2004－ＣＳＥＣ－26 (32)
2004／7／21

# 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—

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PDAや移動体通信機のように記憶資源が比較的乏しい実行環境において，同システムの実装を可能とせしめるための新しい処理方式に関して報告する．

**キーワード** 暗号技術，メモリ保護，マルチタスクOS

# Protecting Memories in Multitask OSes — A Cryptographic Memory System for Resource-poor Platforms—

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 proposed a method, called Cryptographic Memory System (CMS), to protect the data stored in the address spaces of the typical multi-task operating systems. Here we will present a new scheme, which reduces the memory usage by decreasing the number of encryption keys etc., aiming at the application for rather resource-poor platforms such as PDAs or mobile communication devices.

**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)に格納されたデータにアクセスすることが可能である場合が多い．この機能は，特に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同様に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機密情報を盗み出すという形の攻撃を実行する可能性もある．

筆者らはこのような性質を持ったOS上でもアドレス空間上の重要情報を盗み出される危険を排除できるシステムとして，暗号化メモリシステム (Cryptographic Memory System, CMS) を提案している [1]．CMSでは，マルチタスクOSで一般的な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のアドレス空間中で重要情報が格納されている部分を暗号化し，同じく実行再開されるタスクの重要情報格納部を復号する，という処理をOSカーネルが行なうことで，上記のようなアドレス空間からの機密情報の盗み出しを防御している．

しかし，提案したCMSは個々のタスクに対して一つの暗号化鍵を割り当てるような方式となっているため，それら暗号化鍵およびCMSを実現するために必要となる管理領域とがOSカーネル空間に保持される．たとえば典型的なUNIX系OSの場合，同時実行可能なプロセス数の最大値は$2^{15\sim17}$程度であるため，128bit= $2^4$byteの暗号化鍵を用いるならば，最大0.5〜2Mbyte程度のメモリが消費されることになる．さらに，保護領域を管理するためのメモリ領域も必要となることを勘案すると，当該方式でのOSカーネルにおけるメモリ使用は相当量に上ることが明らかである．

このようなメモリ消費はPCプラットフォームでは無視できる程度かもしれない．特に上記数値は最悪値であり，典型的な同時実行中タスク数は100のオーダを超えないのも事実であ

(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

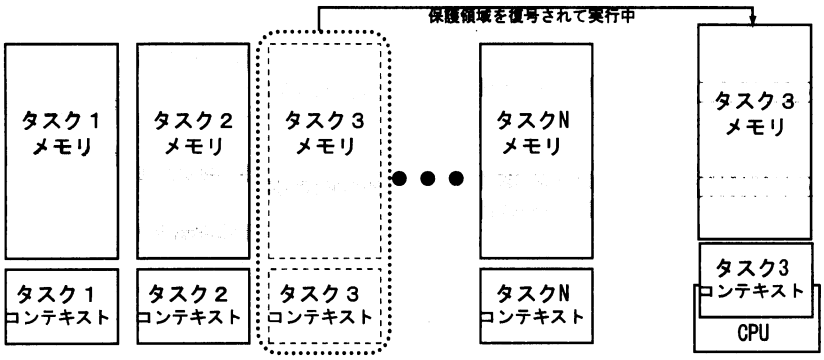


図 1　CMS によるマルチタスク OS 実行時スナップショット
Fig. 1　Snapshot on multi task OS's execution in CMS

る．しかし，PDA や移動体通信機のように資源の限定されたプラットフォームでは極力使用メモリ量を削減する必要があると考える．そこで本稿では，利用する暗号化鍵の数を削減する等の手法を用いることで，少資源 (= 資源の少ない) プラットフォームにも実装可能な新しい CMS の実現方式を提案する．

以降の構成は以下の通りである．まず，2. で新方式のアイデアを述べ，3.〜4. で単純な方式を採る場合の問題点およびそれに対する解決策を示す．5. で全体的な処理方式を説明し，6. で全体のまとめを行なう．

## 2.　処理方式改良アイデア

CMS は，前述の通りマルチタスク OS における複数タスクの仮想的な同時実行が，短い間隔での実行中断と再開を繰り返すことで実現されている点に着目し，実行中断状態のタスクの持つ重要情報保持領域を暗号化してやることで，他タスクからの干渉の排除を実現している．あるタスクが他のタスクのアドレス空間を盗み見しようとする場合，当然当該他タスクは実行中断状態にあるため(注2)，重要情報保持領域は暗号化されており内容を知ることはできない (図 1)．

このように，CMS はタスクの切替えを行なうコンテキストスイッチ処理で実行を中断されるタスクの重要情報保持領域を暗号化し，逆に実行再開されるタスクの同領域を復号するという処理を OS カーネルに追加することで実現されている．

現在の CMS で用いている「各タスクごとに一つの暗号化鍵を割り当てる」という方式は，素直に考えた場合にはごく自然な実現と考えられる．しかし，この CMS という方式が，誰が何を何から保護することを目指したものなのかをあらためて再検討してみると，このような構成は必然ではないことが分かる．

**誰が**　OS カーネル
**何を**　一つのタスク上の重要情報保持領域
**何から**　他タスクによる読み出し

このように処理の主体はあくまでも OS カーネルであり，重要情報を持つ個々のタスクは完全に受動的に操作されるだけで，CMS による暗号化/復号処理が行なわれていることにすら気付くことはない．また，タスクごとに割り当てられた鍵は OS カーネルが自身のみアクセス可能なアドレス空間に保存しているため，割り当てられたタスクであっても鍵の値を知ることはできない．

逆に考えれば，ある瞬間に実行中のタスクにとって，実行中断状態にある複数の他タスクの重要情報保持領域がタスクごとに異なる鍵で暗号化されていたとしても，それらがまったく同じ鍵で暗号化されていたとしても，それらの鍵にアクセスできない以上得られる効果に影響はないのである．

このように一見当然のようにも見えるタスクごとに暗号化鍵を設定するという構成は，必ずしも必然ではないということが明らかとなった．本稿で提案する改良型 CMS は，タスクごとではなくシステム全体で一つ(注3)の暗号化鍵を用いるという構成を取る．

ただし，単純に暗号化鍵を一つとするだけでは安全な方式とはならないことを次に見て行く．

## 3.　単純に一つの鍵とした場合の問題点

CMS は OS カーネルが実行主体となってタスクの持つ重要情報保持領域を暗号化により保護するという仕組みであるため，一つのタスクが自分の力だけで他タスクの保護部分を解読しようというシナリオのみが脅威であるならば，単純に一つの暗号化鍵を用いるといいう方式でも十分な安全性が確保できる．現代的な共通鍵暗号アルゴリズムは，攻撃者にとって極端に有利な条件であっても鍵に関する全数探索より効率的な攻撃方法は存在しないという性質を目指しているため，解読すべき鍵が一つであっても複数であっても攻撃者にとって必要なコストは所詮定数倍でしかないからである．

しかし，マルチタスク OS では攻撃者も複数のタスクを用いて他のタスクに対する攻撃を実施することが可能であるため，単純な単一鍵利用方式には図 2 に示すような方法で破られてし

(注2)：SMP システムの場合，この言明は厳密には偽だが，本稿では単一 CPU システムのみを対象とする

(注3)：もしくは後述の通り少数

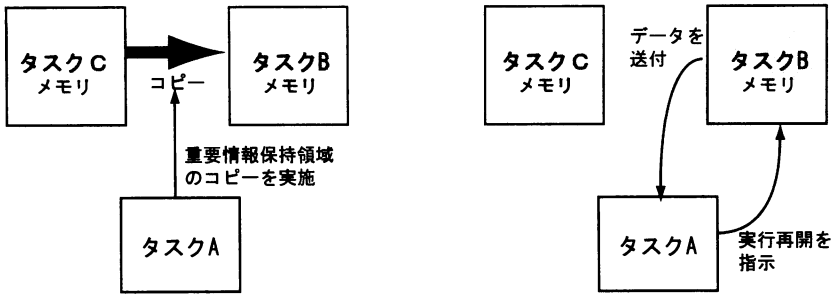


図 2　単純単一鍵方式への攻撃
Fig. 2　A possible attack on the simple single-key method

まうという欠陥が存在する.

図 2 で，攻撃者は攻撃実行タスク A と補助タスク B を利用し，以下のようなステップで対象タスク C の保持する重要情報保持領域を読み出すことができるのである.

（1） 補助タスク B は対象タスク C の重要情報保持領域と同じサイズの重要情報保持領域を確保した後，実行タスク A からの指示があるまで休眠

（2） 実行タスク A は対象タスク C の重要情報保持領域の内容を補助タスク B の重要情報保持領域に複写

（3） 続いて，実行タスク A は補助タスク B の休眠を解除し，重要情報保持領域の内容を送付させる

上記 (2) でタスク A が処理を実施している時点ではタスク B/C ともに実行中断状態にあるため，複写されるデータは OS カーネルによって暗号化された状態にあり，かつ複写先の領域も同じく暗号化されている．その後，(3) で補助タスク B が実行再開されると，当該タスクの重要情報保持領域は OS カーネルによって復号されるが，その時点での同領域の内容は対象タスク C の重要情報保持領域に保持されていたデータ (暗号化済) となっているため，この復号処理の結果得られるデータは対象タスク C の重要情報そのものということになる.

現行の CMS では暗号アルゴリズムとして CBC (Cipher Block Chaining) モード AES を利用しているため，一見，初期化ベクタをタスクごとに異なるものとすることでこの問題を回避できるように感じられるかもしれない．しかし，そのためには初期化ベクタという鍵と同じサイズのデータをタスクごとに用意しなければならないため単一鍵にする意味がなくなるという問題がある他に，CBC モードの持つ自己修復性ゆえに初期化ベクタの差異は 2 ブロック目以降には影響を与えないため，セキュリティ的にも不十分である.

## 4. ブロック暗号 CTR モードの利用

以上のような欠点を克服して単一鍵による暗号化メモリシステムを実現するためには，

（1） タスクごとにデータを割り当てる必要がある場合，暗号化鍵よりも十分短くなければならない

（2） あるタスクの重要情報保護領域が他タスクの保持する同様の領域に複写された場合，複写されたデータは正しく復号されない

という性質を持たせる必要がある．前述の通り，ブロック暗号利用モードとしてもっとも一般的な CBC モードは，このような性質を持たないため，この目的で用いるのは適切ではない.

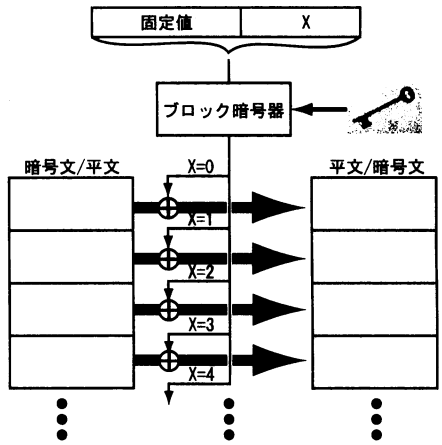


図 3　ブロック暗号 CTR モード
Fig. 3　CTR mode block cipher

そこで，筆者らは望ましい性質を持つブロック暗号利用モードとして，CTR (Counter) モード [2] に着目した．CTR モードでは，カウンタと呼ばれるブロックサイズと同じ大きさのデータを暗号化/復号ルーチンが共有し，メッセージに対する暗号化処理は，利用するブロック暗号アルゴリズムと鍵でカウンタの値を暗号化した結果をメッセージに対して排他的論理和演算することで実現される．カウンタ値は既定のやり方にしたがってブロックごとに初期値から変化させられるため，カウンタ値が一巡するまでは同じデータが現れることなく安全に用いることができる (図 3).

一般に，CTR モードではカウンタの一定部分を固定値，残りをブロックごとに変動する値として用いるが，その固定値部分をタスクごとにユニークな値としてやることで，上述の性質を備えたシステムを実現できるのではないかと考えたのである.

なお，上記の説明でも明らかな通り，CTR モードはブロック

暗号を疑似乱数ビット列生成エンジンとして用いたストリーム暗号と捉えることができるため，ストリーム暗号に関する「同じ疑似乱数ビット列を二回以上の暗号化処理に用いてはならない」という一般的ポリシーに留意する必要がある．

### 4.1 OFB モード利用の可能性に関して

DES 時代からブロック暗号利用モードとして一般的な四つのモードのうち，ECB (Electric Code Book) と CFB (Cipher Feed Back) の二つは前出の CBC と同じく必要な性質を満たさない．しかし，初期化ベクタ $IV$ を利用して

$$C_i = P_i \oplus Enc_K^i(IV)$$

のような形で暗号化処理を行なう OFB (Output Feed Back) であれば，IV をタスクごとにユニークなものとすることで上記 (2) の性質を満たすことはできる．

ただし OFB の場合，その構成から考えて任意の IV から可能なすべての状態を生成できることは保証できないという点が，CTR と比較して劣るところである．

また，OFB は暗号化/復号ともにすべてのブロックを順番に処理するのでなければ効率が悪くなるという性質を持つ．CMS が扱う対象であるアドレス空間は，ページング処理等の結果として一部が二次記憶装置に退避されている可能性があり，そのような領域の復号(注4)は実行中のプログラムが実際にアクセスするまで待つという最適化を行なう場合，この性質は非効率性をもたらす要素となる．

以上から，OFB よりも CTR の方が本方式での利用モードとして適切であると結論できる．

## 5. 処理方式

ここでは，CTR を用いた単一の暗号化鍵による CMS を実現するための処理方式を説明する．

### 5.1 タスクごとのカウンタ固定値の割り当て

前述の通り，タスクごとの CTR モードカウンタ固定値部分を全システムでユニークなものとすることが，単一暗号化鍵版 CMS を実現するためには必要となる．さらに，そのようなユニーク値を保持するために記憶領域を新たに用いるようなことになってはならない．そうなってしまっては暗号化鍵数を減らした効果が半減(注5)するからである．

このようなタスクごとのユニーク値として用いることができ，かつ，現在の一般的な OS で既に管理されているデータとしては

- タスク ID
- タスク生成時点での時刻値

という二種類が考えられる．タスクに関してはこれら以外にも多くの情報が管理されているが，他のデータはタスクの進行にしたがって変動する可能性があり，タスク生成時から終了まで特定の値を取り続けるのはこの二つである．

一般に，タスク ID の最大値は $2^{15}$(注6)～100000(注7)程度であるため，タスク ID のみでユニークな値を実現しようと思うと，タスク ID が一巡するたびに暗号化鍵を更新しなければならないことになり，効率を落す．

逆に，タスク生成時点での時刻値に関しては，マイクロ秒単位の値が得られるためこの精度での生成時刻値が確実に得られるのならば各タスクでユニークな値を時刻値のみで実現することが可能である．しかし，実際には OS カーネルが維持する時刻値の更新はタイマ割込み時に実施されるためマイクロ秒単位の精度は期待できず，一般には特定の時刻値に生成されるタスクは複数存在する可能性があり，タスクごとに完全にユニークな値であることは期待できない．

以上の考察より，本システムでのタスク毎カウンタ固定値としては，タスク ID およびタスク生成時点での時刻値の二つを併用することとした．

なお，タスク ID とタスク生成時刻の両者を併用したとしても，タイマ割込み間隔内に ID が一巡するだけのタスク生成消滅処理が可能であるならば，同一の値が複数のタスクに割り当てられてしまうことになる．しかし，典型的な UNIX 系 OS ではタイマ割込み間隔は 10msec 程度であり，同じく UNIX 系 OS ではタスク生成を行なうために比較的コストのかかるシステムコール呼出が必要となるため，この時間間隔内に 30000～100000 回のタスク生成処理を実行することは不可能である．さらに，多数の休眠タスクの生成により空きタスク ID の数を極端に減少させることで，同一タスク ID&生成時刻を持つ複数のタスクを実現する可能性はあるが，そのような極端な状況は容易に検出可能である．

また，システム時刻を調整するために OS カーネルが保持する時刻値を過去に戻る方向に変動できてしまうと，あるタスク ID とタスク生成時刻とが複数のタスクに割り当てられる可能性が現れる．実際のところ，これは本システムのみに限らず他にも影響を及ぼす可能性のある問題であり，他の方法で解決すべきと考える．具体的には，システム時刻を遅れさせる際に時刻値を過去の値に直接設定するのではなく，刻時間隔を操作することで漸近的に調整する等である．OS によってはセキュリティレベルの設定によってそのような形での時刻調整しか行なえないように強制することも可能であるため，本稿では時刻値に関してはこの漸近的調整が行なわれることを仮定する．

### 5.2 ストリーム暗号に基づく制約

続いて，タスク毎カウンタの変動値部分に関して説明する．コンテキストスイッチにおける暗号化/復号処理の実行の都度，特定の初期値から既定の方法で変動させるという方式(注8)であれば固定値以外の余分なデータは必要なくなる．しかし，このようなナイーブな方式では秘密情報保持部分の内容が変動し得るものである以上，異なるデータに対して同じ疑似乱数ビット

(注4)：基本的にページング処理を受けるのは実行中断中タスクのアドレス空間であるため，対象が CMS 保護領域であればそこはすでに暗号化されていると考えて良い

(注5)：もしくは全滅

(注6)：Linux および OpenBSD

(注7)：FreeBSD

(注8)：たとえば 0 から開始してブロック処理毎に 1 ずつ増加

列による排他的論理和演算が適用される可能性が出る．これはストリーム暗号の誤った利用でありセキュリティ的にこのような方式は好ましくない．

そこで，本方式ではカウンタの変動値部分においてある特定の値の利用は暗号化/復号処理各一回ずつとなることを以下のようにして保証する．

**暗号化対象アドレスの利用**

暗号化対象となる各ブロックの先頭アドレスを，カウンタ変動値部分の下位ビット(注9)としてそのまま利用する．

**単調増加データの利用**

初期値 0，各タスクに関して暗号化/復号処理を各一回実施するごとに 1 ずつ増加される単調増加データを，カウンタ変動値部分上位ビットとして利用する．

カウンタ変動部分をこのように設定することにより，特定のカウンタ値が複数回利用される可能性を排除できる．また，単調増加データ部分は各タスクごとに管理する必要があるが，そのサイズは 32bit 程度(注10)となるため，タスクごとに鍵を割り当てる必要のある現行方式と比較して必要なメモリ量は 1/4 程度に削減できる．

一方，CTR モードでは本来ブロックごとに 1 ずつ異なるカウンタ値を用いるようにすればもっとも効率的な運用を実現できるが，上の方式では

- アドレス空間全体を保護するわけではないのに，アドレスをカウンタ値の下位ビットとして用いている
- ブロックごとのカウンタ値の差分がブロックサイズ (16byte) となる

という二つの点から効率性は劣るものとなっている．

しかし，仮に平均 $\frac{1}{8}$msec ごとにコンテキストスイッチが発生しさらに特定のタスクがその半分でスケジュールされるという状況にあったとして(注11)，前述の条件で当該タスクの単調増加部分が溢れるまでにはおよそ 100 万秒ほどの時間がかかることになるため，上記程度の非効率性は容認できる範囲であろう．

### 5.3 暗号化鍵の更新

前節最後の議論でも触れた通り，この方式では長期間連続して運用される場合，カウンタ変動部分が溢れる可能性がある．そのため，そのような環境が対象であるならば，いずれかのタスクでカウンタ溢れが発生する以前に，暗号化鍵を更新できるような処理方式となっていなければならない．

もっとも単純には，これはいずれかのタスクで最初にカウンタ溢れの発生が迫った時点で単一暗号化鍵の更新処理を行なうことで実現できる．しかし，このようにナイーブな実現方式を取るとすると，鍵を更新する時点で，当該鍵で暗号化されている他タスクの重要情報保持領域に関しても適切な処理を施さなければならない．すなわち，概略

- すべてのタスクの重要情報保持領域を現行鍵で復号
- 新規鍵の生成
- 新規鍵による全タスク重要情報保持領域の暗号化

といった処理である．

この処理が行なわれるのは OS カーネルによるコンテキストスイッチというクリティカルな処理の中であり，かつ，処理の対象となるタスクの数は多数に上るかもしれず，さらには保護領域の一部が二次記憶装置に退避されている可能性をも考慮すると，最悪の状況ではとても許容できないほど長い処理時間を要する可能性のあるこの方式は，現実的ではないことが明らかである．

このような状況に対処するための定石的な手法が，上のような即時的処理方式ではなく必要になるまで処理を遅らせる遅延処理方式である．これは，鍵の更新を行なう際，旧鍵をただちに破棄するのではなく，新鍵と並存させておくということだ．

更新された新鍵は，それ以降に発生するコンテキストスイッチで重要情報保持領域を暗号化する際に用いられる．更新時点で旧鍵により暗号化された状態で実行中断しているタスク群は，その後いつかの時点で OS カーネルによってスケジュールされた際に重要情報保護領域を旧鍵で復号され，それから後は新鍵により処理されることになる．この結果，旧鍵で処理されているタスクは一斉にではなく，ある程度の遅延を伴いながら徐々に新鍵を利用する状態へと移行する．また，移行措置の結果，旧鍵を使用しているタスクがすべて存在しなくなったことを検出するため，鍵データに対しては，当該鍵を用いて何個のタスクを暗号化しているのかを示す参照数管理を行なうことが必要となる．

なお，ここでひとつ注意を要するのは，旧鍵で処理されたタスクが休眠状態に置かれたままでありながら新鍵におけるカウンタが溢れてしまう，という状況を避けなければならないということだ．前述の通り，かなり悲観的な運用状況を仮定したとしても，カウンタ溢れが発生するまでには 10 日以上の連続運転が必要であり，その間一度もスケジュールされないままのタスクが存在するということは，そもそも非常に考え難い状況ではある．

しかし，さらに確実を期すならば，旧鍵の参照数が 0 にならないままで新鍵のカウンタ溢れが近付いた時点で，すべてのタスクを一度スケジューリングするような操作を行なうのが良いだろう．このような操作は OS カーネルではなくユーザタスクでも実施可能であるため，そのための専用のユーザタスクをあらかじめ用意しておくことにより，OS カーネルを不必要に複雑化することなしに実現できる．

### 5.4 CMS 管理用データのユーザ空間への移動

現在の CMS では，保護領域記述子等の鍵以外の管理データも OS カーネルのアドレス空間に保持している (図 4)．これは当該タスク以外のタスクによって鍵が読み出されたり，その他の管理データを操作されたりするのを防ぐためだが，本稿の提案方式では鍵はすべてのタスクで共有されることとなるため，図のようにタスクごとのデータとして保持する必要はなくなる．

ここで提案するのは，鍵以外の図示されている管理データをユーザ空間に持ち出すという方式である．同時に，二重リンクおよびハッシュリンクで動的抜挿時の最適化を図っていた保護

---

(注9)：現在一般的な OS では 32bit

(注10)：カウンタ固定部分を 64bit と仮定

(注11)：かなり悲観的な見積もりであろう

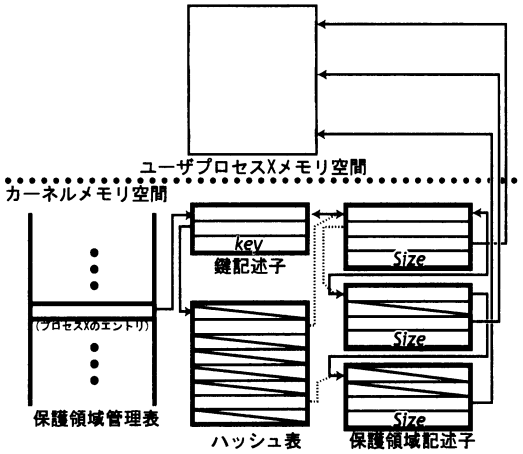


図 4　現行 CMS 保護領域管理方式

Fig. 4　Protected region management scheme in the current CMS

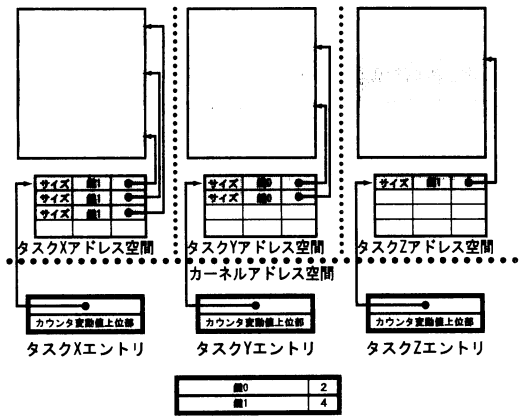


図 5　新保護領域管理方式

Fig. 5　New protected region management scheme

領域記述子を単純な表形式とすることで，管理構造に使用するメモリの削減を狙う．これは少資源プラットフォームへの実装という目的から，保護領域の削除時に線形探索が必要になるという処理コスト増とメモリ利用量削減とのトレードオフから決定した方針である．

この結果，タスク中の各保護領域のアドレスおよびサイズ，そして，新旧二つの鍵のうちどちらを利用しているのかを示すフラグとからなる保護領域記述子をエントリとして持つ表として全体が管理される．各保護領域に加えて，この管理用表データそのものも CMS によって暗号化することで，同領域に対する保護を実現する (図 5)．

なお，この方式では管理用データ部分を含めた保護領域に関して，利用暗号アルゴリズムの安全性の範囲で守秘性を保つことができるが，攻撃者による保護領域改竄の危険性は存在する．ただし，攻撃者が改竄行為を行なえるのはあくまでも暗号化された状態の保護領域であるため，それによって CMS のセキュリティがただちに侵されるわけではない．たとえば管理用データ中のアドレスやサイズを攻撃者が改変したとしても，その結果として得られる効果は，本来の保護領域以外の部分が代わりに暗号化/復号されるようになるだけであり，保護領域は暗号化されたまま放置されることになる．これは同保護領域の所持タスクであっても当該領域を使えなくなることをも意味するため，そこに保存されたデータを利用しようとした時点でそのような攻撃がなされたことが発覚するだろう．

唯一危険をもたらす可能性のある攻撃は，保護領域の確保後で，かつ，その領域に必要なデータが格納される前の時点で管理用データの内容を改竄することで，その後当該保護領域に格納された重要なデータが暗号化されないよう仕向けるというものだが，このような攻撃の実施には適切なタイミングの同定等，かなりの困難が伴う．さらに，筆者らが [3] で提案している改竄検出手法を管理データ部に適用することで，管理データの書き換えによる攻撃を完全に防御することも可能である．

## 6. ま と め

以上，暗号化により各タスクの持つ重要情報保持領域の他タスクによる読み出しを防止するためのシステムである CMS を，PDA/移動体通信機等の少資源環境で実現するための一方式を紹介した．

共通鍵ブロック暗号アルゴリズムの利用モードに関する最近の研究成果である CTR モードを適用することにより，各タスク毎ではなくシステムで一つの鍵のみを用いることで安全に CMS が実現できること，およびそのために必要となるタスクごとへの CTR モードカウンタ値の割り当て方を示せたと考える．

また，最後に従来 OS カーネルのアドレス空間で保持していた管理データを，ユーザ空間に移動するというアイデアも述べた．本来，カーネル空間は容量制限も厳しく，二次記憶領域への退避も容易ではないため，このような形でカーネル空間上のデータをユーザ空間へと外出しできるのならば，それによって全体的な記憶容量の削減には繋がらないとしても，十分な効果を持つ．その意味で，CMS に関連したデータのみではなく他のタスク管理関係のデータに対しても同様のアイデアを適用する可能性を検討するのも価値がありそうである．その場合，CMS とは異なりコンテキストスイッチ処理ごとに暗号化と復号を適用する必要はないため，少ない性能低下で実現することも可能かもしれない．

今後はさらなる記憶領域の削減が可能であるかどうかも含めて，そのような新しい可能性をも検討してみたい．

### 文　献

[1] 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.

[2] M. Dworkin: Recommendation for block cipher modes of operation, Special Publication 800-38A, NIST, December 2001.

[3] 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.