情報処理学会論文誌 Vol. 49 No. 9 2989–3000 (Sep. 2008)

Cell Broadband Engine™ プロセッサを利用したセキュアソフトウェアプラットフォームの実現

村瀬 正名[†1] 阪本 正治[†1] 清水 かんな[†2]
ウィルフレッド プルーフ[†3] ヴラディミール ズバルスキー[†3]

本論文では，マルウェアによる秘密鍵やパスワードなど秘匿データへの不正アクセスを保護する Cell Broadband Engine™（Cell/B.E.）Security SDK について報告する．本 SDK は，アイソレーションモードと呼ばれる特殊なメモリ保護モードを利用して，特権レベルに関係なく秘匿したい情報を他のソフトウェアから隔離することを可能にする．我々は，本 SDK 上にコンテンツ保護システムを試作し，コンテンツ保護の鍵の生成処理や暗号コンテンツの復号処理を OS やハイパーバイザから隠蔽できることを確認した．同時に，本プラットフォームが秘匿データ保護の分野において有効であることを示した．

A New Secure Software Platform on the Cell Broadband Engine™ Processor

Masana Murase,[†1] Masaharu Sakamoto,[†1]
Kanna Shimizu,[†2] Wilfred Plouffe[†3]
and Vladimir Zbarsky[†3]

This paper presents a new secure software platform, namely, the Cell Broadband Engine™(Cell/B.E.) Security SDK. The key feature to our platform is to use the isolation mode of the Cell/B.E. processor, which disables malware including OSes and hypervisors to access application secrets. We have prototyped a content protection system based on the Cell/B.E. Security SDK and proved that it is practical to apply our technology to secrets protection areas.

1. はじめに

コンテンツの不正コピーを防止するコンテンツ保護技術の拠りどころは放送暗号[1),2)] や DTCP，DTCP-IP[9)] などに基づいた暗号技術である．コンテンツ保護技術のライセンスを受けて製品を開発・製造する際には，秘匿データが漏洩しないようにコンプライアンス・ルールやロバストネス・ルールを遵守することが求められる．従来こうした製品では，ハードウェアによる作り込みによって秘匿データを保護してきた．しかし，近年では開発期間の短縮，製造コスト低減の観点から，再利用や拡張しやすいソフトウェアによる実装が望まれている．

秘匿データ（たとえば，秘密鍵やデジタルコンテンツ）を扱うシステムをソフトウェアとして実装する場合，難読化（obfuscation）[8),22)] と呼ばれる手法により製品そのものに含まれる秘匿データを隠す．しかし，デバッガを利用してプログラムの挙動を解析したり，プログラムが使用するレジスタやメモリの内容を盗み見たりすることによって，秘密鍵が漏洩する危険性がある．また，通常，オペレーティングシステム（OS）やハイパーバイザのような特権レベルの高いソフトウェアは，レジスタやメインメモリの全領域を読み出せる．したがって，攻撃者は特権レベルの高いソフトウェアを乗っ取ることで，レジスタやメインメモリに展開された復号鍵や平文の機密情報を容易に取得できる．

こうしたソフトウェア攻撃に対応するため，我々は IBM SDK for Multicore Acceleration バージョン 3.0[11)] の一部として，Cell Broadband Engine™（Cell/B.E.）Security SDK を新たに開発した[5)]．本 SDK では，秘匿したい情報を特権レベルに関係なく他のソフトウェアからいっさいアクセスできなくする環境を実現する．したがって，秘密鍵などの秘匿データを安全に扱える仕組みを実現できる．本論文では，Cell/B.E. プロセッサのセキュリティ機能を利用可能にした Cell/B.E. Security SDK について報告するとともに，コンテンツ保護分野での応用例についても紹介する．

以下では，2 章で想定する脅威モデルについてまとめ，3 章でそれらの脅威に対する関連研究について述べる．次に，4 章で Cell/B.E. を概説し，同時に，Cell/B.E. の独自のセキュ

†1 日本アイ・ビー・エム株式会社東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.
†2 IBM コーポレーションシステムズ&テクノロジーグループ
Systems & Technoly Group, IBM Corporation
†3 IBM コーポレーションアルマデン研究所
Almaden Research Center, IBM Corporation

リティ機能についても解説する．5 章では，我々が開発したセキュアソフトウェアプラットフォームである Cell/B.E. Security SDK の詳細を述べる．6 章で本 SDK を利用したプログラミングモデルを解説し，続く 7 章で我々が実装したコンテンツ保護システムについて紹介する．本 SDK に対する考察を 8 章で述べ，9 章で本論文をまとめる．

2. 脅威モデル

秘匿データを扱うアプリケーションを実行するうえで，ソフトウェアによる攻撃の脅威モデルを分析すると，その攻撃方法として次の 3 つがあげられる．悪意あるユーザはこれらの攻撃手法を単体で，あるいはいくつか組み合わせて攻撃する．

A-1. アプリケーションの動的解析による秘匿データ漏洩

A-2. アプリケーション実行ファイル改ざんによる秘匿データ漏洩

A-3. アプリケーション実行ファイルの静的解析による秘匿データ漏洩

A-1 は，OS やハイパーバイザのような特権レベルの高いソフトウェアが，メモリやレジスタにアクセスして，それらの内容を監視することにより，アプリケーションが何を行っているか推定し，メモリやレジスタ中に存在する暗号鍵やパスワードなどの機密情報を特定する攻撃方法である[16]．難読化により逆アセンブリが困難な場合に使用されることが多い．

A-2 は，アプリケーションの実行コードを書き換えて，秘匿データを外部に送り出すように変更する攻撃手法である．動的に秘匿データを作り出すアプリケーション（たとえば，暗号鍵生成プログラムなど）の場合は，この種の攻撃方法により，暗号鍵などの秘匿データが漏洩する危険性がある．

A-3 は，アプリケーション実行バイナリイメージからデータ領域に含まれる秘匿データやテキスト領域に含まれる機密アルゴリズムを解析する攻撃方法である．商用アプリケーションでは，こうした逆アセンブルによる攻撃[8] をソフトウェア使用契約によって禁止しているが，攻撃を防ぐまでには至らない．そこで，多くの製品では難読化によって逆アセンブル攻撃を困難にする手法がとられている．

3. 関 連 研 究

前章で述べた脅威モデルに対して，アプリケーションの秘匿データを保護する技術は，次の 3 つに大別される．第 1 の手法は，ソフトウェアレベルでの保護手法である．第 2 は，CPU とは別の特殊なハードウェアを追加することで秘匿データ保護を行う手法がある．第 3 の手法は，本研究のようにハードウェアレベルでの完全性検証機構や暗号機能を有した CPU を用いる手法である．以下にそれぞれの手法の詳細をまとめ，本研究との相違点を明らかにする．

3.1 ソフトウェアによる手法

Tripwire[13] はアプリケーションローダにプログラム検証機構を有するソフトウェアシステムである．あらかじめ，起動してもよいアプリケーションのハッシュリスト（グッドリスト）あるいは起動を禁止するアプリケーションのハッシュリスト（バッドリスト）のどちらかを作成しておき，グッドリストあるいはバッドリストのハッシュ値とロードされるアプリケーションのハッシュ値とを比較することで決められたアプリケーションのみ起動を許すシステムである．一方，Trusted JVM[12] は，Java 仮想マシン上に構築されたシステムで，本仮想マシンが Java アプリケーション実行時にアプリケーションの完全性検証や暗号領域の復号処理を行う．

しかし，これらソフトウェアによる手法では，OS あるいはハイパーバイザのような特権レベルの高いソフトウェアからは無防備である．また，検証機構や保護機構そのものが書き換えられてしまう恐れがある．特に検証機構や保護機構そのものの書き換えを検知する仕組みをソフトウェアのみで実現することは困難である．したがって，本手法では我々が想定する脅威から秘匿データを保護することはできない．

3.2 セキュア補助プロセッサによる手法

Dyad[25]，IBM 4758[26]，TPM[21] を用いた NGSCB[15] は，CPU とは別の特殊なセキュアハードウェアを用いて，アプリケーションの完全性検証を実現する手法である．IBM 4758 と NGSCB は，検証に必要となる鍵情報は特殊なセキュア補助プロセッサ内のメモリ空間のみで参照可能になっており，検証鍵の漏洩も防げる．これらの手法は，セキュリティ機能が実装されていない CPU 環境でもセキュア補助プロセッサおよび関連するソフトウェアスタックを追加することで容易に拡張できる点が優れている．しかし，セキュア補助プロセッサを使用しない設定でもアプリケーションを実行することは可能なので，その場合，実行ファイルの改ざんを見抜けず，そのままプログラムが実行されてしまう．すなわち，攻撃者は実行ファイルを自身のマシンにコピーし，実行コードの動的解析，静的解析を自由に行える．一方，我々が開発した SDK では，暗号アプリケーションの実行をサポートしており，アプリケーションの暗号化に用いた鍵が危殆化しない限り，攻撃者は実行コードを動的にも静的にも解析できない．Dyad では，本研究と同じように，暗号アプリケーションをサポートしている．ハードウェアに埋め込まれている鍵で直接アプリケーションの暗号化を行っており，本暗号鍵が漏洩しない限り，暗号アプリケーション中に含まれる機密情報を不正入手

することは難しい．しかし，ハードウェア鍵で直接アプリケーションを暗号化しているため，ハードウェア鍵の異なるマシンで同一の実行バイナリを動かすことはできない．実用上，マシンごとに異なる実行バイナリを配布するのは現実的ではない．

3.3　セキュア CPU による手法

XOM[14]，Cerium[6] は，OS を信頼せず，より信頼度の高いソフトウェアに最上位の特権レベルを与え，このソフトウェアとセキュア CPU を連携させながらアプリケーションの検証や秘匿データの保護，暗号アプリケーションの実行を実現する．具体的には，最上位の特権レベルを有するソフトウェアが，メインメモリから CPU にデータが転送されるたびに改ざんがないかの検証や復号処理を行い，メインメモリに計算結果を書き戻す際には必要に応じて当該結果を暗号化して転送する．XOM では，ハイパーバイザを利用し，Cerium では，最小限の機能を有した小規模のセキュアカーネルを用いている．これらの研究では，DMA 攻撃[17] によるハイパーバイザやセキュアカーネルの改ざんに対処するため，これら実行コードが CPU 内のキャッシュに収まることを前提にしている．しかし，XOM に関しては，メインメモリに書き出されたデータ（実行コードを含む）に対してリプレイ攻撃が可能であることが報告されている[19]．

一方，AEGIS[20] においては，XOM や Cerium のように仮想メモリの管理やアプリケーションのロードを実現する小規模のセキュアカーネルを利用する方式とセキュアカーネル自体をハードウェアで実装する方式の 2 種類が提案されている．XOM や Cerium では CPU とメインメモリ間のデータの読み書きの際に，高信頼のソフトウェアが当データの暗号・復号処理を実行していたが，AEGIS ではセキュア CPU 内に実装された暗号エンジンが自動的に利用され，データの機密性を維持している．

XOM，Cerium および AEGIS では，機密情報を含むデータが CPU 内のキャッシュからメインメモリに書き出される際に，高信頼ソフトウェアあるいは CPU 内の暗号エンジンが自動的に暗号化を行う．したがって，メインメモリ上のデータ暗号処理は，これらがサポートする暗号アルゴリズムに限定される．一方，我々のアプローチでは，アプリケーションローダはアプリケーションが扱うデータの暗号化には関与しない．これは，アプリケーションプログラマが処理速度や暗号強度に応じてアルゴリズムを自由に変えらるようにし，柔軟性を高めるためである．

ブートプロセスにおいては，Cerium や AEGIS では，本研究のようなランタイムセキュアブートをサポートしておらず，別のソフトウェアスタック検証方式を採用している．具体的には，カーネルやハイパーバイザなど下位レイヤで動作するソフトウェアのハッシュ値およびアプリケーション自身のハッシュ値をアプリケーション実行バイナリに埋め込み，アプリケーション実行時にセキュア CPU がアプリケーションに埋め込まれたこれらハッシュ値と現在メモリに展開されているカーネルおよび実行を開始するアプリケーションのハッシュ値を比較，検証する方法である．しかし，この方式はセキュアカーネルなどの下位レイヤのソフトウェアが変更されると，アプリケーション実行ファイルを再度構築する必要がある．一方，我々はアプリケーション実行時には，ランタイムセキュアブートによって各レイヤごとでソフトウェアスタックの検証を行っているため，下位レイヤの変更が上位レイヤに影響しない．

また，TrustZone[23] では，新たな特権モードを追加することで，セキュア環境と非セキュア環境の 2 つの環境を同一マシン上に実現する．TrustZone のセキュア環境では，セキュア環境用のセキュアカーネル，セキュアデバイスドライバ，セキュアアプリケーションが動作し，非セキュア環境からの不正アクセスをハードウェアのメモリアクセス制御機能によって禁止している．セキュア環境では，セキュアブートを実施し，セキュアカーネル，セキュアデバイスドライバ，セキュアアプリケーションなどのソフトウェアの完全性検証や暗号化された実行コードの復号などを実現する．セキュアカーネルなどセキュアアプリケーションのメモリを管理するソフトウェアは，ブート時のみ完全性の検証が行われるのに対し，本研究では，こうしたソフトウェアについても，アプリケーションロードのたびに完全性検証を行う．これによって，システムブート後のソフトウェア改ざん行為についても検知可能としている．

さらに，我々のアプローチでは，アプリケーションそのものを OS やハイパーバイザ，さらに第三者によって提供されるサービス群から切り離し，これらの不正アクセスをいっさい受けない環境を実現するが，TrustZone ではセキュア環境で動作するセキュアアプリケーションをセキュアカーネルやその他のサービス群から切り離して実行することはしない．したがって，第三者が提供するサービス（特にデバイスドライバ）からセキュアアプリケーションの処理内容を監視されたり，第三者のソフトウェアのセキュリティホールの影響でセキュアアプリケーションの機密情報が漏洩したりする危険性がある．

4.　Cell/B.E. セキュリティ機能

我々は Cell/B.E. のセキュリティ機能[3] を利用したソフトウェアスタックを構築することで 2 章で述べた 3 つの脅威に対処する．本章では，Cell/B.E. のセキュリティ機能について概説し，次章でセキュアソフトウェアプラットフォームである Cell/B.E. Security SDK を

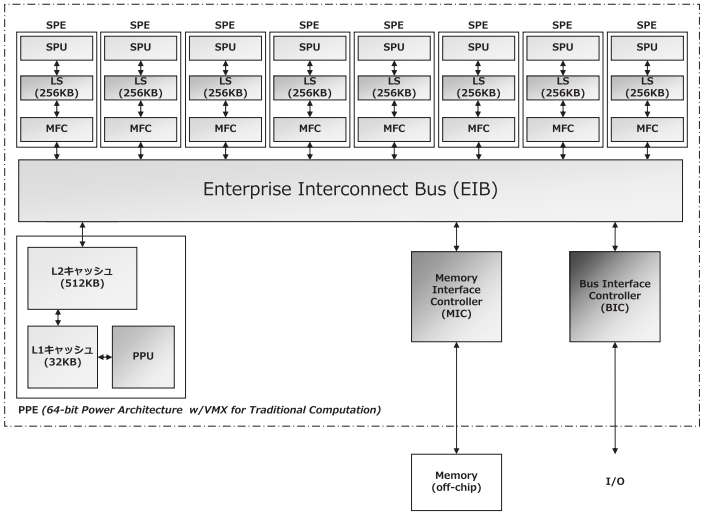

図 1　Cell/B.E. プロセッサ・ブロックダイアグラム
Fig. 1　Block diagram of the Cell/B.E. processor.

詳述する．

Cell/B.E. は，PPE（PowerPC Processor Element）と呼ばれる 64 ビット PowerPC 互換のコアと 8 個の SPE（Synergistic Processor Element）と呼ばれる 128 ビット SIMD エンジンで構成されるマルチコア・プロセッサである[18]．**図 1** のように，PPE，各 SPE，メモリインタフェース，I/O インタフェース間は EIB（Element Interconnect Bus）と呼ばれるリングバスにより接続されている．

SPE は SPU（Synergistic Processor Unit）と呼ばれる演算ユニットとローカルストア（LS）と呼ばれる 256 K バイトのローカルメモリ，DMA コントローラである MFC（Memory Flow Controller）を持ち，メインメモリと SPE 間は DMA によりデータ転送を行う．SPE 上でアプリケーションを動作させるには，SPU がサポートする命令群を利用したプログラムを作成し，本プログラムを LS 上にロードして実行させる．

Cell/B.E. には，アイソレーション実行，ランタイムセキュアブート，ハードウェアによる復号処理の 3 つのセキュリティ機能が備わっている．アイソレーション実行とは，SPE でアプリケーションを実行する際に，アイソレーションモードと呼ばれる特殊なメモリ保護モードでアプリケーションを起動させることにより，PPE や他の SPE 上で稼動しているソフトウェアや外部のハードウェアから，その SPE 上の LS やレジスタにアクセスできなくする機能である（**図 2**）．

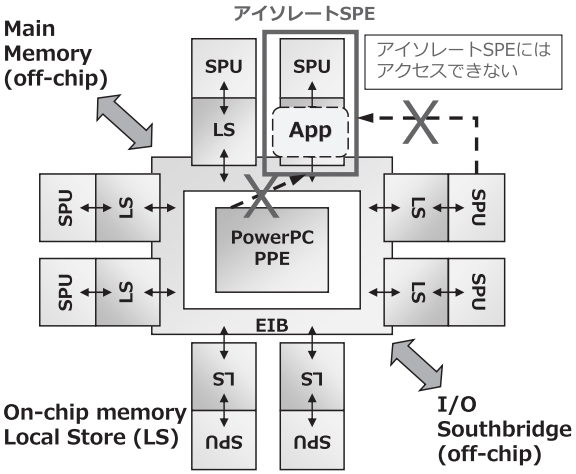

図 2　アイソレーション実行
Fig. 2　Application execution in isolation mode.

ランタイムセキュアブートは，アイソレーションモードでアプリケーションをロードする際にそのアプリケーションが改ざんされていないか検証する機能である．この機能は，アプリケーションをロードするたびに実行され，改ざんを検知するとハードウェアによってアプリケーションの実行が拒絶される．

さらに，ハードウェアのみがアクセスできるハードウェア鍵によって暗号データを復号し，ランタイムセキュアブートによって認証されたアプリケーションに平文のデータを受け渡す仕組みも提供される．この機能は，暗号鍵などの機密性の高い秘匿データを暗号化し，認証されたアプリケーションのみがこのデータを復号することを可能にする．ランタイムセキュアブートやハードウェアによる復号処理もアイソレーションモードで動作するため，特権レベルの高い OS やハイパーバイザであっても，これらの処理過程を盗み見たり，処理内容を書き換えたりすることはできない．

5. Cell/B.E. Security SDK

我々は Cell/B.E. Security SDK を開発し，マルウェアによる秘匿データへの不正アクセ

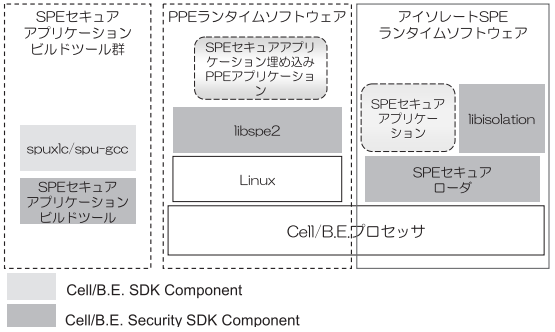

図 3　Cell/B.E. Security SDK ソフトウェアスタック
Fig. 3　Software stack of the IBM Cell/B.E. Security SDK.

スを防ぐセキュアソフトウェアプラットフォームを実現した．本セキュアプラットフォームは，Cell/B.E. のセキュリティ機能を有効にする最初のソフトウェア開発プラットフォームである．本 SDK は，2 章で述べた [A-1] から [A-3] の攻撃に対処するため，以下の機能を提供する．

(1)　アイソレーション実行 API
- アイソレーションモードで SPE アプリケーション*1を実行可能にし，[A-1] のアプリケーション動的解析攻撃に対処する．

(2)　アプリケーション認証，完全性検証
- 認証されたアプリケーションのみ実行可能にし，[A-2] のアプリケーション改ざん攻撃に対処する．

(3)　暗号アプリケーション（以降，SPE セキュアアプリケーション）の実行
- 秘匿データをアプリケーションに埋め込むことを可能とし，アイソレート SPE 内でのみ秘匿データを展開する．[A-3] のアプリケーション静的解析攻撃を防ぐ．

図 3 に，ソフトウェアスタックを示す．本 SDK を構成する重要なコンポーネントは，SPE セキュアアプリケーションビルドツール群，libspe2 および SPE セキュアローダの 3 つである．SPE セキュアアプリケーションビルドツール群は，アプリケーションの暗号化と電子署名の付与を行い，SPE セキュアアプリケーションのビルド環境を提供する．libspe2 は PPE 用のライブラリであり，アイソレーション実行を実現する API 群を提供する．SPE セキュアローダは，SPE セキュアアプリケーションの完全性検証と復号処理を行う．SPE セキュアローダ自身は Cell/B.E. プロセッサのハードウェア鍵によって完全性検証処理が行われるため，改ざんされた SPE セキュアローダを実行することはできない．

*1 SPE 内で動作するアプリケーションを SPE アプリケーション，PPE 内で動作するアプリケーションを PPE アプリケーションと呼ぶ．

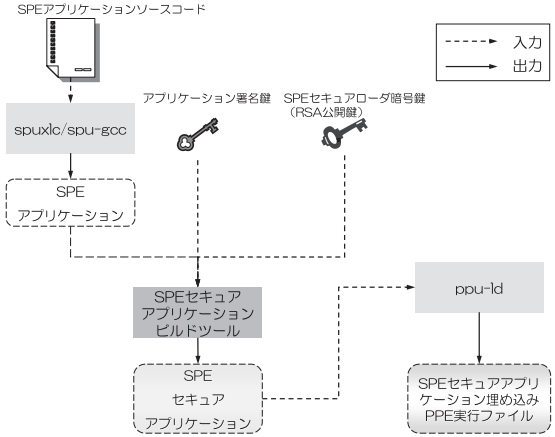

図 4　SPE セキュアアプリケーションビルドプロセス
Fig. 4　Build process of an SPE secure application.

5.1　SPE セキュアアプリケーションのビルド

図 4 に示すように，SPE セキュアアプリケーションのビルドでは，まず，ユーザは SPE アプリケーション用コンパイラである spuxlc [24] や spu-gcc [10] を利用して実行ファイルを作成する．次に，SPE セキュアアプリケーションビルドツール群に，この実行ファイルとアプリケーションを署名する署名鍵，アプリケーション暗号鍵を暗号化する SPE セキュアローダ暗号鍵（RSA 公開鍵）を与え，SPE セキュアアプリケーションの実行バイナリをビルドする．一般に，Cell/B.E. では，SPE の実行ファイルを CESOF（CBEA Embedded SPE Object Format）[4] と呼ばれるオブジェクト形式に変換し，PPE アプリケーションに SPE アプリケーションを埋め込む．これは，SPE アプリケーションと PPE アプリケーションを一体化し，1 つの実行ファイルとして両アプリケーションを扱えるようにするためである．本 Security SDK においても，従来と同様に SPE セキュアアプリケーションを PPE アプリケーションに埋め込んでいる．

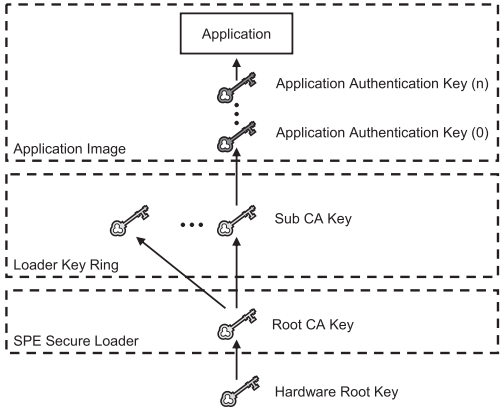

図 5　認証鍵信頼チェーン
Fig. 5　Chain of trust in the authentication key hierarchy.

アプリケーション署名鍵は，スケーラビリティ，管理の容易さを考慮して，**図 5** に示すような階層構造を採用した．図 5 に示すように，ハードウェアルート鍵を基点として，SPE セキュアローダが管理するルート CA 鍵，SPE セキュアローダから認証を受けているサブルート CA 鍵，アプリケーションに署名するアプリケーション認証鍵の順に信頼チェーンを形成する．本 SDK では，CRL（Certificate Revocation List）をサポートしており，秘密鍵が危殆化した場合にはサブルート CA や特定のアプリケーション署名鍵を無効化することも可能である．

5.2　アイソレーション実行

SPE を制御するライブラリである libspe2 にアイソレーション実行用のフラグを追加した．**図 6** に libspe2 を利用した SPE セキュアアプリケーションのプログラミング例を示す．図 6 のように `SPE_ISOLATE` フラグを設定して，`spe_context_create` を呼び出すと，libspe2 はアイソレーション実行のための準備を開始する．その後，通常の SPE アプリケーションの起動と同じように，`spe_program_load`，`spe_context_run` を呼び出すことで SPE セキュアアプリケーションのロードが行える．

5.3　セキュアアプリケーションロード

図 7 のようにアイソレート SPE 内で実行されるアプリケーションは，実行前に次の処理が行われる．

```
extern program_handle_t hello_spu;
// アイソレーションモード使用
spe_context spe = spe_context_create(SPE_ISOLATE, NULL);
// アプリケーションロード
spe_program_load(spe, &hello_spu);
// アプリケーション実行
spe_context_run(spe, &entry, 0, NULL, NULL, &stopinfo);
// 後処理
spe_context_destroy(spe);
```

図 6　libspe2 を利用した SPE セキュアアプリケーション実行例
Fig. 6　Sample code for launching an SPE secure application.

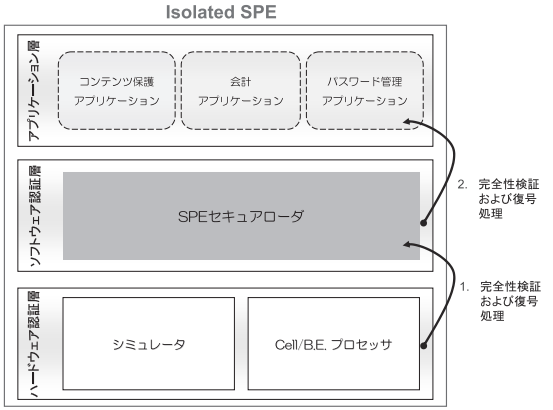

図 7　アイソレート SPE 内認証レイヤ構造
Fig. 7　Chain of verification and decryption in the isolated SPE.

Step1　ハードウェアによる SPE セキュアローダの完全性検証
Step2　SPE セキュアローダの復号
Step3　SPE セキュアローダによるアプリケーションの完全性検証
Step4　SPE セキュアローダによるアプリケーションの復号

完全性検証で改ざんが検知された場合には，プログラムの実行を中止する．これらの処理は，アイソレーションモードで実行されるため，高い特権レベルを有するマルウェアであっても処理内容を覗き見ることはできない．SPE セキュアローダは，ソフトウェア認証層であり，アプリケーション層をハードウェア認証層から切り離している．これによって，ハードウェア認証層の実装方法が変化してもアプリケーションは何ら影響を受けない．また，アプリケーションの性質にあわせて認証，復号アルゴリズムや鍵情報などを入れ替えることも可能になる．本実装では，ソフトウェア認証層の完全性検証処理に SHA-1 および RSA を用い，また復号処理に RSA と AES-128/CBC モードを用いた．

一方，アプリケーション実行イメージの復号処理では，SPE セキュアローダは，まずアプリケーションイメージから暗号化されたアプリケーション復号鍵を取得する．アプリケーション復号鍵は暗号化されたアプリケーション実行コードを平文に戻す鍵であり，本実装では AES-128/CBC モードを用いた．次に，その暗号化された鍵を自身が保持する RSA 復号鍵で復号し，さらにアプリケーション実行コードの復号処理を行う．本 SDK では，アプリケーションの全体を暗号化する機能のほか，コードの一部（ELF のセグメント単位）を暗号化する機能も備えている．

6. セキュアプログラミングモデル

本章では，Cell/B.E. Security SDK が提供する，Decrypt-in/Encrypt-out プログラミングモデルとセキュアコードオーバレイについて述べる．前者は複数の SPE 間で，データの安全な受け渡しを実現する方法であり，後者は単一 SPE 内で複数の SPE セキュアアプリケーションを実行する方法である．両プログラミングモデルは次章で述べるコンテンツ保護システムに応用している．

6.1 Decrypt-in/Encrypt-out プログラミングモデル

各 SPE の LS のサイズは 256 K バイトと比較的小さな作業空間である．アプリケーションによっては，秘匿データを一時的にメインメモリに退避させ，次の計算のために LS 内の作業空間を確保する必要がある．しかし，メインメモリは PPE や他の SPE からアクセス可能であり，メインメモリにデータを退避させる際には秘匿データを暗号化しなければならない．さらに，暗号の処理過程を盗み見られないようにするため，アイソレート SPE 内でデータの暗号化や復号を行い，平文のデータは同 SPE 内でのみ参照可能にしなければならない．

そこで我々は Decrypt-in/Encrypt-out プログラミングモデルを実現する API を本 SDK に実装し，ライブラリとしてユーザに提供している．具体的な動作例を**図 8** に示す．アイソレート SPE 内で動作するアプリケーションが鍵情報と使用アルゴリズムを指定して，`encrypt_out()` API を呼び出す．`encrypt_out()` では，データをアイソレート SPE 内で暗号化し，メインメモリに DMA 転送する．一方，データを受け取る SPE（同一 SPE でも，他 SPE でもよい）は，同じ復号鍵と使用アルゴリズムを指定し，`decrypt_in()` API を呼び出すことで，アイソレート SPE 内でのみ平文のデータが展開される．`decrypt_in()` では，メインメモリから暗号データをアイソレート SPE 内に DMA 転送し，アプリケーションが指定した暗号アルゴリズムに則ってデータを復号する．`encrypt_out()` および `decrypt_in()` での暗号化・復号処理およびこれらの処理に使われる暗号鍵はアイソレート SPE 内に閉じて利用される．したがって，OS やハイパーバイザからもこれらの処理過程や暗号鍵を盗み見ることはできない．今回，我々は，Cell/B.E. に最適化された AES-128/CBC モードと AES-128/ECB モードの実装を拡張して本機能を実現した．AES-128/CBC モードと AES-128/ECB モードの性能を**表 1** に示す．

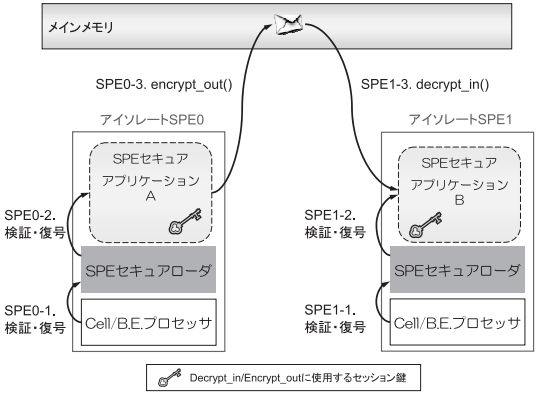

図 8　Decrypt-in/Encrypt-out プログラミングモデル
Fig. 8　Decrypt-in/Encrypt-out programming model.

なお，複数 SPE 間でのセッション鍵を共有する方法としては，たとえば，メインメモリを共有バッファとして利用し，公開鍵暗号方式を利用した鍵交換を行う手法が考えられる．動的なセッション鍵の交換の実装については，アプリケーションレイヤでの実装を仮定している．

表 1　AES-128/CBC モードおよび AES-128/ECB モードの性能[7)]
Table 1　AES-128 performance.

アルゴリズム	SPE (Gb/sec)	Leading Brand CPU with SIMD (Gb/sec)
AES CBC Encrypt 128-bit key	0.795	0.968
AES CBC Decrypt 128-bit key	1.507	0.966
AES ECB Encrypt 128-bit key	2.059	1.029
AES ECB Decrypt 128-bit key	1.499	1.035

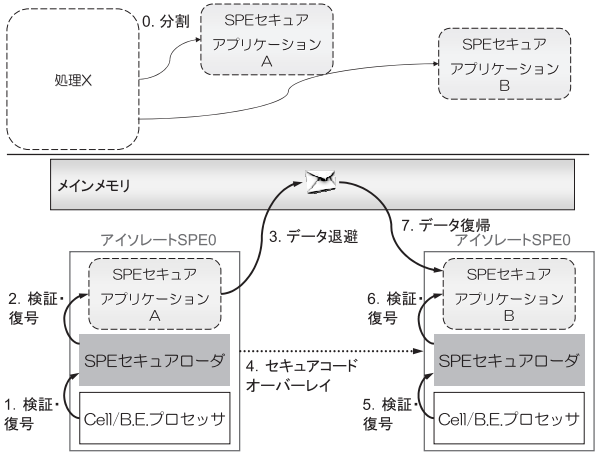

図 9　セキュアコードオーバレイ
Fig. 9　Secure code overlay.

6.2　セキュアコードオーバレイ

セキュアコードオーバレイは，アプリケーションのコードサイズが LS のサイズより大きい場合に有効である．特に，ある逐次処理を複数の SPE セキュアアプリケーションに分割可能な場合には，セキュアコードオーバレイを用いて，安全に次のアプリケーションに処理の続きを依頼できる．セキュアコードオーバレイの仕組みを**図 9** に示す．

たとえば，処理 X を SPE セキュアアプリケーション A と SPE セキュアアプリケーション B に分割する．まず，SPE セキュアアプリケーション A をアイソレーションモードでロードする．SPE セキュアアプリケーション A は何らかの演算を行い，中間データを生成し，このデータをメインメモリに退避させる．

次に，SPE セキュアアプリケーション B に切替え要求を libspe2 に対して行う．具体的には，`spe_program_load`，`spe_context_run` の引数に SPE セキュアアプリケーション B を指定して，再度呼び出せばよい．切替え要求が成功すると，再び，ハードウェア認証層から順に完全性検証処理および復号処理を行い，SPE セキュアアプリケーション B を起動する．SPE セキュアアプリケーション B はメインメモリ中の中間データをアイソレート SPE 内にロードして処理の続きを実行する．先の Decrypt-in/Encrypt-out プログラミングモデルとセキュアコードオーバレイを組み合わせることで，中間データを秘匿した状態で次の SPE セキュアアプリケーションに渡すことも可能である．

7.　応用例：コンテンツ保護

我々は，本 Cell/B.E. Security SDK を利用して，コンテンツ保護システムを構築した（**図 10**）．本システムは，AACS に準拠[2)] したコンテンツ復号鍵を生成する AACS コンポーネントと，コンテンツ復号鍵で MPEG コンテンツを復号し，MPEG コンテンツをデコードする MPEG コンポーネントで構成される．AACS コンポーネント，MPEG コンポーネントはともにハードウェアによって認証された SPE セキュアローダによって完全性検証が行われ，改ざんが認められなった場合に実行が許可される．AACS コンポーネントは，自身の保有するデバイス鍵（秘匿データ）からコンテンツ復号鍵を生成し，その鍵情報を MPEG コンポーネントに渡すことで，暗号化された MPEG コンテンツを復号する．この鍵情報の受け渡しにセキュアコードオーバレイを利用している．なお，コンテンツ復号鍵の受け渡しの際に利用する暗号鍵については，両コンポーネント間であらかじめ共有させている．コンテンツ復号鍵の受け渡しに用いた暗号方式は AES-128/CBC モードである．

図 10 のようにデコードしたフレームデータをメインメモリに展開する場合には，`encyrpt_out()` API を利用して暗号化し，動き保証の計算など過去のフレームが必要になる場合には，`decrypt_in()`API を利用してアイソレート SPE 内にフレームデータを展開する．今回の実装では，AES-128/ECB モードを指定して両 API 群を呼び出している．

デバイス鍵は，AACS コンポーネント中に埋め込まれているが，アプリケーションは暗号化され，かつ，アイソレートされた LS 内でのみ平文に展開されるので，漏洩の危険性はない．我々は，実際にデバイス鍵に対する攻撃として，[A-1] から [A-3] の攻撃を試みた．[A-1] の攻撃では，攻撃スレッドを PPU 上に走らせ，アイソレート LS 内に展開されるデバイス鍵の検索を行った（図 10）．本攻撃は，Cell/B.E. のハードウェアによってアクセスが拒否されるため，攻撃対象となる LS 内のデータ参照は行えない．また，[A-2] の攻撃を

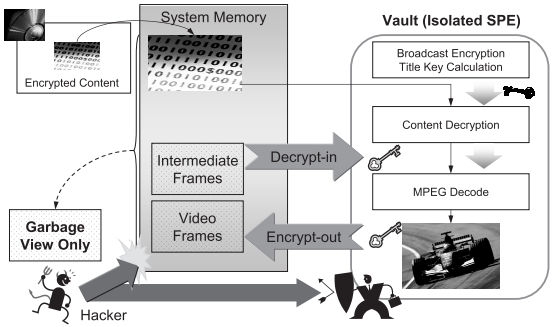

図 10　コンテンツ保護システムにおける動的攻撃への対応
Fig. 10　Protection mechanism against memory snooping in a content protection system.

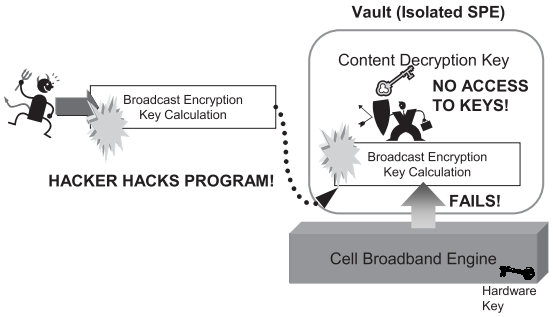

図 11　コンテンツ保護システムにおけるプログラム改ざん攻撃例
Fig. 11　Defence against tampering in a content protection system.

行うため，我々は暗号化し，署名された AACS コンポーネントを改ざんし，LS 内のデータをメインメモリに書き出すコードを追加した（**図 11**）．この改ざん AACS コンポーネントを実行した結果，改ざん AACS コンポーネントのハッシュ値と電子署名内に含まれるハッシュ値とが一致せず，SPE セキュアローダによって実行が中止されることを確認した．最後に [A-3] の攻撃として，AACS コンポーネントに埋め込まれたデバイス鍵の検索を試みた．前述のようにデバイス鍵は AES-128/CBC モードで暗号化されており，この暗号化領域の解読は困難である．また，アプリケーション暗号化処理において，本 SDK で提供される SPE セキュアアプリケーションビルドツール群は，脆弱鍵，準脆弱鍵の使用を拒否する．したがって，これらの鍵を SPE セキュアアプリケーションの暗号鍵として利用することはない．

一方，平文コンテンツをメインメモリあるいはデコード処理中の SPE から奪取する攻撃についても同様に検証した．[A-1] の攻撃で，攻撃スレッドを PPU 上で走らせ，メインメモリ内に展開されるフレームデータのアドレスをつきとめたとする．しかし，メインメモリ内のフレームデータは暗号化されており，モザイク映像しか得られない．また，アイソレート LS 内に展開された平文のフレームデータは，Cell/B.E. のハードウェアによってアクセスが拒絶される．[A-2] の攻撃では，デバイス鍵に対する攻撃のときと同様に，改ざんされた MPEG コンポーネントは，SPE セキュアローダによって実行が中止されるので，平文コンテンツをアイソレート LS から漏洩させるようなコードを実行することはできなかった．なお，MPEG コンポーネントに平文のコンテンツは含まれていないので，[A-3] の攻撃は除外した．

8. 考　　察

SPE にセキュアアプリケーションをロードする場合，アプリケーションの完全性検証と復号処理のため，処理オーバヘッドが生じる．そこで，SPE セキュアアプリケーションのロード時間を計測した．今回の実験では，アプリケーションの一部（.data セグメント）を暗号化し，本セグメント領域を 1 K バイトから 100 K バイトまで 10 K バイト刻みで増加させて計測した．SPE アイソレーションモードをサポートしたシミュレータを用いて計測した結果を**表 2** に示す．表に示すロード時間とは，アプリケーションを LS にコピーし始めてから，アプリケーションの完全性を検証し，アプリケーションの暗号箇所を復号してから，実際にアプリケーションのエントリポイントにジャンプするまでの時間を指す．表 2 から分かるように，SPE セキュアアプリケーションをロードする際に，50 msec 弱のオーバヘッドをともなう．しかし，アプリケーションサイズが増加しても，ロード時間はほとんど増えない．これは，アプリケーション暗号鍵を RSA によって復号する処理に大半の時間が費やされていることを意味する．本 SDK では，2,048 ビットの鍵長を利用しているが，安全な鍵長の下限が 2,048 ビットより大きくなった場合，ロード時間はさらに長くなる．

また，SPE は 256 K バイトのローカルメモリしかないため，デコードされた画像データは暗号化してメインメモリに書き出す必要がある．特に，MPEG のような動き補償があるデコード処理では，メインメモリに書き出したフレームを再びアイソレート SPE 内にロードし，復号して対象となるフレームをデコードする必要があり，本 SDK の Decrypt-in/Encrypt-out API 群によってメインメモリ・LS 間の安全なデータの受け渡し

表 2　3.2 GHz Cell/B.E. を利用した場合の SPE セキュアアプリケーションロード時間
Table 2　SPE secure application boot time on the 3.2 GHz Cell/B.E. processor.

アプリケーション イメージ（K バイト）	3.125	12.125	22.125	32.125	42.125	52.125	62.125	72.125	82.125	92.125	102.125
暗号データ サイズ（K バイト）	1	10	20	30	40	50	60	70	80	90	100
ロード時間 (msec)	47.81	47.90	47.99	48.08	48.18	48.27	48.36	48.46	48.55	48.64	48.74

が実現可能となる．一方，Motion JPEG のようにフレーム間の依存処理がない場合は，最終画像を暗号化してメインメモリに書き出すのみでよいため，MPEG よりもデコード処理において暗号化・復号の計算量を減らせる．また，表示装置が暗号化インタフェースをサポートする場合には，表示装置とアイソレート SPE とで暗号化通信路を確立し，暗号化されたデコード結果をそのまま表示装置に転送でき，end-to-end のコンテンツ保護が期待できる．

9. おわりに

Cell/B.E. Security SDK によって，アプリケーションの動的解析による秘匿データ漏洩，アプリケーション実行ファイル改ざんによる秘匿データ漏洩，アプリケーション実行ファイルの静的解析による秘匿データ漏洩の 3 つの脅威に対抗する手段を実現した．具体的には，アイソレーション実行の有効化，Cell/B.E. が提供するランタイムセキュアブートに基づくアプリケーションセキュアロード，さらに暗号アプリケーション実行のサポートである．また，256 K バイトという小さな LS 空間を有効に使うために，メインメモリを安全に利用する Decrypt-in/Encrypt-out プログラミングモデル，セキュアコードオーバレイという 2 つの機能をライブラリとして実現した．今後は，さらにこれらを拡張し，サポートする暗号アルゴリズムの強化や動的なセッション鍵の交換についても本 SDK 上で実現していく．

Cell/B.E. Security SDK を利用することで，アプリケーションプログラマは秘匿データを扱うシステムを，再利用性や拡張性の点で柔軟に対応できるソフトウェアで安全に実装できるようになった．これによって，システム開発の短縮やコスト削減が期待できる．今後は，コンテンツ保護分野のみでなく，Cell/B.E. の高速な暗号処理能力と高度なセキュリティ機能を組み合わせたアプライアンスやアクセラレータへの応用についても検討していく．

謝辞　本研究を進めるにあたり，株式会社ソニー・コンピュータエンタテインメント社より Cell/B.E. 向け MPEG-2 デコーダをご提供いただいた．ここに記して感謝の意を表する．

参考文献

1) 4C Entity. http://www.4centity.com/
2) AACS Specifications. http://www.aacsla.com/specifications/
3) Cell Broadband Engine Architecture Version 1.02. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/1AEEE1270EA2776387257060006E61BA/$file/CBEA_v1.02_11Oct2007_pub.pdf
4) Cell Broadband Engine Programming Handbook. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/9F820A5FFA3ECE8C8725716A0062585F/$file/CBE_Handbook_v1.1_24APR2007_pub.pdf
5) Cell/B.E. Security SDK. http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/AEBFE7D58B5C36E90025737200624B33/$file/CBE_Secure_SDK_Guide_v3.0.pdf
6) Chen, B. and Morris, R.: Certifying program execution with secure processors, *HOTOS'03: Proc. 9th Conference on Hot Topics in Operating Systems*, Berkeley, CA, USA, USENIX Association, pp.23–23 (2003).
7) Chen, T., Raghavan, R., Dale, J. and Iwata, E.: Cell Broadband Engine Architecture and its first implementation—A performance view, *IBM Journal of Research and Development*, Vol.51, No.5, pp.559–572 (2007).
8) Collberg, C.S. and Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection, *IEEE T. Softw. Eng.*, Vol.28, No.8, pp.735–746 (2002).
9) DTLA. http://www.dtcp.com/
10) GNU Toolchain 4.1.1 and GDB for the Cell BE's PPU/SPU. http://www.bsc.es/projects/deepcomputing/linuxoncell/
11) IBM SDK for Multicore Acceleration V3.0. http://www-03.ibm.com/technology/cell/swlib.html
12) Jin, H., Leake, D.E.J., Lotspiech, J.B., Nin, S.I. and Plouffe, W.E.: Tamper-resistant trusted java virtual machine and method of using the same (2005). United

States Patent Application 20050114683 (May 26, 2005).
13) Kim, G.H. and Spafford, E.H.: Experiences with tripwire: Using integrity checkers for intrusion detection, *System Administration, Networking and Security Conference III* (1994).
14) Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J. and Horowitz, M.: Architectural Support for Copy and Tamper Resistant Software, *Proc. 9th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-IX*), pp.168–177 (2000).
15) Microsoft NGSCB. http://www.microsoft.com/resources/ngscb/default.mspx
16) Percival, C.: Cache Missing for Fun and Profit (2005). http://www.daemonology.net/papers/htt.pdf
17) Panholzer, P.: Physical Security Attacks on Windows Vista (2008). http://www.sec-consult.com/fileadmin/Whitepapers/Vista_Physical_Attacks.pdf
18) Pham, D.: The Design and Implementation of a First-Generation CELL Processor, *Proc. 2005 IEEE International Solid-State Circuits Conference*, pp.184–187 (2005).
19) Shapiro, W. and Vingralek, R.: How to Manage Persistent State in DRM Systems, *DRM '01: Revised Papers from the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management*, London, UK, pp.176–191, Springer-Verlag (2002).
20) Suh, G.E., Clarke, D., Gassend, B., van Dijk, M. and Devadas, S.: The aegis Processor Architecture for Tamper-Evident and Tamper-Resistant Processing, Technical Report LCS-TM-460, MIT Laboratory for Computer Science (2003).
21) Trusted Computing Group. https://www.trustedcomputinggroup.org/specs/TPM/
22) Wang, C., Hill, J., Knight, J. and Davidson, J.: Software Tamper Resistance: Obstructing Static Analysis of Programs, Technical Report CS-2000-12, Dept. Computer science, University of Virginia, Charlottesville, VA, USA (2000).
23) Wilson, P., Frey, A., Mihm, T., Kershaw, D. and Alves, T.: Implementing Embedded Security on Dual-Virtual-CPU Systems, *IEEE Des. Test*, Vol.24, No.6, pp.582–591 (2007).
24) XL C/C++ for Multicore Acceleration, V9.0. http://publib.boulder.ibm.com/infocenter/cellcomp/v9v111/index.jsp
25) Yee, B.: Using secure coprocessors, Ph.D. Thesis, Carnegie Mellon University (1994).
26) Zhang, X., van Doorn, L., Jaeger, T., Perez, R. and Sailer, R.: Secure coprocessor-based intrusion detection, *Proc. 10th Workshop on ACM SIGOPS European Workshop: Beyond the PC* (2002).

商標

- Cell Broadband Engine は，Sony Computer Entertainment Inc. の商標.
- Linux は，Linus Torvalds の登録商標.
- IBM，PowerPC，Power Architecture は，IBM Corporation の商標または登録商標.
- Microsoft は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標.
- その他の会社，製品の名称はその他の会社の商標.

(平成 19 年 11 月 30 日受付)
(平成 20 年 6 月 3 日採録)

村瀬　正名（正会員）

2001 年慶應義塾大学環境情報学部卒業，2003 年慶應義塾大学大学院政策・メディア研究科修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所入所．ユビキタスコンピューティング，センサネットワーク，セキュリティシステムの研究開発に従事．IEEE，ACM 各会員.

阪本　正治（正会員）

1987 年電気通信大学電気通信学部通信工学科卒業，1989 年電気通信大学電気通信学研究科通信工学専攻修士課程修了．同年日本アイ・ビー・エム（株）東京基礎研究所．2000 年東京大学数理科学研究科博士課程修了（数理科学博士）．音声合成・音声認識，コンテンツ保護，並列アプリケーションの研究に従事．日本音響学会，IEEE 各会員.

清水かんな

1997 年米カルフォルニア工科大学電気工学科卒業，1998 年英オックスフォード大学計算機科学科修士課程修了．2002 年米スタンフォード大学電気工学科博士課程修了（Ph.D.）．現在，IBM コーポレーションシステムズ&テクノロジーグループ所属．同社のプロセッサセキュリティアーキテクト．フォーマルメソッドおよびセキュリティアーキテクチャの研究開発に従事.

ウィルフレッド　プルーフ

米イリノイ工科大学数学科卒業，米カリフォルニア大学計算機科学科博士課程修了（Ph.D.）．1981 年に IBM コーポレーションアルマデン研究所入所．ソフトウェアセキュリティ，コンテンツ保護，分散システムの研究開発に従事．現在，リアルタイム分散システム向けミドルウェア研究のグループマネージャ．ACM 会員．

ヴラディミール　ズバルスキー

2002 年米コーネル大学計算機科学科卒業．2002 年から 2008 年まで IBM コーポレーションアルマデン研究所．暗号技術応用，セキュリティアーキテクチャの研究開発に従事．