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

橋本幹生† 春木洋美†

近年，コンピュータソフトウェアおよびコンピュータシステムで扱われる著作物に対する著作権侵害が問題になっている．とりわけ，利用者端末のOSがマルチベンダ・マルチタスクのオープンシステムであるとき，この問題は深刻となる．この解決のため，我々は，ソフトウェアベンダの立場からは利用者端末のOSが信頼できないことを前提として，マルチベンダ・マルチタスクのシステムにおいて，アプリケーションプログラムを解析や改ざんから守ることのできるプロセッサハードウェアアーキテクチャL-MSP（License-controlling Multi-vendor Secure Processor）を提案する．L-MSPでは，マルチタスクOSに必要な資源管理機能と，アプリケーション保護のための秘密保護機能とを分離することで，信頼できないOSを持つ端末システムにおいても安全なライセンス管理を実現するソフトウェア実装の枠組みを提供する．

Multi-vendor Secure Processor under a Hostile Operating System

Mikio Hashimoto† and Hiroyoshi Haruki†

Recently, copyright violation problems has been raised on both computer programs and other copyrighted information processed on computer systems, especially in multitask open systems. Several kinds of secure microprocessor architecture have been proposed to offer a solution to the problem. But previous types of architecture were much inclined to software protection. We focused on both software and content protection including its license control. In this paper, we propose a secret management microprocessor architecture which enables both privacy and secure flexible license control of software under management of a hostile operating system. It is called License-controlling Multi-vendor Secure Processor(L-MSP). This architecture enables secure license control implementation by software itself.

1. はじめに

1.1 ソフトウェア保護

PCや，Linux OSベースの情報家電システムが普及するにつれ，そこでのソフトウェアの解析・改変が問題となりはじめている．一例としては，PC向けのあるDVD再生ソフトにおいて，コンテンツスクランブルの復号鍵を秘匿化する実装の不備から，エンドユーザによってDVDの著作権保護システムが解析された事例が知られている[1]．ユーザ端末で利用されるソフトウェアやコンテンツの著作権保護には，ソフトウェアに埋め込まれた秘密や動作が，ユーザによって解析・改変されることを防止する秘密保護が要求される．オープンシステムではOSもユーザが自由に改変可能であり，アプリケーションソフトの秘密保護に対してOSの権限による攻撃が行われることを前提とした対策が必要となる．

オープンシステムに適用可能な秘密保護技術として，ソフトウェアに組み込まれたアルゴリズムや定数を秘匿化するTRS（Tamper Resistant Software）と呼ばれる技術がある．だが，TRSの安全性は実装方式の秘密性に依存するために，その利用範囲は限られている[2),3)]．また，プロセッサチップが物理的な耐タンパ性を持つことを前提として，プロセッサに埋め込まれた固定の暗号鍵で，プログラムを復号して実行することでプログラムを保護するプロセッサが知られている[4)]．だが，上記暗号鍵がプロセッサに埋め込まれている制約から，PCのように不特定のベンダが提供するプログラムの保護が必要なマルチベンダ環境のソフトウェア保護には適用できない．

1.2 セキュアプロセッサ

マルチベンダ環境に対応可能な，非対称鍵と対称鍵を組み合わせたプロセッサハードウェアによるプログ

† 株式会社東芝研究開発センター
Corporate Research and Development Center, TOSHIBA Corporation

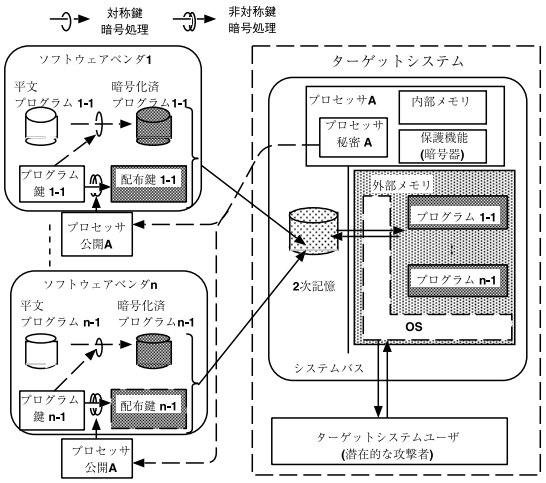

図 1 セキュアプロセッサ運用の概念図
Fig. 1 Schematic view of secure processor.

ラム保護方式が提案されている[5)~7)]．その概念を**図 1** に従って説明する．システムユーザ（ユーザ）は潜在的な攻撃者であり，ターゲットシステム（システム）のストレージや外部メモリ，バスはユーザが読み取り・改変可能である．システムには暗号機能を備えた耐タンパ性を持つプロセッサがあり，秘密鍵 A を保持する．秘密鍵 A は潜在的な攻撃者であるユーザには開示しないが，秘密鍵 A に対応する公開鍵 A は公開する．

ソフトウェアベンダ（ベンダ）1 はプログラム 1-1 を作成し平文実行プログラムを得る．ベンダはプログラム 1-1 に対して，保護のためのプログラム鍵 1-1 を生成し，プログラム鍵 1-1 で平文実行プログラムを暗号化して暗号化済みプログラム 1-1 を作成する．そして，ターゲットシステムのプロセッサが持つ秘密鍵 A に対応する公開鍵 A でプログラム鍵 1-1 を暗号化して，配布鍵 1-1 を得る．これら 2 つの情報がターゲットシステムに送られ実行される．復号にはいずれもユーザの知りえない鍵が必要なため，これらはどこに置かれても安全である．

プログラム実行時には，プロセッサが配布鍵 1-1 を復号してプログラム鍵 1-1 を取り出してプロセッサ内のテーブルに格納する．次にプログラム本体をプログラム鍵 1-1 で逐次復号して読み込みながらプログラムが実行される．ここでプログラム鍵の値を扱う部分はプロセッサハードウェアにあらかじめ組み込まれたメカニズムによって処理されるため，たとえ OS といえどもプログラム鍵を知ることはできない．

したがって，このようなシステムでは復号されてプロセッサ内に読み込まれた情報が安全であると仮定すれば，敵対的な OS の管理下であっても，実行プログラム本体に埋め込まれた秘密を，暗号の強度に基づいて守ることがマルチベンダ環境において可能となる．だが，プログラムの実行過程を詳細に見たとき，この仮定は必ずしも成り立たない[7)]．1 つはプロセッサ内部のメモリにも安全ではないものがあるからである．たとえば，あるプログラムの実行によりキャッシュメモリに読み込まれた平文情報は，外部割込みにより制御が OS に移った後も有効なため，OS 特権を利用すれば容易に読み出されてしまう．

また，マルチタスクシステムのプロセス切替えでは，レジスタ情報の保存・復帰は OS の役目である．実行状態の中のレジスタ情報をプロセスごとに切り替えることはマルチタスクシステムに必須だが，切替えの過程でプログラムのレジスタ値が，信頼できない OS によって解析されたり，任意の値に改変されたりするおそれがある．

我々は，プログラムの秘密保護に関わるこれらの問題を解決し，マルチベンダ・マルチタスク OS の資源管理との両立を可能とするプロセッサアーキテクチャL-MSP（License-controlling Multi-vendor Secure Processor）を提案する．L-MSP の特徴は名称に示すとおりソフトウェアによるライセンス管理を安全に実現できることである．

本稿は以下の構成をとる．2 章では既存マルチタスクシステムの分析に基づいて要求条件を整理し，3 章で提案アーキテクチャL-MSP の安全性モデルと詳細を説明する．4 章では L-MSP と著作権保護やライセンス管理との関係を議論し，5 章で関連研究との比較を紹介し，6 章でまとめを行う．

2. 要求条件

2.1 プロセッサハードウェア，OS の制約条件

現在のマルチタスクシステムは，プロセッサやメモリといった限られた資源を複数のプロセス間で多重化して，効率良く共有することを目的として作られている．我々の目標の 1 つは，資源を効率良く多重化するためのプロセッサや OS の基本的な機能と共存できるような秘密管理方式の提案である．本節では既存マルチタスクシステムのプロセッサアーキテクチャと OS のプロセス管理の制約から，要求条件を整理する．

図 2 に既存プロセッサの構成とプロセスの要素を示す．プロセスの構成要素はプロセッサによる扱いにより，プログラム，データ，レジスタ（コンテキスト）の 3 種類に分類できる．

プログラムはソフトウェアベンダから供給される．同一プログラム由来のプロセス間でプログラムは共通

である．プロセスの実行状態情報はデータおよびレジスタに保持される．状態情報は各プロセスに固有である．プロセスの静的データは，ここでの議論ではプログラムに埋め込まれた定数の一部と見なしてさしつかえない．

プロセスの実行時には，プロセッサのバスインタフェースユニット（BIU）を通じて外部メモリからプログラムが読み取られ，命令キャッシュ（I-Cache）に格納され，コアの実行ユニット（EXU）で解釈実行される．命令の実行を反映したレジスタ値の変更および外部メモリへの読み書きが BIU，D-Cache を通じて行われる．

実行プロセスが切り替えられるときは，プロセスのレジスタ内容は OS によりコンテキスト情報として外部メモリに保存される．メモリ管理ユニット（MMU）はプロセスビューから見える仮想アドレスと外部メモリ上の物理アドレスを変換して，各々のプロセスに他のプロセスから隔離されたメモリ空間を与えている．

図 3 は OS によるプロセス管理を概念的に示したものである．左側はメモリとプロセッサからなるハードウェアのビューを表しており，右側は資源を管理する OS と，OS が管理するプロセスから見える仮想的な環境からなるソフトウェアのビューを表している．

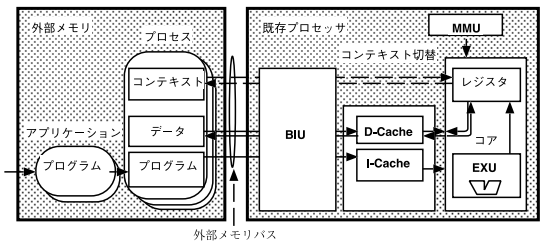

図 2 既存プロセッサのハードウェア構成
Fig. 2 Block diagram of existing processor.

OS はプロセス対応に PCB（Process Control Block）と呼ばれる管理情報のテーブルを保持し，そこにはプロセスのメモリマップなどが格納されている．OS のメモリ管理機能は MMU のページテーブルを書き換えることで各プロセスビューに見える仮想的なアドレス空間を管理する．なお，図上では MMU はキャッシュメモリのバス側に置かれているが，これは説明のための図の制約によるもので，実際のキャッシュメモリには物理アドレスキャッシュを使うことを想定している．すなわちアドレス変換はコアからキャッシュメモリへのアドレス出力で行われる．また，プロセスの切替えにあたっては OS のスケジューラが与えたポリシーに従ってディスパッチャがレジスタ情報の外部メモリへの退避および復帰を行う．これらの資源管理操作はプロセスビューからは透過的であり，プロセスは意識しない．

いったんプロセスの実行が始まると，プログラムを命令キャッシュに読み込み，命令を実行してデータキャッシュを通じてデータにアクセスするのはプロセッサであり，この過程には OS は介在しない．キャッシュ上には頻繁に参照されるデータが置かれ，直前に実行されていた他プロセスのものも混在する．ただし，プロセスがアクセスできるアドレス範囲は MMU により制約されており，他のプロセスのデータを無制限に参

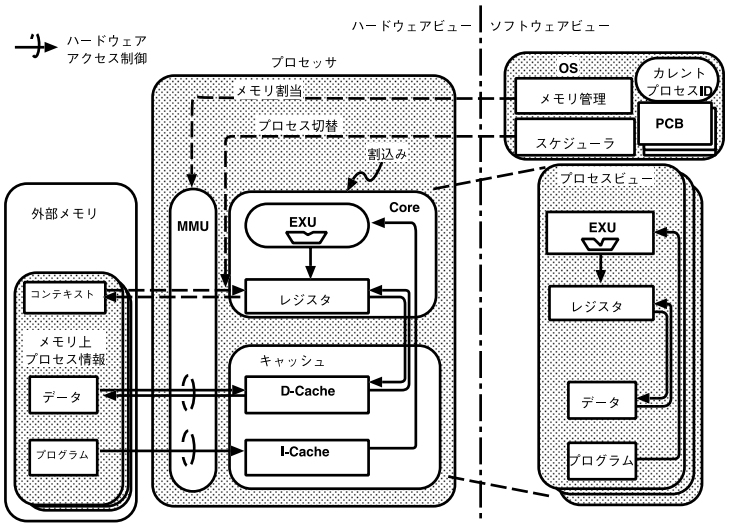

図 3 既存マルチタスクシステムのプロセス管理
Fig. 3 Process management in existing processor.

表 1 既存マルチタスクシステムの資源管理
Table 1 Resource management in existing system.

位置	リソース（多重化）	管理	
		実行	ポリシー
外部	メモリ（アドレス）	MMU	メモリ管理
内部	キャッシュ（アドレス）	キャッシュ制御	MMU
	コア（時分割）	OS ディスパッチャ	OS スケジューラ

照できるわけではない．

このようなプロセッサと OS の協調動作により，複数のプロセスに対して，あたかもそれぞれ専用のメモリおよびプロセッサコアが与えられてプログラムを実行して演算を行い，データの入出力を実行できるかのような仮想的なビューが提供される．上記システムの資源の多重利用と管理を**表 1** にまとめる．

我々の目的は，OS および外部メモリ上の情報は信用できず，プロセッサ内部のメカニズムと情報だけが信用できる前提で，複数のプロセスに安全性を提供する秘密保護アーキテクチャを構築することである．プロセス情報はプロセッサ外部，内部どちらにおかれた場合でも保護されなければならない．そして，既存 OS との整合性の観点から，秘密保護は OS による資源管理操作との整合性を持ち，資源管理操作はアプリケーションから透過的でなければならない．

2.2 秘密保護に関する要求条件

プロセスの秘密保護にはさまざまな観点とレベルがありうる．我々は次にあげる項目を著作権およびプログラムの保護に関わる秘密保護の要求条件とする．これらの条件の背景については 4 章で詳細に議論する．

- プログラム中，レジスタ演算のみの部分の安全性はプロセッサにより保証する．
- プログラムは安全性を意識して記述されることを前提とする．プログラムのバグにより，秘密が漏れることは許容する．
- プログラムの特定部分だけでなく，全体を暗号化保護することでプログラム全体の同一性を保証する．
- 異なる鍵で暗号化されたプログラムは識別される．
- 同一プログラム由来の複数のプロセスを識別し，個別に保護する．たとえば同一プログラム由来のプロセス 1 からプロセス 2 へ，OS が勝手にデータをコピーすることを防止できる機構を備える．後述のソフトウェアによるライセンス条件管理の安全性確保が目的である．

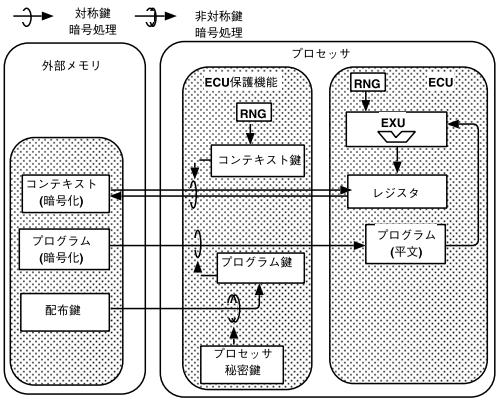

図 4 孤立 ECU の保護モデル
Fig. 4 Isolated ECU protection model.

3. 提案アーキテクチャ

3.1 提案アーキテクチャのプロセス保護モデル

3.1.1 孤立 ECU のモデル

本項では提案アーキテクチャL-MSP 紹介の導入として，暗号化に基づいたプロセス保護モデルを単純化した形で説明する．L-MSP では，プロセッサが保護プロセスを認識してその秘密を保護する．プロセッサの保護対象となる暗号化されたプログラムを保護プログラムと呼ぶ．L-MSP は従来の平文プログラムも実行できるが，平文プログラムは非保護プログラムと呼び，L-MSP の直接管理の対象とはならない．同様にプロセスには保護プロセスと非保護プロセスとがある．

以下の説明では．OS によるプロセス管理と，プロセッサによるプロセス情報管理とを区別するため，プロセッサが認識するプロセスを実行制御単位（ECU: Execution Control Unit）と呼ぶ．ECU の制御情報は OS の PCB とは独立にプロセッサ中に保持される．

図 4 に孤立 ECU の保護モデルを示す．孤立 ECU はデータ入出力を持たない，プログラム，レジスタ，EXU による計算機能だけを持つ仮想的な ECU である．ECU には ECU ID が割り当てられ，プロセッサ内部で一意に識別される．実際のプロセッサでは複数の ECU が時分割多重的に実行されるが，ここではただ 1 つの ECU だけを示している．

プロセッサには ECU 保護機能があり，ECU の状態や暗号鍵を管理している．OS は**表 2** に示す特殊命令の発行を通じて ECU 保護機能に指示を出すことで ECU を生成（登録）したり，実行の中断・再開を行ったりすることができるが，ECU 保護機能内部の情報は ECU 保護機能によりカプセル化されており，OS が ECU の秘密情報を直接操作することはプロセッサ

表 2 ECU 操作特殊命令
Table 2 ECU operation instructions.

名称	内容
ECU 登録（register）	配布鍵復号とプログラム鍵登録，コンテキスト鍵初期化
ECU 実行開始（start）	初期状態からの ECU 実行開始
ECU 実行再開（continue）	中断状態からの ECU 実行再開
ECU 削除（delete）	プロセッサ内部の ECU 情報の削除

の仕様により禁止される．

ECU 生成時には，プログラムの配布鍵と登録先 ECU ID が OS により指定され，ECU 保護機能に登録される．配布鍵がプロセッサ秘密鍵により復号されてプログラム鍵が取り出され，ECU 保護機能に格納される．同時に，ECU 保護機能の乱数発生器（RNG）が生成した乱数が同様にコンテキスト鍵として格納される．

プログラムの実行が始まると，命令フェッチにより，プログラムが逐次復号されてプロセッサ内部に読み込まれ，EXU により解釈実行される．図では単純化のためキャッシュを明示していないが，プログラムはキャッシュラインサイズを単位としてブロック暗号化されており，キャッシュフィルに同期して復号処理される．

割込みにより ECU の実行が中断されるときは，ECU 保護機能がレジスタ値をコンテキスト鍵により暗号化して，外部メモリに退避する．実行再開時はやはり ECU 保護機能がレジスタ値を復号して復帰する．この処理はハードウェアコンテキストスイッチとして実装されるため，OS は ECU の実行再開を指示することはできるが，ECU のレジスタ値を知ることも操作することもできない．

ここで，ECU 要素に対する外部メモリでの情報の偽造攻撃を考える．配布鍵は ECU 開始時にただ一度だけ読み込まれるのでメモリ上の偽造は意味がない．プログラムはプログラム鍵によって復号されて実行されるので，プログラム鍵を知らずにプログラム断片を置き換えたとしても，でたらめな命令列を挿入するにすぎず，有効な攻撃とはならない．また，ECU の実行停止中にメモリ上のコンテキストを置換する攻撃についても，攻撃者が RNG によって設定された鍵を知らない限り，意図したようにレジスタ値を改変することはできない．異なる ECU 間でコンテキストを置換する攻撃については，置換対象がたとえ同一プログラム由来のプロセスのコンテキストでも，個々のプロセスごとに RNG で設定された鍵が異なるため，置き換えは有効ではない．ECU 保護機能は，外部メモリ上の汚染された情報から ECU を保護するいわばフィルタとして機能し，ECU の安全性を保証している．

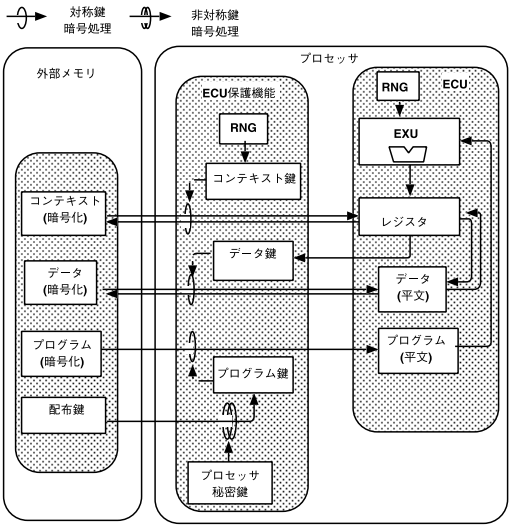

図 5 一般 ECU の保護モデル
Fig. 5 General ECU protection model.

L-MSP の孤立 ECU モデルでは ECU ID とプログラム鍵およびコンテキスト鍵が統合されて管理されることにより，たとえ同一プログラム由来のプロセス間であってもプロセス状態を他のプロセスへ複製することを防止している．これはライセンス管理のソフトウェア実装を容易にする．なお，プロセッサ上の ECU の秘密情報は，OS により ECU の削除が指示されたとき，すべて消去される．

3.1.2 一般 ECU の保護モデル

前項の孤立 ECU モデルでは，配布鍵，プログラム，コンテキストの ECU 要素への置換攻撃に対してプロセッサハードウェアが情報の安全性を検証することができた．この検証処理は実行対象のプログラムからは透過的である．

本項では前節のモデルを拡張して，データの入出力能力を持つ一般 ECU 保護モデルを説明する（**図 5**）．実用的な ECU が扱うデータ入力には，OS から割り当てられたメモリ領域のポインタやユーザのキー入力などがあり，ECU の観点から見て安全とはいえないデータを扱うことが避けられない．これはアプリケーションに関わる問題であり，プロセッサが単独でその情報の安全性を判定することは一般には不可能である．本項では L-MSP アーキテクチャにおいて，汚染されたデータの入出力がある場合でも，プログラミングが十分注意深く行われることを前提とすれば，安全なアプリケーションが構築可能であることを述べる．

一般 ECU でも，プログラムの一部をデータ入出力を行わないように記述することは可能であり，その部

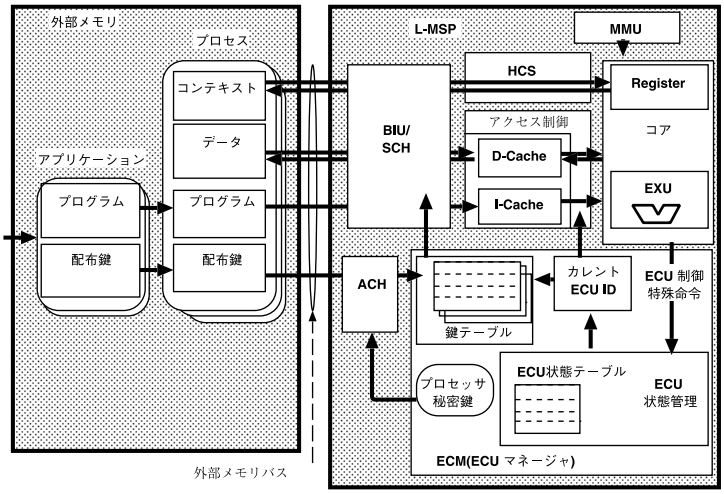

図 6 L-MSP のハードウェア構成
Fig. 6 Hardware block diagram of L-MSP.

分は孤立 ECU として機能する．したがって，外部からの入力値についてプログラム自身が検証を行うことにより安全性を確認できる．入力値に対して検証アルゴリズムが存在し，検証アルゴリズムがプログラムに実装され，かつその部分が孤立 ECU として実装されていれば，そのプログラムは安全に入力値を検証することが可能である．

たとえば通信相手のサーバからのメッセージの署名検証アルゴリズムおよび信頼できるサーバの公開鍵がプログラムに埋め込まれており，それらが孤立 ECU として実装されていれば安全にメッセージを検証することができる．サーバの公開鍵ではなくルート認証局の公開鍵と関連認証アルゴリズムが埋め込まれていれば任意のサーバからのメッセージを検証できる．このような検証済みの入力値はプログラムに埋め込まれた定数と同様に安全なものとして扱うことができる．

もしこの検証過程の中間値や結果がレジスタに収容しきれない場合には，これらの値に署名を付加したうえで，暗号化して外部メモリの作業領域に書き出す．署名および検証に使う秘密値はレジスタに保持し，外部メモリには書き出さない．再度その値を利用する際は外部メモリから読み出した値を秘密値に基づいて検証して利用する．

ECU はデータの暗号化処理入出力のために ECU 保護機能の暗号機能を暗号アクセラレータとして利用することができる．この目的のため，各 ECU 対応にデータ鍵レジスタと呼ばれる特殊レジスタが設けられる．ECU がデータ鍵レジスタに暗号化対象の仮想アドレス範囲と暗号鍵を設定すれば，以後の対象領域のメモリアクセスでは透過的に暗号・復号処理が行われる．

上述のように現実のアプリケーションは汚染されたデータを扱わざるをえない．汚染されたデータを隔離するための，あるいは安全性検証のポリシーをセキュリティプロトコルに基づいて定めるのはアプリケーション自身であり，プロセッサはその実行を暗号ハードウェアなどにより支援する．この隔離もしくは検証動作そのものの安全性を裏付ける機能として，L-MSP ではプログラムとレジスタだけからなる孤立した演算の安全性をプロセッサのメカニズムにより保証している．一般 ECU において，汚染されたデータをフィルタするのはプログラム自身の責任である．

3.2 提案アーキテクチャの詳細

3.2.1 ハードウェア構成

図 6 に提案アーキテクチャのハードウェア構成を示し，図 2 との差分を説明する．なお，本稿の目的は提案アーキテクチャにおける秘密情報の管理方法の明確化であるため，ここでは暗号アルゴリズムや鍵長は議論しない．BIU には対称鍵暗号ハードの SCH（Symmetric Cipher Hardware）が組み込まれている．コアに隣接してコンテキスト切替えのための HCS（Hardware Context Switcher）がある．ECU 情報全般を管理する ECU マネージャ（ECM）は，鍵テーブル，カレント ECU ID，プロセッサ秘密鍵，ECU 状態管理機能などを含んでいる．ECM はコアが発行した ECU 操作命令に従ってカレント ECU ID や鍵テーブルを更新する．非対称鍵暗号ハードの ACH（Asymmetric Cipher Hardware）はプロセッサ秘密鍵を ECM から受け取り，配布鍵の復号結果をプログラム鍵テーブルへ送る．

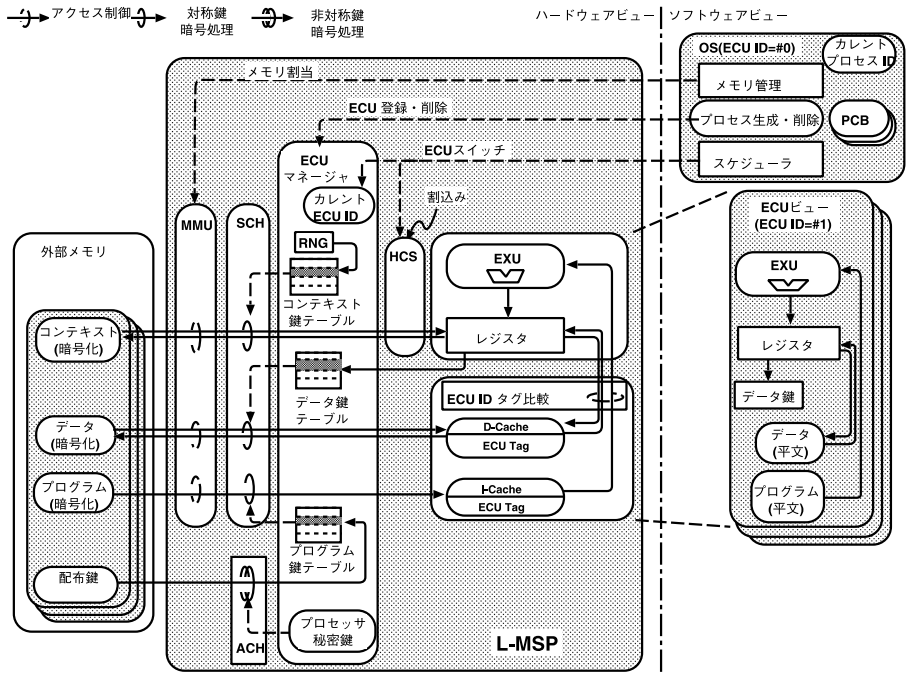

図 8 L-MSP における ECU 管理
Fig. 8 L-MSP ECU management.

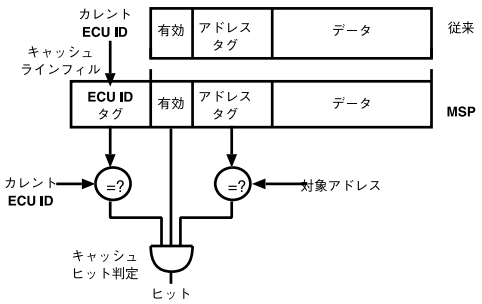

図 7 L-MSP のキャッシュヒット判定機構
Fig. 7 Cache mechanism of L-MSP.

キャッシュにも変更がある．**図 7** にキャッシュのデータ構造と動作を示す．従来のキャッシュにあった有効ビットとアドレスのタグ情報に加えて，本アーキテクチャでは ECU ID のタグが追加されている．キャッシュラインのフィルでは読み込み時のカレント ECU ID が書き込まれる．キャッシュのヒット判定時にはアドレスタグの比較に加えてカレント ECU ID とタグの ECU ID が一致しなければヒットしない．つまり，最初にそのラインを復号して読み込んだ ECU 以外の ECU は，キャッシュに保持されたラインを読むことはできない．その場合，後から同一アドレスをアクセスした ECU は当該 ECU の暗号鍵により復号してそのアドレスにアクセスする．すなわち同一内容であっても読み出し主体の ECU が異なれば読み出し結果が異なる．

図 5 に示した ECU 保護機能は，上記ハードウェアに含まれる SCH，ACH，HCS，ECM，キャッシュコントローラの連携動作の結果として実現されるものである．

3.2.2 ECU 管理

図 8 に L-MSP における ECU 管理の概念図を示す．これは既存システムにおける図 3 に対応する．以下変更点を ECU の動作とあわせて説明する．

ソフトウェアビューにおける変更は，OS から ECU 生成・削除を ECM に指示するプロセス生成・削除機能が加わったことである．また，プロセスビューにはデータ鍵レジスタが追加されている．以下の説明では，ECU ID=#1 を例として ECU の生成・実行動作を説明する．操作対象の ECU ID#1 に相当する鍵テーブルエントリを図上では黒く塗りつぶして表している．

OS が ECU ID=#1 を指定して ECU 登録命令を発行して ECU 生成を ECM に指示すると，配布鍵が ACH により復号され，プログラム鍵がプログラム鍵テーブルの#1 エントリに格納される．OS の ECU 実行開始命令発行によりカレント ECU ID が#1 にセットされ，ECU の実行が始まると，外部メモリから読み込まれたプログラムはカレント ECU ID=#1 で指定されるプログラム鍵により SCH で復号され，キャッ

表 3 L-MSP システムの秘密保護
Table 3 Resource management in L-MSP system.

位置	リソース（保護）	管理	
		実行	動作指示
外部	メモリ（暗号化）	SCH, ECU 鍵テーブル	メモリ参照
内部	キャッシュメモリ（タグアクセス制御）	タグ比較	キャッシュ参照
	コア（レジスタ値退避・復帰）	HCS	割込み，OS スケジューラ
	ECU 鍵テーブル（アクセス制御）	ECU マネージャ	OS プロセス管理

シュラインに格納される．このとき，キャッシュラインの ECU タグには#1 が格納される．MMU はここでも有効であり，秘密保護の有無にかかわりなく ECU は MMU が許可しないメモリ領域を参照することはできない．キャッシュラインの内容が EXU に読み込まれるときには上述のキャッシュヒット判定が行われる．

ECU 実行中に割込みが発生すると，HCS によりレジスタ値がやはり ECU ID #1 のコンテキスト鍵により暗号化されて外部メモリの所定アドレスに退避される．そして，割込みハンドラに制御が移る前にカレント ECU ID は#0 に更新される．ECU ID=#0 は OS を含む非保護プログラムに割り当てられており，ECU ID=#0 のとき暗号処理は行われない．

OS スケジューラが ECU #1 の再開命令を発行すると中断された ECU#1 の実行が再開される．ECM は外部メモリから#1 のコンテキストを読み込み，コンテキスト鍵で復号して HCS がレジスタ値を復帰する．復帰対象には PC（プログラムカウンタ）が含まれているので，中断した状態から実行が正しく再開される．

ECU は，データ鍵レジスタに暗号化対象アドレス範囲と暗号鍵を設定することで，SCH を暗号アクセラレータとして利用しながらデータの読み書きができる．データ鍵テーブルへの書き込みは実行中の ECU と同一のエントリに限定され，他の ECU のテーブルに書き込むことはできない．

これらの機構によって，ソフトウェアビューとしては，個別の ECU ごとに秘密保護された実行環境が提供される．そして，この秘密保護はメモリ管理やスケジュールのような OS の資源管理機構と共存できる．**表 3** に L-MSP の秘密保護対象ハードウェアのまとめを示す．特筆すべき点は保護対象としてプロセッサ内部の ECU 鍵テーブルが追加されたことである．ECU 鍵テーブルは ECM によりカプセル化されており，たとえ OS が信頼できず，指示が不適切であっても ECU の秘密は保護される．

4. 議　　論

4.1 著作権保護とプロセス保護

デジタル著作権保護とはユーザ端末に置かれたデジタル情報を，権利者が認めた形態でのみ利用可能とするような管理と定義できる．現行の DVD-Video の場合には，物理媒体を保持していることがライセンスであり，コンテンツのネットワーク配信においてはサーバからの許諾などがライセンスである．

ライセンス管理の機構を柔軟に実現するには，実装は固定されたハードウェアよりもソフトウェアによることが望ましい．その場合，利用許諾そのものは物理媒体やサーバからのライセンス許可情報として送信されるなどさまざまな形態がありうる．だが最終的にそれらの利用許諾条件を解釈して利用可否を判断するのは端末プログラムの実行であり，判断結果は一時的にせよプロセスの実行状態として保持されるはずである．

したがって，利用許諾を受けた状態のプロセス状態を複製・保存し，任意の時間，任意のシステムで再実行することができれば，ライセンス許諾の仕組みが破壊されることにつながる．L-MSP ではプログラムだけでなく，プロセスの実行過程を改変から保護し，プロセス状態の複製を禁止する機構を設けたことにより信頼できない端末における安全なライセンス管理を容易にするものである．

サーバとの間で再認証を行うなどのプロトコルの工夫により，ライセンス管理における危険のかなりの部分は排除できるが，実行状態が他のシステムに移動不可能であることおよび複製不能であることが保証されればプロトコルを単純化することができ，サーバ負荷の低減およびプログラムの作成効率向上が期待できる．

4.2 秘密保護と資源管理

L-MSP では秘密保護と資源管理の機能の直交化を目標としている．ここではその背景を述べる．著作権保護などの目的により，プロセスの秘密保護を要求しているのはソフトウェアベンダである．一方，OS にはユーザの意思に従って CPU 時間およびメモリ資源をプロセスに割り当てられる資源管理能力が求められる．これらはソフトウェアベンダとターゲットシステムユーザという異なる立場からの要求である．

ユーザの観点からは，秘密保護の有無にかかわらず資源保護機能はつねに有効である必要がある．たとえば，あるプロセスが暴走して多くの CPU 時間とメモリを占有してしまった結果，ユーザがそのプロセスの実行中止を指示した場合は，たとえそのプロセスが秘密保護の対象であっても即座に実行を中止し，メモリ

を回収できなければならない．秘密保護の対象プログラムがいわゆるウィルスであった場合を考えればその必要性が理解されよう．

一方，ベンダの立場からもプログラムの秘密保護機能が資源管理に優先されることは好ましくない．たとえば，秘密保護対象のプログラムが不具合により暴走してしまった場合にユーザが実行の中止手段を持たなければ，ベンダがその結果に責任を負うことになり，プログラム開発の困難度が増えてしまう．ベンダの要求はプログラムの実行が中断されないことではなく，むしろ実行が中断されても秘密が保護されることである．

この状況を，セキュリティの基本 3 要件である機密性（Confidentiality），完全性（Ingetgrity），可用性（Availability）にあてはめて考えたとき，次のように解釈できる．ベンダは，提供したプログラムについて，それが実行されるときに秘密保護である機密性と完全性を要求するが，実行が中断されないという，可用性は要求しない．

一方，ユーザの立場からは，ベンダから提供されたプログラムについて機密性と完全性は要求しないが，そもそもプログラムを実行したのはユーザ自身の意思なので，プログラムの可用性が求められる．なお，この場合のユーザが機密性および完全性を要求しないとは，ベンダから提供されたプログラムについてであって，ユーザ自身に帰属するデータについては当然機密性および完全性を要求する．

このようにシステム外部由来のプログラムのセキュリティ要件については，通常の場合とは異なり，3 要件を分離して考えなければならない．L-MSP は，ユーザとベンダの間に立つ信頼できる第 3 者（TTP: Trusted Third Party）の役割を果たすものである．OS は信頼できない通信路の一部を構成する．

4.3 著作権保護の依頼計算モデル

上記のベンダとユーザ，そして TTP の関係は**図 9** に示す依頼計算の特殊な場合としてモデル化できる．ソフトウェアベンダは依頼計算の依頼者であり，プログラムをターゲットシステムに送付する．また，プログラムが扱うコンテンツはコンテンツホルダから暗号化した形でシステムに送付され，その暗号鍵であるトレード シークレットは別途ベンダに送付される

システム上のバスや外部メモリは信頼できない通信路であり，プロセッサはその通信路を通じて送られた暗号化されたプログラムに改変のないことを検証しながら，外部にその秘密を漏らさないように実行する．OS も同通信路の構成要素である．一般の依頼計算では実行結果が依頼者に返されるが，このモデルでは実行結果は依頼者に必ずしも返されなくてもよい．代わりに，実行の副作用である出力，たとえば画像や音声などが，システムのユーザに渡される．一般のユーザはサービスの提供相手であると同時に，潜在的にはプログラムに対する攻撃者となる．ここではプログラムはベンダの代理人として機能している．

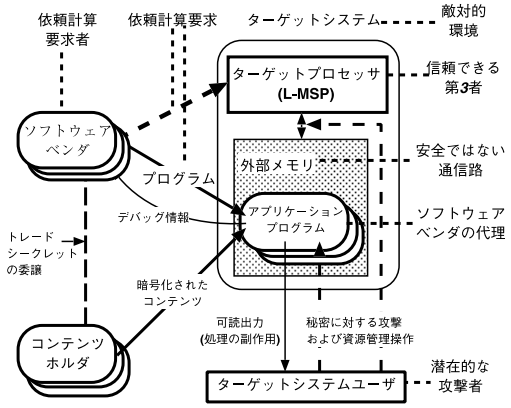

図 9 著作権保護システムの依頼計算モデル
Fig. 9 Remote computation model of copyright protection system.

ここまでプログラム鍵をプロセッサ公開鍵で暗号化したものを配布鍵と呼んでいた．だが，これは鍵の配送よりもむしろ TTP であるプロセッサへのカプセル化されたセキュアなメッセージ転送である．したがって，配布鍵にはプログラム鍵に限らず，プログラムのライセンス条件などの，プロセッサに安全に配送する必要のある情報が含まれてよい．プロセッサに埋め込まれた秘密鍵は，プロセッサに安全に鍵情報を配送する手段だけでなく，ソフトウェアベンダに対してそのプロセッサが安全であることを証明する手段としても機能する．その意味で，ソフトウェアベンダがプロセッサに配布鍵を発行する際には，その秘密鍵が正当なものであることを証明書の検証などで確認する必要がある．

5. 関 連 研 究

Lie らは，XOM アーキテクチャにおいて，プロセスの実行状態保護としてデータおよびコンテキスト情報の保護手法を示している[7]．ここではプロセッサ内の秘密情報の保護にタグを適用することが示されている．XOM においてはレジスタ個別に暗号化してキャッシュ保存することにより保護するメカニズムが提案されている．一方，L-MSP においては，プロセスのレジスタ情報の全体をまとめてキャッシュメモリに一時退避し，キャッシュメモリから外部メモリへの移動時

に暗号化することを想定している．

一般にプロセッサコアのレジスタファイルからキャッシュメモリへの転送と，キャッシュメモリから外部メモリへの転送とを遅延要求で比較した場合，前者の要求がより厳しい．L-MSP ではレジスタ情報の暗号処理を相対的に速度要求の低いキャッシュに担当させることで遅延要求を緩和することを意図している．本稿ではこの機構の詳細に触れることができなかったが，今後検討結果を報告する予定である．

6. まとめと今後の課題

マルチタスク OS の資源管理と共存可能なプロセスの秘密保護手段を提供する L-MSP アーキテクチャを提案した．提案アーキテクチャは，次にあげる機構を備え，プロセスの構成要素（プログラム，データ，コンテキスト）すべてを保護対象として含む．

- 外部メモリ上プロセス情報の暗号化保護
- 内部メモリのプロセス情報のアクセス制御
- 暗号鍵およびアクセス制御に用いる ID の管理

実行状態の保護は，プログラムおよびコンテンツのライセンス管理の要求条件に基づいたものであり，端末に L-MSP アーキテクチャを持つ信頼できるプロセッサが実装されていれば，端末の OS が敵対的な動作をとる信頼できないものであった場合でも，プログラムの秘密を守り，安全なライセンス管理が実現できる．これは，端末 OS が Linux，μITRON のようなユーザが改良可能な OS であっても安全な著作権保護メカニズムの実装が可能であることを意味する．L-MSP は一種の TTP として位置付けられ，応用システムは依頼計算の特殊形態としてモデル化することができる．著作権保護以外にもグリッドシステムやネットワークエージェントなどの保護[8] への応用が可能である．

現在，我々は本アーキテクチャの機能検証および評価を目標とした FPGA による提案アーキテクチャの実装および μITRON マルチタスク OS の適用を完了したところであり，今後実装および評価の詳細を報告する予定である．

謝辞 本研究にあたって多大な技術支援をいただいた東芝 SoC 研究開発センターの宮森高主査，玉井孝典氏に感謝いたします．また有益なコメントをいただいた東芝研究開発センターの白川健治主務，寺本圭一主務，藤本謙作主務に感謝いたします．

参 考 文 献

1) Eskicioglu, A.M.: Protecting Intellectual Property in Digital Multimedia Networks, *IEEE Computer*, Vol.36, No.7, pp.39–45 (2003).
2) Aucsmith, D.: Tamper Resistant Software: An Implementation, *Information Hiding, 1st International Workshop*, pp.317–333, Springer-Verlag (1996).
3) 石間宏之，亀井光久，斎藤和雄：ソフトウェアの耐タンパー化技術，情報処理，Vol.44, No.6, pp.622–627 (2003).
4) Dallas Semiconductor: *Secure Microcontroller Data Book* (1997).
5) Arnold, M.G. and Winkel, M.D.: Computer systems to inhibit unauthorized copying, unauthorized usage and automated cracking of protected software, U.S. Patent, No.4,558,176 (1985).
6) Gilmont, T. and Legat, J.-D.: Hardware Security for Software Privacy Support, *Electronic Letters*, Vol.35, No.24, pp.2096–2098 (1999).
7) Lie, D., Thekkath, C.A., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J.C. and Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software, *ASPLOS 2000*, pp.168–177 (2000).
8) 春木洋美，河口信夫，稲垣康善：耐タンパハードウェアを用いたモバイルエージェント保護手法，情報処理学会論文誌，Vol.44, No.6, pp.1613–1624 (2003).

(平成 15 年 7 月 31 日受付)
(平成 15 年 9 月 25 日採録)

橋本 幹生

平成元年東北大学理学部生物学科卒業．平成 3 年大阪大学大学院基礎工学研究科生物工学科修了．現在（株）東芝研究開発センター所属．ホームネットワーク，著作権保護技術の研究開発に従事．電子情報通信学会，ACM 各会員．

春木 洋美（正会員）

平成 12 年名古屋大学工学部電気電子情報工学科卒業．平成 14 年同大学院工学研究科計算理工学専攻修了．現在（株）東芝研究開発センター所属．著作権保護技術の研究開発に従事．