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

2005－ＣＳＥＣ－30（32）
2005／7／21

マルチタスクOSメモリ保護方式の実装と評価 — 少資源プラットフォーム向けCMSおよびデータ改竄検出手法—

稲村　雄†　江頭　徹†　竹下　敦†

† 株式会社NTTドコモ マルチメディア研究所　〒239-8536 神奈川県横須賀市光の丘3-5
E-mail: †{jane,egashira,takeshita}@mml.yrp.nttdocomo.co.jp

あらまし　筆者らは，マルチタスクOSにおいてアドレス空間に置かれたデータを保護するための手法として暗号化メモリシステム (Cryptographic Memory System, CMS) を提案しているが，本稿では先般発表した少資源プラットフォーム向け暗号化メモリシステムと改竄検出手法に関して，両者を統合した形での試験実装を行なったので，その実装方式および評価を報告する．

キーワード　暗号技術，メモリ保護，マルチタスクOS

Evaluation of Memory Protecting Scheme in Multitask OSes — A Cryptographic Memory System for Resource-poor Platforms & Tampering Detection—

INAMURA YU†, EGASHIRA TORU†, and TAKESHITA ATSUSHI†

† Multimedia Laboratory, NTT DoCoMo, Inc.　3-5, Hikarinooka, Yokosuka, Kanagawa, 239-8536 Japan
E-mail: †{jane,egashira,takeshita}@mml.yrp.nttdocomo.co.jp

Abstract　The authors have been proposing a method, called Cryptographic Memory System (CMS), to protect the data stored in the address space of a process in the typical modern multi-task operating systems. Recently we have reported two new schemes, one of which optimizes the kernel memory usage by dicreasing the number of the keys used and the other makes it possible to detect the tampering attack from another process. Here we will present a new proof-of-concept implementation for these new schemes integrated and also reveal the evaluation results from this implementation.

Key words　Cryptography, Memory Protection, Multi Task OS

1. はじめに

一般に，複数タスクの同時並列的な実行が可能であるマルチタスクOSにおいては，一つのタスクが他タスクの持つアドレス空間(注1)上のデータにアクセスすることが可能である場合が多い．通常，この機能はプログラムの実行時デバッグを実現するために利用されるが，悪意を持ったプログラムがこの機能を利用することで，

- 他タスクのアドレス空間からの機密情報の盗み出し
- 他タスクのアドレス空間上のデータの改竄

という形の攻撃が可能となるという脆弱性も存在する(注2)．

一般に，このような脆弱性を改善するためには，SELinux [1] に見られるように特権を排除した必須アクセス制御技術(注3)が利用されるが，たとえMACの下であったとしても，他タスクのアドレス空間へのアクセス可能性が本質的であるようなタスク(注4)の存在が必要なのであれば，脆弱性は残ることとなる．

このような事情を鑑み，筆者らは一般的な任意アクセス制御技術(注5)のみを提供するようなOS上であっても，アドレス空間上の重要情報を保護可能なシステムとして，暗号化メモリシステム (Cryptographic Memory System, CMS) を提案してい

(注1)：プログラムから順序付けられた指標 (=アドレス) を用いてアクセスすることのできる空間．通常アドレス空間に結び付いているのは物理メモリだが，ファイル等の他のオブジェクトに結び付けて利用することもある

(注2)：4.1.1節で用いるプログラムが同脆弱性の利用例である
(注3)：Mandatory Access Control, MAC
(注4)：デバッガタスク等
(注5)：Descrete Access Control, DAC

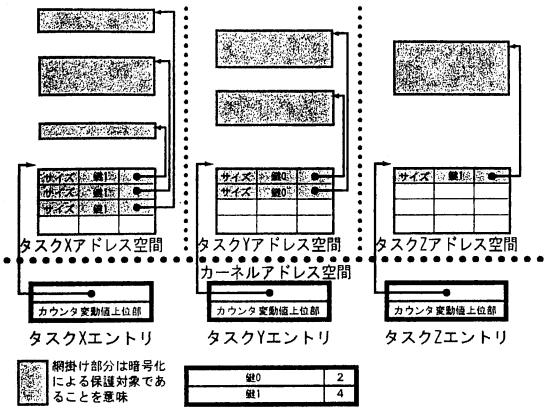

図 1　少資源プラットフォーム向け暗号化メモリシステム管理構造

Fig. 1　Management Structure in Cryptographic Memory System for Resource-poor Platforms

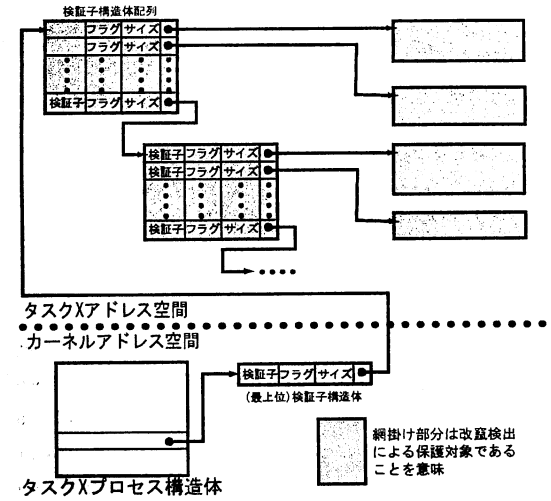

図 2　データ改竄検出手法管理構造

Fig. 2　Management Structure in Tampering Detection System

る [2]．また，その拡張として，

- PDA のように資源の乏しいプラットフォーム向けのメモリ使用量を抑えたバージョン
- 他タスクによるメモリデータ改竄検出機能

を提案した [3], [4]．

CMS およびその拡張方式では，マルチタスク OS で必須のコンテキストスイッチ処理のタイミングで，実行を中断されるタスクアドレス空間中の重要情報格納部分に対する保護処理(注6)を，同じく実行再開されるタスクの重要情報格納部に対して逆処理(注7)を OS カーネルが実施することで，上記のようなアドレス空間上の情報への攻撃に対する防御を提供している．

(注6)：暗号化もしくは検証子生成

(注7)：復号もしくは検証子検証

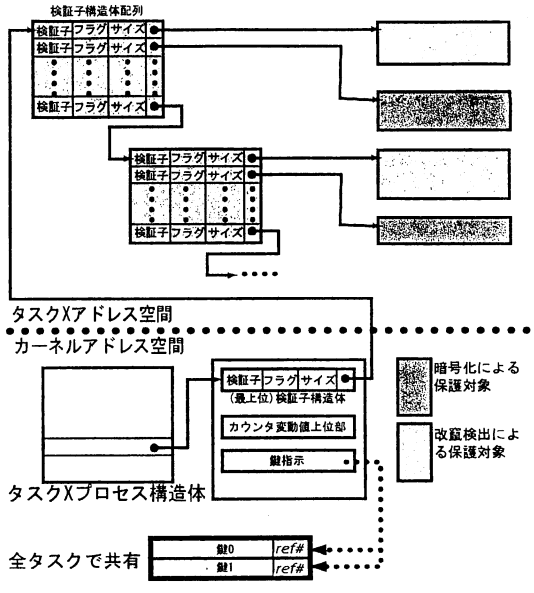

図 3　統合システム管理構造

Fig. 3　Management Structure in the Integrated Method

このため，CMS は OS が MAC/DAC どちらの方式を採用していたとしても，アドレス空間上の重要情報を確実に保護することが可能である．

本稿では，[3] および [4] で報告した処理方式を統合することで，アドレス空間上データに対して秘匿化および改竄検出を同時もしくは個別に適用可能とした実装方式を述べるとともに，同方式を Linux カーネルに試験実装したシステムに基づく評価結果を提示する．

以降の構成は以下の通りである．まず，2. 章で統合実装方式の概要を述べ，3. 章で実際に行なった同方式の試験実装内容を紹介した後，4. 章で同試験実装から得た評価結果を提示し，5. 章で全体のまとめを行なう．

2.　統合処理方式概要

先般報告した [3], [4] における管理構造概要を図 1，2 に示す．コンテキストスイッチ時にカーネルは，ここで示された管理構造を辿ることによって必要な処理(注8)を適用する．上記管理構造への登録はプログラム中での指定を受けてなされるため，アドレス空間中の必要な領域のみを保護することが可能となっている．

図から明らかな通り，この二方式は

- カーネル空間に置かれるルートデータ
- ユーザ空間に置かれる管理用データ配列

という構成上の類似点を持っている．この特徴を有効に活用するために，両者を統合した形で実装を行なうこととした．具体

(注8)：管理構造の末端として示されている保護領域に対して，図 1 では暗号化/復号，図 2 では検証子の生成と検証子構造体配列要素への格納/比較

表 1　試験実装環境

暗号アルゴリズム	CTR モード AES
鍵長	128bit
検証子生成	MD5 ($datasize > 128bit$)
	データ自体 ($datasize \leq 128bit$)

的には，図 3 に示したような形となる．

[3] で提案した通り，保護領域暗号化用の鍵は現行用および鍵更新を遅延型で実現するための旧鍵保存用の二個をシステム全体で共有する．また，プロセス毎のデータとなるカウンタ変動値上位部および鍵指示フラグが暗号化メモリシステム部では必要となる．

改竄検出部に関しては [4] で提案した検証子構造体形式をほぼそのまま踏襲しているが，同構造体のフラグとしては [4] で定義された要素に加えて，保護対象となるデータへの暗号化の必要性を示すためのビットを新たに付け加えている．

図 3 で示した検証子構造体配列に格納された情報には機密性は必要とされないため，同部には改竄検出機能のみを提供することで処理オーバヘッドを軽減した．なお，暗号化のみを適用する領域に関しては，図示した検証子構造体中の検証子格納部分はデッドスペース化するため，効率を損なっている．適用する保護処理に応じて変更するようにこの部分を最適化することも可能だが，それによって増加する処理方式の複雑性と，上述の通り検証子構造体に関しては改竄検出のみ適用という点から予想される処理オーバヘッドの小ささとを勘案し，デッドスペースが存在したとしても許容範囲内のオーバヘッドに抑えられると判断した．

4. 章で示す性能測定結果を考慮すれば，この判断は適切だったと評価している．

3. 試験実装

ここでは，前章で概要を説明した統合処理方式の試験実装に関して，利用アルゴリズム等の詳細をまとめる．

3.1 利用暗号アルゴリズム

試験実装で利用した暗号アルゴリズムは，表 1 の通りである．

[3] で説明した通り，暗号化メモリシステム部分はブロック暗号 CTR モードの利用による鍵数削減実現が特徴的であり，現状十分な安全性を実現可能と評価できる 128bit 鍵 AES を同モードにて使用している．また，[4] で説明した通り，改竄検出部分は衝突一致性は必要とせず汎用的一方向性の確保のみで十分であるため，近年，衝突一致性に関して懸念が表明されている MD5 [5], [6] を用いることとした．その主たる理由は SHA-1 等の一方向ハッシュ関数と比較して動作が高速なためである．

3.2 処理タイミング

今回の試験実装において，OS カーネルが暗号化／復号・検証子生成／検証等の処理を実施するタイミングは，[2] で提示したのと同じく，保護領域を保持するタスクがコンテキストアウト／インされる時点であり，[4] で提案した遅延検証処理，すなわち，保護領域に対する検証処理をコンテキストインのタイミングではなく実際に当該領域がプログラムによってアクセスされる時点まで遅延するという最適化は実装していない．

表 2　試験実装環境

CPU	Intel Pentium III 852 MHz
OS	RedHat9.0
カーネル	2.4.20-8

3.3 改竄検出時の処理

改竄検出方式を実装する場合に問題となるのが，実際に保護領域に対する改竄行為の発生が検出された際の処置方法である．改竄が検出された場合にはタスクをただちに終了させるという選択もあり得るだろうが，本試験実装では [4] で提案した通り，該タスクに対して特定シグナル(注9)を送信するという方式を取った．この方式であれば，ソース内で特に対処していない場合には改竄検出と同時にユーザアプリケーションは終了することとなるし，ソース内で該シグナルに対するハンドラを設定しておくことで，改竄検出後に必要な措置を取るといった対処も可能となる．

4. 評　価

本章では，試験実装から得た実験結果により同方式の効果・性能等を評価する．

まず，試験実装に用いた CPU/OS 等の環境を表 2 にまとめる．[2] およびその試験段階だった [7] ではともに BSD4.4 をベースとした FreeBSD および OpenBSD カーネルに対する実装だったが，それを拡張した方式を今回上記 OS とは出自の異なる Linux カーネルに首尾良く実装できたことから，同コンセプトの高い汎用性を証明できたのではないかと考えている．

4.1 効　果

4.1.1 単一鍵に関する CTR モード/CBC モードの影響の確認

ここでは，[3] で議論した通り，システムで単一の鍵を利用することに由来する問題の存在を確認した．具体的には，本試験実装で実現したような単一鍵利用システムを，CTR モードで利用する場合と，CBC モードで利用する場合で，[3] で述べたヘルパプログラムを利用した攻撃に対する耐性が変わることを確かめた．

まず，ptrace システムコールを用いて一つのタスクのアドレス空間から別のタスクのアドレス空間へデータを複写するような攻撃プログラムを用意した．攻撃の疑似的なターゲットとしては，一定量(注10)の保護領域を暗号化保護属性で確保した後に該保護領域を疑似乱数データで埋め，その後，ユーザの入力がある度に該保護領域の内容を出力するようなプログラムを用意した．また，該プログラムは疑似的なヘルパアプリケーションとしても用いている．

本実装の比較対象としての単一鍵 CBC モード暗号化保護を実施するシステムとしては，[2] で報告した FreeBSD ベースの

(注9)：デフォルトでは USR1 だが，カーネル make 時のオプションとして変更可能

(注10)：160Byte

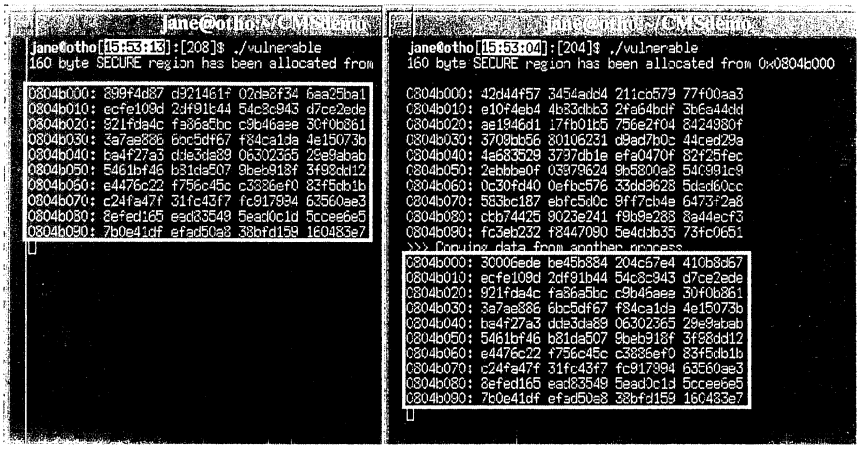

図 4　CBC モード暗号化での実験

Fig. 4　Experiment with CBC Mode Encryption

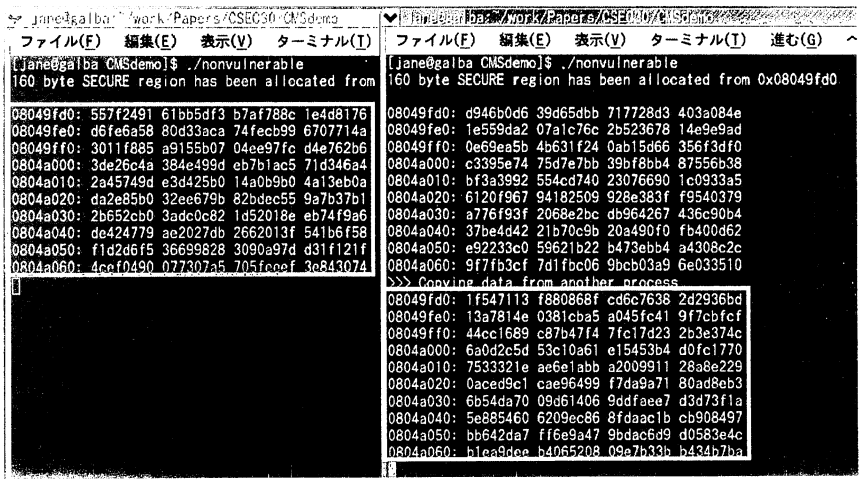

図 5　CTR モード暗号化での実験

Fig. 5　Experiment with CTR Mode Encryption

CMS 実装を，プロセス毎ではなくシステム全体で同じ鍵を用いるように修正した版を作成した．

上記比較対象実装および本実装において，

（1）　ターゲットおよびヘルパとして後者のプログラムを 2 個立ち上げ

（2）　両タスクで一回保護領域の内容を表示

（3）　攻撃プログラムによりターゲットタスクからヘルパタスクへ保護領域の内容を複写

（4）　ヘルパタスクで再度保護領域の内容を表示

という操作を行なった結果を，図 4 および図 5 に示す．図 4 から明らかな通り，CBC モード暗号化を単一鍵で用いるシステムでは，複写操作後にヘルパタスクが表示する内容 (図 4 枠線部) は，IV がタスク毎に異なることに基づく第一ブロックの違い以外は，同図左でターゲットタスクによって表示されている内容と同一である．これは，[3] での議論を裏付ける結果と言えるだろう．

一方，図 5 では，複写操作後にヘルパタスクが表示する保護領域の内容は，ターゲットタスクのものとも複写操作前のヘルパタスクのものとも類時点のないまったく別の内容となっている．

以上から，CTR モードを利用することにより，ヘルパタスクが利用可能な場合ですら安全な単一鍵利用システムが実現できたことが示された．

4.1.2　利用カーネルメモリ容量の削減

ここでは，本実装における利用カーネルメモリ容量が，[2] で提示したバージョンと比較してどの程度削減できたかに関して評価を行なう．

まず，本実装において保護領域を保持するタスク毎に必要となるカーネルメモリ容量は

カウンタ変動値　4byte

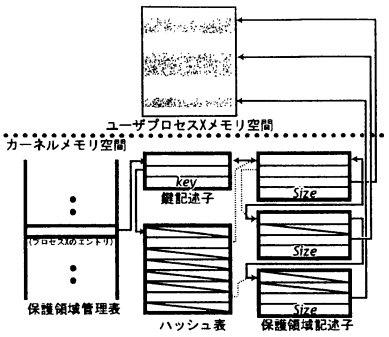

図 6　既存方式における保護領域管理

Fig. 6　Protected region management in old CMS

鍵指示他フラグ　1byte

最上位検証子構造体　24byte

の計 29byte である.

一方，CMS のオリジナルバージョン [2] では，タスク毎に必要となるカーネルメモリ容量は，

鍵記述子　24byte

ハッシュ表　128byte

保護領域記述子　1 保護領域辺り 20byte

であるため (図 6 参照)，$162+20 \times N$[(注11)] byte となる.

以上から，保護領域が一つのみである場合でもカーネルメモリ使用量は $\frac{1}{6}$ 以下に削減できたことになる．また，本実装の場合には利用カーネルメモリ量は保護領域の数に依存しないため，この差は保護領域の数が増加するにしたがってより顕著となる．これは，一般にカーネルメモリには swap 処理を適用し難いことを鑑みれば，非常に有効な最適化と考えることができるだろう.

[2] では保護領域に対する改竄検出は行なえなかったため，保護領域の管理構造をすべてカーネル空間に置くことで安全性を確保する必要があったという点が，利用カーネルメモリ容量の増大に影響していたと考えられる.

また，保護領域の指定解除を高速に実現するために，[2] ではハッシュ表および双方向リンクを用いていたこともカーネルメモリ消費量の増加の一因である．この点に関しては速度と利用メモリ容量のトレードオフが存在するが，本実装でとった方式は，特にメモリ資源の乏しいプラットフォームでの実現には適切なものだと評価できる.

4.2 性　　能

性能に関しては，

- 単一保護領域時での保護領域サイズの影響
- 複数保護領域時での保護領域数の影響

という二つの観点からの評価を行なった.

4.2.1 保護領域サイズに関する処理オーバヘッド

一つだけ確保する保護領域のサイズを 1Kbyte[(注12)]から 4 倍ずつ 1024Kbyte まで増加させたとき，改竄検出および暗号化を独立もしくは同時に適用した場合に，どの程度のオーバヘッドが課せられるかを測定した．測定用プログラムは，指定されたサイズ/保護方式[(注13)]の保護領域を確保した後に，同保護領域全体に対して疑似乱数データを繰り返し書き込む，というだけのものである．疑似乱数データの書き込み回数は，保護領域のサイズに関わらず一定となるように調整している.

比較対象として，単純に `malloc` で確保しただけの同サイズの領域に対して同じ処理を行なった場合にかかる時間と，上記プログラムを改竄検出のみ，暗号化のみ，改竄検出/暗号化同時適用という三種類の条件の下で実行させた場合の実行時間を図示したのが，図 7 である.

図から明らかな通り，改竄検出のみ ⇒ 暗号化のみ ⇒ 同時適用の順で処理オーバヘッドは大きくなっている．改竄検出のみ適用の場合には，保護領域サイズとして [3] で想定する資源の乏しいプラットフォームで通常必要とされるメモリ量より大きくなりそうな 1Mbyte を保護領域として確保したときであっても，25%程度の性能低下で抑えられることが確認できた.

また，改竄検出と暗号化を同時に適用した場合であっても，保護領域サイズが 256Kbyte 程度までは 20%以下の性能低下で済むことも分かった．アプリケーションによって事情は異なるが，特に重要な情報のみを保護するのであれば，256Kbyte 以下というサイズが十分であるケースも多いだろう．また，ニーズに応じて改竄検出のみと使い分けることで，より多くのアドレス空間を保護可能となるため，多くの場面で現実的な保護策として本方式が利用可能となることを期待している.

4.2.2 保護領域数に関する処理オーバヘッド

現在の実装では，検証子構造体配列はページサイズである 4Kbyte 単位で割り当て，そのすべての要素が消費されたときには新たなページを割り当てることとしている (図 3 参照)．そのため，検証子構造体の大きさが前述の通り 24byte であることから，1 ページで管理可能な保護領域の最大値は 169 個[(注14)]となる.

このような処理方式となっているため，確保する保護領域が 1 ページで管理し切れない数に達した場合，コンテキストスイッチ時には複数のページに渡る検証子構造体に対する処理が必要となり，これは保護領域のサイズとは別種のオーバヘッドをもたらす原因となり得るだろう．そこで，ここでは全体としてのサイズが一定量となるような複数の保護領域を保持した場合に，保護領域個数が性能に与える影響を測定した．その際に用いる全保護領域合計サイズとしては，図 7 から改竄検出のみの適用であっても 20%近い性能低下をもたらすと推定できる 512Kbyte を採用した.

保護領域個数としては，64 個から 4 倍刻みで 4096 個まで変化させたときに，改竄検出のみ，暗号化のみ，改竄検出/暗号化同時適用という三種類の条件の下でそれぞれがどの程度の性能低下をもたらすかを表したのが，図 8 である．その際に利用したプログラムは 4.2.1 節の測定で用いたものを，確保した

(注11)：ただし，N は保護領域の数

(注12)：=1024byte

(注13)：改竄検出のみ，暗号化のみ，全適用

(注14)：最終要素は次の検証子構造体配列へのリンクとして予約されているため

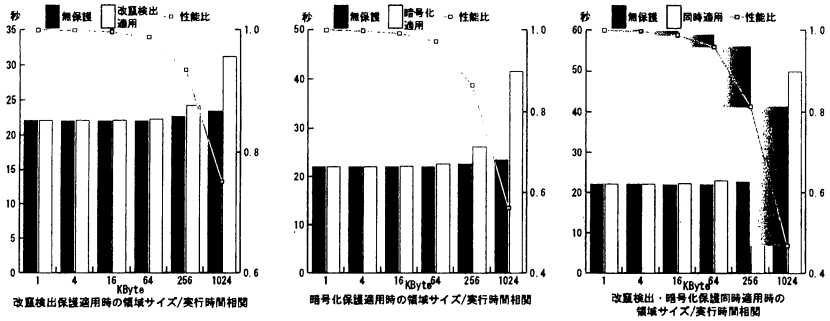

図 7 保護領域サイズに関する処理オーバヘッド

Fig. 7 Overheads with regards to size of protected region

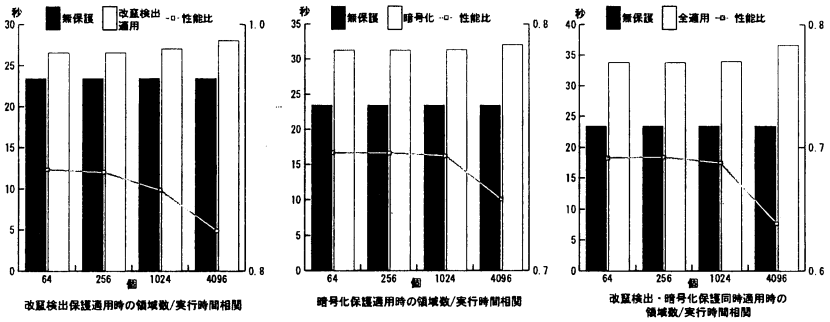

図 8 保護領域個数に関する処理オーバヘッド

Fig. 8 Overheads with regards to number of protected regions

領域が等サイズの複数個の保護領域として分割されるように改修したものである.

図から明らかな通り，保護領域個数に最も敏感なのは改竄検出機能のみを適用したケースだが，その場合でも最大の保護領域個数である 4096 個のときに，最小である 64 個のときと比較して 5% 程度の性能低下を示しているに過ぎない．4096 個の保護領域というのは，検証子構造体配列数としては 25 個に相当するため，保護領域の管理目的にのみ用いられる固定オーバヘッドとみなせる検証子構造体配列の存在は，この規模の保護領域を確保したとしても大きな負荷とはならないことが確かめられた.

現状の実装は，検証子構造体配列の各要素が有効であるか否かに関わらず，常にすべての要素に対して検証子生成／検証処理を行なうようになっており，その単純な最適化方式としては，利用中の要素のみを処理するというものが考えられるだろう．しかし，検証子構造体配列に関しては高速な MD5 による改竄検出処理のみであり，常に全要素に対して処理を行なうのでもそれほど大きなオーバヘッドはもたらさないとの予測の下に現在の実装を選択したのだが，今回の性能測定によりその予測が裏付けられたと考えている.

5. おわりに

以上，先般報告した少資源プラットフォーム向け暗号化メモリシステムと改竄検出手法に関して，両者を統合した形での試験実装を行ない，その効果および実行性能への影響度合を明らかにした．そこで示された結果から，カーネルメモリ使用量に関する劇的な削減の実現と，保護領域サイズ／個数に関して妥当な性能低下範囲で利用するための指標を明らかにすることができた.

関連する将来の仕事としては，[4] で提案した遅延処理方式を導入した試験実装を行なうことで，その有用性評価を実施することが挙げられる.

文　献

[1] National Security Agency: *Security-Enhanced Linux*, http://www.nsa.gov/selinux/index.html.

[2] 稲村 雄，本郷 節之: 暗号技術によるメモリデータ保護方式の提案，情報処理学会論文誌，Vol. 45, No. 8, August 2004.

[3] 稲村 雄，江頭 徹，竹下 敦: マルチタスク OS におけるメモリの保護 — 少資源プラットフォーム向け暗号化メモリシステム，情報処理学会研究報告 2004-CSEC-26, July 2004.

[4] 江頭 徹，稲村 雄，竹下 敦: マルチタスク OS におけるメモリの保護 — データ改ざん検出手法の提案，情報処理学会研究報告 2004-CSEC-26, July 2004.

[5] Xiaoyun Wang et al.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD, Crypto 2004 Rump Session, August 2004.

[6] Dan Kaminsky: MD5 To Be Considered Harmful Today, Bugtraq Mailing List, December 2004.

[7] 稲村 雄: Cryptographic Memory System —Get High with a little help from my kernel, 情報処理学会研究報告 2003-CSEC-22, July 2003.