Computer Security Symposium 2017
23 - 25 October 2017

Trusted Execution Environment 搭載デバイスのための セキュア空間へのアクセス権限設定法の基礎検討

濱本 亮[†1]　佐々木 貴之[†1]　森田 佑亮[†1]　三好 一徳[†1]　小林 俊輝[†1]

概要: 米国の VISA International が中心となって設立された標準化団体である GlobalPlatform によって Trusted Execution Environment (TEE) と呼ばれるハードウェアレベルのセキュリティ機能が定義されており，組み込みデバイス向けの CPU を展開している ARM では TEE の実装例として TrustZone と呼ばれる機能が公開されている．しかし，TEE によって構成されたセキュアなメモリ空間(セキュア空間)はアクセス権限が 1 種類しか設定できず，アーキテクチャ設計の柔軟に欠ける．そこで本稿では，TEE のセキュア空間が持つ権限を多段階で設定できるアクセス制御の基本設計を提案する．

キーワード：IoT, Trusted Execution Environment，権限設定，アクセス制御

A Study on Access Control for Devices with Trusted Execution Environments

Ryo Hamamoto[†1]　Takayuki Sasaki[†1]　Yusuke Morita[†1]　Kazunori Miyoshi[†1]
Toshiki Kobayashi[†1]

Abstract: Trusted Execution Environment (TEE), one of the famous security technologies provided by hardware, is defined by GlobalPlatform. One of the implementations of TEEs for embedded devices, TrustZone, is proposed by ARM. However, secure memory spaces (secure world) which consist of TEE have only one access level. Therefore, it lacks flexibility for designing of device architecture. In order to solve this issue, this paper proposes a multi-level access control method for TEE's secure world.

Keywords: IoT, Trusted execution environment, Access control

1. はじめに

スマートシティ[1]などをはじめとした Internet of Things (IoT) システム[2]の普及に伴って，センサデバイスや家電，車やヘルスケア製品などネットワークに接続される組み込み機器 (IoT デバイス) が増加している．しかしその一方で，デバイスに対する攻撃領域 (Attack Surface) が広がり，デバイス情報の改ざんなどの不正行為が発生しやすくなる．そのためデバイスに対するセキュリティ脅威への対策がますます重要となっている．これまではセキュリティ脅威への対策としてウィルス対策ソフトなどのソフトウェアを利用した対策が広く用いられてきた．一方で最近ではソフトウェア的な対策に加えてハードウェアを用いたセキュリティ対策 (セキュアハードウェア) が検討されつつある．代表的なセキュアハードウェアとして米国の VISA International が中心となって設立された標準化団体である GlobalPlatform によって定義された Trusted Execution Environment (TEE) [3] が知られている．TEE はデバイスのメモリ空間上にユーザですら直接アクセスできないメモリ空間 (セキュア空間) を構築して，セキュア空間内に機密情報やセキュリティ処理を実装する．これによってマルウェアなどを用いた機密情報の流出やデバイスが行う処理の改ざんを防止することが可能となる．TEE の 1 つである Intel SGX では，セキュア空間内に実装された処理に脆弱性がある場合や，初めから悪意あるプロセスが搭載された場合にその動作によってデバイスの安全性が失われることを防ぐため，TEE は最低限の権限しか持たない．つまり安全性に重きを置く設計思想である．しかし TEE を考慮したデバイスの設計を考えた場合に，権限が最低限では TEE に実装できるデバイスの機能が限定的となってしまう (デバイス設計に対する自由度の低下)．例えば，デバイスを安全に制御するためには TEE と入出力ポート間でやり取りされる命令やデータをセキュアに伝達するチャネルが求められるが，最低限の権限ではデバイスにアクセスできず，セキュアなチャネルを構築することは難しい．一方，TEE に特権を与えると，TEE が汚染された場合にデバイスの安全性が担保できない．従って，デバイスの安全性と設計の自由度との間に発生するトレードオフ問題がある．この課題を解決するために，本稿では TEE のセキュア空間に対してその権限を段階的かつ個別に設定する手法を提案する．また，

†1 日本電気株式会社 セキュリティ研究所
Security Research Laboratories NEC Corporation

© 2017 Information Processing Society of Japan

提案手法によってデバイスの安全性と設計の自由度に関するトレードオフ問題がある程度解決できることを説明する．

以後第 2 節ではセキュアハードウェアの概要とその一種である TEE の概要及びその課題を説明する．第 3 節では提案手法を説明し，第 4 節にて提案手法を実装したデバイスのアーキテクチャの紹介と評価を行う．第 5 節では関連研究を紹介し，最後に第 6 節にて本稿のまとめと今後の課題について述べる．

2. セキュアハードウェア

本節ではセキュアハードウェアの概要とセキュアハードウェアの一例である TEE を説明し, TEE を搭載したデバイスを設計する際の課題について述べる．

2.1 セキュアハードウェアの概要

改ざんや情報流出など，様々な脅威がデバイスの持つ情報に対して存在している．これらの脅威が発生する原因の一例として，デバイスがマルウェアに感染することによって，ファイルやメモリ (メモリ上の OS やプロセスを含む) が書き換えられるケースがある．この対策として，従来はウィルス対策ソフトなどをはじめとしたソフトウェアレベルでのセキュリティ対策が主流であった．しかしながら，ソフトウェアレベルでのセキュリティ対策には，ソフトウェアが有するバグに対して脆弱であるという課題がある．例えば，ソフトウェア実装時に含まれたバグによる暗号鍵の流出[4]やバグを利用したファームウェアの書き換え[5]などが攻撃例として知られている．

ここで，IoT デバイスが攻撃対象となる場合の影響を考える．IoT デバイスには車やロボットなど，現実空間へ影響を与えるモノが数多く含まれる．そのため，ソフトウェアのバグが人々の生活へ影響を及ぼすことはもちろんのこと，人間の身体や生命に直接危害を与えるインシデントへと発展することが予想される．

この問題への対策として，ハードウェアレベルでセキュリティ対策を実装することでソフトウェアレベルの脅威を防ぐことが提案されている．セキュリティ機能を実装した，もしくはハードウェアレベルでのセキュリティ機能拡張を行ったハードウェアはセキュアハードウェア[6]と呼ばれており，現在盛んに議論されている．なお，上記の実現のためには専用のハードウェアを用いる例や，専用のハードウェアを利用せず，通常のプロセッサを拡張して柔軟性を確保する方式がある．セキュアハードウェアの登場で，ファイルやそのメタデータのハードウェアを用いた完全性計測や保護機能などが実現できるようになった．セキュアハードウェアの代表例として，Trusted Platform Module (TPM) [7]や Secure Element (SE) [8]，および Trusted Execution Environment (TEE) [3]がある．本稿では，現在様々な企業にてソリューションやユースケースが議論されている TEE に着目し，次節にてその詳細を述べる．

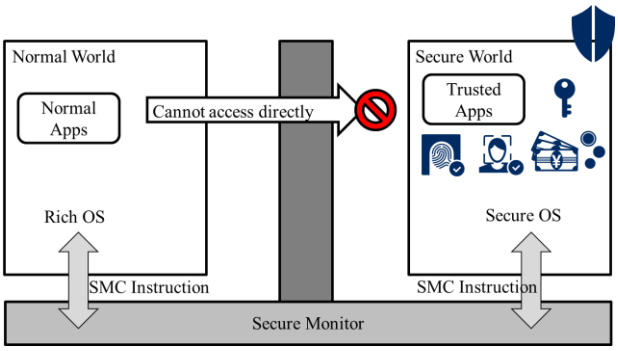


図 1 Cortex-A における TrustZone

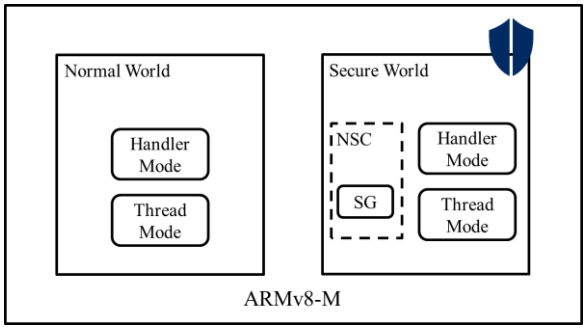


図 2 Cortex-M における TrustZone

2.2 Trusted Execution Environment

セキュアハードウェアの代表例として, GlobalPlatform が定義した TEE がある．TEE は，デバイスに対して汎用 OS が利用するメモリ領域とは別に，汎用 OS からは直接アクセス不可能なセキュアな領域 (セキュア空間) を構築する．このセキュア空間に，例えば秘密鍵や暗号処理プロセスなどを保存することで，外部からの情報窃取やプロセスの改ざんを防止することが可能となる．なお，一般的にはセキュア空間へはユーザですら直接アクセスを行うことができない．実際に TEE を実装している CPU の例として ARM の Cortex-A/M シリーズがある．ARM では，TrustZone[9]と呼ばれるセキュア空間 (Secure World) を構築し，セキュア空間上にセキュリティに関する処理やその処理で用いるデータを実装する．

Cortex-A と M では，TrustZone の実装が異なっている．Cortex-A では，図 1 のように汎用 OS とは別に，セキュリティの処理を行うための OS (セキュア OS) が実装されている．汎用 OS が動作する非セキュア空間 (Normal World) がセキュア空間へアクセスするためには，必ずセキュアモニターを経由する必要がある．つまり，非セキュア空間からはセキュア空間へ直接アクセスすることはできない．一方 Cortex-M では，図 2 にあるように，ハンドラモード (Handler Mode: 例外処理) とスレッドモード (Thread Mode: 通常処理) にセキュリティ保護状態を意味する Secure State (SS) と，従来から存在するセキュリティ保護がされていない状態を意味する Non Secure State(NSS)を実装

© 2017 Information Processing Society of Japan

している．Cortex-M の Cortex-A との一番の違いは，セキュアモニターの有無である．Cortex-M の TrustZone では，セキュアモニターを利用せずメモリのアドレスをベースにしてセキュリティの保護を実施する．具体的には，NSS ではアクセスできないメモリアドレスを設け，その部分にセキュアな処理やデータを実装する．つまり SS と NSS では同じメモリ空間を共有するものの，アクセス許可が異なっている．なお，Cortex-M では，非セキュア側からセキュア側の処理を安全に呼び出すための仲介役としてセキュアモニターの代わりに Secure Gateway (SG) 命令という拡張命令と Non-Secure-Callable attribute (NSC) という特殊な属性のメモリ領域が定義されている．NSC はセキュア空間に実装されているが, NSS でアクセスが可能なメモリ領域である．ただし，NSC には，SG 命令のみが実装されており，SG 命令以外は NSC から実行できない．また，非セキュア側からセキュア側の処理を呼ぶ際には，必ず SG 命令を実行しなければならない．これは，SG 命令でのみ SS に状態遷移できるためである．なお，Cortex-A はいわゆるハイエンド向けのプロセッサとして，Cortex-M はローエンドのプロセッサとして位置づけられており，デバイスに要求されるスペックに応じて実装される CPU が使い分けられる．

この他 Intel SGX[10]も TEE の具体的な実装例として知られている. SGX は Skylake から導入されたセキュリティに関連する x86 拡張命令セットである．SGX はアプリケーションがシステムメモリ上に外界から保護された領域 (Enclave) を作成することを許可する命令である．TrustZone のセキュア空間と同様に，Enclave へはそのシステムで動作中の他のプログラムがアクセスすることは不可能であり, また, Enclave 内で動作しているプロセスやデータはたとえユーザであっても直接操作できない．なお，現在までに多くの IoT デバイスには ARM 系の CPU が，IoT ゲートウェイと呼ばれる製品には Intel 系の CPU が搭載されていることから，様々な IoT デバイス/IoT ゲートウェイに TEE が実装される環境が整いつつある．

2.3 TEE の課題

本節では，TEE の持つ特徴とその特徴が原因となり生じる課題について説明する．一般的に TEE では，安全のためセキュア空間へ実装されている機能の権限は 1 種類のみとなっている．例えば Intel SGX ではリングプロテクションにおける Ring3，すなわち最低権限がセキュア空間に付与されている．この理由は，悪意あるプロセスがセキュア空間に実装された際にその動作を防ぐためである．つまり，アーキテクチャの設計における自由度に比べて，TEE 内部に実装されている機能がマリシャスであった際に，悪意ある動作を防ぐことに重きを置いた設計となっている．しかし視点を変えると，安全性に重きを置くためデバイス設計時の自由度が低下するという課題がある．つまり，ある程度の権限が必要となる機能はセキュア空間外に実装する必要が出てくる．一例としてデバイスドライバはセキュア空間外に実装しなければならない．これは，セキュア空間では直接デバイスのアクチュエータの制御機能 (デバイスドライバ) を利用できないためである．しかし，セキュア空間外は改ざんの脅威に晒されており，改ざんされたデバイスドライバによってデバイスのアクチュエータが正しく動作しない可能性ある．従って，設計の自由度と安全性との間にはトレードオフがある．

ここで，デバイスのアクチュエータを正しく動作させるためには，TEE と入出力ポート (アクチュエータ) 間で命令やデータをセキュアに伝達するチャネルが必要である．しかし，セキュア空間に設定される権限が 1 種類であることが原因となり，そのようなセキュアなチャネルを構築することが現状の TEE では難しい．この解決のために，本稿ではセキュア空間に与える権限をセキュア空間毎に個別かつ段階的に設定できる制御機構を提案する．次節にて提案手法を説明する．

3. セキュア空間への段階的アクセス権限設定

本節では，提案法であるセキュア空間への段階的なアクセス権限設定の概要と，権限設定に利用するポリシーの設定例を紹介する．

3.1 概要

提案手法では安全性と設計の自由度との間に生じるトレードオフという課題に対して TEE のセキュア空間のアクセス権限を段階的かつ個別に設定することによる解決を目指す．具体的には，非セキュア空間からセキュア空間へアクセスする場合，つまり，セキュリティレベルの高いメモリ領域へアクセスする場合は，必ずアクセス権限を管理す

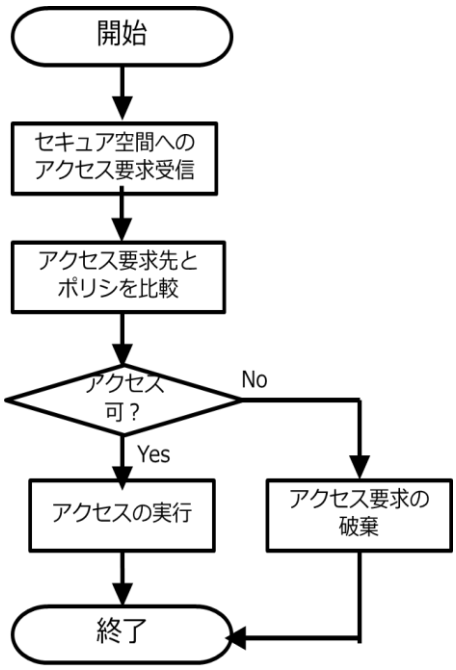


図 3 提案手法のフローチャート

© 2017 Information Processing Society of Japan

る装置 (以後, アクセス判定装置) を通過させる. アクセス判定装置は, 入力されたアクセス要求を解析して要求先にアクセス可能か否かを判定する機能を持っており, 可能であればアクセスを実行, 不可であれば入力された命令を破棄する機能を持つ. アクセス判定装置のフローチャートを図 3 にまとめる. なお, アクセス判定装置にはアクセス権限を記載したポリシーを保存しており, ポリシーに基づいて入力された命令のアクセス判定を行う. ポリシーの記述は, セキュア空間を Intel SGX の様に Enclave という複数の領域に分割し, それぞれの Enclave に対して権限を設定する. ポリシーの設定例を表 1 に示す. ここでは, Enclave を区別するために付与された ID (Enclave ID [10]) を用いてポリシーを記述しており, Enclave 1 は低い権限, Enclave 2 は高い権限を持っている. 次節にて, この提案手法を組み込んだデバイスのアーキテクチャ例を紹介する.

表 1 権限割り当てポリシー

Enclave ID	アクセス権限
Enclave1	低
Enclave2	高

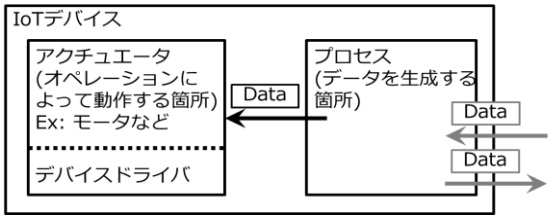


図 4 IoT デバイスの代表的な構成要素

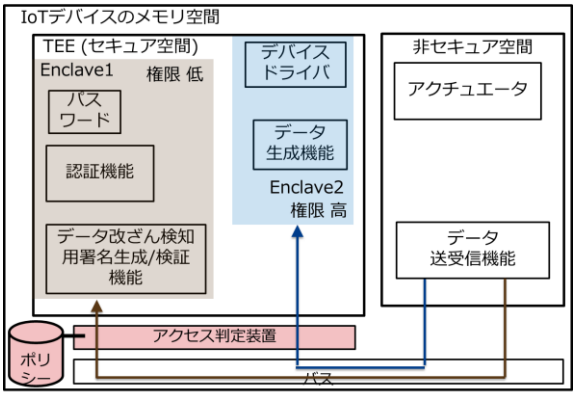


図 5 デバイスアーキテクチャ

4. 提案手法を実装するデバイスのアーキテクチャと評価

本節では, 提案する段階的アクセス権限設定法を実装したデバイスのアーキテクチャについて概要を説明する. また, OpenSGX[11]と呼ばれる qemu を利用した SGX のエミュレータを用いて, 提案手法を実装, 評価する.

4.1 提案手法を実装するデバイスのアーキテクチャ

本節では, 提案手法を実装したアーキテクチャを述べる. ここでスコープとしているのは, 図 4 に示すような 1 つの (単一の) IoT デバイスであり, TEE などの耐タンパ性を持つハードウェアレベルのセキュリティ機能を搭載したデバイスを想定する. IoT デバイスの代表的な構成要素例としてプロセスやアクチュエータ(入出力ポートの一例)があるが, これらは複数存在してもよい. なお, 図 4 における Data は単なる数値や文字列などのセンサ値に加えてアクチュエータへの命令コマンドも含まれている.

図 5 にアクセス判定装置を実装したデバイスの構成例を示す. デバイスにはセキュア空間 (TEE) と非セキュア空間が存在しており, 非セキュア空間からセキュア空間へアクセスするためには, 第 3 節にて提案したアクセス判定装置を経由する必要がある. なお, アクセス判定装置にはポリシーが保存されており, ポリシーに基づいて Enclave の権限を変更する.

アクセス判定装置とポリシーはセキュリティ機能やデバイスドライバなどを保存している TEE の外部に実装する. これは前述の TEE の内部に悪意を持ったプロセスが存在する場合, アクセス判定装置とポリシーが改ざんなどの脅威にさらされるためである. これらを保護するためには (1) TEE から隔離して CPU やセキュア処理専用のハードウェアに保存する, (2) アクセス判定装置とポリシーのみを実装した専用の TEE を作成する, などの対策が考えられる. 本稿では, OpenSGX で利用されている qemu のソースを一部改造することによって, 疑似的に CPU 上にアクセス判定装置とポリシーを組み込んでいる(詳細は 4.2 節). また, 図 5 の例は, 表 1 のポリシー記述を一部組み込んだ構成となっている. 元々の TEE では, デバイスドライバなど, ある程度の権限を必要とする要素は TEE 内部に実装することができなかったため, 入出力ポートと TEE との間にセキュアなチャネルを構築することは難しかった. 提案手法では, 図 5 の構成をとることで, デバイスドライバも TEE 内部に実装可能になるため, デバイスドライバを TEE で保護して改ざんを防ぐことができる. これにより, TEE からデバイスの入出力ポート, および実際に動作するアクチュエータ部までの間にセキュアなチャネルを構築することが可能となる.

加えて, TEE 内に実装された機能に合わせてその権限を設定できるため, TEE の機能と安全性を両立可能である. 具体的には, ドライバなどを実装する場合には TEE に高い

© 2017 Information Processing Society of Japan

権限を与え，それ以外の場合には低い権限を TEE に与えることで，TEE が汚染された際のリスクを低減できる．

4.2 提案手法の有効性評価

本節では OpenSGX を用いて提案手法を実装し，本研究にて提案する権限設定が実行できることを確認する．なお，実験に用いる PC は，Ubuntu 16.04 (64bit) を OS とし，CPU は Intel Core i5-3230M，RAM は 4GB である．

4.2.1 評価手法

まず，図 6 に文献[11]で説明されている OpenSGX のアーキテクチャと，本研究で改造を加えた部分を紹介する．OpenSGX は，一般的な qemu に SGX をエミュレーションする機能を加えた QEMU SGX をベースとしている．そして QEMU SGX 上に OS のエミュレータとライブラリを実装している．OpenSGX を用いたソフトウェアの開発者は Enclave に実装する機能を C 言語によって作成し，OpenSGX のコンパイル機能を用いてコンパイルすることによって Enclave Program を作成することができる．本稿では OpenSGX に付属されている Enclave 内で Hello world を実行するプログラム(図 7)を対象に評価を行う．具体的には，ソース内の puts 文の実行を Enclave の権限に応じて制御する．これによって Enclave から画面出力までにセキュアなチャネルを構築する．アクセス判定装置をハードウェア上に実装するために，OpenSGX の qemu に関するのソースのうち，sgx-helper.c に着目する．このソースは，OpenSGX で Enclave を実現するためのヘルパー関数がまとめられている．その中に，sgx_eenter()という，Enclave へのアクセスを司る関数がある．この中に，Enclave に応じた権限の設定と，権限に応じたアクセス制御を実装することによって，疑似的にではあるが CPU (ハードウェア) 上にアクセス判定装置とポリシーを実装することができる (図 8)．ポリシーの記述は Enclave ID ベースで行うものとし，その管理は get_auth()関数によって実現する．get_auth()関数は，引数となった ID を持つ Enclave の権限を整数値で返す．本実装例では，リングプロテクションと同様に，整数値の値が小さいほど権限が高いものとする．なお，本稿におけるアクセス判定装置は CPU の拡張機能として実装するため，Enclave へ実装する機能 (例えばデバイスドライバなど) そのものの改造は不要である．

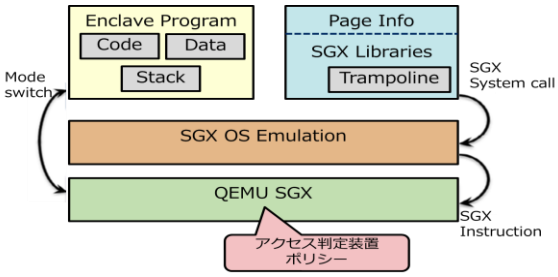


図 6 OpenSGX のアーキテクチャと提案手法実装のための改造部分

opensgx/user/demo/hello.c

```
22  #include <sgx-lib.h>
23  #include <stdio.h>
24
25  void enclave_main()
26  {
27      char *hello = "hello sgx!\n";
28      puts(hello);
29      sgx_exit(NULL);
30  }
```

図 7 Enclave Program の例 (Hello world)

qemu/target-i386/sgx-helper.c

```
static void sgx_eenter(){
    …
    int auth; //policy checker
    …
    // Ensure that the FLAGS field in the TCS does not have any
reserved bits set
    checkReservedBits((uint64_t *)&tcs->flags,
0xFFFFFFFFFFFFFFFEL, env);
    eid = tmp_secs->eid_reserved.eid_pad.eid;
    auth = get_auth(eid); //obtain access level from the policy

    /**/
    if(auth !=0){
       printf("You don't have permission to access the
enclave¥n");
       raise_exception(env, EXCP0D_GPF);
    }
    /**/
    …
```

ポリシを表現する配列の値を読む関数

権限がない場合は例外を発生させて処理を中断

図 8 OpenSGX の改造部分(赤文字部分)

4.2.2 評価結果

OpenSGX による提案手法の実行結果を図 9，図 10 に示す．図 9 は Enclave の権限が高い場合，図 10 は権限が低い場合の実行結果である．まず，図 9 より，Enclave の持つ権限が高い場合は，Hello world の puts 文が実行されていることがわかる．これは sgx_eenter()に追加したアクセス権限設定に関する機能 (図 8 に追記した Enclave の ID に応じた権限を取得する関数と，権限に応じて処理を中断する if 文) によって動作を実施可と判定されたため表示できた．

一方で権限が低い場合は Hello world の puts 文が実行できず，権限が無い旨が代わりに表示されている．なお，権限が無いことを表示するために sgx_eenter()に追記した printf は，Enclave 外に実装されている．この結果から，Enclave の持つ権限が低い場合は，アクセス権限設定に関する機能によって Enclave 内の機能の実行 (本実験では puts 文) を中断させることができたことを意味する．

© 2017 Information Processing Society of Japan

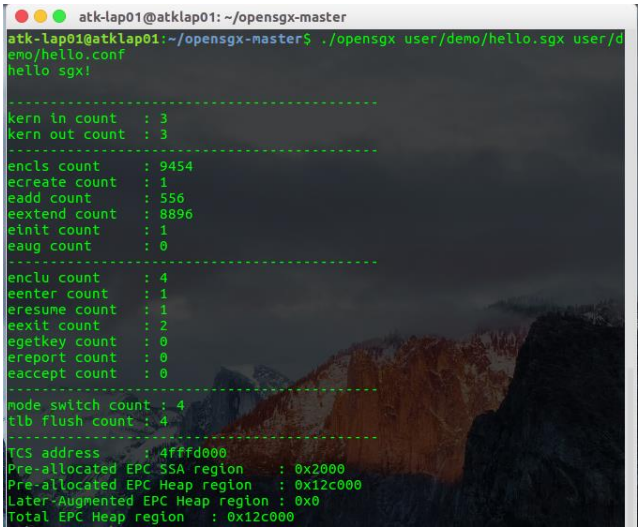


図 9 OpenSGX によるサンプルコードの実行結果
(権限 高)

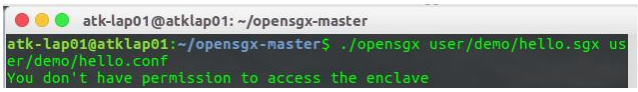


図 10 OpenSGX によるサンプルコードの実行結果
(権限 低)

従って，この結果よりアクセス権限に応じて Enclave 内のプロセスの実行可否を設定できた．つまり，Enclave の権限を個別かつ段階的に設定できることを示した．なお，今回の実験では Enclave から画面表示 (出力) までの一方向を制御したが，ポリシーの設定を変更することで，例えば Enclave への入力を制御することも可能である．つまり，双方向のセキュアなチャネルを構築することも可能である．

5. 関連研究

本節ではセキュアハードウェアを用いた関連研究を紹介する．特にセキュア空間から入出力までの間にセキュアなチャネルを構築することに着目した研究を紹介する．

まず，Intel SGX を対象とした関連研究として文献[12]が挙げられる．[12]では，Intel SGX の Enclave から入出力デバイスまでの間にセキュアなチャネルを作る方法の提案している．具体的には，信頼できるハイパーバイザーを構築するために，hypervisor attestation というハイパーバイザーへの Attestation を行うことを提案している．これによって，信頼できる対象を (a) TEE に保存されている Trusted Boot(TB)に必要な情報だけでなく，(b) ハイパーバイザー (c) 信頼できるデバイスドライバまで拡張している．なお，[12]では，権限管理はハイパーバイザーで実施することは記載されているが，本稿における提案手法とは異なり，ポリシー設定による Enclave への段階的な権限付与は行っていない．

次に，ARM TrustZone を対象にした関連研究として，文献[13]が知られている．[13]では，ユーザのインプットから画面のアウトプットまでの間に TrustZone を利用してセキュアなチャネルを構築する方法を提案している．具体的には，デバイスドライバをフロントエンドとバックエンドに分けてフロントエンドをセキュア領域に保存することで，画面表示をセキュアに実現させる．なお，デバイスドライバのフロントエンドとバックエンドは proxy を介してやり取りを行う．これによってユーザとサービス間でセキュアな Interaction が実現可能となる．ただし，実現のためにはデバイスドライバを一部改造する必要があり，またドライバを呼ぶためのラッパー関数を実装する必要がある．

セキュアなチャネルを構築するために利用されるものは TEE だけではない．例えば文献[14]では TPM と Intel Trusted eXecution Technology (TXT) [15]を併用することで，クライアントのデバイス内部から遠隔にあるサーバまでの間にセキュアなチャネルを構築する方法を議論している．なお，[14]で構築されるセキュアなチャネルは一方向のみを対象としており，双方向ではない．

TEE や TPM などのセキュアハードウェアを利用せずにセキュアなチャネルを構築する研究例も存在する．例えば文献[16]では，入出力デバイスからデバイス内部のアプリケーションまでの間にセキュアなチャネルを確立するために，セキュアなハイパーバイザーをアプリケーションによって構築する方法を提案している．ただし，実装方法によってはソフトウェアのバグによって脆弱性が発生する可能性が含まれる．

6. おわりに

IoT の普及に伴ってネットワークに接続される組み込み機器がますます増加している．その一方で，機器情報の改ざんなどの不正行為への対策が重要となってきており，特にセキュアハードウェアが期待されている．本研究ではセキュアハードウェアの一種である TEE に着目し，IoT デバイスを安全に制御するために TEE と入出力ポート間の命令やデータをセキュアに伝達するチャネルを構築することを目指して，セキュア空間の権限をセキュア空間毎に個別かつ段階的に設定する制御機構を提案した．さらに提案手法を実装したデバイス構成についてその基本設計を紹介し，OpenSGX を用いてその有効性を評価した．本稿における提案によって，IoT システムを構成するデバイスの入出力ポートから TEE までに構築されるチャネルを保護することが可能となる．

本研究の今後の課題として，現在以下の点について検討中である．

- 提案手法の実機による有効性評価
- TEE を中心とした IoT デバイス向けの汎用的セキュアデバイスアーキテクチャの確立

© 2017 Information Processing Society of Japan

参考文献

[1] H. Arasteh, V. Hosseinnezhad, V. Loia, A. Tommasetti, O. Troisi, M. Shafie-khah, and P. Siano``Iot-based smart cities: A survey,'' Proc. IEEE 16th EEEIC 2016, pp.1–6, 2016.

[2] L. Atzori, A. Iera, and G. Morabito, ``The Internet of Things: A survey,'' Comput.Netw., vol.54, no.15, pp.2787–2805, 2010.

[3] GlobalPlatform official site, available at https://www.globalplatform.org/, cited 2017.

[4] Z. Durumeric, J. Kasten, D. Adrian, J.A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman,M. Payer, and V. Paxson, ``The matter of heartbleed," Proc. ACM IMC 2014, pp.475–488, 2014.

[5] K. Nohl, ``BadUSB – On accessories that turn evil,'' Proc. Blackhat USA 2014, 2014.

[6] FFRI 2013-04-05 Monthly Research ``セキュアハードウェアの登場とその分析,'available at http://www.ffri.jp/assets/files/monthly_research/MR201303_TrustZone.pdf, cited 2017.

[7] W. Arthur, D. Challener, and K. Goldman, A Practical Guide to TPM 2.0—Using the New Trusted Platform Module in the New Age of Security, Apress, 2015.

[8] S. Rohilla, ``Secure element An evolution to existing secure technology,'' International Journal of Scientific and Research Publications, vol.5, no.7, 5pages, 2015.

[9] ARM TrustZone Technology Overview, available at https://www.arm.com/products/security-on-arm/trustzone, cited 2017.

[10] V. Costan and S. Devadas, ``Intel SGX explained,'' IACR Cryptology ePrint Archive 2016/086, 2016.

[11] P. Jain, S. Desai, S. Kim, M.-W. Shih, and J. Lee, ``OpenSGX: An open platform for SGX research," Proc. NDSS2016, 16pages, 2016.

[12] S. Weiser and M. Werner, ``SGXIO: Generic trusted I/O path for Intel SGX,'' Proc. ACM CODASPY 2017, pp.261–268, 2017.

[13] W. Li, M. Ma, J Han, Y. Xia, B. Zang, C.-K. Chu, and T. Li, ``Building trusted path on untrusted device drivers for mobile devices,'' Proc. ACM APSys 2014, no. 8, 7pages, 2014.

[14] A. Filyanov, J.M. McCune, A.-R. Sadeghi, and M. Winandy, ``Uni-directional trusted path (UTP): Transaction confirmation on just one device,'', Proc. IEEE/IFIP DSN 2011, pp.1–12, 2011.

[15] White paper, ``Intel® Trusted Execution Technology, Hardware-based technology for enhancing server platform security,'' available at http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html, cited 2017.

[16] Z. Zhou, V.D. Gligor, J. Newsome, and J.M. McCune, ``Building verifiable trusted path on commodity x86 computers,'' Proc. IEEE S&P 2012, pp.616–630, 2012.

© 2017 Information Processing Society of Japan