに耐タンパ性を確保することが出来る．

本論文では，スーパスカラ・シミュレータである Simplescalar を用いて耐ソフトウェアタンパ・プロセッサの動作速度を計測し，既存手法に比べて平均で 6.7%，最大で 24.7%の速度向上を確認した．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景

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多数存在する．

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 [8, 16, 10, 18]．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．暗号化することによって，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，高い耐タンパ性を持つ．しかし，メモリアクセスにおいて暗号化や復号化が必要になるため，アクセス速度の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，より多くの情報を得られるが，これを行うためには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，実行前のバイナリに対して静的に行うものと，実行中に動的に行うものがある．前者には逆アセンブラなどを用いる方法，後者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，非常にタンパしやすい．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，対象のソフトウェアが動作しているハードウェアの前にいなくても実行可能である．

1.2 目的

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実行方法で，メモリ上のデータを暗号化せず，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，メモリ上での暗号化を行うためにメモリアクセス速度が低下したセキュア・プロセッサよりも高速な実行を可能とする．

1.3 構成

以下，本論文は次のように構成される．2 章で，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 章で，セキュア・プロセッサについての解説を行う．4 章では，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 章で提案手法の評価を行い，6 章でまとめと今後の課題について述べる．

第2章　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 章で触れたセキュア・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 章で詳細を説明し，ここでは残りの 2 つ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**が挙げられる．

2.1.1　難読化

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 [6, 17]．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 [4] もある．

2.1.2　自己暗号化

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 [3]．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，その強度は劣る．一方で，自己暗号化はソフトウェアで復号化を行う以上，復号化した瞬間のデータを読まれてしまう危険性がある．

2.2 OSの完全性を保証する手法

2.2.1 Trusted Platform Module

Trusted Platform Module (TPM)[13, 14, 15] は，Trusted Computing Group (TCG)[2, 12] の提唱する Trusted Computing において，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．

第3章　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・プロセッサは，ハードウェアによって耐タンパ性を確保するプロセッサである．これまでに，**XOM**[5],**L-MSP**[19, 16], **AEGIS**[10] などといったセキュア・プロセッサが研究されてきている．

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，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

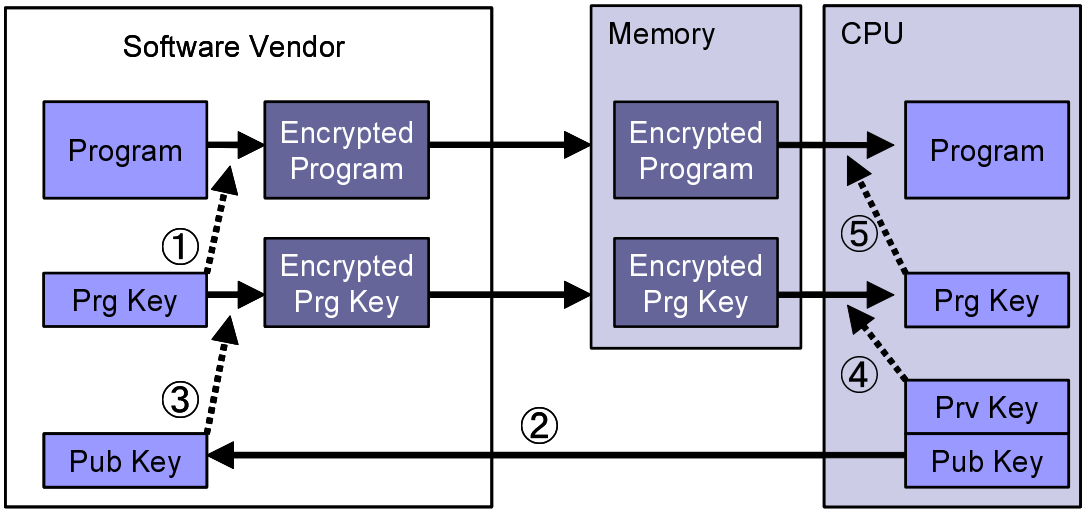

図 3.1: 暗号化済みプログラムの実行

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 3.1)．

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．

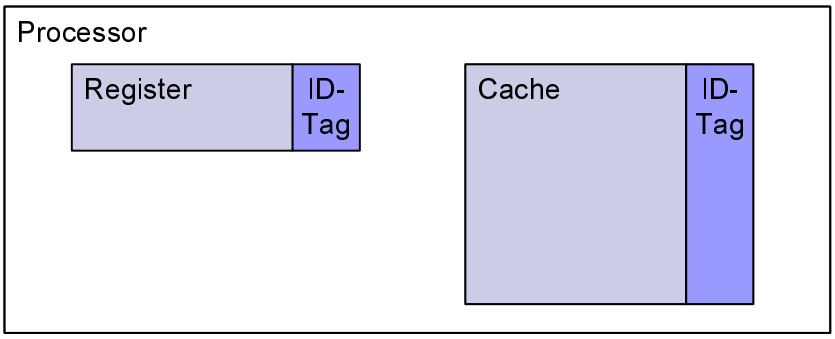

図 3.2: レジスタ／キャッシュのプロセス識別タグ

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[5, 9, 8] 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 (図 3.2)．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 (図 3.3)．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．

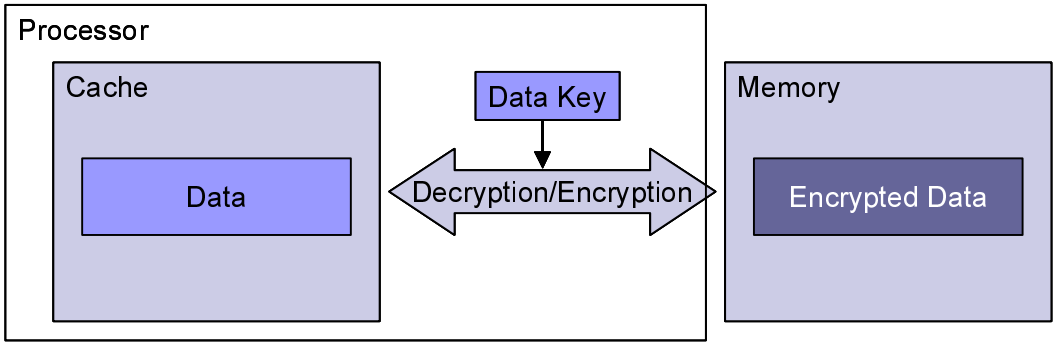

図 3.3: メモリの暗号化

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 (図 3.4)．これは，メモリの書き込み時にデータのハッシュ値を計算し，データが読み込まれるときにそのハッシュ値と比較することによって改ざんされていないかどうかを確かめるものである．改ざんされていないことを確かめるためには，ハッシュ値が正しいことが保証できなければならないため，これをプロセッサで管理する．しかし，そのために全てのハッシュ値をプロセッサで管理するのはサイズが大きすぎるという問題がある．

Suh らの提案する AEGIS[10, 7, 11] 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 (図 3.5)．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．

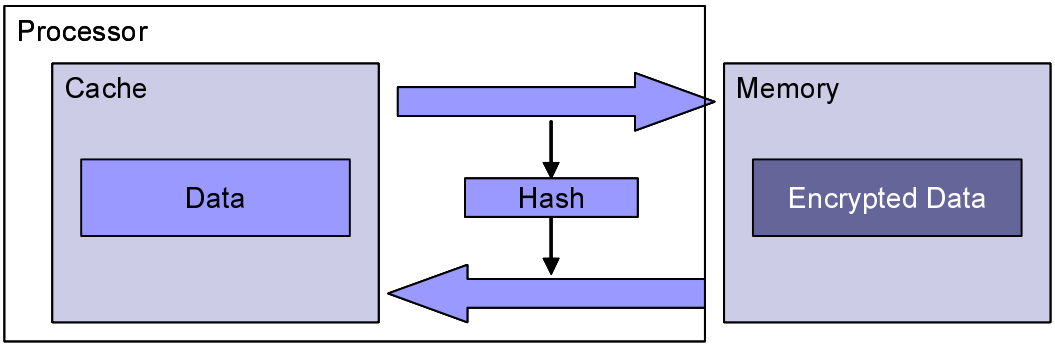

図 3.4: メモリの完全性検証

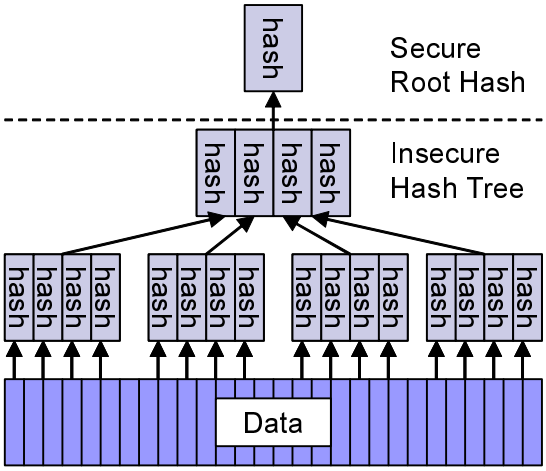

図 3.5: AEGIS のハッシュツリー

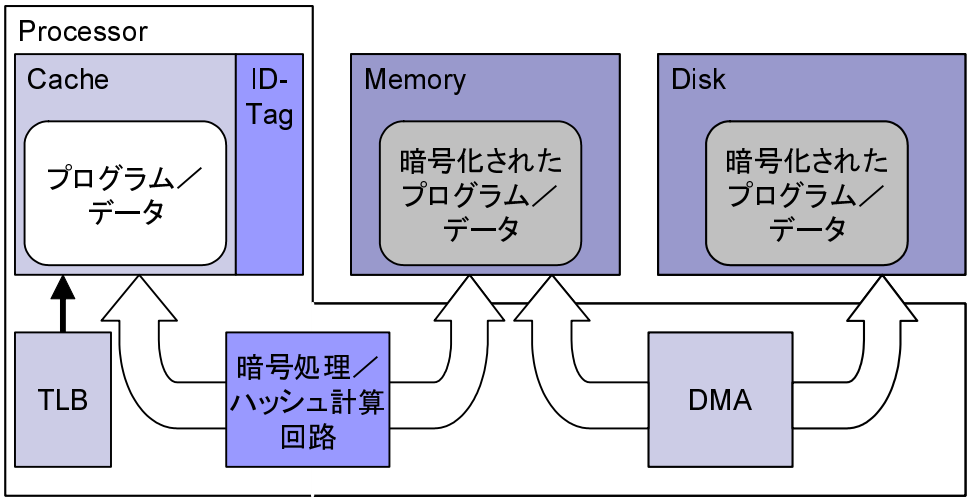

図 3.6: セキュア・プロセッサ全体の動作

3.5 セキュア・プロセッサ全体の動作

図 3.6 に，ここまでに述べたセキュア・プロセッサ全体の動作を示す．

まず，キャッシュやレジスタにタグを追加することで他のプロセスからのアクセスを制限する．これらがメモリに書き出される際には，プロセッサ内部で暗号化を行い，暗号化されたデータを書き出す．さらに，この時にハッシュの計算を行ってハッシュ値をプロセッサで管理し，再び読み込まれたときにデータが改ざんされていないかどうかを検証する．

3.6 セキュア・プロセッサの問題点

3.3.2 で述べたように，メモリアクセスに暗号処理が必要になるため，プログラムの実行速度が低下するという問題がある．これは，サイズの大きなコンテンツを扱うようなプログラムにおいて，特に大きな速度低下を招く．

第4章　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 章で述べたように，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ハードウェアで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，耐ソフトウェアタンパ・プロセッサに用意する機能は次の通りである．

- 機密性の保持 (解析の防止)
  - アクセス制限による秘匿
  - アクセス制限の及ばない場所への暗号化
- 完全性の保持 (改ざんの防止)
  - アクセス制限の及ばない場所での **Message Authentication Code** による改ざん検出

Message Authentication Code Message Authentication Code (MAC) とは，秘密情報を共有する二者間での改ざん検出に用いられる手段である．完全性を検証したいデータと，暗号鍵などの秘密情報を組み合わせたもののハッシュ値をとり，これを MAC としてデータとともに相手に送信する．送られた相手は，届いたデータと自分の持つ秘密情報によってハッシュ値を計算し，送られてきた MAC と一致するかどうかを確認する．これが一致しなければ，改ざんされたことがわかる．

本手法では，二者間の通信ではなく，プロセッサが生成したデータを OS が管理する場合に，OS によって改ざんされていないかどうかを検出するために用いる．図 4.1 に示すように，プロセッサが生成した MAC をデータと共に OS にあずけ，再びプロセッサに戻ってきたときに，データから生成する MAC と受け取った MAC が一致するかどうかを確認する．データ鍵を秘密情報としており，データ鍵は OS に知られることがないため，この方法でデータおよび MAC の改ざんを検出できる．

本研究では上に挙げた機能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 4.2)．

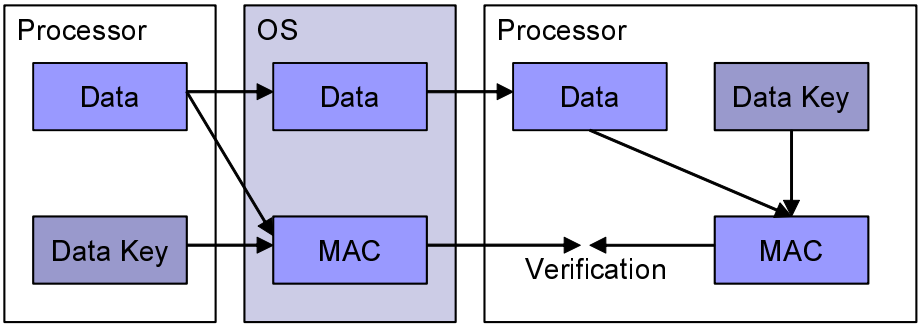

図 4.1: Message Authentication Code の利用

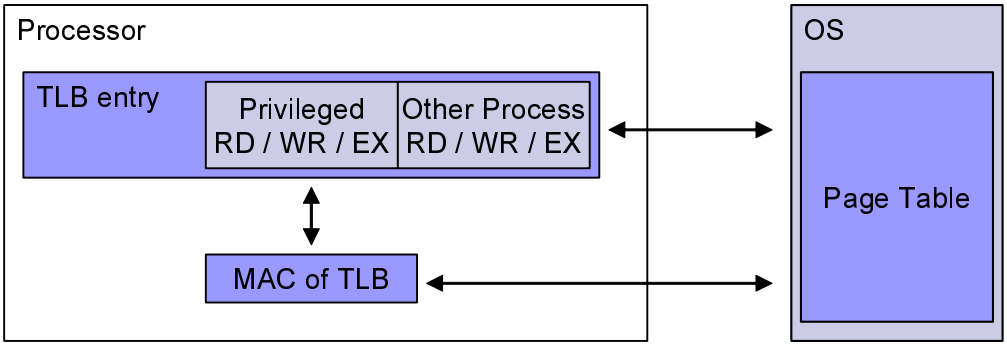

図 4.2: Secure TLB

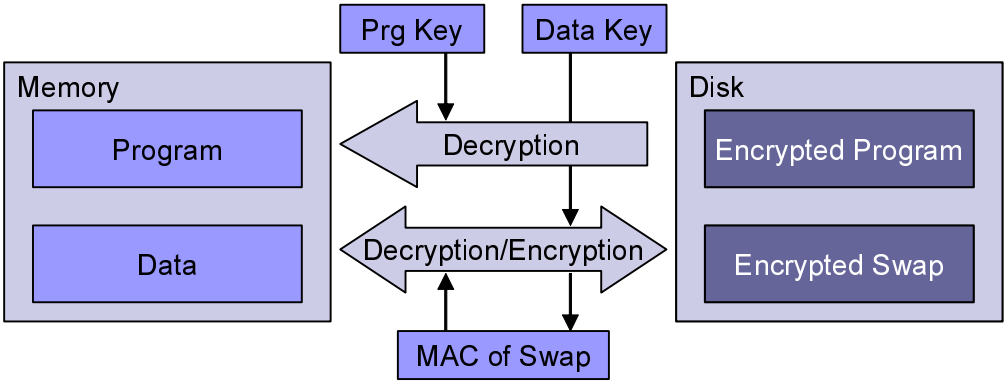

図 4.3: Secure DMA

Secure DMA　Secure DMA は，暗号処理機能と 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 4.3).

以下で，先に述べた耐タンパ性を確保する機能のそれぞれについて説明する.

4.3　アクセス制限による秘匿

まず，プロセッサがアクセス制限できる範囲に関しては，それによってタンパからの保護を行う．具体的には，メモリ保護の拡張を行い，キャッシュやメモリを統一的に秘匿する.

4.3.1　メモリ保護の拡張

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
- 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．

また，この機能は耐ソフトウェアタンパ実行を行っている間には強制されるものであり，OS であってもこれを変更することはできない．

4.4 アクセス制限の及ばない場所への暗号化／MAC による検証

上述のメモリ保護の拡張によってアクセス制限をすることのできない場所にデータを移動するときには，データを暗号化して保護する必要がある．この必要があるのは，

- プログラムファイル
- レジスタ退避
- スワップ

の 3 つである．

さらに，解析を防ぐだけではなく改ざんをも防ぐために

- レジスタ退避
- スワップ
- ページテーブル

に関しては，MAC によって改ざんが行われていないことを確かめる必要がある．プログラムファイルの完全性については，必要ならば自身で検証を行うこととする．暗号化によって解析を防ぐことはできており，意味のある改ざんはできないため，プログラム自身による検証で十分である．

以下で，暗号化と MAC による検証について説明する．

4.4.1 プログラムファイルの暗号化

ディスク上，あるいはネットワーク上などでのプログラムファイルを保護するために，暗号化を行う．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後で述べる．

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．

プログラムファイルの読み込み

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 (図 4.4)．

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 4.4.4 で述べる．

暗号鍵の管理

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．

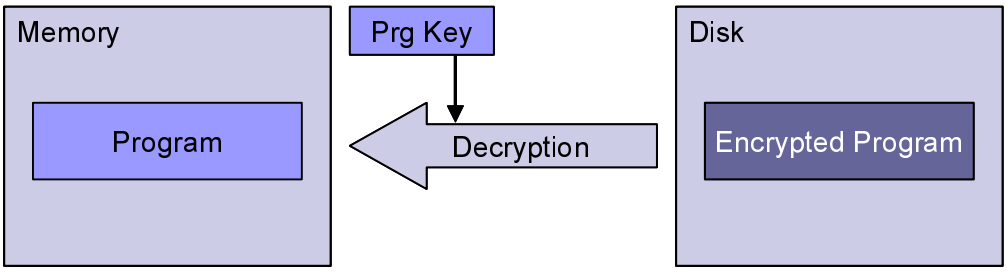

図 4.4: プログラムファイルの読み込み

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．

4.4.2 レジスタ退避

プロセス切り替えが発生するとき，レジスタの退避を行うが，このレジスタ値は OS が管理して再度プロセスが実行される際に読み込まれるため，OS による解析や改ざんが行われないようにする必要がある．

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 の生成を行う．暗号化は，割り込み直前のプロセス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 (図 4.5)．

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．

4.4.3 スワップ処理

物理メモリが不足すると，OS によってスワップ処理が行われ，メモリ上のデータが二次記憶上に書き出されるが，二次記憶上にはメモリ保護のアクセス

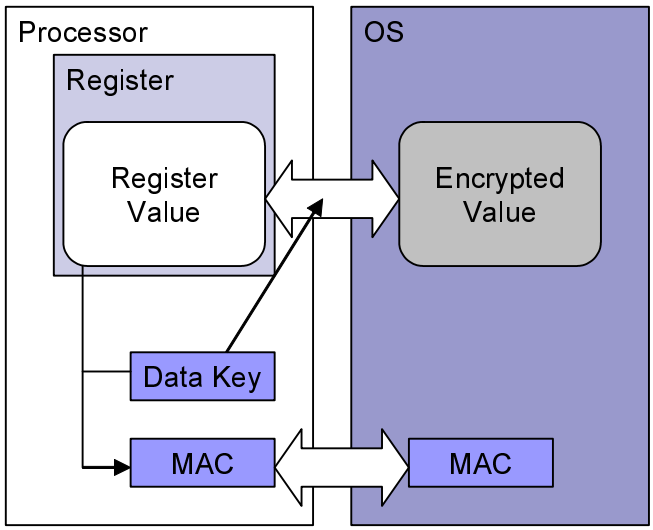

図 4.5: レジスタの退避

制限は及ばない．そこで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 (図 4.6)．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．

4.4.4 ページテーブルの改ざん防止

メモリ保護の拡張によって，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 が管理している．ページテーブル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．これを防ぐために，ページテーブルの改ざん防止が必要となる．

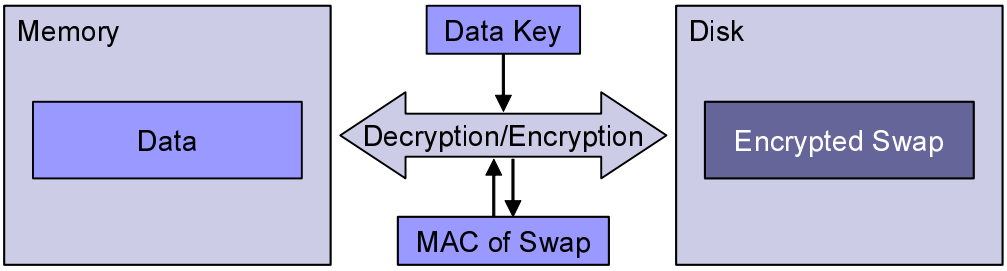

図 4.6: スワップ処理

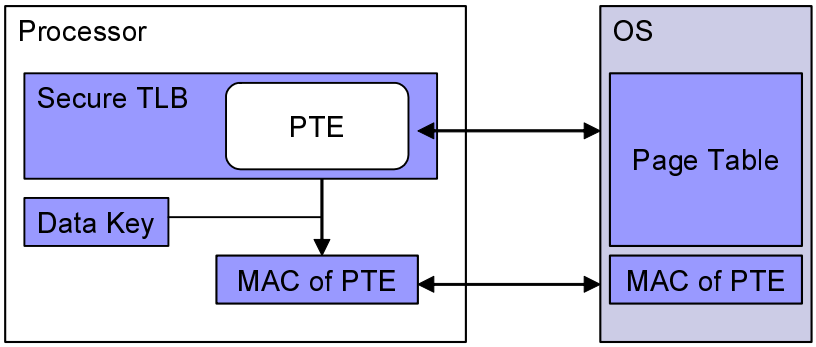

図 4.7: ページテーブルの改ざん防止

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 (図 4.7)．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．

4.5 Secure TLB と Secure DMA の協調

上述の方法によって，アクセス制限や暗号化，MAC 検証などが行われるが，DMA に関する問題が残っている．

- アクセス権限のないページの DMA 転送

- DMA 転送に関する PTE の改ざん

の 2 点について，以下で Secure TLB と Secure DMA の協調による対策を述べる．

4.5.1 アクセス権限を持たない DMA 転送の防止

メモリに関しては，先に述べたようにメモリ保護の拡張によって特権を持つプロセスからのアクセスであっても防ぐことが可能になったが，DMA 転送によるアクセスにはこれは影響していない．

そのため，DMA 転送によってアクセス権限を持っていないページを転送できてしまう．これができると，保護されるべきページであっても，DMA 転送によってディスクに転送してからアクセスすることが可能になる．

これを防ぐために，Secure TLB が持つアクセス権限情報を Secure DMA が利用し，転送するページのページテーブルエントリを参照して DMA を行うプロセスが正しく権限を持っているか確認する．

この確認によって，権限を持たない DMA 転送は防げる．

4.5.2 DMA に関するページテーブルの改ざん防止

ページテーブルの改ざん防止については 4.4.4 で説明したが，DMA 転送が行われるとページテーブルを変更しなければならない．この DMA 転送に関するページテーブルの変更を OS に任せると，改ざんが行われる可能性がある．例えば，DMA 転送していないのにしたことにすると，その領域は空き領域扱いになり，全てのプロセスからアクセス可能になってしまう．

これを防ぐため，Secure DMA が行った転送の情報を，Secure TLB 上の PTE に反映させることを行う．DMA に関するページテーブルの変更は Secure DMA からの情報を受けた Secure TLB が行い，OS に任せることはしない．

この方法をとることで，DMA 転送におけるページテーブルの改ざんを防ぐことができる．

4.6 耐ソフトウェアタンパ・プロセッサのまとめ

耐ソフトウェアタンパ・プロセッサでは，以下に挙げた機能によってソフトタンパを防ぐ．全体図は図 4.8 のようになる．

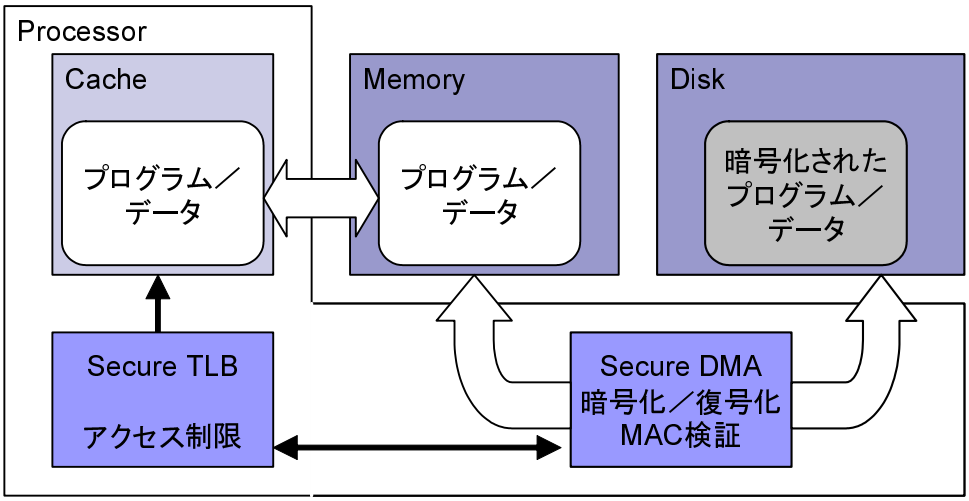

図 4.8: 耐ソフトウェアタンパ・プロセッサの全体図

- プログラム鍵／データ鍵の管理
- メモリ保護の拡張
- ディスク上／ OS 管理下での暗号化
- ディスク上／ OS 管理下にあったデータの MAC による検証
- Secure TLB と Secure DMA の協調

第5章　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 5.1 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，150 サイクル多い 300 サイクルとした．

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検証機能があるため，TLB のアクセスレイテンシをベースラインよりも 15 サイクル多い 45 サイクルとした．

5.1　プログラム実行速度の変化

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 5.1 である．ベースラインの IPC(instruction per cycle) 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・プロセッサからは平均で 6.7%，最大で 24.7%の速度向上があった．

5.2　メモリアクセス速度の影響

メモリアクセス速度が実行速度に与える影響を調べるため，ベースラインにおいてメモリアクセスによる遅延を 100 クロックから 400 クロックまで変化させたときの IPC の相対的な変化を示した結果が図 5.2 である．bzip2，gap，mcf といったベンチマークでは，メモリアクセス速度が遅くなるにしたがって実行速度が大きく低下しており，メモリアクセス速度の影響を大きく受けるプログラムであると言える．これらのベンチマークは，図 5.1 に示したセキュア・プロセッサと耐ソフトウェアタンパ・プロセッサの速度比較においてセキュア・

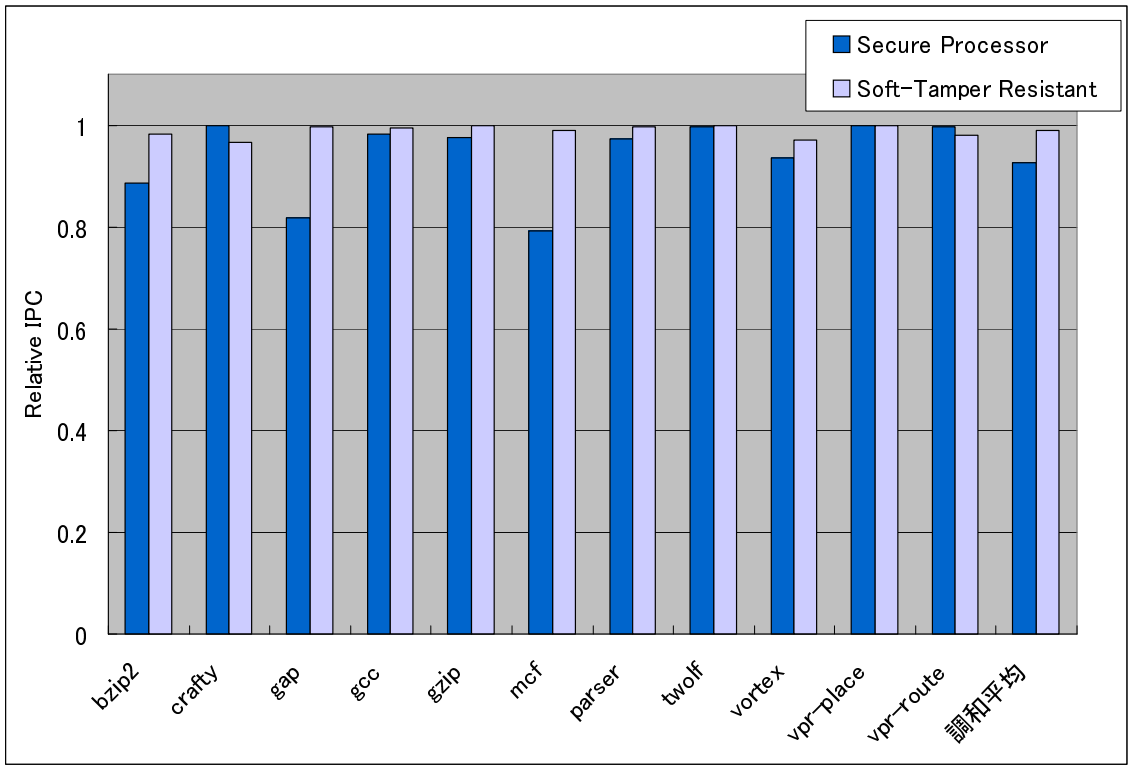

図 5.1: 相対 IPC の比較

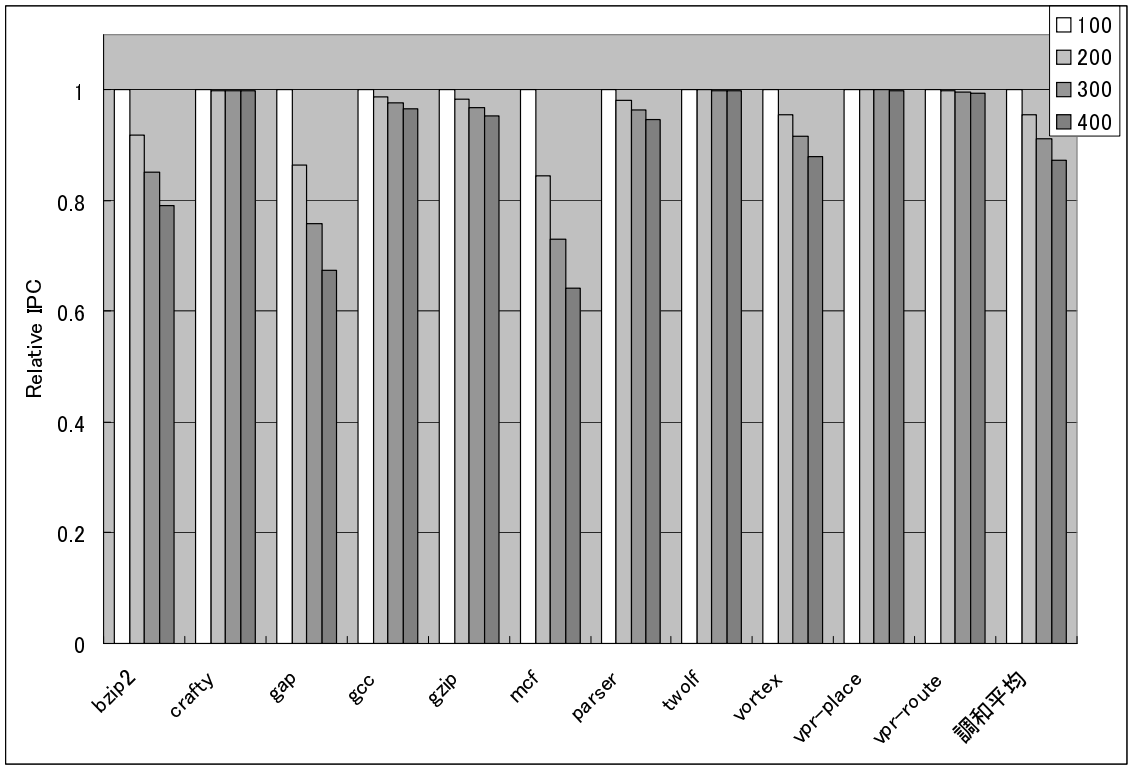

図 5.2: メモリアクセス速度の与える実行速度への影響

プロセッサの速度が大きく低下しているものであることから，メモリアクセス速度が実行速度に与える影響の大きいプログラムは，セキュア・プロセッサでは大きく速度が低下することがわかる．

5.3 TLB ミス遅延増大の影響

TLB ミス遅延の増大が実行速度に与える影響を調べるため，ベースラインにおいて TLB ミス遅延を 20 クロックから 60 クロックまで変化させたときの IPC の相対的な変化を示した結果が図 5.3 である．vortex や vpr-route などのベンチマークでは，TLB ミス遅延が大きくなるにつれて実行速度が低下しており，TLB ミス遅延の影響を受けるプログラムであると言える．図 5.1 に示した速度比較において，これらのベンチマークでは提案手法の速度がベースラインから低下しており，TLB ミス遅延の影響を受けて遅くなったと考えられる．

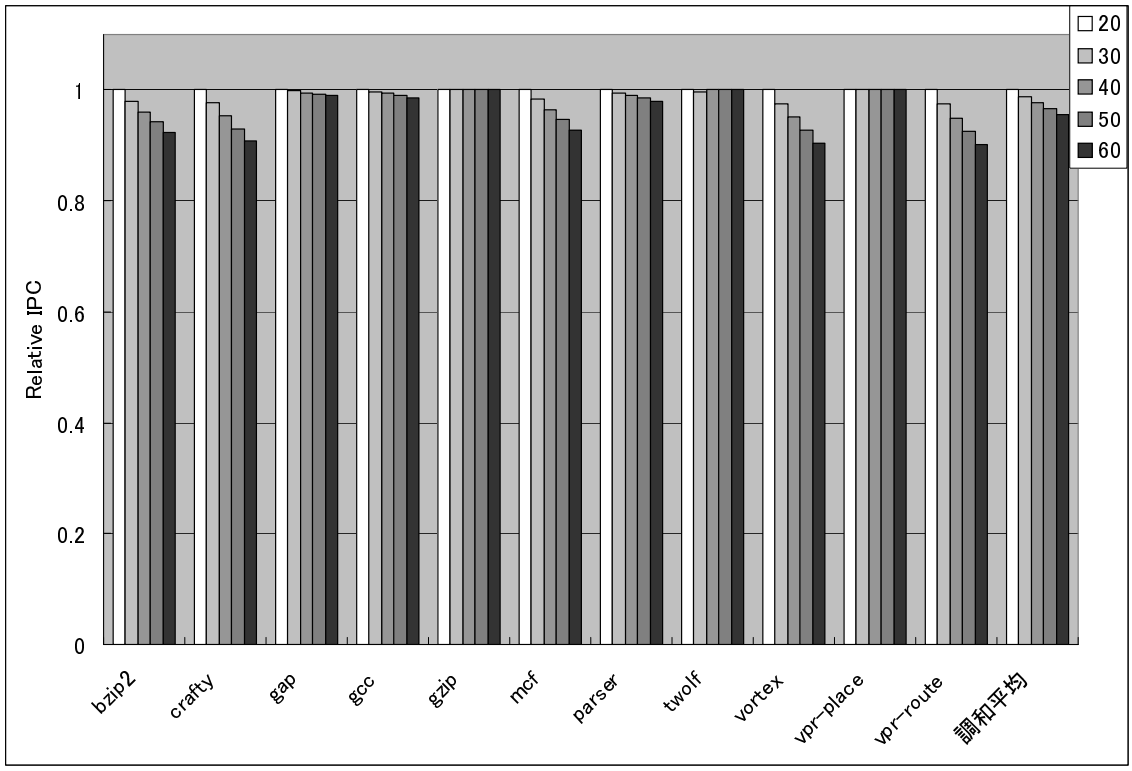

図 5.3: TLB ミス遅延の与える実行速度への影響

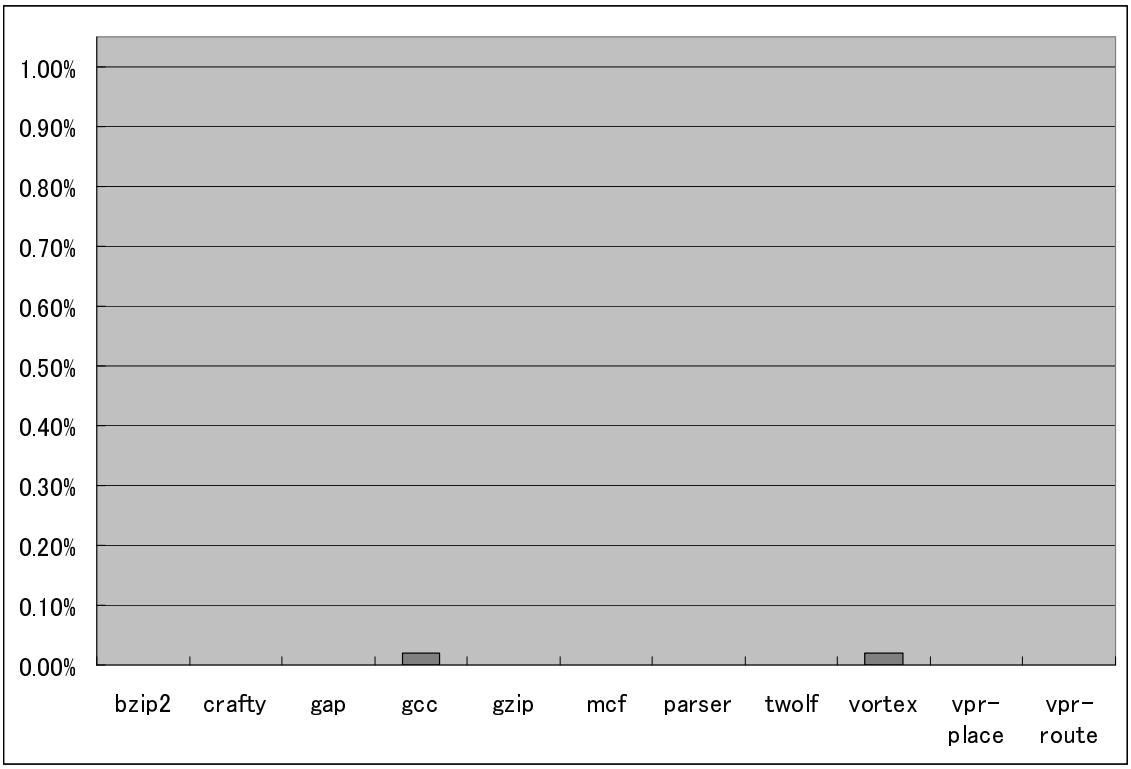

図 5.4: Instruction TLB ミス率

しかし，TLB ミス遅延の影響はメモリアクセス速度ほど大きくはない．これは，TLB ミス率が非常に低いためである．図 5.4 に，ベースラインでの各ベンチマークの Instruction TLB(iTLB) ミス率，図 5.5 に Data TLB(dTLB) ミス率を示す．iTLB ミス率は，すべてのベンチマークで無視できるほど小さく，最大でも 0.02%であるため，影響はないと言える．dTLB ミス率は，bzip2 や crafty，mcf といったベンチマークで 0.5%を超えており，図 5.3 に示したように影響が現れているが，一部のベンチマークを除いては，メモリアクセス速度の影響に比べて小さい．

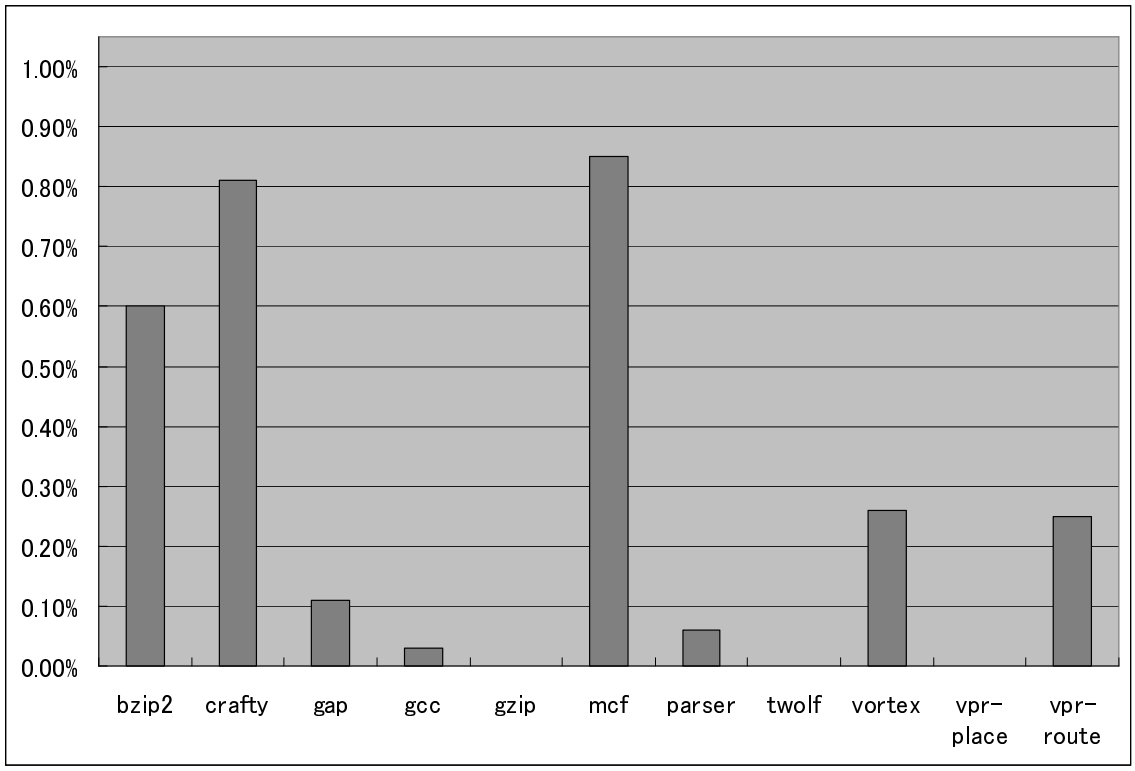

図 5.5: Data TLB ミス率

表 5.1: 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		

第6章　おわりに

6.1　まとめ

本論文では，耐タンパの重要性について述べ，その中でも特に耐ソフトタンパが重要であることについて説明した．そこで，従来手法であるセキュア・プロセッサよりも高速に実行が可能な，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．この耐ソフトウェアタンパ・プロセッサについて，耐タンパ性を確保するために行う

- メモリ保護の拡張
- 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／スワップの検証

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平均で 6.7%，最大で 24.7%の速度向上が見込めることをシミュレータによって確認し，この手法のセキュア・プロセッサに対する優位性を示した．

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．これらは，OS までを含んだシミュレータを用いることによって評価を行うことができると考える．

さらに，Secure TLB、Secure DMA の詳細について検討を行う余地がある．たとえば，エントリの検証と並行しての実行が動作の完全性に影響を及ぼさないことを示すことができれば，Secure TLB におけるオーバヘッドはなくなり，さらに本手法の実行速度は向上する．

また，MAC 対するリプレイ・アタックの効率的な防止策についても詳細を検討する必要がある．

参考文献

[1] Simplescalar. http://www.simplescalar.com/.

[2] Trusted Computing Group. https://www.trustedcomputinggroup.org/.

[3] David Aucsmith. Tamper Resistant Software: An Implementation. In *Information Hiding*, pp. 317–333, 1996.

[4] 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.

[5] 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.

[6] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical report, July 1997.

[7] 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.

[8] 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.

[9] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John C. Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Architectural Support for Programming Languages and Operating Systems*, pp. 168–177, 2000.

[10] 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.

[11] G. Edward Suh, Charles W. O'Donnell, Ishan Sachdev, and Srinivas Devadas. Design and implementation of a single-chip secure processor using physical random functions. In *Proceedings of the International Symposium on Computer Architecture*, 2005.

[12] Trusted Computing Group. *TCG Specification Architecture Overview*. Trusted Computing Group, 1.2 edition, 2004.

[13] Trusted Computing Group. *TPM Main Part 1 Design Principles*. Trusted Computing Group, 1.2 revision 85 edition, 2005.

[14] Trusted Computing Group. *TPM Main Part 2 TPM Structures*. Trusted Computing Group, 1.2 level 2 revision 85 edition, 2005.

[15] Trusted Computing Group. *TPM Main Part 3 Commands*. Trusted Computing Group, 1.2 level 2 revision 85 edition, 2005.

[16] 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.

[17] 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.

[18] 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. 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. 情報処理学会論文誌 コンピューティングシステム, Vol. 47, No. 18, November 2006.

[19] 春木洋美, 橋本幹生, 川端健. 耐タンパプロセッサ (L-MSP) システムの実装とプロセス管理. 暗号と情報セキュリティシンポジウム, 2004.

発表文献

主著論文

- プログラムの振る舞い秘匿のための動的アドレス変換
  清水 一人, 卒業論文, 東京大学 工学部 (Feb. 2005)
- プログラムの振る舞い秘匿のための動的アドレス変換
  清水 一人, 高田 正法, 入江 英嗣, 坂井 修一
  情報処理学会 研究会報告 ARC 2005-ARC-164(Aug. 2005)
- 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
  情報処理学会 研究会報告 ARC 2007-ARC-172(Mar. 2007)(発表予定)

共著論文

- クラスタ型プロセッサにおける SMT 実行
  高田 正法, 入江 英嗣, 服部 直也, 渡邊 翔太, 清水 一人, 坂井 修一
  情報処理学会 研究会報告 ARC 2005-ARC-162(Mar. 2005)
- 超ディペンダブル・プロセッサアーキテクチャの構想
  入江 英嗣, 荻野 健, 勝沼 聡, 清水 一人, 栗田 弘之, 五島 正裕, 坂井 修一
  電子情報通信学会技術研究報告 CPSY2006-1〜12, Vol.106, No.3, pp.49-54 (Apr. 2006)
- アドレスオフセットに着目したデータフロー追跡による注入攻撃の検出
  勝沼 聡, 栗田 弘之, 塩谷亮太, 清水 一人, 入江 英嗣, 五島 正裕, 坂井 修一
  先進的計算基盤システムシンポジウム 2006(SACSIS2006), Vol.2006, No.5, pp.515-524 (May. 2006)

- Base Address Recognition with Data Flow Tracking for Injection Attack Detection
  Satoshi Katsunuma, Hiroyuki Kurita, Ryota Shioya, Kazuto Shimizu, Hidetsugu Irie, Masahiro Goshima and Shuichi Sakai
  IEEE International Symposium on Pacific Rim Dependable Computing (PRDC 2006), pp.165-172 (Dec. 2006)

謝辞

非常に多くの方々から多大なご指導，ご協力，励ましを頂き，本論文を完成させることができました．この場を借りて，感謝の意を表したいと思います．

指導教官である坂井修一教授には学部 4 年からの 3 年間に渡って多くのご指導，ご助言を頂きました．いろいろとご心配をおかけしたこともありましたが，非常に細かく気を遣ってくださり，様々な面で助けられました．本当にありがとうございました．

また，五島正裕助教授からも，大変多くのご指導を頂きました．いろいろと至らない点の多い私に根気強く指導してくださいました．常にネガティブな思考の私に対し，ポジティブな考えで研究を直接引っ張っていただきました．本研究がこの形になったのは，五島助教授のおかげです．ありがとうございました．

八木原晴水さん，月村美和さん，内田杏さんには，研究室における設備の導入や各種事務手続きなど，研究室で過ごすための様々なご支援を頂きました．

入江英嗣博士，栗田弘之氏，勝沼聡氏をはじめ，ディペンダブル・グループのメンバーの皆様には，ミーティングにおける議論を通して，貴重なご意見を頂きました．特に入江博士には，3 年間を通して，グループのリーダーとして指導していただきました．ありがとうございました．

研究室の計算機，及びネットワークの構築・管理のために，塩谷亮太氏，渡辺憲一氏，一林宏憲氏には多大なご尽力を賜りました．心より感謝致します．

その他の方々にも，色々な面でお世話になりました．

また，塩谷亮太氏には研究室に入る前からの付き合いがあり，研究とは関係のない所でもいろいろとお世話になりました．“ よい時も悪い時も，塩谷はそこにいた． スパシーボ，塩谷． ”

本研究は，21 世紀 COE「情報科学技術戦略コア」，及び，科学技術振興機構 CREST,「ディペンダブル情報基盤」による支援を受けて行われました．