情報処理学会論文誌 Vol.56 No.8 1604–1620 (Aug. 2015)

ハードウェアセキュリティ機能を利用した長期安全性の確保が可能な組込みシステム

磯崎 宏[1,2,a)] 金井 遵[1]

受付日 2014年11月19日, 採録日 2015年5月9日

概要：近年，暗号通信機能を備えた組込み装置が普及しつつあるが，特に，産業用制御システムで利用される組込み装置はPCなどの情報系システムと比較して長期間敷設されることが想定される．このため長期安全性を確保するために暗号処理のような保護対象の処理を安全に更新することが課題となる．暗号モジュールをハードウェアとして実装すると不正な解析や改変に対する強度は高いが，1台あたりの製造コストに加えて物理的な交換が必要となるため更新コストが高い．一方，ソフトウェアとして実装すると更新は容易だが，解析されるリスクが高い．そこで本論文では，組込み向けプロセッサのセキュリティ機能であるTrustZoneを活用することで，解析や改変に対するセキュリティ強度を保ちつつ保護対象の処理とデータを部分的に更新可能とするソフトウェアシステムを提案した．提案手法では汎用処理から，更新対象の暗号モジュールと暗号モジュールの正当性を確認する検証処理を分離させてセキュア環境で実行させることにより，汎用処理が不正に改変された場合にも更新対象の処理とデータの保護を可能としている．さらに提案手法を実装して評価した結果，提案手法はセキュリティ機能を実行するハードウェアの大部分が置き換え可能であり，オーバヘッドは許容可能で小さいとの評価結果を得た．また，更新対象のモジュールを暗号化することでモジュール内部の機密情報を秘匿したまま配布できるほか，検証者は任意のタイミングで更新対象のモジュールのみ検証することができるため，従来手法と比較して装置外部から設計者の意図したとおりモジュールが更新されているかを確認するリモート検証のコストが低いことも明らかとなった．

キーワード：セキュリティ，組込み，TrustZone

Embedded System with Long-term Security Utilizing Hardware Security Function

Hiroshi Isozaki[1,2,a)] Jun Kanai[1]

Received: November 19, 2014, Accepted: May 9, 2015

Abstract: Recently, embedded devices with crypto function are widely deployed. More specifically, it is assumed that the embedded devices used in industrial control system will be deployed for a long period of time comparing to information technology system such as PC. Therefore, making secure updatable method for security sensitive process like cryptographic calculation to keep long term security is a significant challenge. When crypto module is implemented in hardware, it is difficult for an attacker to illegitimately analyze or modify the module while it will be more costly to update it since physical replacement is necessary in addition to the per-device cost. On the other hand, when it is implemented in software, it is easy to update it while it is more vulnerable to the attack. To solve this problem, we propose a software system which is able to update security sensitive module only with enough robustness against tampering to utilize TrustZone which is a security feature of an embedded processor. In our proposed system, the system is divided into general-purpose processes and a crypto process that is to be updated and a verification process which verifies the integrity of the crypto process, and the crypto process and the verification process are executed in a secure environment. As a result, even if the general-purpose processes are modified or the control is taken over, it is possible to keep the crypto process secure. We demonstrate a full implementation of our proposed system. The evaluation clarified that the proposed system can replace most of functions implemented in hardware and it clarified overhead cost is reasonably small. Furthermore, the system makes it possible to keep confidentiality of the crypto module when distributing it. Moreover, the system minimizes a remote attestation cost to verify the integrity of the system from outside since the verifier is required to have less knowledge about verification target and can verify it at arbitrary time.

Keywords: security, embedded, TrustZone

© 2015 Information Processing Society of Japan

1. はじめに

従来のコンピュータセキュリティは，主として暗号アルゴリズムや暗号プロトコルといった暗号要素技術によって支えられてきた．暗号処理は計算量が多いため，従来の組込みシステムでは，暗号処理を専用のハードウェアで実装し，それをソフトウェアからデバイスドライバを介して利用するといった構成で実現されることが多かった．しかしながら，暗号処理をハードウェアで実装してしまうと，鍵などの秘匿データや処理内容をハードウェア内に隠ぺいすることで秘匿性は確保できるものの，処理内容を更新するには物理的な交換が必要となってしまう欠点がある．特にスマートメータなどの産業用制御システムで利用される組込み制御装置は長期間にわたり利用されることを前提としており，装置敷設後の暗号アルゴリズムや，その実装に対する脆弱性の発見に備えて更新可能な仕組みを備えておく必要がある．実際に，北米ではスマートグリッドで利用するシステムをアップデート可能なように設計・実装することがガイドラインなどで規定されている [1], [2] が，スマートメータは各家庭に設置されるため，鍵の漏えいなどセキュリティ事故が発生した場合，膨大な数の装置を物理的にいっせいに交換することは現実的ではない．また，産業用制御システムはスマートメータなどの組込み制御装置以外にも，端末装置と短距離通信で接続される PC などの操作装置，端末装置と長距離通信で接続されるサーバから構成される．特に組込み制御装置と通信する操作装置が汎用 PC で動作するアプリケーションソフトウェアであった場合には不正な解析などによって鍵などの機密データが漏えいするリスクは高くなる．このように，組込み制御装置には問題がなかったとしても，操作装置やサーバでデータが漏えいした場合には組込み制御装置のデータや処理内容を更新する必要がでてくる．さらに，組込み制御装置と通信する対向装置が DES のような古い暗号アルゴリズムしか備えていない場合，相互接続性を確保するために古い暗号アルゴリズムをサポートする必要があり，対向装置の更新に合わせて暗号アルゴリズムや鍵長を更新することも考えられる．そこで，暗号処理をソフトウェアで実装し，通信機能を利用してソフトウェアをアップデートさせることにより，新しい鍵に更新したり暗号アルゴリズムを変更したりすることが考えられる．しかしながら，ソフトウェアはハードウェアと比較して解析や改変が容易であり，信頼の基盤を構築することが困難であるため暗号処理の実行環境である OS やミドルウェアに脆弱性が存在すると，攻撃者によって脆弱性が悪用され，アップデートの仕組みそのものが改変されたり，更新対象のソフトウェアに含まれる鍵などの機密データが不正に取得されたりするリスクが高い [3]．このように，産業用制御システムで利用される組込み制御装置においては，暗号処理をソフトウェアで実装しつつ，長期的に安全性を保つために更新可能な仕組みを実現することが課題となっている．

暗号計算などの保護対象の処理の秘匿性を確保する手段として，従来はハードウェアとして実装していた保護対象の処理を，ファイルアクセスや通信など汎用処理と分離させ，保護対象の処理のみ解析や改変から保護されたセキュアソフトウェア環境で実行させるアーキテクチャが考えられる．この実現のため，我々は ARM プロセッサに備わっているセキュリティ機能である TrustZone を利用し，1 つのプロセッサで 2 つの OS を並行に実行させることのできるソフトウェアプラットホーム LiSTEE^{TM} を提案し，汎用 OS から機密性の高い処理を切り離して実行するプラットホームが実現可能であることを示した [4]．さらに OS 切替え時に退避・復帰させるレジスタの数を削減することで，LiSTEE^{TM} の高速な OS 間遷移が実現可能であることを検証した [5]．

しかし文献 [5] では，すべての機能をモノリシックに構成したアーキテクチャとなっている．このため，アップデート対象のモジュールに鍵など機密データやアルゴリズムの高速化手法といった実装上のノウハウが含まれる場合，保護対象の処理のみ後から配布して更新することはできない．また，サーバはシステム全体のバイナリオブジェクトを配布する必要があり，特に配信先の装置数が多い場合，処理負荷や通信量が高くなってしまう．この課題を解決するにはモジュール性を向上させる必要がある．文献 [4] では概略構成に触れているものの，汎用処理の性能低下を抑制しつつ，保護対象の処理を部分的に更新する仕組みと保護対象に含まれる鍵や処理内容の秘匿性を確保する仕組みを実現するには，具体的にどのように処理をノンセキュア環境とセキュア環境で実現すべきか，その構成が明らかになっていない．また，保護対象の処理を更新した後，管理者がシステムの状態を調査し，期待どおりに更新されているかを確認することも求められるが，文献 [4], [5] では具体的な構成が述べられていない．たとえば，スマートメータから収集した電力消費量などのデータはサーバ側で集計されるが，そのデータが正当なスマートメータから送信されているかどうかをシステム管理者が確認するためには，正しい鍵を持ち安全な実装がなされた保護モジュールによって暗号化されたデータであることをサーバ側で確認することが必要である．そのための手法としてトラステッドブートが提案されている [6] が，トラステッドブートではブート時にシステム全体の正当性を検証することができる一方

1 株式会社東芝研究開発センター
Corporate R&D Center, Toshiba Corporation, Kawasaki, Kanagawa 212–8582, Japan
2 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University, Fujisawa, Kanagawa 252–0882, Japan
a) hiroshi.isozaki@toshiba.co.jp

© 2015 Information Processing Society of Japan

で保護対象の処理とは無関係な処理を含むシステム全体を検証する必要があり，検証者のコストが高い．特にスマートメータのような複雑なシステムでは保護対象の処理とは無関係の様々な機能が搭載され，それらの機能はセキュリティ上の理由とは独立に個別に更新されていくと予想されるが，システム管理者がそれらすべての機能のバージョンを管理することは現実的ではない．検証者のコストを最小化するには，保護対象の処理に限り健全性を検証できる仕組みが求められる．さらに，トラステッドブートではブート時の状態しかシステムの健全性を確認できない致命的な制約もある．このため，健全性を確認した後に攻撃によって保護対象の処理が改変され，改変された状態で運用され続けたとしても検証者は気が付くことができない．これを解決するには，検証時に保護対象の処理の状態を取得し，健全であるかを確認できる必要がある．また，この確認はネットワークを介して任意のタイミングで行えることが望ましい．その場合，保護対象の処理が健全であることを示す情報が通信経路上や汎用処理を実行する OS 上で改ざんされては困るため，何らかの方法で保護する必要がある．このように，検証者のコストを低減させつつ，任意のタイミングでシステムの健全性をネットワーク経由で確認できることが理想的だが，その手法が明らかになってない．

そこで本論文では，LiSTEE^{TM} を用いて汎用処理から，保護対象の暗号処理とその処理の正当性を確認する検証処理を分離させ，保護対象の処理と検証処理をセキュア環境で実行させることにより，汎用処理が改変された場合でも更新対象の処理とデータを保護することが可能なシステムを実現する．提案手法では，汎用処理の機能低下を最小限に抑え，ハードウェアを交換せずに，暗号処理に含まれる情報を秘匿しつつ配布して安全にアップデートさせる方法を提供することで，長期安全性を確保する制御装置を実現することができる．さらに，保護対象の処理を更新した後，保護対象の処理に限定して状態を確認する手段を提供することで検証者のコストを最小化するとともに，ブート時に限定せず任意のタイミングで保護対象の検証を行うことができる方法を提供することにより，システム全体の安全性を向上することができる．

以下，2 章と 3 章では本研究で利用する TrustZone と TPM の機能について述べる．4 章では提案手法の設計について述べる．5 章で実装について述べ，6 章で評価を行う．7 章で関連研究について述べ，8 章でまとめと今後の課題について述べる．

2. TrustZone

本章では，LiSTEE^{TM} が動作する ARM プロセッサが持つ TrustZone の機能について述べる．

TrustZone とは，一部の ARM プロセッサに備わっているハードウェアセキュリティ機能である [7], [8]．ARM プロセッサは，ユーザモードと特権モードの 2 種類のモードが定義されている．特権モードではすべての命令の実行と，メモリのアクセスが許可されているが，ユーザモードでは実行可能な命令やレジスタに対するアクセスが制限されている．汎用のシステムでは，OS を特権モードで実行し，アプリケーションプログラムをユーザモードで実行するといったように，ユーザモードと特権モードを切り替えながら様々な処理を行う．

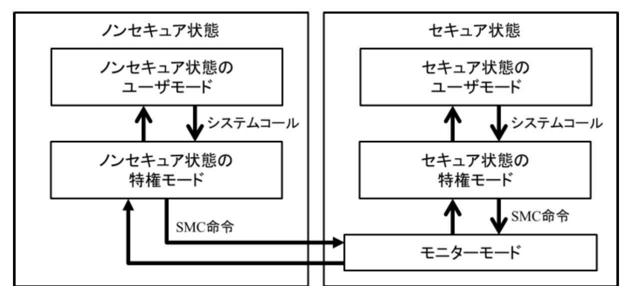

図 1 TrustZone 対応プロセッサの状態遷移
Fig. 1 State transition of TrustZone capable processor.

TrustZone 対応の ARM プロセッサでは，この 2 種類のモードに直行する概念として，新たにセキュア状態とノンセキュア状態が定義されている．図 1 に TrustZone 対応プロセッサの状態遷移を示す．プロセッサの状態は NS ビットと呼ばれるプロセッサのレジスタの設定によって決定され，プロセッサはこの 2 つの状態を選択的に切り替えながらプログラムを実行する．この仕組みを利用し，メモリアクセス制御機構に対してセキュア状態からのみアクセス可能な領域，セキュア状態・ノンセキュア状態両方からアクセス可能な領域といったように各メモリ領域のアクセス権限を適切に設定することで，保護対象と非保護対象のデータとコードを分離することができる．たとえば，プロセッサが特権モードであってもノンセキュア状態の場合にはセキュア状態で管理するメモリ領域へのアクセスを禁止するといった設定が可能となる．

さらに，TrustZone 対応のプロセッサはセキュア状態とノンセキュア状態を相互に切り替えるためのモニタモードを定義している．モニタモードでは前述の NS ビットの設定に関係なく，プロセッサはセキュア状態で動作する．

プログラムは SMC 例外命令と呼ばれる特殊命令を実行することによってプロセッサをモニタモードに遷移させることができる．SMC 命令が実行されると，プロセッサは状態を強制的にモニタモードに遷移させ，さらに，あらかじめ登録されているモニタ用のベクタに格納されている命令を実行する．モニタモードで動作するプログラムは現在のモードの状態をメモリに保存し，メモリに退避してあった遷移後のモードの状態をリストアする処理を行う．このようにして，1 つのプロセッサの中に，セキュア状態とノンセキュア状態を仮想的に構築し，切り替えながら実行することができる．

© 2015 Information Processing Society of Japan

3. TPM

本章では，TPM（Trusted Platform Module）の機能と，その利用例について説明する．

TPMとは，乱数生成機能，鍵生成機能，セキュアハッシュ計算機能などの暗号処理機能をハードウェアとして備えたセキュリティチップである [9]．非営利団体 Trusted Computing Group で機能や暗号アルゴリズム，インタフェースが規格化されており，PCをはじめとする多くの民生機器に搭載されている．TPMは，ホストCPUとは独立したハードウェアであり，TPMを操作するためのインタフェースが仕様で規定されているため，それ以外の経路でTPM内部のデータを操作したり，処理内容を変更したりすることはできない．つまり，ホストCPUで動作するソフトウェアからTPMの処理内容を取得したり，変更したりすることができないようになっており，TPMをソフトウェアレベルの攻撃からシステムを保護するためのセキュリティアンカーとして利用することができる [10]．

また，TPMはPCR（Platform Configuration Register）と呼ばれる，プログラムやデータなどある一時点でのメモリの状態を保存するレジスタを備えている．入力データは，入力前のPCRの値と結合され，その値に対するハッシュ値が新しいPCRの値となる．PCRに任意の値を設定したり，任意のタイミングでリセットしたりすることはできない．このPCRを使い，BIOS，OS，ミドルウェア，アプリケーションといった順で装置の起動時から実行したプログラムのハッシュ値をPCRに積算値として順々に蓄積し，最終的に検証したい状態のPCR値と，あらかじめ計算しておいた予測値を比較する．このようにして，設計者の意図したとおりにシステムが起動したことを確認するトラステッドブートの実現手段としてTPMを利用することができる．さらに，PCRの値にTPMの秘密鍵によって公開鍵アルゴリズムで署名を計算することもできるため，ネットワークで接続されたサーバなど検証対象のシステムが意図したとおりに起動したかどうかをリモートで検証することもできる．

4. 提案方式 LiSTEE^{TM} 仮想ハードウェアの設計

本章では，提案方式 “LiSTEE^{TM}” の設計について説明する．なお，本論文では一連の機能をひとまとめにした概念的な用語としてモジュールを定義する．

4.1 全体アーキテクチャ

図 2 に LiSTEE^{TM} の全体アーキテクチャを示す．LiSTEE^{TM} は，暗号処理などの保護対象の処理を実行する LiSTEE^{TM} 仮想ハードウェア，ファイルアクセスやネットワーク処理などの汎用の処理を行うリッチ OS，LiSTEE^{TM} 仮想ハードウェア処理とリッチ OS の切替え処理を行う LiSTEE^{TM} モニタの3つから構成される．前述したセキュリティ要件を満たすためには，保護対象のモジュール，その保護対象のモジュールを検証，復号，実行する LiSTEE^{TM} 仮想ハードウェア，LiSTEE^{TM} モニタ，そして初期化コードとそれらの実行状態を信頼の基盤となる TCB（Trusted Computing Base）に含める必要がある．

- LiSTEE^{TM} 仮想ハードウェア
  LiSTEE^{TM} 仮想ハードウェアは，再暗号処理などの処理と鍵などの秘匿データからなる保護対象モジュール，保護対象のモジュールを検証したり復号したりする保護・更新モジュール，暗号処理を行う TPM モジュール，保護・更新モジュールの実行時間を監視する時間管理モジュールと，それらの処理モジュールをメモ

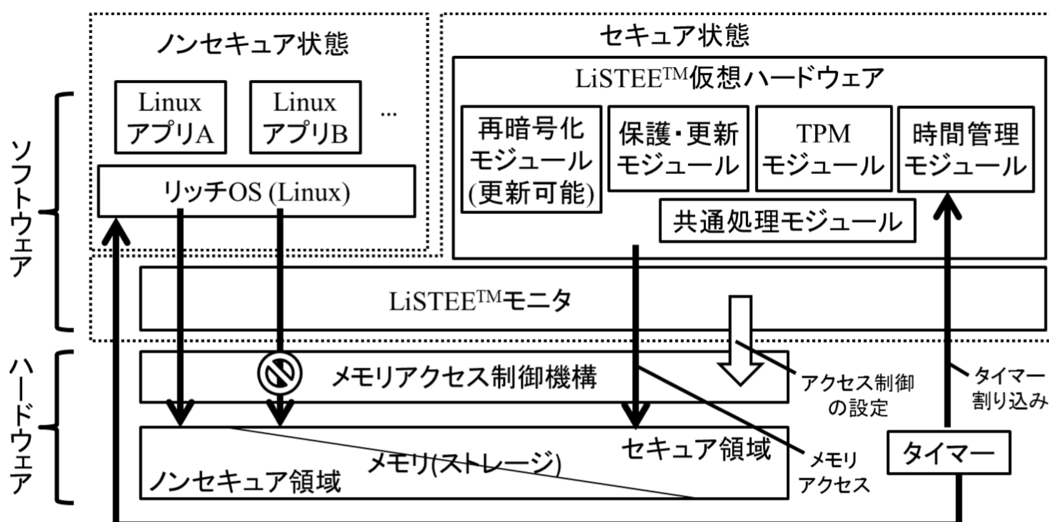

図 2 LiSTEE^{TM} の全体アーキテクチャ

Fig. 2 System architecture of LiSTEE^{TM}.

© 2015 Information Processing Society of Japan

リからロードして実行したり LiSTEE^{TM} モニタに対してリッチ OS に遷移したりするよう指示する処理を行う共通処理モジュールから構成される．LiSTEE^{TM} 仮想ハードウェアはセキュア状態の特権モードで動作する．

- リッチ OS
  ファイルアクセスやネットワーク通信などの汎用処理と，保護対象の処理の実行を依頼する処理を行う．LiSTEE^{TM} ではリッチ OS として Linux をサポートしている．デバイスドライバが LiSTEE^{TM} 仮想ハードウェアとのデータ送受信や LiSTEE^{TM} モニタと通信して OS 間遷移命令を送信することで，リッチ OS からは LiSTEE^{TM} 仮想ハードウェアが仮想的にハードウェアモジュールとして動作するように見せることが実現できる．
- LiSTEE^{TM} モニタ
  LiSTEE^{TM} 仮想ハードウェアとリッチ OS は排他的に実行する必要があるため，LiSTEE^{TM} 仮想ハードウェアとリッチ OS からの要求に基づいて，TrustZone の機能を使いコンテキスト切替え処理を行う．また，起動時にメモリアクセス制御機構に対してメモリのアクセス権限を適切に設定する．

4.2 LiSTEE^{TM} の機能

LiSTEE^{TM} は保護対象モジュールの実行機能，保護対象モジュールの更新機能，保護対象モジュールの検証機能の 3 つの機能から構成される．

(1) 共通基本機能

アクセス制御ポリシ設定機能

LiSTEE^{TM} モニタは初期化処理として装置起動時に TrustZone のメモリアクセス制御機構を利用してメモリアクセス制御のポリシを設定する．メモリを，LiSTEE^{TM} 仮想ハードウェアとリッチ OS のデータ交換に用いる共有領域，リッチ OS およびリッチ OS 上で動作するアプリケーション用のノンセキュア領域，そして LiSTEE^{TM} 仮想ハードウェア用のセキュア領域の 3 つの領域に分割する．共有領域とノンセキュア領域は，LiSTEE^{TM} 仮想ハードウェアとリッチ OS 双方からアクセス可能なように設定する．なお，共有領域とノンセキュア領域のアクセス制御の設定は同一であるが，ここでは目的を明確化するために別々の用語として説明する．一方，セキュア領域は LiSTEE^{TM} 仮想ハードウェアからのみ参照可能なように設定しておく．

データ転送機能

リッチ OS と LiSTEE^{TM} 仮想ハードウェアの通信機構として，LiSTEE^{TM} ではデータ転送機能を提供している．前述のように，LiSTEE^{TM} 仮想ハードウェアはコードサイズの肥大化によって脆弱性を包含してしまうリスクを最小化するため，デバイス制御用のデバイスドライバを内包しない．このため，LiSTEE^{TM} 仮想ハードウェアがディスクデバイスやネットワークデバイスなどからデータを取得するには，まずリッチ OS がデバイスの制御およびデータのリード・ライトを実行する．そして，共有領域と設定されているメモリ領域にリッチ OS がデータをライトする．なお，リッチ OS のデバイスドライバによって共有領域をリッチ OS 上の仮想アドレス空間にマップしているため，リッチ OS からは，通常のメモリ領域にアクセスするのと同様の仕組みでデータのリード・ライトを行うことができる．その後，OS 間遷移機能を用いて LiSTEE^{TM} 仮想ハードウェアに処理を遷移させる．LiSTEE^{TM} 仮想ハードウェアは共有領域からデータをリードし，セキュア領域内でデータを処理する．LiSTEE^{TM} 仮想ハードウェアからリッチ OS にデータを送信する場合には，まず LiSTEE^{TM} 仮想ハードウェアが共有領域にデータをライトする．その後，OS 間遷移機能を用いてリッチ OS に処理を遷移させる．リッチ OS は共有領域と設定されているメモリ領域からデータをリードする．このようにして LiSTEE^{TM} 仮想ハードウェアとリッチ OS の間でデータを交換する．

OS 間遷移機能

リッチ OS と LiSTEE^{TM} 仮想ハードウェアを切り替えながら実行するための機構として，モニタモードで動作する LiSTEE^{TM} モニタは OS 間遷移機能を提供している．NS ビットの設定によるプロセッサの状態切替えとともに，OS のコンテキスト保存と復帰処理も LiSTEE^{TM} モニタが行う．TrustZone では，NS ビットによって仮想的にセキュア状態とノンセキュア状態の 2 つの状態を提供しているが，リッチ OS と LiSTEE^{TM} 仮想ハードウェアは同一のレジスタを利用しているため，そのまま遷移先のコードを実行させると遷移元のコンテキストを破壊してしまう恐れがある．したがって，LiSTEE^{TM} モニタは切替え元のコンテキストをメモリに退避し，メモリに退避してあった切替え先のコンテキストを復帰する処理を行う必要がある．また，リッチ OS からモニタモードに遷移するための SMC 命令を発行するためにリッチ OS 用のデバイスドライバも実装している．

(2) 保護対象モジュールの実行機能

LiSTEE^{TM} 仮想ハードウェアの実行手順について説明する．ここでは LiSTEE^{TM} 仮想ハードウェアとして再暗号化処理を例にとって説明する．スマートメータなどの組込み制御装置では，センサ装置などの近接装置と通信する際に通信プロトコルとして ZigBee で通信し，サーバなど宅外装置と通信する際には通信プロトコルとして TCP/IP で通信するといったように複数のネットワークインタフェースを備える場合がある．各ネットワークインタフェースで用いる暗号アルゴリズムやデータ長はプロトコルによって異なるため，プロトコル変換の過程でデータの再暗号化が必要となることが多い．再暗号化を行うには，データを一

© 2015 Information Processing Society of Japan

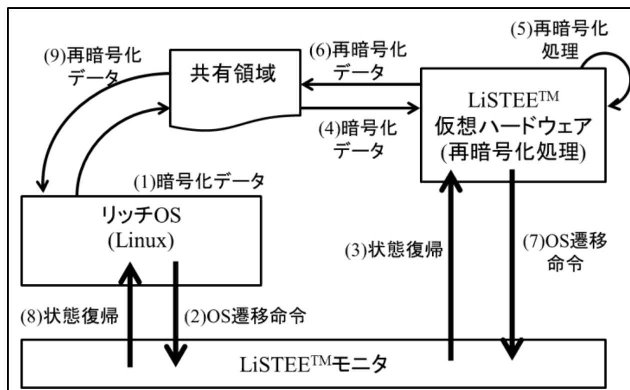

図 3 保護対象モジュールの実行の流れ
Fig. 3 Execution flow of protected module.

度平文に戻す必要があるため，この処理を保護することが重要となる．また，暗号処理と同時に通信処理も行うため，リッチ OS の処理も同時並行して行う必要がある．この処理について**図 3** を用いて説明する．

まず，リッチ OS 側でセンサデバイスやネットワーク，ディスクから暗号化されたデータをロードし，そのデータをメモリの共有領域に書き込み（処理 (1)），OS 間遷移命令を呼び出す（処理 (2)）．処理が LiSTEE™ モニタに遷移し，LiSTEE™ モニタは OS 間遷移機能でリッチ OS から LiSTEE™ 仮想ハードウェアに処理を切り替える（処理 (3)）．LiSTEE™ 仮想ハードウェアの再暗号化モジュールは共有領域から暗号化されたデータを読み込み（処理 (4)），セキュア領域内で再暗号化処理を実行した後（処理 (5)），再暗号化済みデータを共有領域に書き戻して（処理 (6)），共通処理モジュールに戻り，OS 間遷移命令を呼び出す（処理 (7)）．すると，再び LiSTEE™ モニタに処理が遷移し，OS 間遷移機能でリッチ OS に処理を切り替える（処理 (8)）．最後にリッチ OS は再暗号化されたデータを共有領域から読み込む（処理 (9)）．再暗号化対象のデータが終了するまで，この一連の手順を繰り返し実行する．リッチ OS が OS 間遷移命令の実行を依頼してから，再び状態が復帰されるまでの間，リッチ OS の処理は中断されたままとなる．再暗号化対象のデータサイズが大きい場合，再暗号化処理に要する時間が長くなるため，リッチ OS の処理が中断する時間も長くなる．リッチ OS の処理が停止可能な許容時間はアプリケーションによって異なるが，再暗号化するデータを分割して転送することで OS 間遷移を含めた 1 回の再暗号化処理にかかる時間を短くし，リッチ OS の処理が長時間にわたり停止することを防止できる．なお，一般的に暗号処理では 1 回の処理で暗号化するデータサイズは暗号アルゴリズムによって異なる．たとえば，鍵長 128 ビットの AES の場合，データの暗号化サイズは 128 ビット単位となる．したがって，データサイズが 128 ビットの倍数でしか暗号化できず，分割の最小単位は 128 ビットとなる．よって，リッチ OS 側の最小停止時間は 128 ビットのデータを処理する時間となる．なお，LiSTEE™ 仮想ハードウェア内でデータをパディングすることにより，128 ビット以下のデータも処理することが可能であるため，リッチ OS から LiSTEE™ 仮想ハードウェアに転送するデータサイズは暗号アルゴリズムに依存しない．このように，再暗号化処理はセキュア状態で実行されるため，再暗号化処理とは直接関係のない OS などの汎用処理部分に脆弱性があり，攻撃者がその汎用処理部分を完全に制御してしまったとしても，不正なデバッグによる解析攻撃や処理フローの改変攻撃を防止することが可能である．

(3) モジュール更新機能

LiSTEE™ では，LiSTEE™ 仮想ハードウェア全体ではなく，再暗号化モジュールなど保護対象のモジュールのみを更新する方法を提供している．前述のように，LiSTEE™ 仮想ハードウェア自体にはディスクデバイスにアクセスするデバイスドライバ機能を備えていない．そこで，LiSTEE™ 仮想ハードウェアの再暗号化モジュールを更新するには，まずリッチ OS が更新対象のモジュール，この場合，新しい再暗号化モジュールのバイナリオブジェクトファイルをディスクからロードし，共有領域と設定されているメモリ領域にライトする．LiSTEE™ モニタは初期化処理時に再暗号化モジュールを配置するメモリ領域をセキュア領域に割り当てている．LiSTEE™ 仮想ハードウェアの保護・更新モジュールは，このセキュア領域のメモリマップに関する情報を有しており，共有領域に書き込まれた新しい再暗号化モジュールのバイナリオブジェクトをセキュア領域の元々再暗号化モジュールが配置されていた領域に上書きする．この際，再暗号化モジュールのバイナリオブジェクトが再暗号化モジュール用に割り当てられたサイズ以下であることを確認する．再暗号化モジュール用に割り当てられた領域のサイズに収まるのであれば，新しい再暗号化モジュールのバイナリオブジェクトのサイズが更新前の再暗号化モジュールよりも大きかったとしても問題なく上書きすることができるが，それ以上のサイズの場合には再暗号化モジュールの更新を行わず，保護・更新モジュールはエラーとして OS 間遷移機能を利用して LiSTEE™ 仮想ハードウェアからリッチ OS に遷移する．LiSTEE™ 仮想ハードウェアの共通処理モジュールが再暗号化モジュールを呼び出す際，再暗号化モジュールの先頭アドレスの命令を実行する．したがって，更新後の再暗号化モジュールが先頭アドレスから実行するように構成されていれば，更新後の再暗号化モジュールを実行することができる．

また，保護対象のモジュールは通信経路上やリッチ OS 上で改変される危険性があるため，認証されたモジュールのみアップデートを許可する検証の仕組みを備えている．まず，保護対象モジュールの開発者に保護対象モジュールのバイナリオブジェクトに署名するための鍵（署名用の秘密鍵）をあらかじめ渡しておく．LiSTEE™ 仮想ハード

© 2015 Information Processing Society of Japan

ウェアのTPMモジュールには署名用の秘密鍵に対応する公開鍵が埋め込まれている．保護対象モジュールの開発者は，バイナリオブジェクトのハッシュ値を計算し，署名用の秘密鍵を使って保護対象モジュールに署名を施し，バイナリオブジェクトに添付して配布する．保護・更新モジュールは再暗号化モジュールのバイナリオブジェクトを共有領域から読み込んでセキュア領域に上書きする際，再暗号化モジュールのハッシュ値を計算し，LiSTEE^{TM} 仮想ハードウェアのTPMモジュールが管理する公開鍵を使って署名の検証処理を行い，一致した場合のみロードを許可する．これにより，LiSTEE^{TM} 仮想ハードウェアが未検証の再暗号化モジュールを実行してしまうことを防止することができる．なお，署名検証処理はセキュア状態で実行される．したがって，保護対象の処理とは直接関係のないOSなどの汎用処理部分が改変されたとしても，検証処理が改変されたり検証処理に用いる鍵などのデータが改変されたりすることはない．また，更新対象を保護対象モジュールである再暗号化モジュールに限定することができるため，リッチOSとは独立にアップデートすることができる．また，署名に用いる暗号アルゴリズムと鍵はその時点で標準として推奨されているものの中でも，最新のアルゴリズムと長い鍵長を選択しておく．

(4) モジュール保護機能

保護対象モジュールには秘密のデータが含まれている場合がある．たとえば，再暗号化モジュールにはデータを復号するための鍵と，再暗号化するための鍵が含まれている．しかし，保護対象モジュールのバイナリオブジェクトはリッチOSによってファイルとして管理されているため，逆アセンブルなどの静的解析によってそれらの鍵が不正に解読される危険性がある．そこで，LiSTEE^{TM} では保護対象モジュールをあらかじめ暗号化して保護する方法を提供している．まず，保護対象モジュールの開発者に保護対象モジュールを暗号化するための鍵（暗号鍵）をあらかじめ渡しておく．保護対象モジュールの開発者は，その暗号鍵を使って，保護対象モジュールのバイナリオブジェクトを暗号化して配布する．リッチOSがネットワーク経由で暗号化された保護対象モジュールのバイナリオブジェクトをダウンロードし，共有領域に暗号化された保護対象モジュールのバイナリオブジェクトを書き込む．LiSTEE^{TM} 仮想ハードウェアの保護・更新モジュールは保護対象モジュールのバイナリオブジェクトをセキュア領域にコピーする際，LiSTEE^{TM} 仮想ハードウェアのTPMモジュール内に埋め込まれた鍵（復号鍵）によって，バイナリオブジェクトを復号する．セキュア領域以外では保護対象モジュールが平文になることはないため，保護対象モジュール内の定数や，モジュールの処理内容を秘匿することができる．なお，バイナリオブジェクトの暗号・復号に用いる暗号アルゴリズムと鍵はその時点で標準として推奨されているものの中でも，最新のアルゴリズムと長い鍵長を選択しておく．

(5) 保護対象モジュールのリモート検証機能

TPMを用いたトラステッドブートでは，装置起動時からロードした全モジュールのハッシュ値をPCRに順々に格納し，そのPCRの値に対してTPMの秘密鍵で計算した署名値を検証することで，検証者がシステム上で意図したモジュールがロードされていることを確認していた．一方，LiSTEE^{TM} では，LiSTEE^{TM} 仮想ハードウェアの保護・更新モジュールが保護対象モジュールのバイナリオブジェクトをセキュア領域にコピーする際にそのモジュールの平文のバイナリオブジェクトのハッシュ値を計算する．そして，TPMモジュールのPCRの値をリセットし，計算したハッシュ値をPCRに格納する．保護対象モジュールの更新は装置起動後から任意のタイミングでなされるが，従来のPCRは任意のタイミングでリセットすることができないため，PCRの機能をそのまま移植するだけでは十分でない．そこで，TPMモジュールのPCRでは，PCRの値を任意のタイミングでリセットできるようにする代わりに，リセットの呼び出しを保護・更新モジュールに限定している．これにより，PCRの不正なリセットを防ぎつつ，保護対象モジュールの更新時にハッシュ値を格納できるようにしている．さらにシステム外部からPCRの値を検証できるようにするために，まず保護・更新モジュールはTPMモジュールにPCRの署名値を要求する．TPMモジュールは自身に格納された秘密鍵を使い，公開鍵アルゴリズムによってPCRに対する署名を生成して，保護・更新モジュールに返す．保護・更新モジュールは，その値を共有領域に書き込む．リッチOSは共有領域から署名済ハッシュ値を取得する．このようにして，意図した保護対象モジュールが動作していることをシステム外部から確認することができる．

4.3 処理フロー

これまでに述べた機能について，モジュール更新時とリモート検証時の処理フローを整理して説明する．

(1) モジュール更新時の処理フロー

まず，保護対象モジュール開発時の処理について述べる．保護対象モジュールの開発者は保護対象モジュールを開発し，バイナリオブジェクトを生成する．そして，保護対象モジュールを暗号化するための鍵（暗号鍵）を用いてバイナリオブジェクトを暗号化する．さらに，暗号化されたバイナリオブジェクトに対するハッシュ値を計算して署名用の秘密鍵を用いて署名を生成し，バイナリオブジェクトに添付する．

次に，保護対象モジュールの更新処理について**図 4** を用いて説明する．前処理として，LiSTEE^{TM} モニタが初期化処理時に再暗号化モジュールを配置するメモリ領域をセキュア領域に割り当てる（処理 (1)）．さらにリッチOS

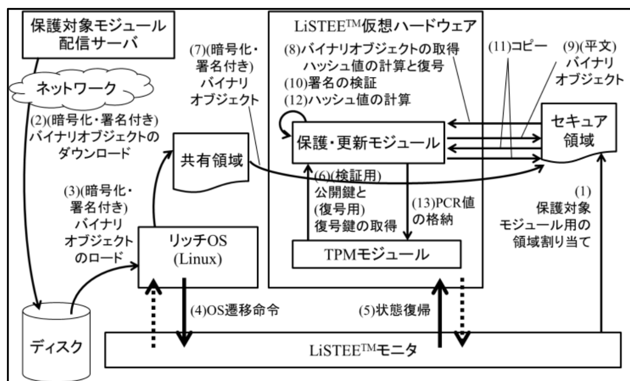

図 4　保護モジュールの更新処理のフロー図

Fig. 4　Execution flow of update process of protected module.

が暗号化・署名付きバイナリオブジェクトをネットワーク経由でダウンロードし，ディスクに保存しておく（処理 (2)）．以上が前処理であり，以降は実際にモジュールを更新する処理となる．まず，リッチ OS がディスクから暗号化・署名付きバイナリオブジェクトを読み込み，共有領域に書き込む（処理 (3)）．リッチ OS が SMC 命令を発行し，$\mathrm{LiSTEE^{TM}}$ の OS 間遷移機能で $\mathrm{LiSTEE^{TM}}$ 仮想ハードウェアに遷移する（処理 (4)，(5)）．$\mathrm{LiSTEE^{TM}}$ 仮想ハードウェアの保護・更新モジュールは TPM モジュールからバイナリオブジェクトを復号するための復号鍵と，バイナリオブジェクトに添付されている署名を検証するための公開鍵を取得する（処理 (6)）．次に暗号化・署名付きバイナリオブジェクトの復号とハッシュ値の計算を行うが，バイナリオブジェクトのサイズは復号やハッシュのブロックサイズと比較して大きい．したがって，暗号化・署名付きバイナリオブジェクト全体を一度に読み込むのではなく，ブロック単位で一部分を共有領域から読み込み，セキュア領域の保護・更新モジュールが管理するデータエリアにコピーし（処理 (7)），ハッシュ値を計算しつつ復号する（処理 (8)）．そして，平文となったバイナリオブジェクトの一部分をセキュア領域の保護・更新モジュールが管理する一時領域に追加していく（処理 (9)）．この処理はバイナリオブジェクト全体の復号が完了するまで繰り返し行う．最終的に得られた暗号化バイナリオブジェクト全体に対するハッシュ値と，バイナリオブジェクトに添付された署名，それに TPM モジュールから取得した公開鍵を使って署名の検証を行う（処理 (10)）．署名の検証が失敗した場合には，セキュア領域にコピーした平文バイナリオブジェクトを消去し，エラーとして OS 間遷移機能で $\mathrm{LiSTEE^{TM}}$ 仮想ハードウェアからリッチ OS に遷移する．検証が成功した場合には，保護・更新モジュールは平文となってセキュア領域の一時領域に保存したバイナリオブジェクトを読み込み，ハッシュ値を計算しつつ更新前の再暗号化モジュールが配置されていた部分にコピーする（処理 (11)，(12)）．この処理はバイナリオブジェクト全体のコピーが完了するまで繰り返し行う．そして，保護・更新モジュールは TPM モジュールの PCR の値をリセットし，最終的に得られた平文のオブジェクトファイルに対するハッシュ値を TPM モジュールの PCR に格納する（処理 (13)）．

保護対象モジュールのサイズが大きい場合，保護対象モジュールの復号処理や署名の検証処理に時間がかかってしまう可能性がある．組込み制御装置は本来の機能を実現するための様々な処理を実行しているが，モジュールの更新処理を実行している間，リッチ OS は停止してしまうため，問題となる場合がある．たとえば，本研究の応用例であるスマートメータの場合，家庭内の電力消費状況をネットワーク経由でサーバに送信したり，サーバからの依頼に基づいて家庭内の電力調整を行ったりするなど，いわゆるデマンドレスポンスサービスに対応する必要がある．保護対象モジュールの検証処理に要する時間が長くなってしまうと，リッチ OS の停止時間が長くなり，サーバからの指示に対応できなかったり，サーバとの通信が途絶えたりしてしまい，このデマンドレスポンスサービスを実現することができない．具体的な要求性能はシステム全体の運用形態や実現形態によって異なるが，たとえばスマートグリッドを想定したシステムでは管理サーバから送信されるデータに対する許容遅延が 50〜300 ミリ秒と報告されている [11]．プロセッサの処理性能を向上させることで遅延時間を短縮することも可能だが，コスト増しになってしまう．そこで，$\mathrm{LiSTEE^{TM}}$ 仮想ハードウェアの時間管理モジュールがタイマー割り込みを利用して検証処理の実行時間を計測し，一定の時間が経過すると，OS 間遷移命令を呼び出して強制的にリッチ OS に処理を切り替える．このため，リッチ OS の停止時間を許容範囲に抑えることができる．なお，バイナリオブジェクトを復号する際のデータ，ハッシュ値を計算するデータ，そして平文となったバイナリオブジェクトは，リッチ OS からはアクセスできないセキュア領域に格納される．したがって，仮にリッチ OS が攻撃者によって不正に改変され，保護・更新モジュールが復号やハッシュ値の計算をしている途中でリッチ OS に遷移したとしても，リッチ OS は復号の際に計算するハッシュの中間値，PCR に格納するハッシュの中間値，平文となったバイナリオブジェクトを盗み見たり不正に書き換えたりすることはできない．また，不正に改変されたリッチ OS が共有領域にロードされた暗号化・署名付きバイナリオブジェクトを書き換えたとしても，保護・更新モジュールが平文となったバイナリオブジェクトを更新前の保護対象モジュールが配置されていた部分にコピーする処理は，署名検証が成功した場合に限り実行されるため，不正な保護対象モジュールが実行されることもなければ，不正な保護対象モジュールによって以前の保護対象モジュールが上書きされることもない．

(2)　リモート検証時の処理フロー

保護対象モジュールのリモート検証処理について**図 5** を

© 2015 Information Processing Society of Japan

表 1 メモリマップ
Table 1 Memory map.

データ項目	開始アドレス	サイズ	保護属性
ベクタテーブル＋初期化コード (code/data)	0x60000000	0x00008000	セキュア領域
リッチ OS(Linux) (code/data)	0x60008000	0x2FFF8000	ノンセキュア領域
LiSTEE^{TM} モニタ＋保護･更新モジュール＋TPM モジュール＋共通処理モジュール (code/data)	0x90000000	0x01000000	セキュア領域
再暗号化モジュール (code/data)	0x91000000	0x0E200000	
共有領域	0x9F200000	0x00E00000	共有領域

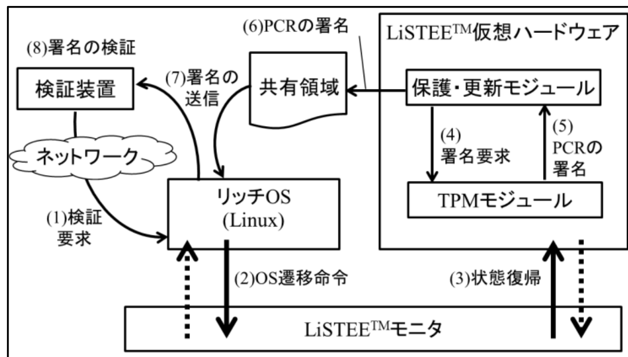

図 5 リモート検証のフロー図
Fig. 5 Execution flow of remote attestation.

用いて説明する．前提として，検証装置は TPM モジュールに格納された署名用の秘密鍵に対応する公開鍵を有しているものとする．まず，リッチ OS が検証装置からネットワーク経由でリモート検証の要求を受信する（処理 (1)）．リッチ OS は SMC 命令を発行し，LiSTEE^{TM} の OS 間遷移機能で LiSTEE^{TM} 仮想ハードウェアに遷移する（処理 (2)，(3)）．LiSTEE^{TM} 仮想ハードウェアの保護・更新モジュールは TPM モジュールに対して PCR の署名値を要求する（処理 (4)）．TPM モジュールは自身に格納されている署名用の秘密鍵を用い，現在の PCR の値に対する署名を生成し，保護・更新モジュールに送信する（処理 (5)）．保護・更新モジュールは TPM モジュールから受信した署名を共有領域に書き込み（処理 (6)），OS 間遷移機能でリッチ OS に遷移させる．リッチ OS は共有領域から PCR の署名を読み込み，ネットワーク経由で検証装置に送信する（処理 (7)）．検証装置は公開鍵を用いて署名を検証する（処理 (8)）．PCR の値は現在実行中の保護対象モジュールのハッシュ値であるから，検証者は意図した保護対象モジュールが実行されていることをリモートから任意のタイミングで検証することができる．

5. 実装

本章では LiSTEE^{TM} の実装について述べる．

LiSTEE^{TM} モニタ，LiSTEE^{TM} 仮想ハードウェアのコンパイラとして ARM C/C++ Compiler 5.01 を利用し，リッチ OS（Linux）のコンパイラとして gcc 4.4.1 を利用した．

また，LiSTEE^{TM} ではノンセキュアの OS として Linux 3.6.1 をサポートしている．実行環境として，TrustZone の機能をサポートしている CoreTile Express A9x4 プロセッサを搭載した評価ボード Motherboard Express uATX を選択した．メモリマップは，0x60000000～0xA0000000 が DRAM に割り当てられている．メインメモリのメモリマップを**表 1** に示す．

リッチ OS には，モニタモードに遷移するために SMC 命令を発行するデバイスドライバを実装した．

LiSTEE^{TM} 仮想ハードウェアの再暗号化モジュールは XOR で暗号化されたデータを復号し，平文のデータを鍵長 128 ビットの AES ECB モードで暗号化する再暗号化処理を実装した．なお，パディング処理は実装していない．TPM モジュールは鍵長 1,024 ビットの RSA 暗号を使った署名生成処理と署名検証処理を実装した．保護・更新モジュールは再暗号化モジュールを更新する際，元々再暗号化モジュールが配置されていた領域に更新対象の再暗号化モジュールのバイナリオブジェクトを上書きする処理を実装した．更新後の再暗号化モジュールが更新前と同様に実行されるようにするために，更新後の再暗号化モジュールのバイナリオブジェクトをコンパイルする際，元の暗号化モジュールと同じアドレスマップを与えてコンパイルする必要がある．また，更新される再暗号化モジュールはそのままメモリ上に配置されるイメージそのものとなっており，保護・更新モジュールはコード部分とデータ部分を区別せずに再暗号化モジュールをメモリにロードする．保護・更新モジュールは TPM モジュールに格納された復号鍵を取り出し，鍵長 128 ビットの AES CBC モードで暗号化された再暗号化モジュールを復号する処理を実装した．さらに，再暗号化モジュールに付与された署名を検証し，署名検証処理が成功した場合に限り，新しい再暗号化モジュールのバイナリオブジェクトを更新前に割り当てたメモリ領域に上書きする処理を実装した．なお，領域を動的に変更する機能は備えていないため，再暗号化モジュールの復号済みバイナリオブジェクトが，再暗号化モジュール用に割り当てられた領域サイズに収まるのであれば上書きすることができるが，あらかじめ割り当てた以上のサイズになってしまうと再暗号化モジュールを更新することはできな

© 2015 Information Processing Society of Japan

い．また，再暗号化モジュールとリッチ OS との間で共有領域を介してデータの送受信を行う際のデータサイズは共通の値となるように再暗号化モジュールの実行前にあらかじめ設定しておく．このデータサイズは共有領域のサイズを超えないように設定する必要がある．たとえば，リッチ OS から再暗号化モジュールにデータを送信する際，設定したサイズのデータ送信が完了すると，OS 間遷移機能を利用して再暗号化モジュールに処理を遷移させ，再暗号化モジュールは設定されたサイズ分だけデータを読み込む．

時間管理モジュールが実行経過時間を計測するために，タイマー割り込みは 2 つのタイマーを利用している．1 つ目のタイマーは IRQ に割り当て，Linux の割り込みハンドラが呼び出されるようにしておく．Linux はタイマー割り込みが IRQ に割り当てられることを想定しているため，Linux のソースコードを修正する必要はない．2 つ目のタイマーは FIQ に割り当て，FIQ が発生した場合に時間管理モジュールにジャンプするようベクタテーブルを設定しておく．時間管理モジュールは保護・更新モジュールが連続して一定時間以上実行されていると判断すると，共通処理モジュールを介して OS 間遷移命令を呼び出す．これにより，リッチ OS が長時間停止することを防止できる．保護・更新モジュールが保護対象モジュールを更新している最中にタイムアウトしてリッチ OS に遷移し，その後，リッチ OS から再暗号化処理を行う目的で OS 間遷移命令が呼び出されないようにする必要がある．この排他制御として，リッチ OS のデバイスドライバは対象モジュールを更新している間，リッチ OS 上で動作するアプリケーションから再暗号化処理を行う目的での OS 間遷移要求があった場合に OS 遷移命令を発行せず，当該タスクをブロック状態にし，他のタスクに切り替える処理のみを行う．また，保護・更新モジュールの更新を再開するため，一定時間後に OS 遷移を実行する．リッチ OS 側のデバイスドライバは更新完了後に，ブロック状態にしたタスクを実行可能状態にし，実行を再開する．

なお，LiSTEE™ モニタと LiSTEE™ 仮想ハードウェアは同一のバイナリオブジェクトとして構成した．また，マルチコアには対応していない．

6. 評価

本章では LiSTEE™ を実機上で試作した評価結果について述べる．設計上の項目については機能で評価し，実装上の項目についてはパフォーマンス測定を行った．そして，保護対象の処理をハードウェアで実現した場合と比較した評価結果について述べる．

6.1 評価の方針と評価環境

評価環境としては，実行環境と同様に評価ボード Motherboard Express uATX を用いた．コアクロックは 400 MHz，一次命令キャッシュは 32 KB，一次データキャッシュは 32 KB，二次キャッシュは 512 KB である．またメインメモリは 1 GB を搭載しており，共有領域，ノンセキュア領域，セキュア領域として，それぞれ 14 MB，768 MB，242 MB を割り当てた．

6.2 機能評価

(1) モジュールの安全な更新について

保護対象モジュールの更新機能とモジュール保護機能によって保護対象モジュールを安全に更新することができる．LiSTEE™ 仮想ハードウェアはセキュア状態で実行されるため，仮にリッチ OS の制御が攻撃者に奪われたとしても，署名検証処理をスキップされたり，結果を改変させられたりすることはなく，安全にアップデートすることができる．また，保護対象モジュールの更新時に保護対象モジュールに付与された署名を検証することで，不正な保護対象モジュールに更新することを防止できる．これらを検証するために，実際に 5 章で示した実装に従い，リッチ OS から保護対象モジュールが更新できるか確認した．その結果，LiSTEE™ 仮想ハードウェア内で更新した保護対象モジュールが動作することを確認した．また，リッチ OS を意図的に改変し，セキュア領域に格納された保護対象モジュールをリッチ OS から改変することを試みたが，保護対象モジュールを改変できないことを確認した．さらに，暗号化された保護対象モジュールを共有領域にロードし，保護対象モジュール部分および署名部分を共有領域のうえで改変した．そして，この改変した保護対象モジュールで更新されるかそれぞれ試みたが，いずれの場合も署名検証処理でエラーとなり，保護対象モジュールが更新されないことを確認した．

また，保護対象である再暗号化モジュールは暗号化された状態でサーバからネットワーク経由で配信することができ，アップデート時の復号処理はセキュア状態でなされ，平文となった実行モジュールはセキュア領域に配置される．このように，保護対象モジュールは，通信経路上では暗号化されており，実行時にはアクセス制御によって保護されているため，配布時と実行時の両方で改変されたリッチ OS から保護対象モジュールに含まれる秘密のデータが取得される恐れはない．さらに提案方式ではリッチ OS や，再暗号化モジュール以外の LiSTEE™ 仮想ハードウェアを更新する必要もなく，保護対象モジュールのみ更新すればよい．このため，アップデート時にシステム全体を再起動する必要もなく，ダウンタイムを最小限に抑えることができる．また，保護対象モジュールの更新に時間がかかる場合，更新処理を分割して実行することができるため，リッチ OS が長時間停止することを防止できる．

なお，5 章で示したように，更新対象である保護対象モジュールが備える暗号アルゴリズムおよび鍵長と，更新対

© 2015 Information Processing Society of Japan

象のモジュールの保護に用いる暗号アルゴリズムおよび鍵長は別々のものを用いている．特に保護対象モジュールが利用する鍵長に関しては一般的に処理性能やコストなどを考慮して，その当時で利用可能な最新の暗号アルゴリズムが採用されるとは限らない．たとえば，2000 年代後半には AES が存在していたにもかかわらず，産業用システムでは Triple DES が利用可能な暗号アルゴリズムと見なされていた [12]．そこで，保護対象モジュールの秘匿に用いる暗号は保護対象モジュールが利用するものよりも最新のアルゴリズムや長い鍵長を採用することで，ハードウェアを交換せずに長期にわたり運用することができる．他にも，正当な鍵を持つ装置のみ特定のサービスを提供可能とする仕組みも実現できる．たとえば，文献 [13], [14] ではスマートメータが収集した計量データの収集や，デマンドレスポンス処理などアプリケーションごとに通信用の鍵を別々に管理する仕組みを提供している．アプリケーションごとに別々な鍵を用意し，それぞれ異なる鍵を持つ保護対象モジュールを提供することで，装置が提供可能なアプリケーションを追加・削除できる．このようなサービスの追加と削除は，装置の寿命と比較して頻繁に行われると予想されるため，鍵を更新するたびに装置を更新しなければならない場合と比較して，コストを抑えることができる．一方で，保護対象モジュールの更新処理の頻度は更新対象モジュールの利用頻度と比較して頻繁に発生するわけではない．このため，保護対象モジュールの保護に用いる暗号アルゴリズムおよび鍵長を長く設定したとしても，全体性能に与える影響は少ない．

(2)　モジュール検証について

保護対象モジュールのリモート検証機能によってモジュールを検証することができる．トラステッドブートでは，OS を含むシステム全体の正当性を検証することが可能である．しかしながら，デバイスドライバなどセキュリティ機能には直接関係のない汎用的なモジュールを更新した場合でも検証値が変化してしまうため，モジュールを更新するたびに期待値も更新する必要があり，検証者のコストが高かった．一般的に，システム全体からすべての脆弱性を排除することは困難であり，検証者はセキュリティ上，信頼の基盤となる部分の改変が起きていないか，すなわち保護対象のモジュールのみ検証できることが理想的である．提案方式では，保護対象モジュールのみが検証対象となっているため，保護対象の処理とは関係のないリッチ OS を更新したとしても期待値を更新する必要はない．したがって，検証者の検証コストを大幅に削減することができる．また，トラステッドブートではブート時にモジュールのハッシュ値を計算するため，仮にブート後にシステムが改変されてしまうと，ブート時と改変後のシステムの状況が異なるにもかかわらず検証値と期待値が一致してしまい，システムが改変されたことを検出できないという欠点がある．一方，提案方式では，保護対象モジュールのロード時にハッシュ値を計算して実行直前の状態を検証値として利用する．このため，トラステッドブートと比較して，検証時と実行時の時間を短縮することができ，検証値と現在のシステムの状態が異なってしまうリスクを大幅に削減することができる．もちろん，トラステッドブートはシステム全体の正当性をブート時に検証する手法としては有効であるため，リッチ OS を含むシステム全体の検証にはトラステッドブートを用い，保護対象モジュールの検証には提案手法を用いるといったように相補的に利用することも可能である．

これらを検証するために，実際に 5 章で示した実装に従い，リッチ OS を介してリモートから署名を要求した．その結果，LiSTEE™ 仮想ハードウェアがリッチ OS に署名を返し，その署名の値を TPM モジュール内の秘密鍵に対する公開鍵と保護対象モジュールのハッシュ値で検証し，成功することを確認した．また，保護対象モジュールを更新したが，署名の値が更新した保護対象モジュールに対応したハッシュ値であることも確認した．さらに，リッチ OS を更新したが，LiSTEE™ 仮想ハードウェアがリッチ OS に返す保護対象モジュールのハッシュ値には変化がなかった．これにより，検証対象以外のモジュールを更新した場合でも検証値が変化しないことを確認した．なお，提案方式ではリッチ OS が署名に用いる秘密鍵を有していないため，正当な署名を計算することはできないが，署名済みハッシュ値を検証者に送信せずに破棄したり，意味のない値に書き換えて送信したりすることが可能である．この場合，LiSTEE™ 仮想ハードウェアが正しい署名済みハッシュ値を共有領域に書いたとしても管理者は署名済みハッシュ値の検証に失敗してしまうが，そのようなサービス妨害攻撃は今後の課題とする．なお，管理者はモジュールの検証に失敗したことで，組込み制御装置が正当な状態ではないと判断することができるため，サービス妨害攻撃を受けたとしても検証自体の目的は果たしている．同様に，リッチ OS が暴走したり改変されたりして，LiSTEE™ 仮想ハードウェアを無意味に何度も呼び出すことにより，その実行を妨害する攻撃も考えられるが，そのようなサービス妨害攻撃も今後の課題とする．

(3)　モジュールのデータ保護について

保護対象モジュールの実行機能によって保護対象モジュールのデータを保護することができる．リッチ OS がアクセス可能な共有領域には暗号化されたデータと暗号化された再暗号化モジュールしか配置されない．LiSTEE™ 仮想ハードウェアの実行イメージと，中間データを格納しておくためのスタックおよびヒープ領域はセキュア領域に確保されている．したがって，リッチ OS が改変されて攻撃者の制御下にあったとしても，リッチ OS がこれらセキュア領域に置かれたデータにアクセスすることはできず，暗号

© 2015 Information Processing Society of Japan

処理を行う過程で生成する中間データを含め LiSTEE^{TM} 仮想ハードウェアが扱うデータが，リッチ OS に取得されたり改変されたりする恐れはない．なお，提案方式ではリッチ OS を攻撃することによりシステムを停止させたり，再暗号化したデータを無効な値に置き換えたり，再暗号化したデータの送信を遮断したりすることが可能だが，そのようなサービス妨害攻撃は今後の課題とする．

(4) モジュールのサイズについて

一般的に，ソフトウェアの規模が増大するにつれ，実装上の不備などの要因により脆弱性が含まれるリスクも高くなる．システム全体の規模を最小化し，綿密にレビューすることが理想的ではあるが，OS などの汎用的な機能はオープンソースなど外部のソースコードを活用することも考えられるため，開発者自身の努力によってバグを削減することが困難な場合が多い．そこで，綿密に検証すべき部分のソースコードのサイズを最小化して，その部分に限定してレビューを行えるような構成になっていることが求められるが，LiSTEE^{TM} 仮想ハードウェアのサイズはシステム全体に比べて十分に小さい．前述のように保護対象モジュール，LiSTEE^{TM} 仮想ハードウェア，LiSTEE^{TM} モニタが TCB に相当する．LiSTEE^{TM} 仮想ハードウェアと再暗号化モジュールのソースコード量は 6,300 行程度であり，コードとデータのサイズはそれぞれ 12 KB，約 5 KB である．このうち，再暗号化モジュールのソースコードは 1,200 行で，コードとデータのサイズはそれぞれ 2 KB，200 B である．また，LiSTEE^{TM} モニタのソースコード量は 900 行程度であり，コードとデータのサイズはそれぞれ 5 KB，19 KB である．一方，Linux 3.6.1 のソースコードは 1,500 万行以上であるため，TCB のサイズは相対的に小さく，コードレビューによってバグを除去したモジュールを構築することが現実的に可能なサイズであるといえる．

6.3 性能評価

暗号処理をセキュアな環境で実行する際のオーバヘッドを最小化する必要がある．仮に性能低下が発生してしまうと，プロセッサの処理性能を向上させる必要があり，結果的にコスト増になってしまう．保護対象モジュールをセキュアな環境ではない一般的な環境で実行した場合と比較して，大きな性能低下がないことが求められる．そこで実行性能が低下しないことを確認するために，再暗号化処理の実行性能を測定した．また，リッチ OS の停止時間も短いことが理想である．そこでリッチ OS の停止時間が許容範囲であることを満たしているか確認するためにリッチ OS 側で ping プログラムを用いてネットワーク通信のレイテンシ低下を測定した．それぞれの性能評価結果について以下で述べる．

(A) 再暗号化処理の実行性能評価

再暗号化処理の実行性能を評価するために，リッチ OS のみで再暗号化処理を行った場合と，リッチ OS と LiSTEE^{TM} 仮想ハードウェアが連携して，LiSTEE^{TM} 仮想ハードウェア上で再暗号化処理を行った場合について性能計測を行った．本評価では 10 [MB] のランダムなデータをあらかじめメモリ上に配置し，16 [B]，32 [B]，64 [B]，256 [B]，1 [KB]，4 [KB]，16 [KB] ごとのブロックに分割して再暗号化処理を行った場合について，それぞれ実行性能（スループット）を計測した．リッチ OS と LiSTEE^{TM} 仮想ハードウェアが連携して処理を行う場合には，たとえばブロックの大きさを 16 [B]，合計データサイズを 10 [MB] とした場合には，合計 $2 \times 10 \times 1{,}024^2/16 = 1{,}310{,}720$ [回] の OS 遷移が発生することとなる．ここで，$10 \times 1{,}024^2/16$ を 2 倍している理由は，一往復でリッチ OS から LiSTEE^{TM} 仮想ハードウェアに遷移し，またリッチ OS に戻ってくるため，2 回の OS 遷移が発生するためである．実行性能の評価結果を**図 6** に，リッチ OS と LiSTEE^{TM} 仮想ハードウェアが連携した場合と，リッチ OS のみで処理を行った場合の性能比を**図 7** に示す．

評価よりブロックサイズが小さい場合には OS 間遷移のオーバヘッドが大きく現れ，リッチ OS のみで処理を行う場合に比べてブロックサイズが 16 [B] では 62%程度性能が低下する．一方，ブロックサイズが 256 [B] では性能低

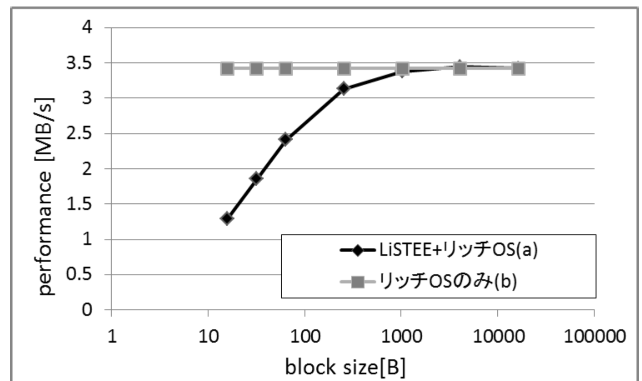

図 6 再暗号化の実行性能

Fig. 6 Throughput of re-encryption process.

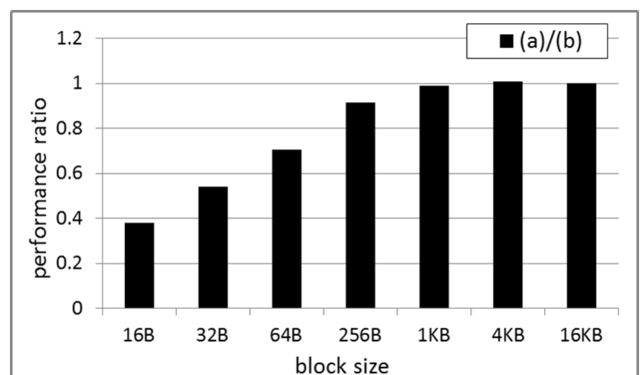

図 7 再暗号化の実行性能比（リッチ OS のみ = 1.0）

Fig. 7 Performance ratio of re-encryption process (Rich OS only = 1.0).

© 2015 Information Processing Society of Japan

下が8.5%，1 [KB] では1.2%，16 [KB] では0.1%とリッチOSのみで処理を行う場合に比べて，ほぼ差異がなくなる．暗号アルゴリズムの計算量にもよるが，適切なブロックサイズを選択することで，オーバヘッドは許容範囲内に収まると考えられる．

一方，ブロックサイズを大きくするとリッチOSとLiSTEE^{TM}仮想ハードウェアの間でデータ受け渡しを行うためのメモリ共有領域を大きくする必要があることや，次に評価するリッチOSの停止時間が長くなることから，一概にブロックサイズを大きくすればよいわけではない．暗号アルゴリズムによって，性能低下率やメモリ使用量，リッチOSの停止時間に鑑みて適切なブロックサイズを選択する必要があるといえる．

なお，TrustZoneではセキュア状態とノンセキュア状態で実行性能に差はないことから，セキュア状態で再暗号化処理を行った場合の性能比はリッチOSのみで処理を行った場合と変わらない．

(B) ネットワークレイテンシの評価

リッチOSが長時間停止すると，ネットワークアクセスやGUI操作に対する反応などの応答速度が遅くなることが予想される．リッチOSの停止時間がリッチOS上で動作するアプリケーションの応答速度に与える影響を評価するため，ネットワークアクセスの応答速度を計測するpingプログラムを用いてネットワークアクセスのレイテンシ（RTT）を計測した．アクセス先はローカルネットワーク内のサーバとし，1 [Gbps] で有線接続されている．評価はリッチOSがアイドル状態（idle），リッチOSのみで再暗号化処理を行った場合（リッチOSのみ），16 [B]，32 [B]，64 [B]，256 [B]，1 [KB]，4 [KB]，16 [KB] ごとのブロックに分割してLiSTEE^{TM}仮想ハードウェアと連携して再暗号化処理を行った場合について行った．評価結果を図**8**に示す．

評価結果より，アイドル状態とリッチOSのみで処理を行った場合のRTTは0.43 [ms] と0.44 [ms] でほとんど違いはない．一方，LiSTEE^{TM}仮想ハードウェアとリッチOSで連携して再暗号化処理を行った場合にはブロックサイズが16 B で0.45 [ms]，256 B で0.48 [ms] まではリッチOSのみで処理を行った場合と大きな差はない．しかし，1 KB で0.75 [ms]，4 KB で1.3 [ms] と，ブロックサイズが大きくなるにつれてリッチOSの停止時間が長くなるため，RTTも大きくなる．ブロックサイズが1 [KB] の場合，1ブロックあたりの処理時間，すなわちリッチOSの停止時間は，1 [MB] あたりの遷移回数が1,000 [回] であることと図7より性能が3.38 [MB/s] であることから，(1,000/3.38)/1,000 = 約0.30 [ms] であり，この値はリッチOSのみで処理を行った場合とのRTTの差である0.31 [ms] とほぼ一致する．

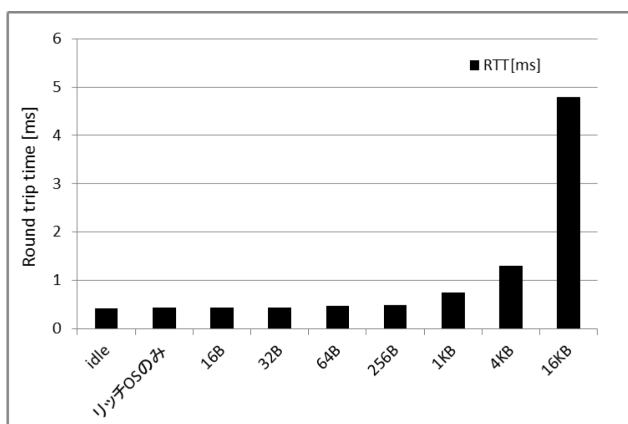

図 8 ネットワークレイテンシ評価結果
Fig. 8 Evaluation result of network latency.

再暗号化処理の実行性能評価でも述べたように，ブロックサイズを大きくするとスループットは高くなるが，リッチOSの停止時間は長くなるトレードオフの関係にある．今回のXORからAESへの再暗号化処理の場合，リッチOSのみで処理を行った場合に比べ，ブロックサイズが256 [B] でスループットは8%低下，RTTは0.04 [ms] 増大，1 [KB] でスループットは1.2%低下，RTTは0.32 [ms] 増大する．どちらもセキュリティが向上することを考えると，スループットの低下率，RTTの増加率ともに十分小さいといえる．応答速度の制約や性能低下の許容範囲，さらにメモリ使用量の制約などからブロックサイズを決定することになるが，たとえば保護対象モジュールの実行性能を優先する場合にはブロックサイズ1 [KB] を，リッチOSの実行性能を優先する場合にはブロックサイズ256 [B] を選択すればよい．

また，提案方式ではOS間遷移に要する時間は往復で1.66 us であり，サイクル数に換算すると664サイクルとなる．提案方式以外にもソフトウェアのみで実現する方法として，暗号計算などの機密データを扱うプロセスを機密情報にアクセスしないプロセスに分離することによって機密データを保護するアプローチが考えられる．このようなリッチOSのみで処理を行う構成ではデータの交換時にプロセスのコンテキストスイッチが発生するが，Linuxではコンテキストスイッチに1,000サイクル弱のサイクル数を要する [15] ため，提案方式ではこれよりも低いコストでOS間遷移を行うことができる．また，要件の一例として文献 [11] では許容遅延が50～300ミリ秒と示されている．仮にデータサイズが1 KB で，128ビット単位で処理したとしても，図7で示した実行性能評価から，1.29 MBps/1,024 = 約1.26ミリ秒で再暗号化処理が完了し，50～300ミリ秒の許容遅延に比べ十分小さい．したがって，OS間遷移に要する時間と再暗号化処理に要する時間を考慮しても十分応答できる．このように提案方式では安全性の向上に加え，暗号計算以外の処理が停止する時間も短く抑えることができるため，リッチOSが停止する時間は許容範囲といえる．

© 2015 Information Processing Society of Japan

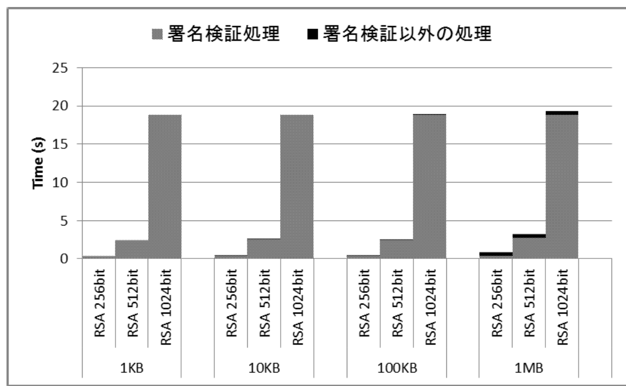

図 9 モジュール更新の処理時間
Fig. 9 Performance result of module update process.

(C) モジュール更新の処理時間

モジュール更新の処理時間を評価した．本評価では，サイズが 1 [KB]，10 [KB]，100 [KB]，1 [MB] の保護対象モジュールのバイナリオブジェクトをディスクからロードし，セキュア領域の再暗号化モジュール用に割り当てられたメモリ領域にコピーするまでの処理時間を計測した．評価結果を図 **9** に示す．

評価結果より，鍵長が 256 ビットでバイナリオブジェクトのサイズが 1 [KB] と 10 [KB]，100 [KB]，1 [MB] の場合，それぞれ 0.35 [s]，0.36 [s]，0.4 [s]，0.9 [s] となり，バイナリオブジェクトのサイズが大きくなるとモジュール更新処理に要する時間も長くなる．また，鍵長を変えて処理時間を計測した．バイナリオブジェクトのサイズを 100 [KB] とし，鍵長を 256 ビット，512 ビット，1,024 ビットとした場合，それぞれ 0.4 [s]，2.51 [s]，18.89 [s] と処理時間が長くなる．次に処理時間の要素を分析した．処理時間はハッシュの計算，復号処理，RSA の署名検証処理，メモリのコピーから構成されるが，このうち RSA の署名検証処理のみを計測した．鍵長が 1,024 ビットでバイナリオブジェクトのサイズが 100 [KB] の場合，全体の処理時間 18.89 [s] のうち，18.84 [s] を署名検証処理が占める．同様にバイナリオブジェクトのサイズを 1 [MB] とした場合，全体の処理時間 19.4 [s] のうち，18.9 [s] を署名検証処理が占める．これにより，処理時間の大半を RSA の署名検証処理が占め，ハッシュの計算，復号処理，メモリのコピーに要する時間は相対的に短いことを確認した．このように，モジュール更新処理には時間がかかるが，前述のとおりモジュール更新の実行頻度は低く，仮に処理時間のかかる公開鍵暗号を用いたとしても，時間管理モジュールの機能によって，一定の時間が経過すると検証処理を中断して強制的にリッチ OS に処理を切り替えることができるため，リッチ OS の停止時間を許容範囲に抑えることができる．なお，図 9 で示した評価結果はタイマーによる定期的なリッチ OS への遷移を行っていない．そこで，時間管理モジュールのタイマーを設定し，定期的にリッチ OS に遷移させる場合の処理時間を計測した．その結果，鍵長が 1,024 ビットでバイナリオブジェクトのサイズが 1 [MB] の場合，タイマーなしで 19.4 [s] のところ，1 [ms] ごとに遷移させた場合には 19.43 [s] となり，100 [us] ごとに遷移させた場合は 19.7 [s] となった．これにより，仮にタイマーによる定期的なリッチ OS への遷移を行ったとしても，OS 遷移に要する時間は短いため署名検証処理に与える影響は少ないといえる．

6.4 ハードウェアで実現した場合の比較評価

保護対象の処理をハードウェアで実現した場合の比較評価として，速度性能，機能，コストの観点から評価した．

(1) 速度性能評価

鍵長 128 ビットの AES 暗号をアクセラレーターの目的でハードウェアとして実装した場合，実装方法にも依存するがハードウェアモジュール単体で数百 Mbps～数 Gbps のスループットが実現でき，提案方式と比較して 100～1,000 倍以上，高速に動作させることができる [16], [17]．しかしながら，スマートメータのような応用を想定した場合，暗号化対象のデータ量は少ないため，ハードウェアアクセラレーターと比較して実行速度が遅い点は問題ない．

(2) 機能評価

ハードウェアと同等に実現可能な機能

LiSTEE^{TM} 仮想ハードウェアの再暗号化モジュールは入力された暗号データを復号し，別の鍵と暗号アルゴリズムで暗号化するという意味においてハードウェアとして実現されている再暗号化ハードウェアと機能的に等価である．LiSTEE^{TM} 仮想ハードウェアの TPM モジュールに含まれる署名生成処理はハードウェアとして実現されている TPM と機能的に等価である．ただし，4.2 節で述べたように，PCR については保護モジュールのロード時に値がリセットできるように機能を変更している．

一般的に，ソフトウェアがハードウェアの処理内容を置きかえることは不可能である．提案方式でも同様に，アクセス制御ポリシ設定機能により保護対象処理を利用するリッチ OS から LiSTEE^{TM} 仮想ハードウェアのメモリ領域へのライトは禁止されるため，保護対象の処理が改変されることはない．また，ハードウェアによる実装では，保護対象処理の実行中に鍵などの秘匿データの漏えいを防止するために秘匿データをハードウェア内に隠ぺいすることが一般的である．提案方式では，セキュア領域として設定された LiSTEE^{TM} 仮想ハードウェア内に秘匿データを含めるため，ノンセキュア状態で実行されるリッチ OS に秘匿データが漏えいすることはない．また，多くの汎用 SoC ではファームウェアを暗号化してフラッシュメモリに保存する機能が備わっており，その機能を利用して LiSTEE^{TM} 仮想ハードウェアと LiSTEE^{TM} モニタを保護して格納することができる．このようにハードウェアが本質的に備えている保護機能は LiSTEE^{TM} 仮想ハードウェアでも同様

© 2015 Information Processing Society of Japan

に実現できている．

さらに，保護対象の処理を利用するアプリケーションプログラムはデバイスドライバなどのインタフェースを介してハードウェアの機能を利用する構成になっていることが一般的である．提案方式でもデバイスドライバを介してLiSTEE^{TM} 仮想ハードウェアを呼び出す構成をとっているため，LiSTEE^{TM} 仮想ハードウェアを利用するアプリケーション開発者は従来のハードウェア機能と同様のインタフェースで呼び出すことができる．

ハードウェアでのみ実現可能な機能

一般的なハードウェア実装に対して追加コストはかかるものの，ハードウェアでしか実現できない機能がある．本論文ではデータの再暗号化処理を保護対象の処理としているため乱数を生成する必要はないが，他の利用シーンでは精度の高い乱数を必要とされる場合が考えられる．ハードウェアで実現する場合，ハードウェアモジュール内で生じる物理現象を利用して真正乱数を生成することが可能である．

また，TPMのように暗号機能を備えたセキュリティハードウェアの中にはハードウェアレベルの耐タンパー機能を備えているものもあり，電子プローブなどを利用した物理解析や，電力消費量などを計測して鍵を推測するサイドチャネル攻撃などの専門的なスキルと専用ツールを備えた攻撃者による高度な実装攻撃を防ぐことが可能である．

提案方式でのみ実現可能な機能

ハードウェアで実現する場合，暗号処理に用いる鍵をハードウェアモジュールの中に含めてしまうと，鍵の値を更新することはできなくなる．多くの暗号ハードウェアモジュールは鍵をソフトウェアから設定することで，その値を更新することが可能だが，この場合，ソフトウェアで平文の鍵を扱う必要があるため，鍵が漏えいするリスクが高い．一方，提案方式では鍵を含めた保護対象モジュールを暗号化してサーバなどから配布することが可能であるため，安全に鍵を更新することが可能である．同様に提案方式ではアルゴリズムを置換することも可能である．さらに保護対象モジュールはAESなどの暗号アルゴリズムだけでなく，一連の処理を含めることができるため，処理内容を置換したり，機能を追加したりすることも可能である．たとえば，スマートメータの鍵をグループで効率的に管理したり，特定のスマートメータをリボークしたりするために，MKB（Media Key Block）を用いたブロードキャスト暗号が提案されている [18]．MKBでは，各装置がMKBに含まれる鍵を導出するために複雑な処理を実行する必要があるが，提案方式を利用すればMKBの高速処理アルゴリズムなどのノウハウを隠ぺいしつつ，鍵導出処理の一部分のみ更新することが可能である．

(3) コスト評価

コストは製造時と運用時に分類できる．

製造時のコストとは，1台あたりの装置の製造コストである．提案方式では，ARMプロセッサ以外の追加ハードウェアを必要としないため追加製造コストはゼロである．一方，暗号アルゴリズムをハードウェアとして実装する場合，その分の追加製造コストが1台あたりに必要となる．台数が少ない分には総コストは少ないものの，スマートメータのように何千万台も設置する場合，総コストは莫大なものとなる．

運用時のコストとは，鍵やアルゴリズムをアップデートする際に必要となるコストである．提案方式では，ネットワーク経由で保護対象モジュールをアップデートすることを想定しているため，モジュールを配布するサーバのメンテナンスコストが必要となる．一方，暗号アルゴリズムをハードウェアで実現する場合，サービスマンが物理的にハードウェアを交換する必要がある．製造時のコストと同様，台数が少ない分にはそのコストは小さいが，台数に応じて交換コストも増えてしまう．さらに，交換の原因がセキュリティ上の理由の場合，装置を交換するまでの間，その装置は攻撃を受けるリスクがあるため早期に交換することが求められるが，何千万台もの装置を短期間でサービスマンが交換することは現実的ではない．

このように，提案方式ではハードウェアで実現する場合と比較して，製造時と運用時の両方でコストを削減することができる．

7. 関連研究

汎用PCの世界では，仮想化技術を用いて2つのOSを並列に動作させる研究が数多くなされており，その一部は実用化され広く普及している [19], [20]．しかし，現在市場に広く流通している組込み向けプロセッサでは，ハードウェアによる仮想化支援機能がサポートされていない．ソフトウェアのみで仮想マシンを実現することも考えられるが，メモリ管理などの処理をすべてソフトウェアで実装する必要があり，実行性能上課題がある．一方，組込み向けシステムで2つのOSを同時実行させる方式としてSafeGが提案されている [21], [22]．SafeGはTrustZoneの機能を利用して，制御系OSと情報系OSを分離し，制御系OSを保護する手法を提案しているが，制御系OS側のモジュールを動的かつ安全に更新することは想定しておらず，また連続的にOS間遷移を実行させ，リッチOSと制御系OS間で大量のデータ交換を行う方法については明らかにされていない．本研究の提案方式では，システムを再起動させることなくセキュア状態で実行するモジュールを安全に更新することができる．さらに，本研究ではノンセキュア状態とセキュア状態で大量のデータを交換する際のオーバヘッドを測定し，ノンセキュア状態のOSが長時間停止することを防止できることを明らかにした．また，SafeGの機能を利用し，制御系OSとリッチOSとの間で安全かつ効率

© 2015 Information Processing Society of Japan

的にデータ交換を行う仕組みが提案されている [23]．しかしながら，本研究では共有領域に保護対象モジュールと暗号化されたデータしか置かれない．保護対象モジュールがリッチ OS によって改変された場合には，モジュール更新機能によって改変されたか否かを検出することができる．暗号化されたデータが改変された場合，モジュール更新機能はこれを検出することはできないが，MAC（Message Authentication Code）を付与するなどの既存手法によって改ざんを検出することができる．一方，保護対象のデータが平文で共有領域に配置される場合や，高機能・効率的なデータの交換を行うには，文献 [23] で示されているような保護された制御系 OS でデータをフィルタして効率的にデータを交換したり，制御系 OS のリアルタイム性を確保したりする方式を組み合わせて，より効果的なシステムを構築することが可能である．

また，保護対象のモジュールを検証する仕組みとして，汎用 PC 向けのプロセッサと TPM を使い，保護された実行環境で保護対象モジュールを実行させるアーキテクチャが提案されている [24]．しかしながら，汎用 PC 向けのプロセッサはセキュア環境をブート時に利用することを想定しており，システム実行中にセキュア環境とノンセキュア環境を切り替えることは想定しておらず，実験では切替え時間に 10 ミリ秒以上必要であることが明らかとなっている．この間，ノンセキュア状態の OS は処理を中断させる必要があるため，制御システム向け組込み装置に用いることは現実的ではない．

さらに，TPM の機能をソフトウェアで置き換える研究もなされている [25], [26]．これらの研究ではハードウェアと比較したパフォーマンス評価は行われているものの，テストやデバッグを目的としていたり，クラウド上の仮想サービスとして利用することを目的としたりしているため，不正な解析や改変を防止するための手法については触れられていない．

8. まとめと今後の課題

本論文では，組込み向けプロセッサのセキュリティ機能を活用し，長期安全性の確保が可能なソフトウェアシステムを提案した．提案方式では，暗号処理を汎用処理と分離してセキュア環境で実行させることにより，保護対象の処理が解析耐性を備えつつ更新可能なソフトウェアモジュールとして実現可能であることを示した．

さらに，提案方式を実装し，汎用処理部分からハードウェアを利用する従来方式と同様の仕組みで，セキュア環境で実行する暗号処理が利用可能であることを示すとともに，オーバヘッドが十分に許容範囲であるとの評価結果を得た．

今後の課題としては，マルチコア対応による実行効率の改善や，再暗号化処理以外に適用していくことが考えられる．

参考文献

[1] National Institute of Standard and Technology: NIST IR 7628 Guidelines for Smart Grid Cyber Security (2010).

[2] National Electrical Manufacturers Association: Requirements for Smart Meter Upgradeability (2009).

[3] Anderson, R. and Fuloria, S.: Smart meter security: A survey (online), available from ⟨http://www.cl.cam.ac.uk/~rja14/Papers/JSAC-draft.pdf⟩ (accessed 2015-05-11).

[4] 金井 遵，磯崎 宏：セキュアプラットホームソフトウェア LiSTEE^{TM}，東芝レビュー，Vol.69, No.1, pp.27–30 (2014).

[5] 金井 遵，磯崎 宏：高速な OS 切り替え機構を有する組み込み向けセキュアモニタ LiSTEE，並列/分散/協調処理に関するサマー・ワークショップ，2013-OS-126, No.19, pp.1–8 (2013).

[6] Sailer, R., Zhang, X., Jaeger, T. and van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture, *Proc. 13th Conference on USENIX Security Symposium*, pp.223–238 (2004).

[7] ARM Security Technology (online), available from ⟨http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf⟩ (accessed 2015-05-11).

[8] Alves, T. and Felton, D.: TrustZone: Integrated Hardware and Software Security, *Information Quarterly*, Vol.3, No.4 (2004).

[9] Trusted Computing Group: TPM Main Specification Part 1 Design Principles, Revision 116 (2011).

[10] Kleidermacher, D. and Kleidermacher, M.: *Embedded Systems Security*, Newnes (2012).

[11] 大谷哲夫：自動検針用国際標準通信プロトコルの基本特性と次世代グリッドへの適用可能性評価，電力中央研究所研究報告 R09009 (2010).

[12] Dzung, D., Naedele, M., von Hoff, T.P. and Crevatin, M.: Security for Industrial Communication Systems, *Proc. IEEE*, Vol.93, No.6, pp.1152–1177 (2005).

[13] 神田 充，大場義洋，田中康之：相互認証と暗号化処理を統合するスマートメータ用統合鍵管理技術 AMSO^{TM}，東芝レビュー，Vol.65, No.9, pp.23–27 (2010).

[14] Forsberg, D., Ohba, Y., Patil, B., Tschofenig, H. and Yegin, A.: Protocol for Carrying Authentication for Network Access (PANA), IETF RFC 5191 (online), available from ⟨https://tools.ietf.org/html/rfc5191⟩ (accessed 2015-05-11).

[15] 金井 遵，佐々木広，近藤正章，中村 宏，並木美太郎：統計情報に基づく省電力 Linux スケジューラ，情報処理学会「システムソフトウェアとオペレーティング・システム」第 106 回研究報告，2007 年並列/分散/協調処理に関する『旭川』サマー・ワークショップ，Vol.2007-OS-106, pp.9–16 (2007).

[16] Ghewari, P.B., Pati, J.K. and Chougule, A.B.: Efficient Hardware Design and Implementation of AES Cryptosystem, *International Journal of Engineering Science and Technology*, Vol.2, No.3, pp.213–219 (2010).

[17] Gielata, A., Russek, P. and Wiatr, K.: AES hardware implementation in FPGA for algorithm acceleration purpose, *International Conference on Signals and Electronic Systems* (*ICSES '08*), pp.137–140 (2008).

[18] Zhao, F., Hanatani, Y., Komano, Y., Smyth, B., Ito, S. and Kamibayashi, T.: Secure Authenticated Key Exchange with Revocation for Smart Grid, *The 3rd IEEE PES Conference on Innovative Smart Grid Technologies* (*ISGT 2012*), IEEE Power & Energy Society (PES), pp.1–8 (2012).

© 2015 Information Processing Society of Japan

[19] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *19th ACM Symposium on Operating Systems Principles*, pp.164–177 (2003).
[20] Waldspurger, C.A.: Memory resource management in VMware ESX server, *5th Symposium on Operating Systems Design and Implementation*, pp.181–194 (2002).
[21] Sangorrin, D., Honda, S. and Takada, H.: Dual Operating System Architecture for Real-Time Embedded Systems, *Proc. 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications* (*OSPERT2010*), pp.6–15 (2010).
[22] 中嶋健一郎，本田晋也，手嶋茂晴，高田広章：セキュリティ支援ハードウェアによるハイブリッドOSシステムの高信頼化，電子情報通信学会論文誌D，Vol.J93-D, No.2, pp.75–85 (2010).
[23] Sangorrin, D., Honda, S. and Takada, H.: Reliable and Efficient Dual-OS Communications for Real-Time Embedded Virtualization, コンピュータソフトウェア, Vol.29, No.4, pp.182–198 (2012).
[24] McCune, J.M., Parno, B., Perrig, A., Reiter, M.K. and Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization, *ACM SIGOPS Operating Systems Review*, Vol.42, No.4, pp.315–328 (2008).
[25] Liu, D., Lee, J., Jang, J., Nepal, S. and Zic, J.: A New Cloud Architecture of Virtual Trusted Platform Modules, *IEICE Trans. Information & Systems*, Vol.E95-D, No.6, pp.1577–1589 (2012).
[26] Strasser, M. and Stamer, H.: A Software-Based Trusted Platform Module Emulator, *Trust '08 Proc. 1st International Conference on Trusted Computing and Trust in Information Technologies: Trusted Computing – Challenges and Applications*, pp.33–47 (2008).

磯崎 宏

2001年慶應義塾大学大学院政策・メディア研究科修了．同年（株）東芝入社．2008年カーネギーメロン大学電気&コンピュータ工学部修士課程修了．ホームネットワーク，セキュリティの研究開発に従事．

金井 遵 （正会員）

2006年東京農工大学工学部情報コミュニケーション工学科卒業．2008年日本学術振興会特別研究員．2009年東京農工大学大学院工学府電子情報工学専攻博士後期課程修了．同年（株）東芝入社．博士（工学）．オペレーティングシステム等のシステムソフトウェア，セキュリティ，ネットワークの研究開発に従事．電子情報通信学会会員．

© 2015 Information Processing Society of Japan