Doctoral Dissertation Academic Year 2015

Security Platform for Embedded End-point Devices in a Smart Grid

Hiroshi Isozaki

Graduate School of Media and Governance
Keio University

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY IN
MEDIA AND GOVERNANCE

Thesis Committee:
Prof. Yoshiyasu Takefuji, Chair
Prof. Yasushi Kiyoki
Prof. Tatsuya Hagino
Prof. Keiji Takeda

Copyrighted
by
Hiroshi Isozaki
All rights reserved.
©Hiroshi Isozaki, 2015

Abstract **Academic Year 2015**

Security Platform for Embedded End-point Devices in a Smart Grid

In this dissertation, we present a software security platform for embedded end-point devices in a smart grid. Specifically, we propose a method to isolate security-sensitive processes from general-purpose processes utilizing security functions of a commodity embedded processor, identify the functions to be included in the security-sensitive processes, perform a full implementation of a system based on the proposed method, and present evaluation results with respect to both security functions and performance.

The proposed security platform provides secure updatability and high availability as well as satisfying legacy security requirements, such as confidentiality and integrity, to enable a fault-tolerant system with long-term security. In order to keep long-term security, the method provides a function to dynamically load and update a legitimate security-sensitive module only with sufficient robustness against tampering. Since the security-sensitive processes are executed in a secure environment, illegitimate modification and information leakage of the security-sensitive processes can be prevented even if the general-purpose processes are modified or their control is taken over. To keep availability, the system introducing our proposed method monitors the status of the operating system and recovers even if the operating system stops working owing to unexpected behavior or cyber-attacks.

Based on the proposed methods, we further propose an autonomous distributed smart grid architecture by introducing a secure mobile agent system in which the protection-required module of a mobile agent can be executed securely without interference by attackers. The proposed secure mobile agent system is very useful and enables new applications in field area networks of smart grids, such as privacy information protection and pay-per-use software charging.

Keyword: Security, Trusted Computing, Embedded Software, Smart Grids, Mobile Agents

Hiroshi Isozaki
Graduate School of Media and Governance
Keio University

論文要旨　2015年度

スマートグリッドで活用する
組込み機器向けセキュリティプラットフォームの提案

本論文では，スマートグリッドで活用する組込み機器向けソフトウェアセキュリティプラットフォームについて述べている．本論文では，汎用の組込み向けプロセッサのセキュリティ機能を活用してセキュリティを確保すべき処理と汎用の処理を分離する方法を提示し，セキュア状態で実行させるべき機能を明確化した．さらに，提案手法に基づくシステムを実装し，セキュリティ機能と性能の両面から評価を行った．

提案するセキュリティプラットフォームは，長期安全性と耐障害性を備えたシステムを実現するために，従来のセキュリティ要件である秘匿性と完全性の確保に加え，セキュアにモジュールをアップデートする方法と高い可用性を実現している．長期安全性を確保するために，正規のセキュリティモジュールに限定して動的にロードし，アップデートする機能を実現している．提案手法では，セキュリティに関連する処理はセキュア環境で実行されるため，汎用処理が不正に改変されたり制御が奪われたりしても，モジュールの不正改変やモジュールに含まれる情報の漏えいを防ぐことができる．また，提案手法を適用したシステムでは予期せぬエラーや攻撃によってオペレーティングシステムが停止したとしても，オペレーティングシステムの監視と回復処理が行える仕組みを提供することで，高い可用性を実現している．

さらに本論文ではこれらの提案手法を要素技術とし，セキュアモバイルエージェントシステムを実現することで自律分散型のスマートグリッドアーキテクチャを提案している．このシステムでは，攻撃者に妨害される事なくモバイルエージェントに含まれる保護対象モジュールをモバイルエージェントシステム上で実行することができる．また，提案するセキュアモバイルエージェントシステムにより，プライバシ保護，利用状況に応じたソフトウェア課金のような新しいアプリケーションがスマートグリッドのフィールドエリアネットワークで実現できる．

キーワード：セキュリティ，トラステッドコンピューティング，組込みソフトウェア，スマートグリッド，モバイルエージェント

慶應義塾大学　大学院
政策・メディア研究科
磯崎宏

Acknowledgements

First and foremost, I would like to express my great appreciation to my supervisor, Professor Yoshiyasu Takefuji for his valuable and constructive suggestions throughout this work. He has been always supportive since I started my research career as an undergraduate student. Ever since, he has guided me with his great patience and tolerance. This dissertation would not have been possible without him. I would also like to express my appreciation to thesis committee members, Professor Yasushi Kiyoki, Professor Tatsuya Hagino, and Professor Keiji Takeda for their services and help they have given to me.

I would like to thank all the people in Toshiba Corporation who have helped me to finish this work, especially Dr. Mutsumu Serizawa, Mr. Atsushi Inoue, and Mr. Takashi Yoshikawa for their encouragement and supports to keep up both my studies and my jobs. I would also like to thank the co-authors of my journal articles and a conference paper. Especially, Dr. Jun Kanai is gratefully acknowledged for his tremendous help and insightful suggestions.

Lastly, and most significantly, I would like to thank my family. To my parents, for the gift of an education, and the example of a work ethic. To my wife, Miho, for her love and faithful support. To my children, Riku and Rui, who brought joy into my life and inspired me to hurry up to complete my study. Without them, none of this study would have happened.

Hiroshi Isozaki
Graduate School of Media and Governance
Keio University

Contents

Chapter 1 Introduction 1
1.1 Motivation 1
1.2 Objective 4
1.3 Main contributions 5
1.4 Dissertation overview 9

Chapter 2 Problem definition 11
2.1 Security problems in smart grids 11
2.2 Problem of keeping long-term security 14
2.3 Problem of keeping availability 17

Chapter 3 Background 20
3.1 ARMv7 architecture 20
3.2 TrustZone 22
3.3 Trusted Platform Module (TPM) 24

Chapter 4 The proposed method 25
4.1 A method to keep long-term security 25
4.1.1 Framework of the virtual security hardware system 25
4.1.2 Functions of the virtual security hardware system 27
4.1.3 Process flow 33
4.1.4 Prototype implementation 37
4.1.5 Evaluation 40
4.2 A method to keep availability 56
4.2.1 Framework of the recovery system 56
4.2.2 Functions of the recovery system 57
4.2.3 Prototype implementation 63
4.2.4 Evaluation 71

Chapter 5 Secure mobile agent system 80

5.1 Security threat to mobile agent system ... 80
5.1.1 Mobile agent system ... 80
5.1.2 Security threat ... 81
5.1.3 Apply secure mobile agent system to smart grids ... 84
5.2 Architecture of the secure mobile agent system ... 86
5.3 Functions of the secure mobile agent system ... 88
5.4 Process flow ... 93
5.5 Prototype implementation ... 96
5.6 Evaluation ... 98
5.6.1 Security and cost analysis ... 98
5.6.2 Performance analysis ... 103
5.7 Application examples ... 108

Chapter 6 Related work ... 111

Chapter 7 Future work ... 117

Chapter 8 Conclusion ... 120

References ... 123

List of Figures

Figure 1.1: Overview of a smart grid......2
Figure 3.1: Mode and world in ARM......22
Figure 4.1: Architecture of the virtual security hardware system......27
Figure 4.2: Execution flow of the Re-encryption module......30
Figure 4.3: Execution flow of the update process of......35
Figure 4.4: Execution flow of remote attestation......37
Figure 4.5: Throughput of the re-encryption process......49
Figure 4.6: Performance ratio of the re-encryption process......49
Figure 4.7: Evaluation result of the network latency......50
Figure 4.8: Performance result of the module update process......51
Figure 4.9: Architecture of the recovery system......56
Figure 4.10: Flowchart of the periodic surveillance and recovery process......60
Figure 4.11: Memory protection mechanism......62
Figure 4.12: Assignment of the timer interrupt......66
Figure 4.13: Flowchart of the access violation handling......70
Figure 4.14: Result of the performance degradation......77
Figure 4.15: Result of the performance degradation with message notification......77
Figure 5.1: Network architecture of a smart grid......85
Figure 5.2: Architecture of the secure mobile agent system......87
Figure 5.3: Process to develop and install the SMA module......91
Figure 5.4: Execution flow of Secure Mobile Agent module installation......95
Figure 5.5: Performance result of SMA module installation (plaintext)......105
Figure 5.6: Performance result of SMA module installation (encrypted)......105
Figure 5.7: Performance result of SMA module installation (encrypted and signed)......106
Figure 5.8: Performance ratio of the SMA module execution......108

List of Tables

Table 2.1: Differences between information and communication system and control system 12
Table 3.1: ARM processor modes and bank registers 21
Table 4.1: Memory map of the virtual security hardware system 38
Table 4.2: Configuration of access control policy 61
Table 4.3: Memory map of the recovery system 63
Table 4.4: Configuration of TZASC 65
Table 4.5: Configuration of hardware interrupt 67
Table 4.6: CPSR and SCR register configuration 68
Table 5.1: Access control policy and its mapping 88
Table 5.2: Memory map of the virtual security hardware module 97

Chapter 1

Introduction

Security is a prerequisite for the commercialization of smart grids. This dissertation focuses on a secure system to keep embedded end-point devices in a smart grid secure, and pays particular attention to realization of a robust mechanism against attacks, including illegitimate modification and eavesdropping, at reasonable cost in terms of development, deployment and maintenance. This chapter provides an overview of the research presented in this dissertation.

1.1 Motivation

A smart grid is a next-generation energy-providing system combining energy infrastructure and communication network infrastructure. Widespread deployment of smart grids is expected in view of the need for clean energy through large-scale use of renewables and optimization of distributed energy resources to deliver electricity efficiently and reliably.

Figure 1.1 shows an overview of a smart grid. A smart grid consists of many components: end-point devices in the household, such as a smart meters, Home Energy Management Systems (HEMSs); head-end systems including Energy Management Systems (EMSs) that control power flow in order to balance and optimize power generation and consumption, and Meter Data Management Systems (MDMSs) that receive meter data from smart meters and utilize utility operations, such as billing and outage management; concentrators that relay data between the end-point devices and the head-end system; distributed power generation equipment, such as

wind turbines and photovoltaic generators; conventional power plants, such as thermal power plants, nuclear power plants, and coal-fired power plants; factories; buildings; and electric vehicle charging stations. These various components are interconnected and transmit control signals and data through networks.

Applications of smart grids vary. The typical applications are home energy management, demand response management, outage management, electric vehicle charging and overhead transmission line monitoring [1][2]. All applications effectively utilize the advantages of the network.

Since the assets handled in smart grids are valuable and the transmitted control signals and the data include confidential information, security breaches in smart grids may involve serious damage to utilities, government, and end-users. Examples of such damage include massive outages or power shortages, reconfiguring or compromising of equipment, economic damage due to a utility company's loss of revenue, and privacy-related information leakage as a result of theft of power usage [3]. Therefore, security is a prerequisite for the commercialization of smart grids.

In order to address the security concerns, much work involving many

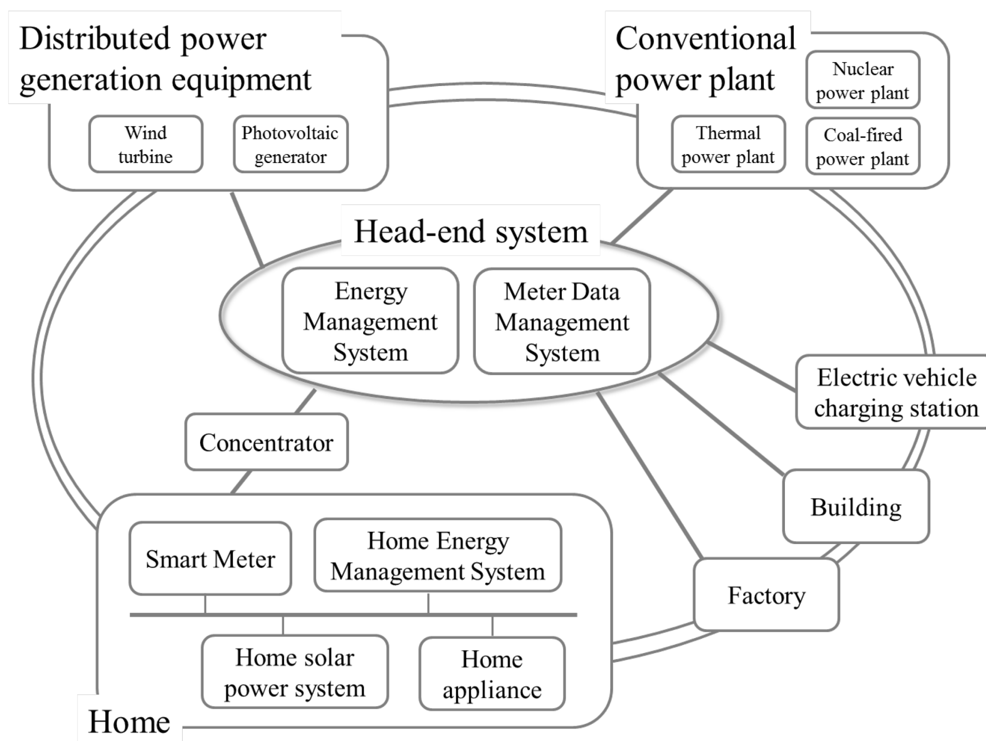

Figure 1.1: Overview of a smart grid.

researchers and engineers has been done to establish security standards for smart grids. For example, in the U.S., in 2010 the National Institute of Standards and Technology (NIST) published Interagency Report 7628, Guidelines for Smart Grid Cyber Security, providing an analytical framework including characteristics, risks, and vulnerabilities related to smart grids [4]. In Europe, in 2014 the European Smart Grid Coordination Group (SG-CG) finalized a report, Smart Grid Information Security, providing guidance, standards landscape and requirements [5].

Although those documents are useful for sharing the basic concept, making specifications, and designing an entire system securely, they are insufficient because systems are not always implemented correctly. It is very difficult to exclude implementation errors from a system even if developers carefully design the system and review its source code. Furthermore, whereas most previous research proposed crypto algorithms and security protocols, there have been few proposals of security architecture to withstand attacks that exploit implementation vulnerability and the matter is left largely to the efforts of individual vendors [6]. In light of the history of information and communication systems, it is inevitable that implementation vulnerability will lead to security incidents in control systems such as smart grids in the near future. Stuxnet, the computer worm that attacked Iran's nuclear facilities in 2010, is well known because the importance of control-system security is widely recognized. Besides Stuxnet, attacks on critical infrastructure have been reported from many places in the world and those attacks exploit implementation vulnerability. Although Stuxnet targeted head-end systems, end-point devices can be targets too. In fact, the damage of an attack is more serious than the head-end server since tremendous numbers of embedded end-point devices will be deployed in the near future. In fact, the deployment of a smart meter, which is a typical example of an embedded end-point device in a smart grid, is in progress in many countries. In Japan in April 2014, Cabinet approved the new Basic Energy Plan that sets a target to introduce smart meters to all households by the early 2020s. Followed by the approval, Ministry of Economy, Trade and Industry (METI) disclosed the introduction plan of a smart meter of each power utility in December 2014. The plan describes that Tokyo Electric Power Company will complete the introduction of 28 million smart meters by 2020. If such a large number of smart meters are deployed without security protection mechanism, there is a great threat to the industrial infrastructure. Actually, it was reported that researchers found a security flaw in a smart meter installed in Spain, which potentially could cause

widespread power outages [7]. Therefore, a security system for embedded end-point devices is strongly desired. However, since the problems and the requirements concerning the keeping of security have yet to be clearly defined, reasonable and effective methods have yet to be developed.

This research is motivated by the intrinsic difficulties of keeping security of embedded end-point devices in smart grids and social needs associated with this issue. With a view to accelerating the spread of smart grids, the approach in this dissertation realizes sufficient robustness against attacks on embedded end-point devices.

1.2 Objective

The objective of this research is to propose a method of enhancing the security of embedded end-point devices in smart grids. Although the importance of security is widely recognized and many cryptographic techniques and authentication protocols have been proposed, security problems concerning implementation vulnerability have not been well defined. As well as enhancement of robustness, cost is an important aspect when evaluating security architecture. If the new security architecture requires costly dedicated hardware or a complete software rebuild, it will never be commercialized. It is necessary to strike the right balance between security and cost.

Moreover, security is often regarded as add-on functions to complement applications. In other words, security functions are sometimes viewed as interference with applications or network architecture and the existing system is restricted because of security concerns. However, there is a way to utilize security that enables the introduction of new applications to smart grids. To achieve the above objectives, the following goals have been set.

(1) To clarify security problems unique to embedded end-point devices in smart grids and to develop methods and demonstrate security systems for embedded end-point devices on existing platforms, including software and hardware, with minimum performance degradation and at reasonable cost.
(2) To propose a software security platform for embedded end-point devices that enables the introduction to smart grids of new applications that work efficiently and reliably.

Several types of embedded end-point devices are used in smart grids. Some are PC-based systems with high-end processors, some are

embedded systems with embedded application processors, and others are small-scale embedded systems with microcontrollers. Since existing security technologies are applicable to PC-based systems, they are beyond the scope of this dissertation. In addition, since small-scale embedded systems with microcontrollers have limited functions, capabilities, and network interfaces, security threats are minimized. Therefore, they are beyond the scope of this dissertation too. In traditional control systems, as it is assumed that embedded devices neither connect to the network nor have rich functions, small-scale embedded systems have had sufficient processing power. In smart grids, however, the requirements for supporting various applications increase, network connectivity becomes a mandatory function, and small-scale embedded systems are insufficient. Embedded systems with application processors will replace small-scale embedded systems and be widely introduced in the near future. In view of this situation, the primary target of this dissertation is set to embedded systems with embedded application processors.

1.3 Main contributions

The research in this dissertation makes a number of contributions to the security platform for embedded end-point devices: keeping long-term security, satisfying the three pillars of information security, demonstrating feasibility with full implementation, and a security platform enabling the introduction of new applications to smart grids.

Keeping long-term security

It is assumed that embedded end-point devices in smart grids are kept in use for a long period of time once they are deployed in the field. Even if all bugs and vulnerabilities are eliminated from the embedded end-point devices at the time of shipment, new vulnerabilities may be found after deployment. In fact, new vulnerabilities are frequently reported [8][9]. Therefore, a mechanism to prevent an attack found after deployment is required. Furthermore, tremendous numbers of embedded end-point devices will be deployed in the field for smart grids. In the case of a cyber-attack, since many embedded end-point devices could be targets of the attack and the attack could be done in a very short period of time through the network, it is impracticable in terms of both cost and time for field service engineers to physically visit each site and replace the devices with new ones.

Although providing an update mechanism through the network is a solution to keep devices at the latest status, it is not easy to implement it since there is no trust anchor in a software system. Furthermore, previous methods require rebooting an entire system, including an operating system, when installing update modules, including security-sensitive functions, since general-purpose functions and security-sensitive functions are integrated. However, rebooting the system is unacceptable for embedded end-point devices in smart grids since it would be necessary to stop services while rebooting the operating system.

To solve these problems, we propose a method to dynamically and securely load and update a module, which includes security-sensitive processes and data, without requiring rebooting the system by providing a secure isolated execution environment, in which a protection-required module transmitted through the network is securely loaded, updated, and executed on working memory [10]. The overhead of executing the protection-required module in the environment is sufficiently small. In particular, depending on the applications, the performance degradation is less than 10% in a severe case compared with a solution based on vulnerable legacy software. Furthermore, in order to minimize the development cost, the proposed method provides the same interface as the traditional security hardware although implemented as software with robustness equivalent to hardware, so that developers can use the protection-required module in the traditional manner.

Satisfying the three pillars of information security

The purpose of information security is to ensure the three pillars of information security: confidentiality, integrity, and availability. Whereas most previous research endeavored to keep confidentiality and integrity, few attempts were made to keep high availability of the devices although it is strongly desired since they must always be working in order to provide various services in smart grids. As a single vulnerability may cause the system to go down, developers need to exclude any vulnerability from the system when designing and implementing it using legacy approaches. However, requirements for the support of various protocols and their associated functions increase the complexity of embedded end-point devices in smart grids, making it very difficult to exclude vulnerability. Therefore, the need to keep device availability in smart grids poses a significant challenge. We propose a novel method involving isolation from general-purpose processes, including the operating system,

of a surveillance process observing the state of the system and a recovery process that reboots the operating system when it detects the system freezes [11][12]. In the proposed method, even if the operating system is attacked and crashes, it is possible to prevent interference in the surveillance and recovery processes, and consequently sufficient robustness is kept against attacks. Furthermore, the overhead of the surveillance process is sufficiently small in our proposed method. In particular, the performance degradation is under 0.2% in a normal use case. Furthermore, we propose a method of notifying administrators when a crash through the network is detected, thus helping administrators investigate the result of the crash.

Demonstrating feasibility with full implementation

Developers desire the total architecture in order to evaluate the feasibility in terms of both security and performance. We demonstrate a full implementation of the proposed methods on existing widely available embedded processors to show that it is applicable to various embedded end-point devices. The evaluation with the implementation clarified that the architecture makes it possible to keep the three pillars and performance degradation is sufficiently small to be ignored. Moreover, the implementation shows that it is possible to reuse existing software assets, which contributes to reduction of the development cost. The embedded end-point devices need to support many functions, which are not directly related to security functions, in order to provide various services. For example, smart meters have a two-way communication function, a data collection function, a data storing function, a display function, and a billing function [13]. Therefore, it is unreasonable to force developers to discard existing software assets and build software from scratch when introducing security functions. We design a system in which developers can reuse all existing programs, including operating system, libraries, middleware, and applications, enabling them to minimize development costs.

Enabling new applications in a smart grid with a security platform

In PC-based systems, it is possible to include many security features with the supports of powerful processing power and sufficient computational resources. However, it is unreasonable to have rich functions in embedded end-point devices in smart grids due to the limited computational

resources. Furthermore, such complex systems require large verification costs, which is unacceptable for developers. We propose security architecture where a function can be dynamically added to embedded end-point devices, depending on the role of devices by applying a concept of a secure mobile agent system to smart grids. A mobile agent system is a distributed system where a program called a mobile agent autonomously moves from one host to another connected through a network. Although many mobile agent systems have been proposed, few studies address the problem of keeping secrecy and integrity of mobile agents. If this problem is addressed properly, it will be useful for many applications since it is possible to reduce the risk of illegitimate interception or modification of a mobile agent’s data and code through the network or on the host. Smart grids are a potential application of a secure mobile agent system. In smart grids, several network architectures are considered. The lack of security may restrict the network architecture even if some architectures offer great advantages, causing utilities to hesitate to introduce network architecture. In particular, a wireless mesh network is considered to be well suited to mobile agent systems. We propose a secure mobile agent system in which a mobile agent is divided into a general-purpose module and a security module. Only the security module is executed in a secure isolated execution environment. Our proposed method enables mobile agents to execute their processes on untrusted mobile agent platforms. As the result, developers can dynamically add functions securely before and after deployment, improving flexibility of system design significantly. Furthermore, since the security module is transmitted in an encrypted manner with the general-purpose module when migrating, it is possible for the mobile agents to protect security-sensitive processes and data. We show that the secure mobile agent platform enhances the capability of the Field Area Network (FAN) in smart grid network architecture and enables many new applications that have not been achieved previously, such as privacy information protection, pay-per-use software charging and software activation, and safety vault. Whereas security is frequently regarded as cost, we successfully present that security features can bring additional values to smart grids by showing new useful and valuable application examples.

1.4 Dissertation overview

Chapter 2: Problem definition

In this chapter, we define the problems to be solved in this dissertation. First, we show an overview of security problems in smart grids and clarify the differences between information and communication systems and control systems. Then, we primarily focus on two critical problems unique to the control system: problems concerning the keeping of long-term security and problems concerning the keeping of availability.

Chapter 3: Background

In this chapter, we provide background information necessary to understand the remainder of this dissertation. Since the target devices of this dissertation are embedded end-point devices in smart grids, we briefly explain available security functions of embedded processors. We also introduce available security hardware whose functions our proposed method replaces.

Chapter 4: The proposed method

In this chapter, we present a security platform for embedded end-point devices in a smart grid. We propose two methods: a method to dynamically load and update a security-sensitive module only without rebooting an operating system, and a method to achieve a fault-tolerant system. We also demonstrate a prototype implementation of those methods. Finally, we present the results of experiments in terms of both qualitative and quantitative evaluation. In the qualitative evaluation, we verify that our proposed methods resolve the problem described in the previous chapter. In order to enhance robustness, performance degradation is unavoidable. In the quantitative evaluation, we evaluate the extent to which our proposed methods degrade performance.

Chapter 5: Secure mobile agent system

In this chapter, we propose a secure mobile agent system utilizing the proposed methods. First, we summarize the security threat to a mobile agent system. Although mobile agent systems have a long history and many systems with security mechanisms have been proposed, the threat that a mobile agent platform will attack mobile agents has yet to be solved. We propose the provision of a secure isolated execution environment for

mobile agents to protect their resources. We also demonstrate prototype implementation and show experimental results. Furthermore, we show that the FAN architecture in smart grids is well suited to a secure mobile agent system and present application examples that have not been achieved previously.

Chapter 6: Related work

In this chapter, we refer to papers related to this dissertation. The research field of this dissertation is trusted computing. We introduce technologies to enable trusted computing, including dedicated security hardware and virtualization. The proposed methods achieve a fault-tolerant system. We will refer to related work on existing methods to enable fault-tolerant systems. We will also refer to related work on secure mobile agent systems.

Chapter 7: Future work

In this chapter, we refer to future work to be addressed in order to improve the proposed method.

Chapter 8: Conclusion

In this chapter, we present our conclusions.

Chapter 2

Problem definition

In this chapter, we define the problems to be solved in this dissertation. First, we clarify security problems unique to embedded end-point devices in smart grids. Then, we focus on two major problems that have not been fully solved in previous research: problems concerning the keeping of long-term security and problems concerning the keeping of availability.

2.1 Security problems in smart grids

Keeping security is a great challenge in smart grids. Since smart grids are complex systems, analyzing security of smart grids is also a complex matter. Much work has been done to expose threats to smart grids and clarify requirements for smart grids in order to minimize security risks [14]. For example, requirements in smart grids can be summarized as high-level security requirements in general and the major security requirements and vulnerabilities with respect to confidentiality, integrity, availability, authentication, authorization, auditability, nonrepudiability, privacy, third-party protection, and trust components for smart grid communications [15]. In order to fulfill the requirements, countermeasures have been analyzed [16][17]. However, many problems still remain since smart grids have many assets to be protected. In particular, smart meters are important assets to be protected because vulnerability can easily be monetized, making them extremely attractive targets for attackers [18]. For example, if smart meters are attacked, attackers can manipulate the energy cost or fabricate energy meter

readings by compromising smart meters. In fact, at a security conference in 2009, researchers showed that it is potentially possible to gain control of all exposed smart-meter capabilities, including remote power on, power off, usage reporting, and communication configurations, by exploiting the vulnerability of smart meters [19].

Many security technologies and solutions have been developed for information and communication systems and are widely used. Although some of them are applicable and useful for control systems, since there are many differences between control systems of smart grids and those of information and communication systems, features unique to control systems make it difficult to apply them without modification. Table 2.1 summarizes the differences from the viewpoint of security [20][21].

Table 2.1: Differences between information and communication system and control system

		Information and communication system	*Control system*
Features	Life cycle	3-5 years	Over 20 years
	Availability	Accepted	Need to work on 24h/365d
	Function	Changed by user operations (e.g. install application)	Controlled and fixed
	Delay	Accepted	Not accepted
Countermeasure	Virus protection	Widely used	Rarely, vendor dependent
	Security patch	Periodically applied	Slow, vendor dependent
	Update	Periodically update	No update
	Vulnerability check	Check periodically	Non-periodical

A typical example of the differences concerns device lifetimes. Whereas device lifetimes in smart grids is expected to be over 20 years, they are expected to be 3-5 years in the case of information and communication systems. Therefore, a method of updating the system is essential. However, it is not easy to introduce an updating mechanism in a software system. For example, when a module to be updated has confidential information, it is necessary to encrypt the module when distributing it and to decrypt it when installing it in the system. However,

since there is no trust anchor in the current software system, it is difficult to protect a key necessary to decrypt the module. Furthermore, an update mechanism sometimes requires rebooting an operating system since general-purpose functions and security-sensitive functions are integrated and it is difficult to replace security-sensitive functions only. However, it is unacceptable in the case of a control system since the unplanned outage could have catastrophic effects on the control system [22]. Moreover, a tremendous number of devices are deployed at remote sites, as opposed to being deployed in a local area network in the case of an information and communication system, which means administrators are unable to operate devices on site. Therefore, a mechanism for secure remote updating of embedded end-point devices is essential in smart grids although the method is as yet unclear.

Another major difference concerns the acceptable level of availability. As in the case of information and communication systems, the high-level security objectives for smart grids, including smart meters, are the keeping of confidentiality, integrity and availability [23][24][25]. To ensure confidentiality and integrity, many cryptographic techniques and authentication protocols have been proposed. Furthermore, they have been evaluated to check their applicability to embedded devices, particularly to those with low processing power [26][27][28][29]. Besides confidentiality and integrity, high availability of the devices is strongly desired since they must always be working to provide various services [16][30]. Although embedded end-point devices in smart grids have fewer functions than devices used in information and communication systems, requirements concerning the support of various protocols and the associated functions make their systems more complicated. As a single vulnerability may cause the system to go down, it is very difficult to keep high availability in a complicated system. Furthermore, unlike in the case of interactive devices, such as PCs or smartphones, it is unreasonable to expect end users to reset and restart embedded end-point devices once they freeze or hang since end users cannot recognize the status of the embedded end-point devices and cannot determine whether they should be rebooted or not. Thus, the need to keep the availability of embedded end-point devices in smart grids poses a significant challenge.

In the following section, we describe the two major problems in detail: the problem of keeping long-term security and the problem of keeping availability.

2.2 Problem of keeping long-term security

Hardware vs. software

In most security systems, a cryptosystem is used to ensure confidentiality and integrity. Since crypto processes incur high calculation cost, the usual approach has been for crypto processes to be implemented as dedicated hardware and for software to use the hardware functions in the case of embedded devices in legacy systems. However, in view of the evolution of embedded processor and development of fast calculation algorithms, developers increasingly tend to implement crypto processes as software.

When crypto processes are implemented as hardware, it is possible to keep high confidentiality since confidential processes and data are embedded inside hardware and it is very difficult to modify hardware functions without specialized tools, but there is a disadvantage in that physical replacement is required when updating crypto functions. This disadvantage is critical for embedded end-point devices in smart grids since a tremendous number of embedded end-point devices are deployed at households and it is impracticable to physically replace all of them in a short period of time in the event of security incidents, such as information leakage including the key value.

On the other hand, when crypto processes are implemented as software, it is easy to implement an update mechanism to replace a crypto algorithm, key length, or key values, but the risks of information leakage and illegitimate modification become high since it is much easier to analyze or modify software than hardware. Furthermore, it is difficult to establish a trusted base in the case of a software-only implementation. If crypto processes and confidential data are implemented in application programs and an operating system or middleware has vulnerability, attackers may exploit these weaknesses, resulting in modification of the update mechanism, invalidation of the crypto system, or eavesdropping of secret information.

To make matters worse, embedded end-point devices introduced in smart grids are used for much longer than those applied in information technology systems. There is a great risk that new vulnerability will be found while end-point devices are in service. Therefore, it is desirable to make the system updatable in order to prevent attacks caused by misuse of new vulnerability found after the deployment of end-point devices. In fact, according to guidelines in the U.S., vendors are required to design and implement systems for application in smart grids that are updatable [4][31].

Difficult to exclude vulnerability in a large system

In order to keep a system secure, a system needs to be implemented without vulnerability. Various testing methods have been proposed to detect and eliminate vulnerability in source code [32][33]. However, since embedded end-point devices will be deployed without maintenance over a long period of time in smart grids and new vulnerabilities are frequently found, there is a large risk that such devices will continue operating without vulnerabilities being fixed even if those devices had no vulnerabilities when shipped. For example, Ret2Libc is a well-known attack against x86 processors that enables an attacker to inject and execute code. ARM processors had been considered invulnerable to Ret2Libc. However, since a new attack extending Ret2Libc against ARM processors has been proposed, the buffer overflow of ARM processors has been regarded as a threat [34]. Therefore, attackers may exploit vulnerability, such as a buffer overflow or a malformed network input, in order to cause an embedded end-point device to crash. To make matters worse, attackers are in a somewhat advantageous position in launching a large attack since the number of device vendors is limited and the software installed in the embedded end-point devices is uniform. Furthermore, attackers can reverse engineer code without administrators noticing in order to find vulnerability because, unlike a server application, embedded end-point devices are located at the user side. Therefore, when attackers find a vulnerability in a single device, they can exploit it on many embedded end-point devices.

Besides the problem at the end-point device's side, information leakage could occur at the head-end system side. In order to authenticate each other, a head-end system shares secret information with end-point devices. In particular, when a head-end system is implemented as a server application working in an operating system, there is a risk of an attack on the application by means of widely available exploit tools. Therefore, even if the end-point devices have no security flaw, their data or software needs to be updated once information leakage occurs at the head-end system side.

Risk of compromising cryptosystem

There is a risk that the cryptosystem itself will be compromised since embedded end-point devices are deployed for a long period of time. The latest algorithm may not be used in some cases in smart grids. The newly deployed embedded end-point devices may need to support an old crypto

algorithm in order to communicate with old devices. When old devices are physically replaced with new ones, the embedded end-point devices need to update their algorithm and keys. For example, although Data Encryption Standard (DES) is no longer secure, when considering the long lifetime of a control system it should be recognized that old devices supporting the DES algorithm remain in legacy systems.

Whereas there are many techniques to update a key, a process that implements an algorithm must be updated besides the key in order to update the cryptosystem. If a module implementing the process is illegitimately modified, the key is vulnerable since there is a risk of the modified module stealing the key. Furthermore, since it is impossible to estimate the workload of the future cryptosystem in advance, flexibility is required to support the update function for an unknown cryptosystem. However, it is hard to introduce the function in legacy system.

Difficult to attest the system

Since embedded end-point devices are deployed at sites remote from a head-end system, it is desirable that administrators check whether the embedded end-point devices are modified or not. For example, when a head-end system collects meter data from each smart meter through a network, an administrator is eager to verify that the unmodified software is installed in the smart meter from the head-end system so that the administrator has sufficient confidence that the data is transmitted from legitimate smart meters.

In order to verify the status of the system, trusted boot is proposed [35]. In trusted boot, the expected status of the system is calculated in advance. When booting the system, the status of the system is calculated and stored in dedicated trusted hardware. Since the hardware is tamper-proof, the attacker cannot modify the stored value. Therefore, a verifier can have sufficient confidence that the system is in the expected status by checking whether the stored value is identical to the value calculated previously. Although trusted boot provides a method of verifying the status of a system when booting, the verification cost is high since it requires verifying an entire system, including modules unrelated to protection-required modules. Especially for complicated systems, such as the embedded end-point devices used in smart grids, which have various functions whose protection is not required, modules are updated frequently in order to add or delete services or to fix problems regardless of security reasons. Therefore, it is impracticable for security

administrators to manage all versions of the modules. In order to minimize a verifier's cost, it is desirable to provide a method that enables the verifier to check the status only of the modules whose integrity is required.

Furthermore, a trusted boot has fatal restrictions in that only the state at the time of system boot can be checked. Therefore, even if the system is attacked and functions whose integrity is required are modified after booting the system, the verifier cannot notice the modification since the stored value in the dedicated tamper-proof hardware does not reflect the current status of the system whereas the stored value is identical to the expected value.

In order to solve these problems, it is necessary to provide a mechanism whereby only modules to be protected can be verified and the status to be verified reflects the current status so that verification can be performed at an arbitrary time. Moreover, in order to reduce a verifier's cost, it is desirable that a verifier can attest the system status at an arbitrary time from a remote host through a network. In order to enable remote attestation, it is necessary to protect the verification data by a certain method, since there is a risk of data being modified on a network or an operating system that executes general-purpose processes. However, the technique is as yet unclear.

2.3 Problem of keeping availability

In a legacy system including a server system, surveillance and recovery processes and their execution environment are monolithically configured. In other words, the reliability of surveillance and recovery processes depends on the reliability of their execution environment. This is a reasonable approach in the case of a tiny system. However, since embedded end-point devices used in smart grids require a large system, such an approach becomes infeasible. In the following, we explain the problems concerning the keeping of availability and the reasons why they are difficult to solve.

Difficult to keep a high level of surveillance continuity

End-point devices need to support various network protocols and data formats depending on countries or use cases in smart grids [36][37][38]. In order to minimize the implementation cost of a complicated application program or a minor network protocol on end-point devices, Linux will be used as a software execution environment. In Linux, the surveillance and

recovery processes can be implemented as a user task executed on the operating system or as an interrupt handler in the operating system. When a surveillance target process is implemented as a user task running on the operating system, then support functions in the operating system, such as the "cron" service in Linux, can be used to detect a failure of the user task and to automatically restart the target process. When the surveillance process is implemented as an interrupt handler in the operating system, then a more sophisticated implementation is necessary than for an application program; it is automatically and periodically called by a timer interrupt as long as the operating system works.

Another legacy approach is implementation of a monitoring and detecting mechanism in the operating system. For example, in order to find buffer overflow attacks, an anomaly detection method is proposed where a protection element monitors system call frequencies, and if the frequencies are different from normal behavior, it determines that an attack occurs [39].

However, the fundamental problem of a legacy approach is that there is no way to restart the process if the operating system itself crashes for any reason. Furthermore, the protection mechanism itself could be a target of the attack, and as a result the protection mechanism could be invalidated. Thus, there is a large risk of devices in a smart grid breaking down and the attack may cause an extensive blackout in the worst case.

In order to prevent devices breaking down, a robust method of recovering the system from failure is required in order to keep a high level of availability. Still, some existing hardware devices support a watchdog timer function that detects the status of the operating system and automatically reboots the operating system [40]. Since not all devices support the function and it is difficult to implement complicated functions in the system as discussed below, a new approach is desired. To clarify the conditions, only a software failure including an attack is assumed in this dissertation. A physical fault, such as a hardware failure or loss of power, or a hardware attack, such as physically destroying devices or cutting cables, are beyond the scope of this dissertation.

Difficult for an administrator to detect when an incident occurs

End-point devices are connected with a head-end system through the network to provide demand-response services. When the devices detect an error status, such as a surveillance target process being stopped for an unknown reason, it is desirable that these devices send a report to the

head-end system so that an administrator can realize the situation and use the report to investigate the reason for the failure. However, for the reason described above, there is no way for devices to send a message to the head-end system if the operating system crashes in a system where the network connectivity function is implemented as a user task or it is implemented within the operating system. Even in such a case, it is desirable to provide a method enabling devices to send a message to acknowledge the error situation to the system administrator. In addition to the unexpected failure, attacks on the network connectivity function need to be considered. When an attacker gains full access to the system under control, the attacker may try to disable the network connectivity function in the operating system or totally destroy the system in the worst case. Therefore, it is desirable not simply to provide a method of sending a message but to keep the network connectivity function secure to protect it against the attack even if the operating system is modified or the control of the operating system is taken over.

Besides notification of the error situation to the system administrator, a software update function is also desirable. However, since many existing hardware devices already support a secure firmware update function and its method is highly dependent on each device, it is beyond the scope of this dissertation.

Development and production cost

Cost is an important aspect in evaluating the proposed security architecture. Generally, there are two types of cost: development cost, consisting primarily of personnel expenses, and production cost, which is charged per device. When implementing an end-point device, if the new security architecture requires a complete software rebuild, the architecture will never be commercialized. Thus, it is desirable to reuse existing software assets, such as libraries, middleware and applications, as much as possible in order to minimize the development cost, including the verification cost. In the case of smart grids, the verification cost is large since reliability is strongly required. Besides the development cost, we need to consider the cost per device. One approach to solve the problems described above is to utilize a dedicated hardware security chip. However, since such chips tend to be very expensive, their use may raise production cost per device. Therefore, the use of widely available existing commodity hardware is desirable in order to minimize the production cost.

Chapter 3

Background

In this chapter, we provide background information on the hardware technologies leveraged by the proposed method. Since the target of this dissertation is embedded end-point devices in a smart grid, we present available security functions of hardware devices. ARM processors are widely used in embedded end-point devices because of low power consumption, midrange performance, and the cost benefits. We present security functions of ARM processors. Besides the central processing unit, dedicated security hardware is widely available and some is used in personal computers. That dedicated security hardware can be used as a trust anchor to enable trusted computing. Although it is too expensive to introduce the dedicated security hardware in embedded end-point devices in smart grids, the concept is very useful and applicable to them. Thus, we present the functions of typical dedicated security hardware.

3.1 ARMv7 architecture

ARM processors support different processor modes depending on the architecture version. The ARMv7 architecture on which the proposed system is implemented supports the seven processor modes shown in Table 3.1.

The processor is executed by selectively switching the modes depending on the process. The processor mode is changed either when a program, such as an operating system, calls a dedicated instruction or when software or hardware exception occurs. The seven modes are

categorized as either non-privileged mode or privileged mode by privilege level. In general, an operating system is executed in privileged mode and application programs are executed in unprivileged mode. In privileged mode, execution of all instructions and access to all memory regions are allowed, whereas in unprivileged mode availability of instructions and accessibility of memory regions are restricted.

The ARMv7 processor has 40 registers, consisting of 33 general registers and 7 status registers. These registers are arranged in partially overlapping banks. For example, r13, which is a bank register and usually used for a stack pointer, refers to different physical registers in User mode and Supervisor mode. For non-banked registers, which refer to the same physical register in different modes, an operating system needs to save and restore in working memory when switching from one mode to another mode so that execution can be subsequently resumed from the same point. On the contrary, the operating system does not need to save the context of banked registers. For example, the operating system does not need to save the context of r13 when switching from User mode to Supervisor mode. Therefore, rapid context switching is enabled.

Table 3.1: ARM processor modes and bank registers

Mode	*level*	*description*	*Bank register*	*# of bank registers*
USR	unprivileged	User mode	r8-r14	7
SVC	privileged	Supervisor mode	r13-r14, spsr	3
IRQ	privileged	IRQ mode	r13-r14, spsr	3
FIQ	privileged	FIQ mode	r8-r14, spsr	8
ABT	privileged	Abort mode	r13-r14, spsr	3
UND	privileged	Undefined mode	r13-r14, spsr	3
MON	privileged	Monitor mode	r13-r14, spsr	3

3.2 TrustZone

TrustZone is a hardware security function supported by a part of the ARM processor [41][42]. In addition to unprivileged mode and privileged mode, a TrustZone-enabled ARM processor supports two worlds that are independent of the modes. One is the secure world for the security process and the other is the non-secure world for everything else. Each processor mode shown in Table 3.1 is available in both the secure world and the non-secure world. Figure 3.1 shows the relationship between worlds and modes conceptually. The world in which the processor is executing is indicated by the NS-bit in the Secure Configuration Register (SCR) except when the processor is in monitor mode. When the processor is in monitor mode, it is in the secure world regardless of the value of the NS-bit of SCR. The processor is executed by selectively switching the worlds if necessary. For example, it is assumed that the key calculation process is executed in the secure world and all other general processes, such as storage access or network access are executed in the non-secure world.

The software that manages switching between the secure world and the non-secure world is called the monitor. The monitor is executed in monitor mode. TrustZone provides a dedicated instruction, the Secure Monitor Call (SMC) instruction, to transit between the worlds. As soon as the SMC instruction is called, the processor switches to monitor mode. The monitor saves a context of the program running in the current world

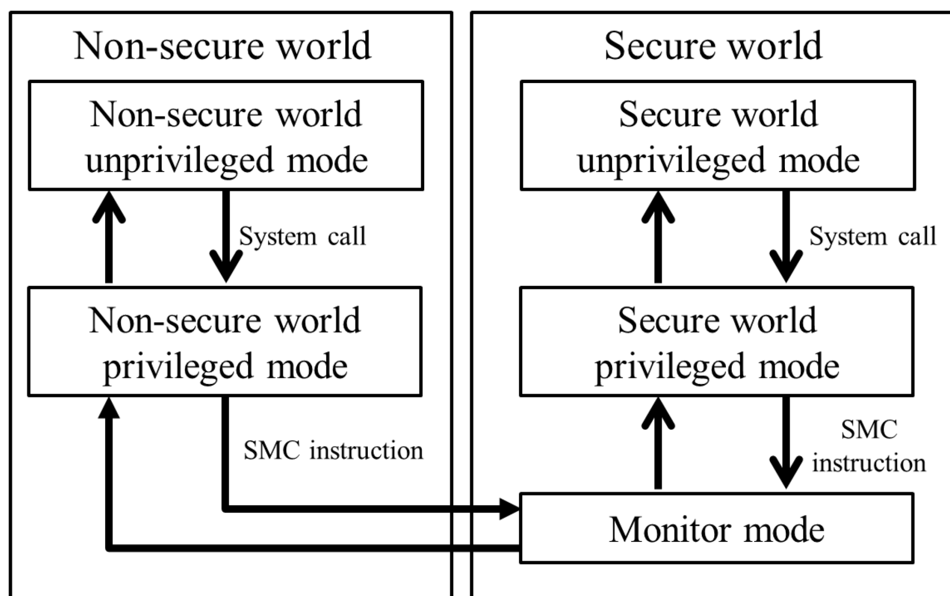

Figure 3.1: Mode and world in ARM.

on the memory and restores a context of the program running in the previous world, then changes the world to set the NS-bit of SCR, and finally executes the program running in the previous world. Besides the SMC instruction, hardware exceptions can be configured to cause the processor to switch to monitor mode.

Note that general registers and Saved Program Status Register (spsr) are not banked between worlds. For example, when r13 in User mode of the secure world is referred and the monitor switches from the secure world to the non-secure world, and then r13 in User mode of the non-secure world is referred, the same physical register is referred. Therefore, the monitor needs to save and restore both bank registers and non-bank registers when it switches worlds.

By using TrustZone-capable hardware, it is possible to make a system where a process running in the secure world can access all system resources, such as memory or peripherals, whereas a process running in the non-secure world can access only a part of system resources. For example, when SCR is used in combination with the TrustZone Address Space Controller (TZASC), access to a particular region of working memory can be restricted for a process running in the non-secure world even if the process runs in privileged mode. When a process running in the non-secure world accesses a memory region that it is configured to be prohibited from being accessed from a process running in the non-secure world, TZASC generates an interrupt signal and it is sent to the processor. As a result, the violation causes an external asynchronous abort. Similar to TZASC, when used in combination with the TrustZone Protection Controller (TZPC), access to a peripheral can be restricted for a process running in the non-secure world. In contrast to TZASC, the access control policy of TZPC can be configured per peripheral, such as DRAM, Timer, or Real-Time Clock (RTC). That is, the configuration of TZPC is performed peripheral by peripheral. There is a correlation between TZASC and TZPC. For example, when configuring a policy such that access to a particular region of DRAM is restricted, the access control of TZPC corresponding to DRAM is set to off and the proper access control policy with the corresponding region is installed on TZASC. TZPC is configured as secure when booting the system. Therefore, for all peripherals whose access controls are valid by TZPC, access by a process running in the non-secure world is prohibited by default. TZASC and TZPC can only be configured by a process running in the secure world, in order to protect those configurations from illegitimate modification.

3.3 Trusted Platform Module (TPM)

Trusted Platform Module (TPM) is security hardware with various crypto functions, such as an encryption function, a random number generation function, a key generation function, and a secure hash calculation function. A non-profit organization, Trusted Computing Group, standardizes its functions and interface, and publishes specifications [43]. TPM is widely used in many consumer appliances, including personal computers. TPM is a hardware module independent of a host CPU. Since the interface for operating TPM is defined in the specification and it is implemented as hardware, it is impossible to eavesdrop or modify data and functions processed inside TPM from outside. That is, software running on a host CPU cannot acquire or change data and functions processed inside TPM. Therefore, TPM can be used as a security anchor for protecting a system from software level attacks [44].

Moreover, TPM has registers called Platform Configuration Register (PCR) inside, which temporarily save the state of the memory at a certain point, in which a program or data are stored. When new data are input in PCR, it concatenates the old value of PCR with the input data, and calculates the hash value of the concatenated value, serving a new value or PCR. It is prohibited to set an arbitrary value for PCR. PCR can be reset only when booting the system and cannot reset the value at arbitrary timing. TPM can be used as a tool to realize trusted boot that verifies the status of the system, ascertaining whether it is in the same status as the system designer intended. First, hash values of programs loaded from the time of booting the system are stored in PCR one by one in order, such as BIOS, OS, middleware, and applications. Then, the final result is compared with the predicted value calculated beforehand. If the values match, it can be regarded that the system's status is legitimate. Otherwise, some programs have been modified and the system is regarded as not being in an expected status. Furthermore, TPM has a function to calculate a signature for the value of PCR with the secret key embedded in TPM by a public key algorithm. TPM can be used to realize remote attestation that verifies the status of the system from a remote host, such as a server connected in the network.

Chapter 4

The proposed method

Security platform for embedded end-point devices in a smart grid

In this chapter, we present a method to achieve a security platform for embedded end-point devices in a smart grid with commodity hardware. We describe a method for a system enabling dynamic loading and updating of a security-sensitive module only with sufficient robustness against tampering. Furthermore, it does not require rebooting the entire system, including an operating system. We also describe a method for a fault-tolerant system enabling the embedded end-point devices to monitor the status of the operating system and to recover even if they stop working owing to unexpected behavior or cyber-attacks, including zero-day attacks. We demonstrate a full implementation of the proposed methods on a commodity embedded processor. We also show experimental results and verify that the proposed methods satisfy the goal.

4.1 A method to keep long-term security

4.1.1 Framework of the virtual security hardware system

The proposed system provides a method for an embedded end-point device to distribute a protection-required module through a network and update it only while keeping its secrets, including secret information, such as keys, and secret processes, such as crypto processes, securely without exchanging hardware. It also minimizes performance degradation of

general-purpose processes. Figure 4.1 shows the entire architecture of the proposed system. It consists of three components: Rich OS, Virtual Hardware, and Monitor. In order to solve the problem described in Chapter 2, it is necessary to include the protection-required module, Virtual Hardware, Monitor, and initialization code in Trusted Computing Base (TCB), which is the basis of trust and provides a secure environment.

- Virtual Hardware: Virtual Hardware consists of the following components: The Re-encryption module, which is a protection-required module containing confidential data and processes, such as keys and re-encryption processes; the Update module, which verifies, decrypts, and updates the Re-encryption module; the TPM module, which processes crypto processes; the Time management module, which supervises the execution time of processes in the secure world; and the Common module, which loads other modules from a memory and executes them, and requests Monitor to context switch to Rich OS. Virtual Hardware runs in privileged mode in the secure world. In the following explanation, we use the Re-encryption module as an example of a module to be updated and protected running in Virtual Hardware.
- Rich OS: An operating system that executes general-purpose processes. It also requests Monitor to execute the protection-required processes. The proposed system supports Linux as Rich OS. Since a device driver in Rich OS has functions to exchange data with Virtual Hardware, and to request Monitor to invoke context switch to Virtual Hardware, applications running on Rich OS can use functions of Virtual Hardware as if they use physical hardware devices, although Virtual Hardware is implemented as software actually.
- Monitor: In order to execute Virtual Hardware and Rich OS exclusively and concurrently, Monitor provides a context switching function between Virtual Hardware and Rich OS based on the request from them by utilizing functions of TrustZone. Moreover, it installs an access control policy to configure TZASC appropriately when booting a system.

The proposed system has five functions: baseline common functions, execution of the protection-required module, dynamic load and update of the protection-required module, protection of the protection-required module, and remote attestation.

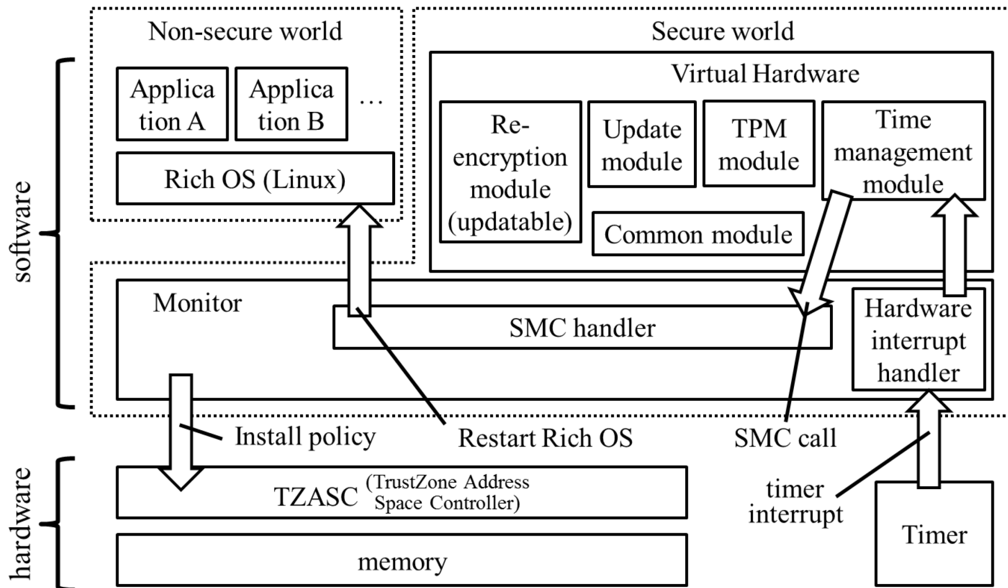

Figure 4.1: Architecture of the virtual security hardware system.

4.1.2 Functions of the virtual security hardware system

Baseline common functions

Monitor serves baseline common functions to execute Rich OS and Virtual Hardware concurrently and exclusively. It has three baseline common functions: configuration of access control policy, data transmission, and context switching between worlds.

1) Configuration of access control policy

   When booting the system, Monitor sets up a policy of memory access control as one of the initialization processes using TZASC of TrustZone. Monitor divides a memory area into three regions: a shared region used for the data exchange between Virtual Hardware and Rich OS, a non-secure region, in which Rich OS and applications running on Rich OS are used, and a secure region for Virtual Hardware. The shared region and the non-secure region are configured so that both Virtual Hardware and Rich OS can access data in them. Although an access control policy installed in the shared region and the non-secure region is the same, we use different names for clarification. On the other hand, the secure region is set up so that only Virtual Hardware can access the data on it.

2) Data transmission

Monitor provides a data transmission function to exchange data between Virtual Hardware and Rich OS. Since the more complicated the system becomes, the larger the risk of including vulnerability becomes, Virtual Hardware does not include device drivers to control devices in order to minimize the risk. Therefore, Virtual Hardware cannot directly access hardware devices. In order for Virtual Hardware to access a disk drive, or a network device, Virtual Hardware indirectly uses device drivers in Rich OS to exchange data with Rich OS. First, Rich OS controls hardware devices, and reads data from the hardware devices. Then, Rich OS writes data on the memory area configured as the shared region. Since the device driver maps the shared region into a virtual memory address region on Rich OS, programs running on Rich OS can read and write data in the same manner as they access the normal memory area. The device driver requests Monitor to invoke context switch to Virtual Hardware. Virtual Hardware reads data from the memory area configured as the shared region and processes it in the secure world. When Virtual Hardware sends data to Rich OS, Virtual Hardware writes data on the shared region. Then, Virtual Hardware requests Monitor to invoke context switch to Rich OS. Finally, Rich OS reads data from the shared region. In this manner, data are exchanged between Virtual Hardware and Rich OS.

3) Context switching between worlds

Monitor provides a context-switching function between worlds. In addition to changing the state of the processor by configuring NS bit, Monitor also saves and restores context of Rich OS and Virtual Hardware. Since Rich OS and Virtual Hardware use the same register, either one could destroy the context of another one without saving the context when context is switched. Therefore, Monitor needs to save the content of registers belonging to the current world on working memory, and restores the content of registers belonging to the transition destination world from the working memory while changing the value of NS bit. Moreover, we implemented a device driver in Rich OS in order to execute SMC instruction to transit the status of the processor from the non-secure world to monitor mode.

Execution of the protection-required module

We explain a re-encryption process as an example of a process to be updated and protection-required running in Virtual Hardware.

Embedded end-point devices, such as smart meters, need to support multiple network interfaces with different communication protocols. For example, they support ZigBee to communicate with proximity devices, such as sensor devices, in addition to TCP/IP to communicate with devices outside the home, such as head-end systems. Since the cryptographic algorithm and data length used by each network interface are different depending on protocols, it is necessary to re-encrypt data when translating protocols. Therefore, protection of the re-encryption process is required since data become plaintext when re-encrypting data. Moreover, it is desirable to execute Rich OS concurrency in order to process network communication functions simultaneously, since a network protocol requires responding in a certain period time, or otherwise it is disconnected.

Figure 4.2 depicts how the re-encryption process works in the proposed system. First, Rich OS loads encrypted data from sensor devices, a network, or disks and writes data on the shared region of a memory area (process (1)), and executes SMC instruction to request Monitor to invoke context switch (process (2)). When SMC instruction is called, a processor calls Monitor. Monitor context switches from Rich OS to Virtual Hardware (process (3)). The Common module in Virtual Hardware is initiated and calls the Re-encryption module that reads data from the shared region (process (4)). After executing the re-encryption process in the secure world (process (5)), the Re-encryption module writes re-encrypted data on the shared region (process (6)). It calls back to the Common module. The Common module requests Monitor to invoke context switch to Rich OS (process (7)). The processor calls Monitor again, and Monitor context switches from Virtual Hardware to Rich OS (process (8)). Finally, Rich OS reads the re-encrypted data from the shared region (process (9)). A series of these processes is repeated until the re-encryption of the data is completed.

Rich OS is suspended after Rich OS requests Monitor to invoke context switch until Monitor restores the context of Rich OS. Since the time of the re-encryption process becomes long when the data size for re-encryption is large, suspension time of Rich OS also becomes long. Since acceptable length of time for suspension of Rich OS to be allowed depends on applications, we cannot specify the target value. However, it is possible to shorten the time required for one re-encryption process

including context switch by dividing data to be re-encrypted into small chunks, thus preventing Rich OS from being suspended for a long time. In general, the block size of data to be encrypted depends on cryptographic algorithms. For example, in the case of AES with 128 bit key length, block size of data must be 128 bits. It can encrypt only by the multiple whose data size is 128 bits, and the minimum unit of division is 128 bits. Therefore, the minimum suspension time of Rich OS is the time necessary to encrypt 128 bits data. For clarification, it is possible to switch back to Rich OS in the middle of the encryption process. Furthermore, it is possible to process data of less than 128 bits by padding data within the Re-encryption module, the data size that Virtual Hardware accepts to handle does not depend on a cryptographic algorithm.

In this manner, even if general-purpose processes, such as the operating system, which is not directly related to the re-encryption process, have vulnerability and attackers completely take control of the processes by misusing the vulnerability, it is possible to prevent attacks, such as eavesdropping key or plaintext data, or modifying the protection-required process flow by illegitimate debug.

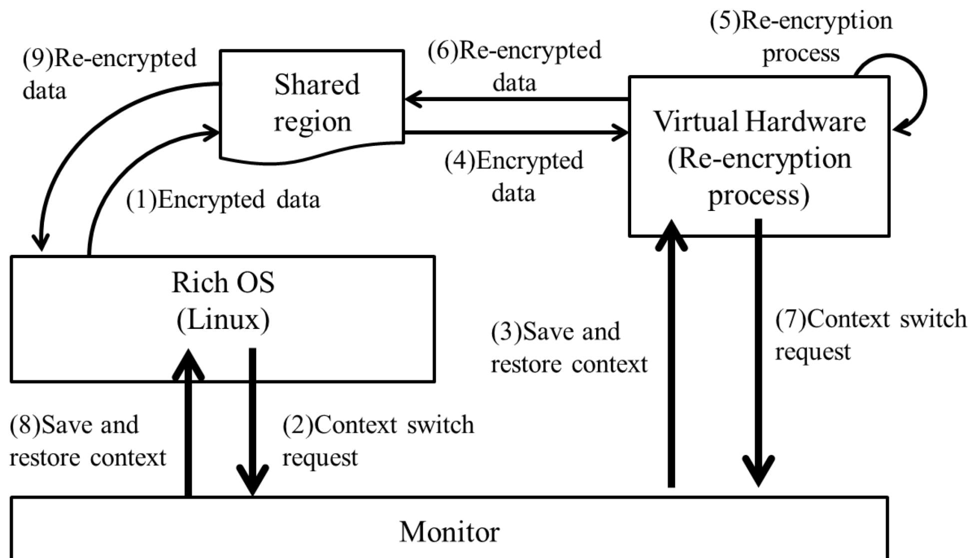

Figure 4.2: Execution flow of the Re-encryption module.

Dynamic load and update of the protection-required module

The proposed method provides a function for dynamic loading and updating the protection-required module only, such as the Re-encryption module, without requiring updating of the whole system. As described above, Virtual Hardware itself does not have the device driver that accesses a disk device. Therefore, in order to update the Re-encryption module in Virtual Hardware, Virtual Hardware indirectly uses the device driver in Rich OS. First, Rich OS loads the binary object file of a new protection-required module, the Re-encryption module in this case, from a disk, and writes it on the shared region of a memory area. Monitor assigns a memory area in which the Re-encryption module will be placed in the secure region in an initialization process. The Update module in Virtual Hardware has the information about the memory map of this secure region, and overwrites the binary object of the new Re-encryption module written in the shared region to the secure region for which an old Re-encryption module is arranged. The Update module checks that the binary object of the size of the new Re-encryption module is smaller than the size assigned to the Re-encryption module. If it fits the size of the area assigned to the Re-encryption modules, the Update module can overwrite the new Re-encryption module even if the size of the binary object of the new Re-encryption module is larger than that of the old Re-encryption module. However, if the size exceeds it, the Update module quits the update process, and requests Monitor to invoke context switch to Rich OS with an error code. When the Common module in Virtual Hardware calls the Re-encryption module, it executes the first instruction placed on the first address of the memory area assigned to the Re-encryption module. Therefore, if it is configured so that the Re-encryption module starts its execution from the first address, the Common module can call the new updated Re-encryption module.

Moreover, the proposed method provides a function to update the protection-required module only when it is successfully verified since there is a risk that the protection-required module is illegitimately modified on a network or on Rich OS. First, a secret key (a secret key for a signature) to sign the binary object of the protection-required module is assigned to a developer of the protection-required module. A public key corresponding to the secret key for a signature is embedded in the TPM module in Virtual Hardware. The developer of the protection-required module calculates the hash value of the binary object, signs the hash value with the secret key for a signature, attaches it to the binary object, and distributes it.

When the Update module reads the binary object of the Re-encryption module from the shared region, it calculates the hash value of the Re-encryption module, verifies the signature with the public key that the TPM module in Virtual Hardware manages, and loads it only when the verification succeeds. In this manner, it is possible to prevent Virtual Hardware from installing the illegitimately modified Re-encryption module. For clarification, the signature verification process is executed in the secure world. It is recommended to choose the latest cryptographic algorithm and longer key for the signature process, which is available and defined as a standard at the time.

Therefore, attackers can modify neither verification process nor data, such as the key used in the verification process, even if general-purpose processes, such as Rich OS, are modified. Moreover, it is possible to update the protection-required module independently with Rich OS.

Protection of the protection-required module

The protection-required module could contain secret data. For example, the Re-encryption module contains keys for decryption and re-encryption. However, since the binary object of the protection-required module is managed as a file by Rich OS, there is a risk of those keys being exposed by analysis with tools, such as a disassembler or a debugger. To solve this problem, the proposed method provides a function to protect the secrecy of the protection-required module by encryption. First, a key (an encryption key) for encrypting the protection-required module is assigned to a developer of the protection-required module. The developer encrypts the protection-required module with the encryption key and distributes the encrypted protection-required module. Rich OS downloads the encrypted binary object of the protection-required module via a network, and writes it on the shared region. The Update module in Virtual Hardware copies the encrypted binary object of the protection-required module to the secure region, decrypts it with a key (a decryption key) embedded in the TPM module in Virtual Hardware. Except for the secure region, since the protection-required module does not become a plaintext, it is possible to keep the keys and processes in the Re-encryption module secret. It is recommended to choose the latest cryptographic algorithm and longer key for the decryption process, which is available and defined as a standard at the time.

Remote attestation

In the case of trusted boot by using TPM, a verifier checks whether the expected modules are loaded on a system as follows: calculate hash values of all the modules loaded from the time of booting, store the hash value in PCR, verify a signature for the hash value, which TPM signs to the value of PCR with a secret key, check the result of the verification.

On the other hand, in the proposed method, hash value of the binary object of the plaintext protection-required module is calculated by the Update module in Virtual Hardware when the Update module copies the binary object of the protection-required module to the secure region. And the value of PCR in the TPM module is reset and the newly calculated hash value is stored in PCR. Since the protection-required module is expected to be updated at arbitrary timing, PCR must be updatable at arbitrary timing. However, existing physical TPM is not allowed to reset PCR at arbitrary timing. Therefore, it is insufficient just to transplant the function of the existing TPM as it is. To solve this problem, we introduced the new restriction that only the Update module can reset PCR at any time and removed the restriction that reset of PCR in the TPM module is allowed only when booting a system, thus enabling reset of the value of PCR at arbitrary timing. As a result, a hash value can be stored at the time of update of the protection-required module, thus preventing illegitimate resetting of PCR.

Furthermore, the proposed method provides a function of verifying the value of PCR from outside the system. The Update module requests the TPM module for the signature of PCR. First, the TPM module generates a signature for PCR with a secret key stored in itself by a public key algorithm, and gives it back to the Update module. The Update module writes the signature value on the shared region. Finally, Rich OS gets the signature from the shared region. In this manner, a verifier can check that the intended protection-required module is loaded from outside the system.

4.1.3 Process flow

In this section, we describe the process flows of update and remote attestation.

Process flow of updating the protection-required module

First, we describe the process flow of development of the protection-required module. A developer develops a protection-required module, and

generates a binary object. A hash value of the binary file is calculated and the binary object is encrypted with a key (an encryption key) for encrypting the protection-required module. Furthermore, the hash value of the encrypted binary object is calculated, a signature is generated with the secret key for a signature, and the signature is attached to the binary object.

Next, we describe the update process of the protection-required module. Figure 4.3 depicts the process. When booting the system, a memory area where the Re-encryption module is placed has been assigned to the secure region (process (1)) by Monitor. Rich OS downloads an encrypted and signed binary object via a network, and saves it on the disk (process (2)). This preprocessing is followed by processes to update the protection-required process. Rich OS reads the encrypted and signed binary object from a disk, and writes on the shared region (process (3)). Rich OS executes SMC instruction and a processor transits to monitor mode. Monitor context switches to Virtual Hardware (process (4, 5)). The Update module in Virtual Hardware gets a decryption key for decrypting the encrypted binary object and a public key for a signature for verifying a signature attached to the binary object from the TPM module (process (6)). The Update module decrypts and calculates a hash value of the encrypted and signed binary object. The size of a binary object could be larger than the block size of decryption or hash algorithm. Therefore, rather than reading the whole encrypted and signed object at one time, it reads a part of the encrypted and signed object from the shared region, it copies it to a part of the secure region that the Update module manages (process (7)), and it decrypts and calculates a hash value (process (8)). It appends a part of the binary object, which becomes plaintext, to the temporary area assigned in the secure region that the Update module manages (process (9)). The hash calculation and decryption continue until the entire binary object is processed. It verifies a signature attached to the binary object with the hash value to the whole encrypted binary object with the public key retrieved from the TPM module (process (10)). If the verification of the signature fails, it deletes the plaintext binary object copied to the secure region, and requests Monitor to invoke context switch from Virtual Hardware to Rich OS with an error code. When verification succeeds, the Update module reads the plaintext binary object placed in the temporary area assigned in the secure region, calculates a hash value for the plaintext binary object, and copies it to the area where the old Re-encryption module is placed (process (11, 12)). This process continues until the entire plaintext binary object is copied. The Update module resets the value of PCR in the TPM module, and it stores the hash value to the plaintext

object file in PCR (process (13)).

Embedded end-point devices execute various processes to achieve various functions. When the size of the protection-required module is large, the decryption and verification processes may take a long time. Since those processes are suspended while executing the update process, a problem arises. For example, in the case of the smart meter that is an application of the proposed method, it is necessary to execute various tasks: transmitting home area power consumption data to a head-end system via a network, or performing home area electric power adjustment based on the request from a head-end system, for example in order to process a demand-response service. The longer the time taken by the verification processing of the protection object module, the longer the suspension time of Rich OS becomes. Therefore, it may miss receiving commands from a head-end system or disconnect from the head-end system, thereby disabling the demand-response service. Since the performance depends on applications, management system of the entire system, implementation, and hardware performance, it is difficult to determine the specific target performance value. As an application example, it is reported that the acceptable delay to the data transmitted from a head-end system is 50-300[ms] in a smart grid [45]. Although it is also possible to shorten the delay by introducing a high-performance

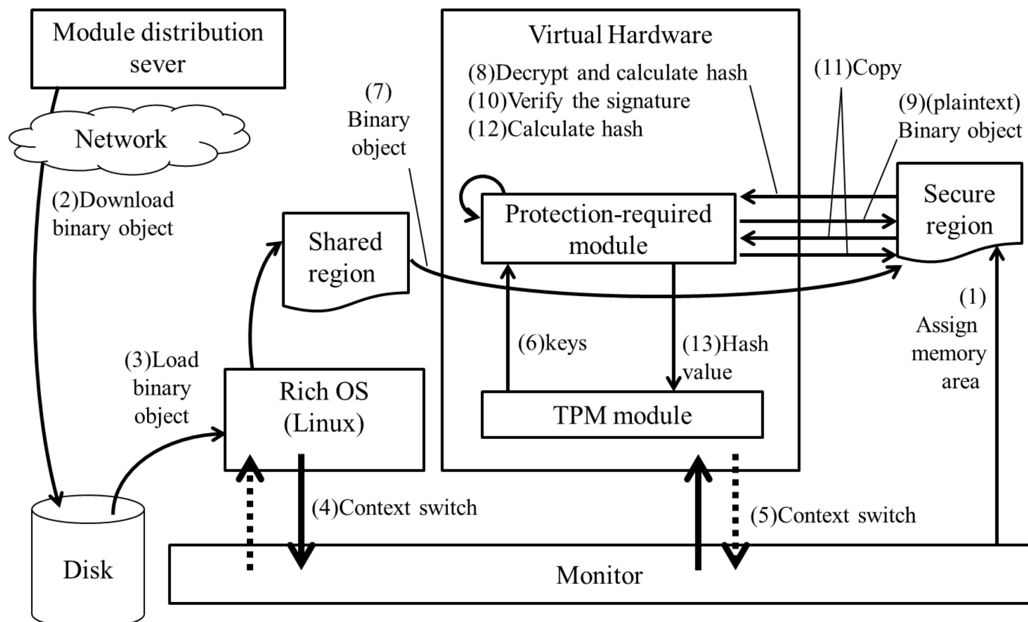

Figure 4.3: Execution flow of the update process of the protection-required module.

processor, it will increase cost. To avoid a long suspension time of Rich OS, the Time management module in Virtual Hardware measures the execution time of the updating process using hardware timer interrupt. When a particular time passes, it requests Monitor to invoke context switch, resulting in compulsorily returning to Rich OS. In this manner, it mitigates the effect of lengthy suspension of Rich OS.

Furthermore, the data used for decrypting the binary object, calculating the hash value, and the plaintext binary object are stored in the secure region that Rich OS cannot access. Therefore, even if attackers illegitimately modify Rich OS, the attackers cannot get or modify intermediate value of hash, decryption, or the plaintext binary object. Moreover, even if Rich OS modifies an encrypted and signed binary object loaded to the shared region, the attackers can neither execute an illegitimate module, nor overwrite the existing protection-required module in the modified module, since the process copies the plaintext binary object to the memory area where the old protection-required module is placed only when the verification succeeds.

Process flow of remote attestation

Figure 4.4 depicts the execution flow of remote attestation. Premising, it is assumed that a verification device has a public key corresponding to the secret key for a signature stored in the TPM module. First, Rich OS receives a command that indicates a request for remote attestation from the verification device through a network (process (1)). Rich OS executes SMC instruction to invoke context switch to Virtual Hardware (process (2, 3)). The Update module in Virtual Hardware requests the TPM module for the signature for the PCR value (process (4)). The TPM module generates the signature for the value of the present PCR with the secret key for a signature stored in itself, and transmits to the Update module (process (5)). The Update module writes the signature received from the TPM module on the shared region (process (6)), and requests Monitor to invoke context switch to Rich OS. Rich OS reads the signature for PCR from the shared region, and transmits to the verification device through the network (process (7)). The verification device verifies the signature with the public key (process (8)). Since the value of PCR is a hash value of the protection-required module that is currently executing, a verifier can verify from a remote location that the expected protection-required module is executed at arbitrary timing.

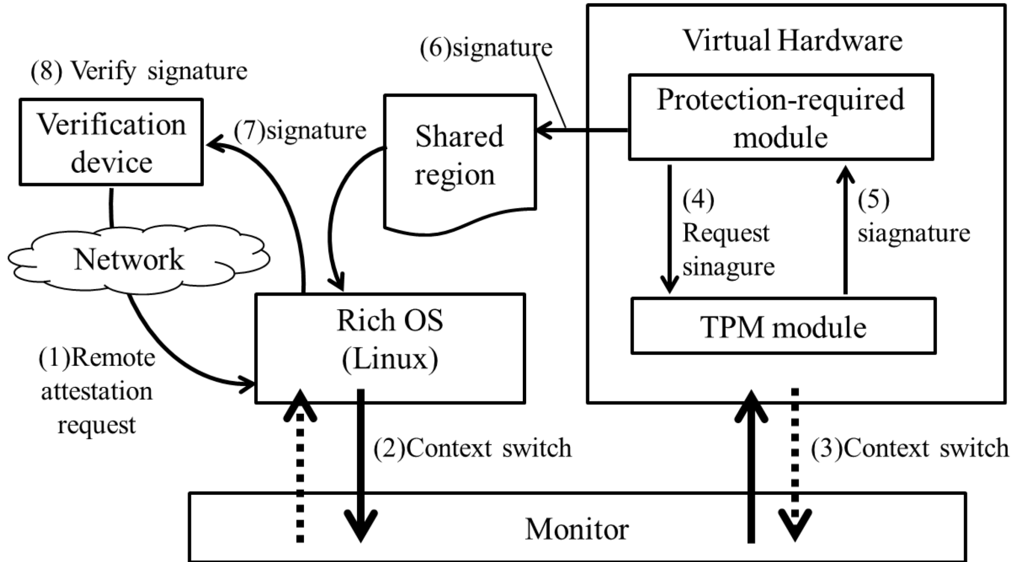

Figure 4.4: Execution flow of remote attestation.

4.1.4 Prototype implementation

We used ARM C/C++ Compiler 5.01 to build Monitor and Virtual Hardware. We used gcc 4.4.1 to build Rich OS. Linux 3.6.1 is supported as Rich OS. We chose Motherboard Express uATX with the CoreTile Express A9x4 processor that supports TrustZone as an execution environment. As for the memory map, from 0x60000000 to 0xA0000000 is assigned to DRAM. Table 4.1 shows the memory map of main memory.

Table 4.1: Memory map of the virtual security hardware system

Data	***Start address***	***Size***	***Region***
Vector table + Initialization code (code/data)	0x60000000	0x00008000	Secure region
Rich OS(Linux) (code/data)	0x60008000	0x2FFF8000	Non-secure region
Monitor + Update module + TPM module + Common module (code/data)	0x90000000	0x01000000	Secure region
Re-encryption module (code/data)	0x91000000	0x0E200000	
Shared memory	0x9F200000	0x00E00000	Shared region

To demonstrate feasibility of the proposed method, we implement the following components: a device driver in Rich OS, the Re-encryption module in Virtual Hardware, the TPM module, and the Update module.

- Device driver in Rich OS: execute SMC instruction in order to transit to monitor mode

- Re-encryption module in Virtual Hardware: a decryption process with XOR and an encryption process with 128 bit AES in ECB mode. Padding process is not implemented

- TPM module: a signature generation process and a signature verification process with 1024 bit RSA algorithm.

- Update module: A process to update the Re-encryption module. It takes out the decryption key stored in the TPM module, and decrypts the Re-encryption module with 128 bit AES in CBC mode. It overwrites the binary object of the new Re-encryption module in the memory area where the old Re-encryption module is placed. Furthermore, it verifies the signature of the new Re-encryption module.

- Time management module: A process to monitor execution time

of processes running in Virtual Hardware. We use two timers for timer interrupt. One is assigned to IRQ and the interrupt handler of Linux is called when IRQ occurs. Since timer interrupt is assigned to IRQ in the current Linux implementation, the modification of Linux source code is unnecessary. The second timer is assigned to FIQ and the vector table is set up to jump to the Time management module when FIQ occurs.

It is necessary to give and compile the same address map as that of the original Re-encryption module, when building the binary object of the Re-encryption module. The content of the binary object of the Re-encryption module is the same as that of the binary image placed in a memory. Therefore, the Update module loads the Re-encryption module to a memory, without distinguishing code region and data region. The Update module overwrites the binary object of the new Re-encryption module to the memory area assigned for the old Re-encryption module only when the signature verification process succeeds. Since it does not have the function to change the size of the memory area dynamically, if the size of the decrypted binary object of the new Re-encryption module fits the predefined size of the memory area assigned for the protection-required module, it is able to overwrite it. However, if the size exceeds the memory area, it cannot update the new Re-encryption module.

It is necessary for the Re-encryption module and Rich OS to agree on the size of the data they transmit and receive through the shared region before executing the re-encryption process so that the data size does not exceed the size of the shared region. When transmitting data to the Re-encryption module from Rich OS, Rich OS requests the invoking of context switch after completion of data transmission, whose size is predefined. Then, the Re-encryption module reads data of the size agreed with Rich OS.

When the Time management module determines that the Update module continues to execute beyond the predefined period of time, it requests the Common module to invoke context switch to Rich OS. Therefore, it is possible to prevent Rich OS suspending for a long time.

While Rich OS is executing its processes after context switch to Rich OS caused by the time out, the other processes may invoke another context switch to Virtual Machine to execute the re-encryption process whereas the Update module has not yet completed the update process of the protection-required module. To prevent this invocation, we implement an exclusive control function in the device driver in Rich OS. While the

Update module is updating the new Re-encryption module, it does not accept a request of context switch from a process for the purpose of re-encryption, blocks the execution of the process and switch, and executes another process. The device driver invokes context switch after a predefined period of time in order to resume the execution of the Update module. Once the Update module has completed updating the new Re-encryption module, it sets the status of the blocked task to executable, and invokes context switch to Virtual Hardware.

We build Monitor and Virtual Hardware as the same binary object. The implementation does not support multi-core.

4.1.5 Evaluation

In this section, we describe the result of the evaluation. We evaluate functions for the design items, and show performance results. We also describe the result of evaluation compared with the case where the protection-required processes are implemented by hardware.

Evaluation environment

As well as the implementation environment, we used Motherboard Express uATX that contains the ARM Cortex-A9x4 processor running at 400 MHz as an experimental environment. Level 1 instruction cache, level 1 data cache, and level 2 cache are 32[KB], 32[KB], and 512[KB], respectively. It contains 1[GB] DRAM as the main memory and we assigned the same memory map as that previously described. The size of the shared region, the non-secure region, and the secure region are 14[MB], 768[MB], and 242[MB], respectively.

Functional analysis

1) Secure update

   A protection-required module can be securely updated by the Update module. Since the Update module is executed in the secure world, even if attackers can successfully take control of Rich OS, they can neither skip the signature verification process, nor modify Virtual Hardware, and therefore the update is performed securely. Moreover, it is possible to prevent updating to an illegitimate protection-required module by verifying the signature given to the protection-required

module when updating it. In order to verify the correctness of the implementation of the update function described in the previous section, we checked whether the protection-required module could be modified from Rich OS, or not. As a result, we confirmed that the protection-required module is loaded, updated, and executed in Virtual Hardware. We also confirmed that modification of the protection-required module running in the secure world fails if Rich OS is modified intentionally and an attempt is made to modify the protection-required module from Rich OS. Furthermore, we modified the protection-required module and the signature portion on a memory area, respectively, after loading the protection-required module to the shared region. And we checked whether the modified protection-required code is updated. As a result, we confirmed that the signature verification process fails with an error code in each case and the protection-required module was not updated and the old protection-required module correctly remains in the secure region.

The Re-encryption module, which is an example of the protection-required module, is distributed via a network from the head-end system in the encrypted manner, the decryption process is executed in the secure world when updating it, and the protection-required module is placed in the secure region. In this manner, the protection-required module is encrypted on the communication path and it is protected by access control when executed. Therefore, there is no risk of the secret data contained in the protection-required module being acquired by illegitimately modified Rich OS either at the time of distribution or execution. Furthermore, since it is necessary to update neither Rich OS nor Virtual Hardware, and the proposed method requires the protection-required module only to be updated on working memory, it does not require rebooting of the entire system at the time of update, and thus it can minimize downtime. Since the update process can be divided into small processes in the case where the update process takes much time, the proposed method prevents lengthy suspension of Rich OS.

Besides, as described in the previous section, it is possible to introduce a cryptographic algorithm and a key length that are different both from those of the protection-required module that is the target of update and those being used for protection of the protection-required module. In particular, the latest available cryptographic algorithm and the key length are not necessarily generally adopted by that time due to the restriction of performance, implementation cost, or hardware

cost. In fact, in the late 2000s, Triple DES was still considered to be an available cryptographic algorithm with sufficient robustness for industrial systems [15], although AES was already standardized and is used in information and communication systems. In our proposed method, the system can continue to be employed for a long period of time without having to replace hardware to introduce the newer encryption algorithm and longer key length in order to protect the protection-required module, than those implemented in the protection-required module.

Besides, the proposed system enables provision of a service available only for specific devices. For example, Kanda [46] and Forsberg [47] provide a method of managing a key to protect the communication channel separately for each application, such as collecting data measured by smart meters, or demand-response service. It enables addition and deletion of services that service providers provide to devices by preparing a different key for each application, and distributing the protection-required module encrypted with a different key. Since services are expected to be added and deleted frequently during the lifetime of devices, our proposed method can minimize the cost compared with the legacy case where devices must be updated whenever a key is updated. Therefore, even if long processing time is required to update the protection-required module to introduce the latest cryptographic algorithm and long key length used for protection of the protection-required object module, the side effect on the whole performance can be mitigated.

2) Module verification

The remote attestation function serves as a method to verify the protection-required module. It is possible to verify the integrity of the entire system including an operating system in trusted boot. However, in trusted boot, the verification cost is high since a verifier needs to update the expected value whenever it updates a module, even if a general-purpose module not directly related to security functions, such as a device driver, is updated, causing the verification value to change. It is ideal that we can provide a method whereby a verifier checks only the changed portion that is the base of the trust anchor, namely, the protection-required module. In our proposed method, since only the protection-required module is the target of verification, even if a system administrator updates Rich OS, which is unrelated to

the protection-required process, it does not need to update an expected value. Therefore, cost of the verifier can be greatly reduced. Moreover, trusted boot has a disadvantage in that it is impossible to detect system modification if it occurs after boot, since it calculates hash value of the module at the time of boot, and a verification value is the same value as the expected value, even if the value indicating the current modified status and the original status when booting are different. On the other hand, our proposed method calculates hash value of the protection-required module and uses the hash value as the target verification value when loading and executing it. Therefore, our proposed method shortens the time between execution and verification, greatly mitigating the risk that the verification value becomes different from the value indicating the current status of the system. Since it is valuable to use trusted boot for verifying the entire system including Rich OS, it is also possible to use our proposed method and legacy trusted boot complementarily for different purposes. For example, in order to make the system more robust, it is possible to use our proposed method for the verification of the protection-required module in combination with trusted boot for verification of the entire system.

In order to verify the correctness of the implementation of the remote attestation described in the previous section, we implemented a small application program that requests a signature of the protection-required module from a remote location through a network and Rich OS. Based on the request, the Update module in Virtual Hardware returns the signature to Rich OS. The application program verifies the signature with the hash value of the protection-required module and a public key corresponding to the secret key in the TPM module. As a result, we confirmed that the verification succeeds. Moreover, we also checked that the hash value is updated and it corresponds to the new protection-required module. We confirmed that Rich OS returns the same hash value of the protection module in Virtual Hardware, even if Rich OS is updated. Thus, even when modules except the protection-required module were updated, it is confirmed that the verification value did not change. Since Rich OS does not have a secret key to calculate a signature, it is impossible for Rich OS to generate a legitimate signature. However, in the proposed method, it is possible to discard the signature without transmitting it to a verifier, or to modify the signature on a network. In this case, even if the Update module writes the signature to the shared region,

the verification fails. In future work, we intend to consider this type of DoS attack. When the verification fails and an administrator receives the error messages, since the administrator can recognize that the device is in error status, the purpose of the verification is achieved. Similarly, it is possible for attackers to call Virtual Hardware unnecessarily and repeatedly by modified Rich OS in order to disrupt the execution of the protection-required module. In future work, we also intend to consider such DoS attacks.

3) Module data protection

It is possible to protect the data of the protection-required module by the execution function of the protection object module. Only the encrypted Re-encryption module and encrypted data are written on the shared region that Rich OS can access. The binary object of Virtual Hardware, and the stack and heap area for storing intermediate data are reserved in the secure region. Therefore, even if Rich OS is illegitimately modified and it is under attackers' control, Rich OS cannot access the data placed on these secure regions, and there is no risk of Rich OS acquiring or modifying the data including the intermediate data generated in the crypto process, which Virtual Hardware handles. The proposed method cannot prevent attacks, such as disrupting a system, replacing the re-encrypted data with invalid data, or blocking data transmission of the re-encrypted data by modifying Rich OS. In future work, we intend to consider those DoS attacks.

4) Module size

It is reported that defect density, which is a rate computed by dividing the number of defects found by the size of the code base in 1000 lines of code, is 0.35 to 0.75, depending on the size of codebase [48]. Although it counts all bugs including vulnerability, it indicates that the greater the code size, the greater is the risk that software includes vulnerability caused by implementation errors. Although ideally the size of the entire system should be minimized and reviewed carefully, since developers sometimes need to use the source code developed by other organizations, such as an open source, organizations, to implement general-purpose functions, such as an operating system, it is difficult to reduce implementation faults by the developers' efforts

in many cases. Therefore, it is desirable to build a system in which the size of the source code of only the portion that developers review carefully is minimized. In our proposed method, the size of Virtual Hardware is sufficiently small compared with the entire system. The protection-required module, Virtual Hardware, and Monitor are TCB in the proposed method. The source code of Virtual Hardware and the Re-encryption module are 6300 lines in total, and the code and data are 12[KB] and 5[KB], respectively. Among these, the source code of the Re-encryption module is 1200 lines, and the code and data are 2[KB] and 200[B], respectively. Moreover, the source code of Monitor is 900 lines, and the code and data are 5[KB] and 19[KB], respectively. On the other hand, since the source code of Linux 3.6.1 is more than 15 million lines, TCB is sufficiently small to be practicable for building a module from which the implementation faults are removed by source code review.

Performance analysis

It is necessary to minimize the overhead at the time of executing crypto processes in a secure environment. If performance degrades, a high-performance processor is necessary but raises the cost. Therefore, it is desirable to minimize the performance degradation compared with the case where the protection-required module is executed in a general environment that is not secure. In order to check the extent to which the proposed method affects the performance degradation, we measured the execution performance of the re-encryption process. Besides minimizing performance degradation, it is ideal to minimize suspension time of Rich OS. In order to check whether it is possible to keep the suspension time of Rich OS within an acceptable range, we measured the degradation of network latency by using a ping program. The following shows the results of the experiments.

1) Performance of re-encryption process

In order to evaluate the execution performance of the re-encryption process of the Re-encryption module, we measured two cases: the case where the re-encryption process is executed only in Rich OS, and the case where Rich OS and Virtual Hardware communicate with one another and the Re-encryption module in Virtual Hardware executes

the re-encryption process. In this evaluation, we prepared the random data of 10[MB] and placed it on the shared memory region beforehand, divided the data into several blocks whose sizes are 16[B], 32[B], 64[B], 256[B], 1[KB], 4[KB], and 16[KB], executed the Re-encryption module in Virtual Hardware that retrieves data from the shared region and executes the re-encryption process with context switch between the worlds, and measured the performance (throughput) of the re-encryption process in each block size. When the block size is 16[B], for example, context switch occurs 1310720 times (2×10×1024×1024/16). Since context switch occurs twice in one round-trip, from Rich OS to Virtual Hardware and from Virtual Hardware to Rich OS, we need to double 10×1024×1024/16. Figure 4.5 shows the performance results of the two cases: the case where the re-encryption process is executed only in Rich OS, and the case where the Re-encryption module executes it by exchanging data between Rich OS and the Re-encryption module. Figure 4.6 shows the performance ratio of those two cases.

In Figure 4.5 and Figure 4.6, block size indicates a size of data Rich OS sends to Virtual Hardware via the shared region in one transition. When block size is small, the overhead of context switch is large and performance degrades about 62% in the case where block size is 16[B] compared with the case where only Rich OS executes the re-encryption process. On the other hand, when the block size is 256[B], 1[KB], and 16[KB], the performance degrades 8.5%, 1.2%, and 0.1%, respectively, indicating that the performance degradation is negligible compared with the case where only Rich OS executes the re-encryption process. Although the amount of calculation depends on the crypto algorithm, it is considered that the overhead can be kept within an acceptable range by choosing a suitable block size depending on applications.

On the other hand, when block size becomes large, it is necessary to prepare a large shared region to exchange data between Rich OS and Virtual Hardware, and the suspension time of Rich OS becomes long. Therefore, it is not necessarily true that large block size is better. There is a tradeoff between block size and other parameters, such as a crypto algorithm, the suspension time of Rich OS, performance degradation, and memory size for the shared region, and thus it is necessary to choose suitable block size depending on those parameters.

For clarification, since there is no difference in performance

between the secure world and the non-secure world in TrustZone, there is no difference in performance between the case where the re-encryption process is executed in the secure world only and the case where it is executed in the non-secure world only.

2) Performance of network latency

When Rich OS is suspended for a long time, response time, such as network access and Graphical User Interface (GUI) operation, becomes slow. In order to evaluate the extent to which the suspension time of Rich OS affects response time of applications running on Rich OS, we measured the network latency (Round Trip Time (RTT)) using the ping program, which is a network utility program that measures the response time of network access. A host that sends a ping command is on the same local network as the proposed method and they are connected by 1[Gbps] wired network. We measured the case where Rich OS is idle, the case where the re-encryption process is executed by Rich OS only, and the case where the re-encryption process is executed by the Re-encryption module cooperating with Virtual Hardware and Rich OS with data divided into several blocks whose sizes are 16[B], 32[B], 64[B], 256[B], 1[KB], 4[KB], and 16[KB].

Figure 4.7 shows the result of the experiment. As shown in Figure 4.7, RTT value at the time of idle state and at the time of processing Rich OS only is 0.43[ms] and 0.44[ms], and thus there is no difference between them. When the Re-encryption module executes the re-encryption process, cooperating with Virtual Hardware and Rich OS, with data divided into several blocks whose sizes are 16[B] and 256[B], the RTT values are 0.45[ms] and 0.48[ms], indicating there is no difference from the case where it is executed by Rich OS only. However, the larger the block size, the longer the suspension time of Rich OS becomes. In particular, when the block size is 1[KB] and 4[KB], the RTT values are 0.75[ms] and 1.3[ms]. When block size is 1[KB], the throughput is 3.38[MB/s] based on Figure 4.7 and context switch is executed 1000 times. Therefore, the processing time per block, which is equivalent to the suspension time of Rich OS per context switch is (1000/3.38) / 1000 = about 0.30[ms] in total. This value is mostly identical with 0.31[ms], which is the difference of RTT compared with the case where it is processed only by Rich OS.

As indicated by the result of the performance of the re-encryption

process, there is a trade-off between throughput and suspension time of Rich OS, since the performance improves in the case of larger block size whereas throughput improves in the case of smaller block size. In this experiment, we implemented decryption for XOR and AES for re-encryption. When block size is 256[B], throughput degrades 8% and the RTT value increases 0.04[ms], whereas throughput degrades 1.2% and the RTT value increases 0.32[ms] when block size is 1[KB], compared with the case where processing is only by Rich OS. Considering that security improves in both cases, the degradation of throughput and RTT is sufficiently small. In the real world, developers need to determine the block size in light of various restrictions, such as response time, performance degradation, and physical memory size. For example, if a high priority is accorded to the performance of the protection-required module, they will choose the block size of 256[B], whereas they will choose the block size of 1[KB] if a high priority is accorded to the performance of Rich OS.

In the proposed method, context switch in a round trip takes 1.66[us], and converted into the number of processor cycles, it is 664 cycles. Other than the approach we show in this dissertation, there is a legacy method implemented by software only in order to isolate protection-required processes and their data by classifying the user tasks handling secret data, such as crypto process, into those that do not access the secret data. In such a system, context switching of processes occurs to exchange data between processes in Rich OS. Kanai shows that context switching takes slightly less than 1000 cycles on Linux [49]. Therefore, the proposed method enables context switch at lower cost than that of an operating system. Moreover, Otani indicates that acceptable delay is 50-300[ms] in a smart grid [45]. Even if data size is 1[KB] and it is divided into 128 bits blocks, the re-encryption process completes at 1.29[MBps] / 1024 = about 1.26[ms], re-encryption processing is completed from the performance result shown in Figure 4.5, and it is sufficiently small compared with the acceptable delay, which is 50-300[ms]. Therefore, even if the time for context switch between worlds and the re-encryption processing time are added, the delay can be kept within an acceptable range. In summary, the evaluation shows that the proposed method enables reduction of the suspension time of Rich OS, in addition to enhancing confidentiality and integrity.

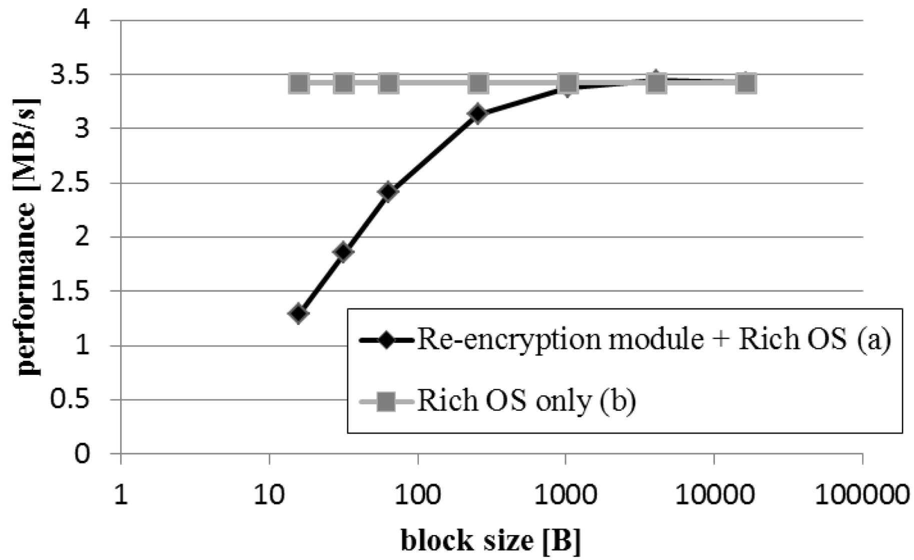

Figure 4.5: Throughput of the re-encryption process.

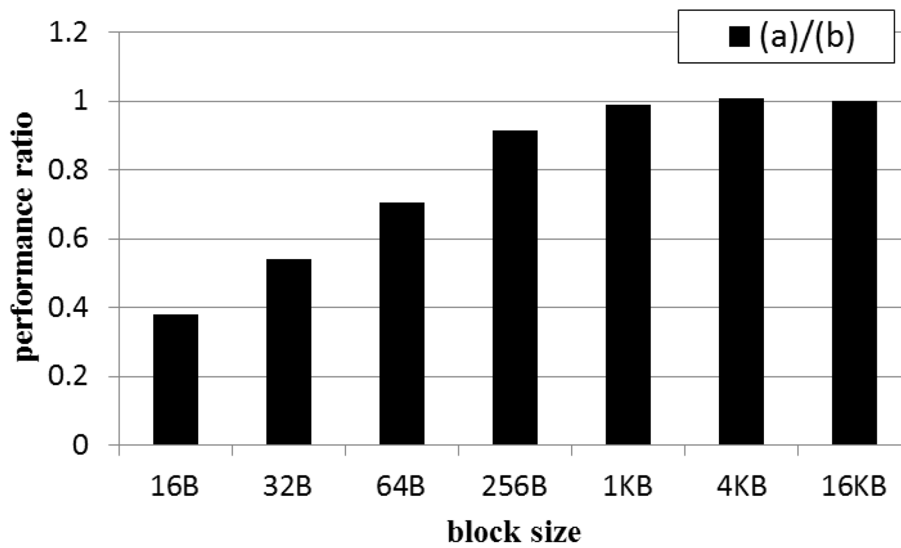

Figure 4.6: Performance ratio of the re-encryption process (Rich OS ony = 1.0).

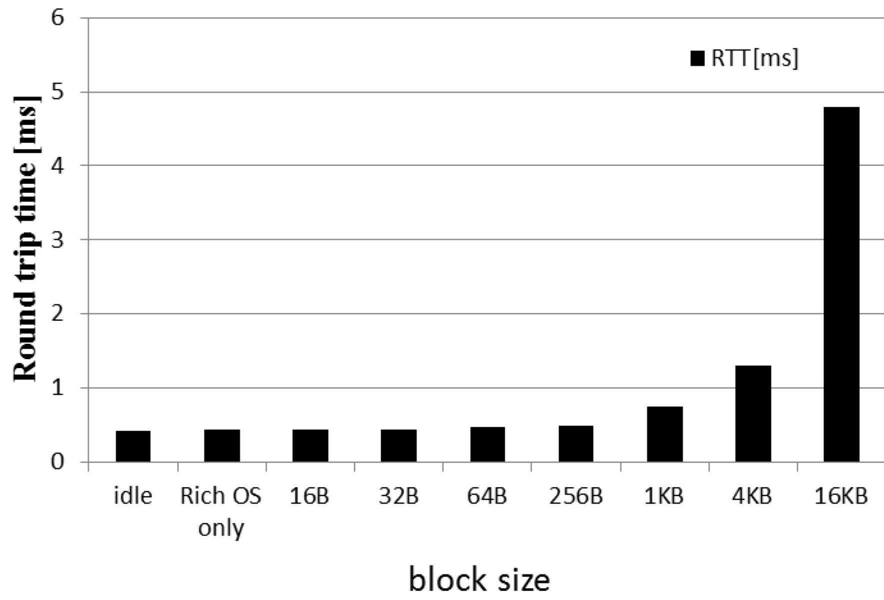

Figure 4.7: Evaluation result of the network latency.

3) Performance of module update

We evaluated the processing time of the update of the security-required module. In this evaluation, we prepared binary objects whose sizes are 1[KB], 10[KB], and 100[KB]. We measured processing time from loading the binary object from the disk through copying the decrypted protection-required module to the memory area of the secure region assigned to the protection-required module.

Figure 4.8 shows the result of the experiment. When the size of the binary object is 1[KB] and 10[KB], 100[KB] and 1[MB], the processing time is 0.35[s], 0.36[s], 0.4[s], and 0.9[s], respectively with 256 bit key length. The larger the binary object, the longer the time to update the protection-required module becomes. We also measured processing time with different key lengths. When the binary object is 100[KB] and key length is 256 bits, 512 bits, and 1024 bits, the processing time is 0.4[s], 2.51[s], and 18.89[s], respectively. The longer the key length, the longer the processing time becomes. Next, we analyzed the processing time in detail. Processing time consists of a hash calculation process, a decryption process, an RSA signature verification process, and a memory copy process. We measured the RSA signature verification process only. When key length is 1024 bits and the binary object is 100[KB], the signature verification process accounts for 18.84[s] of the whole processing time of

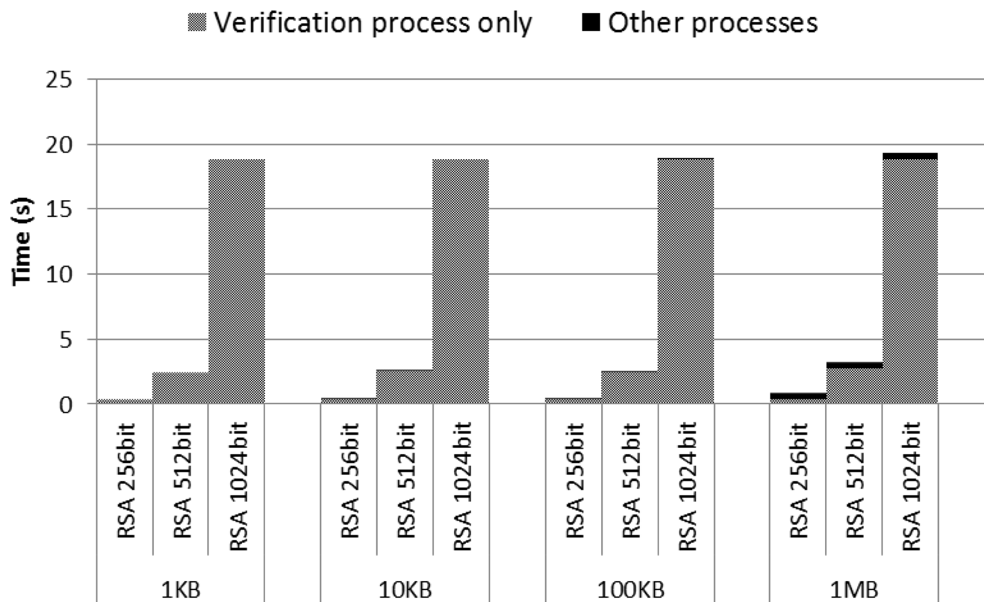

Figure 4.8: Performance result of the module update process.

18.89[s]. Similarly, when the binary object is 1[MB], the RSA signature verification process accounts for 18.9[s] of the whole processing time of 19.4[s]. The result shows that the RSA signature verification process accounts for the greater portion of processing time whereas a hash calculation process, a decryption process, and a memory copy process together takes a relatively small portion of processing time. Although a module update process takes a long time, the update process is rarely executed as described above. Therefore, even if the public key crypto process that requires processing time is used, the suspension time of Rich OS is mitigated by invoking context switch from Virtual Hardware to Rich OS after executing the Update module for a certain period of time by the Time management module.

In the result of the experiment shown in Figure 4.8, the Time management module does not periodically invoke context switch by timer interrupt. We configured the timer in the Time management module and measured the processing time in the case of invoking periodic context switch to Rich OS. As a result, when the key length is 1024 bits and the binary object is 1[MB], the processing time is 19.4[s] without timer interrupt, whereas the processing time is 19.43[s] and 19.7[s], when the timer is set to 1[ms] and 100[us]. The result shows the additional performance degradation of RSA signature verification process is small, since the processing time of context switching is short, even if context switch to Rich OS is invoked periodically.

Comparative evaluation with hardware implementation

We evaluated our proposed method, comparing it with the case where it is implemented by hardware, in terms of performance, functions and cost.

1) Performance

   When 128 bit AES algorithm is implemented as a hardware accelerator, throughput of hundreds of Mbps through several Gbps can be realized, which is 100 to 1000 times faster than the proposed method although it depends on the implementation techniques [50][51]. However, when an application such as a smart meter is assumed, since data size to be encrypted is small, the disadvantage of performance speed is not a big issue in practice.

2) Functions

   Functions realizable equivalent to those of the crypto hardware:

The Re-encryption module of Virtual Hardware is functionally equivalent to that implemented as hardware in terms of decrypting input data and encrypting the data by different keys and algorithms. In addition, the signature generating process included in the TPM module of Virtual Hardware is functionally equivalent to that implemented as physical hardware TPM. However, as described in section 4.1.2, we modified the function so that PCR value of TPM can be reset at the time of loading of a protection module. Generally, it is impossible for software to change hardware functions physically. Similarly, since writing a memory area of Virtual Hardware from Rich OS that uses a function of the protection-required module by configuring memory access control policy is restricted, it is prohibited for software to change hardware functions. Moreover, in order to prevent information leakage of confidential data, such as a key during execution of the protection-required processes, confidential data are generally implemented inside hardware in the case of hardware implementation. In the proposed method, confidential data are not revealed to Rich OS executed in the non-secure world since they are managed by Virtual Hardware whose memory area is configured as secure. Many general-purpose SoCs have a function that encrypts firmware and saves it in a flash memory. The proposed method can be used in combination with the function to store Virtual Hardware and Monitor to realize a more robust system. From the viewpoint of software development, when application programs use crypto functions implemented as hardware, it is common to access the functions through interfaces device drivers in most systems with crypto hardware. In the proposed system, application programs access functions of Virtual Hardware via device drivers. Therefore, application developers using Virtual Hardware can use the same interface as the conventional hardware features.

Functions realizable by crypto hardware only:
There are functions realizable only by hardware in exchange for additional cost. It is unnecessary to generate a random number in an application example in our proposed method. However, there are use cases where a high-quality random number generator is required. It is possible to generate a true random number using the physical phenomenon when it is implemented as hardware. Moreover, some physical crypto hardware has a hardware-level tamper-proof function, such as TPM. Such hardware is robust against advanced physical side-channel attacks, such as a power analysis attack or a timing attack, by attackers having special skill and dedicated tools such as an

electron probe, whereas the proposed method is vulnerable to the attacks since there is no hardware protection mechanism.

Functions realizable by the proposed method only:
It becomes impossible to update the value of a key used for encryption and decryption when the key is included in a hardware module. Although many crypto hardware modules are configured to update the value of the key from software, since it is necessary to process the key in a plaintext manner by software, there is a great risk that the key will be leaked. On the other hand, since the protection-required module that includes a key is distributed from a server in an encrypted manner in our proposed method, it is possible to update the key securely. It is also possible to update a crypto algorithm in the proposed system. Furthermore, not only cryptographic algorithms, such as AES, but more complex protection-required processes are updatable. For example, in order to manage keys of smart meters efficiently in a group or to revoke a specific smart meter, applying broadcast encryption with Media Key Block (MKB) is proposed [52]. It requires execution of complicated processing to retrieve the key value from MKB. Whereas it is difficult to replace the processing when implemented as hardware, it is easy for the proposed method to update a part of key derivation processing while concealing know-how, such as by a high-speed processing algorithm of MKB.

Cost analysis

Cost can be classified at the time of manufacture and deployment. The cost at the time of manufacture is a manufacturing cost per device. In the proposed method, since additional hardware is unnecessary except for an ARM processor, the additional manufacturing cost is zero. On the other hand, when crypto functions are implemented as hardware, there is an additional manufacturing cost per device. Although a small quantity of parts will have only a slight impact on total cost, total cost will become immense when installing tens of millions of devices such as smart meters.

The cost at the time of deployment is the cost incurred when updating a key and an algorithm. When realizing crypto functions as hardware, field engineers need to replace hardware physically. Similar to the cost at the time of manufacture, a small quantity of parts will have only a slight impact on the total replacement cost, but it will increase according to the number of devices to be installed. Furthermore, when replacement is required for a security reason, immediate replacement is required since the equipment is at great risk of succumbing to attack until replacement is

executed. However it is unrealistic to expect field engineers to replace tens of millions of devices in a short period of time. Since the proposed method enables remote updating of embedded end-point devices securely and immediately, deployment cost can be greatly reduced. On the other hand, since the proposed method assumes updating of a protection-required module via a network, the maintenance cost of a server that distributes the module is additionally incurred. However, the maintenance cost of a server decreases because of the evolution of cloud computing.

Thus, compared with hardware implementation, cost at both the time of manufacture and at the time of deployment can be reduced by using the proposed method.

4.2 A method to keep availability

4.2.1 Framework of the recovery system

The proposed system provides a method for an embedded end-point device to automatically recover from an error status. It also provides a high-level memory protection mechanism. Hence, the recovery process is securely executed without interference. Figure 4.9 shows the entire architecture of the proposed recovery system. It consists of three components: Rich OS, Tracker Application, and Monitor.

Since the proposed recovery system reuses some functions of the method described in the previous section, functions of each component are similar with the ones described in the previous section. However, we will describe each component in detail for clarification in this section.

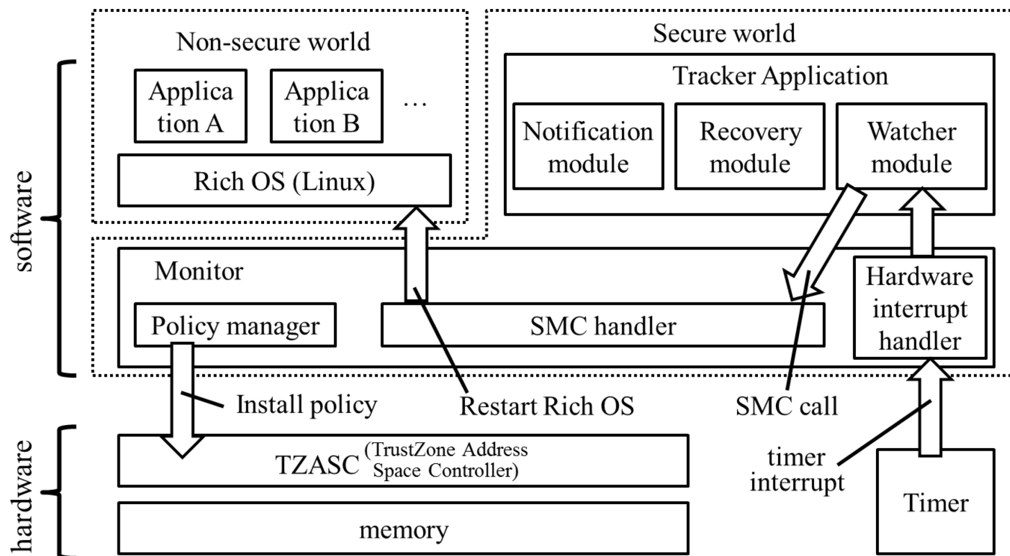

Figure 4.9: Architecture of the recovery system.

- Rich OS: An operating system that executes general-purpose processes, such as storage access or network communication. It is executed in the non-secure world. All applications implementing smart meter functions or concentrator functions run on this operating system.

- Tracker Application: Surveillance and recovery processes executed in privileged mode in the secure world. Tracker Application includes three modules: the Watcher module, the Recovery module, and the Notification module. The Watcher module is an entry point of Tracker Application. It is executed periodically by a timer interrupt through Monitor. Whenever it is called, it investigates the status of Rich OS. If it detects Rich OS is not working, it calls the Recovery module to reboot the entire system. Otherwise, it calls the SMC instruction to switch to Rich OS. Moreover, the Notification module is called before the Recovery module reboots the system. It sends a message to notify that the system is about to reboot to the head-end system through network

- Monitor: A program running in the monitor mode. It initializes configurations of TrustZone-related hardware when booting the system. It also provides a context switching function between worlds in the hardware interrupt handler and the SMC handler. Moreover, Monitor manages the access control policy and installs the policy on TZASC when booting. Policy Manager takes on their roles.

The primary feature of the proposed recovery system is to provide a method for the end-point device to detect the status of Rich OS and to recover it even if Rich OS crashes or stops working. Furthermore, it provides two additional functions. One is to enhance the security protection for Monitor, Tracker Application and Rich OS against attacks. The other is to send a message to the head-end system when an incident occurs. The details of these functions are described below.

4.2.2 Functions of the recovery system

Baseline common functions

Monitor has the role of providing baseline common functions to operate Rich OS and Tracker Application concurrently. Monitor has two functions: system initialization and context switching between worlds.

1) System initialization

   When booting the system, the processor is in the secure world and Monitor is firstly executed. To run Rich OS and Tracker Application concurrently, it needs to load and execute both of them. It first

initializes the status of the processor in both worlds, and loads Tracker Application in the secure world. Then, it invokes context switching to transit from the secure world to the non-secure world, loads the boot loader program of Rich OS, and executes it in the non-secure world. Finally, the boot loader program loads Rich OS and executes it.

The TrustZone-enabled processor supports the function that is either Monitor or Rich OS traps each processor exception (IRQ, FIQ, and external abort). When booting the system, Monitor configures that hardware interrupt handler in Monitor traps timer interrupt so that Rich OS cannot interfere with the execution of Tracker Application when timer interrupt occurs. As well as timer interrupt, Monitor configures that hardware interrupt handler in Monitor traps external abort. Since the access violation causes external abort as described above, this configuration enables Tracker Application to detect the occurrence of a memory access violation.

TZPC is configured to be accessed from the secure world only when booting the system. Since Rich OS needs to use peripherals, Monitor needs to change the configuration of TZPC to non-secure. The only exception is Timer, which triggers periodical execution of Tracker Application. Since it is necessary to prevent the configuration of Timer from changing by a process running in the non-secure world, Monitor remains the configuration of TZPC corresponding to Timer as secure.

2) Context switching between worlds

The trigger of context switching between worlds is either the SMC instruction or the Timer interrupt caused by the hardware timer. The SMC handler in Monitor is executed when the SMC instruction is called and it transits from the secure world to the non-secure world. In contrast to the SMC handler, the timer interrupt triggers transit from the non-secure world to the secure world. In both cases, Monitor invokes context switching between worlds. It first determines the current world. As described in Chapter 3, general registers and Saved Program Status Register are not banked between worlds. Therefore, Monitor needs to save the contents of the registers belonging to the current world on working memory to prevent loss of the previous context, and then change the world. Finally, it restores the contents of the registers belonging to the transition destination world and resumes

the execution.

Periodical surveillance and recovery

While executing Rich OS, whenever the timer interrupt occurs, the processor jumps to the hardware interrupt handler in Monitor. The hardware interrupt handler context switches from the non-secure world to the secure world and calls Tracker Application. Specifically Monitor saves a context of Rich OS to memory and restores a context of Tracker Application, then changes the world and finally calls the Watcher module of Tracker Application. The Watcher module checks the status of Rich OS. If it judges that Rich OS is not working, the Watcher module calls the Recovery module that reboots the system. Otherwise, it calls the SMC instruction. Then, the SMC handler in the Monitor is executed. It context switches from Tracker Application to Rich OS, and restarts Rich OS at the point just before the timer interrupt occurred. While executing Monitor and Tracker Application, the execution of Rich OS is suspended. That is, Rich OS continues to be processed as if nothing were executed during the execution of Tracker Application. Figure 4.10 shows the flowchart of the periodic surveillance and recovery process.

There are many ways for the Watcher module to determine whether Rich OS is working or not. One of the methods is to check the data area of Rich OS. In general, when an operating system is working, there must be a certain data area that is updated regularly. By checking this data area, it is possible for the Watcher module to judge whether Rich OS is working or not.

Memory protection

By utilizing TZASC, Monitor provides an access control function such that access of Rich OS running in non-secure mode to the working memory area, which Tracker Application running in the secure world uses, is subject to restrictions. Policy Manager in Monitor manages three kinds of access control policies: full access, access denied, and read-only. When booting the system, Policy Manager divides working memory into several regions and it installs one of the three access control policies for each working memory region on TZASC before loading Rich OS.

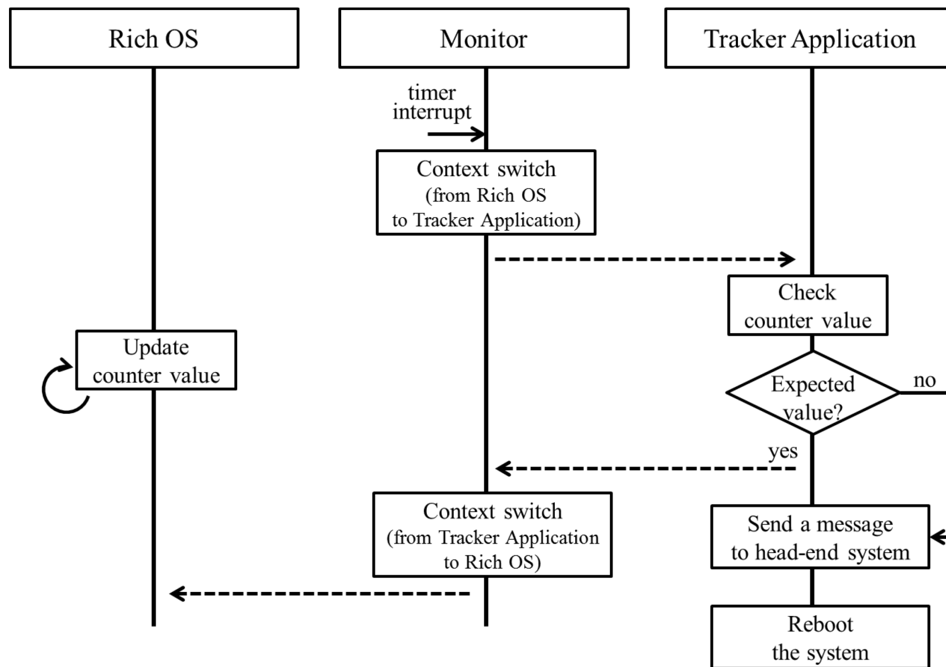

Figure 4.10: Flowchart of the periodic surveillance and recovery process.

Table 4.2 shows how each policy works. Full access indicates no restriction. A process running in both non-secure world and secure world can freely access the region configured according to this policy. This policy is primarily used to share data between Rich OS and Tracker Application. Access denied indicates full restriction. A process running in the non-secure world can neither read nor write to a region configured according to this policy, whereas a process running in secure world can read and write to the region. Read-only indicates a process running in the non-secure world cannot overwrite the content on the memory but is allowed to read it, whereas a process running in the secure world can freely access the region using ordinary random access memory, such as DRAM or SRAM, as the working memory which is, of course, physically writable memory.

Table 4.2: Configuration of access control policy

Policy	***From secure world process***	***From non-secure world process***	
		Read	***Write***
Full access	OK	OK	OK
Access denied	OK	NG	NG
Read-only	OK	OK	NG

Using these policies, the proposed system provides two memory protection mechanisms. Figure 4.11 shows how these memory protection mechanisms work. One is protection for the kernel area of Rich OS. The other mechanism is protection for Monitor and Tracker Application.

To realize protection for the kernel area of Rich OS, Monitor provides read-only memory. In general, when a program is loaded into memory, a data region (data segment) and a code region (code segment) are assigned. In the initial state before booting the system, all regions are allowed to be accessed from the non-secure world by default. In order to allow the boot loader to write the code segment into the memory, Monitor leaves the memory region as is until the code segment is loaded. Just after executing the kernel of Rich OS, Monitor sets the memory region as read-only for kernel code segment of Rich OS. As a result, even Rich OS is prohibited from overwriting its own code segment.

To protect Monitor and Tracker Application, Policy Manager in Monitor installs an access control policy such that Rich OS cannot access the memory area allocated to Monitor and Tracker Application, whereas Tracker Application and Monitor can access all areas when booting the system. This policy protects Monitor and Tracker Application from illegitimate falsification by Rich OS, even if Rich OS is attacked and under the control of an attacker.

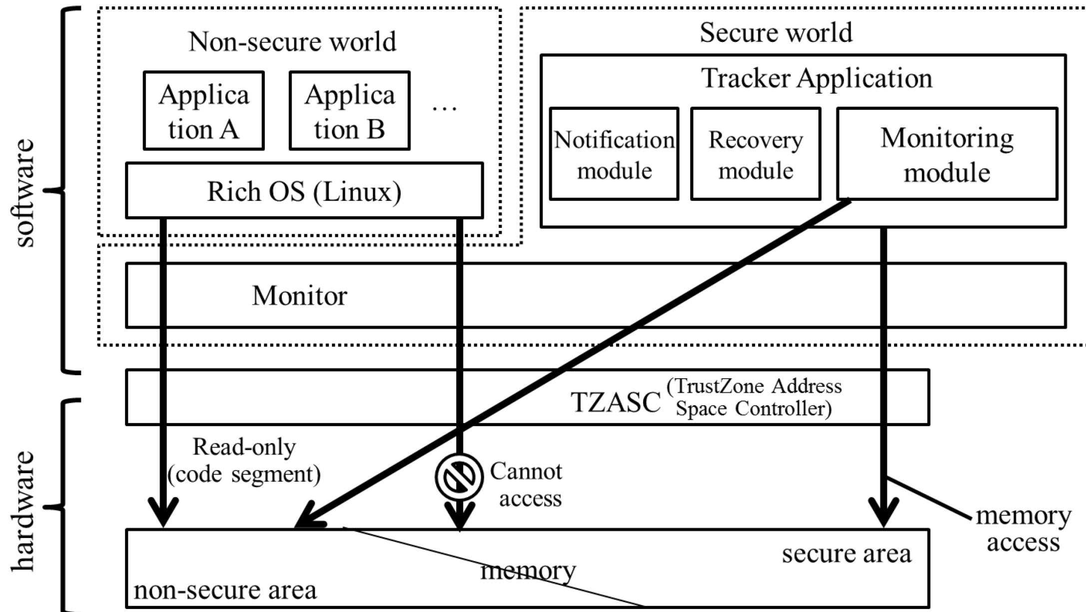

Figure 4.11: Memory protection mechanism.

Besides the protection for Monitor and Tracker Application, memory protection provides a hardware access control mechanism. One of the possible attacks to disable end-point devices is that of shutting down the system. To prevent such an attack, Policy Manager in Monitor installs an access control policy so that Rich OS cannot access the registers corresponding to power management. Thus, it is possible to protect the system against the shutdown attack even if Rich OS is under the control of an attacker.

In the case of policy configured to access denied or read-only, TZASC generates an interrupt signal when the access violation caused by a process running in the non-secure world occurs. Monitor configures the hardware interrupt handler in Monitor to trap the interrupt so that the system will continues to work without crashing even if access violation occurs, and Monitor can detect the access violation.

Message notification

The proposed recovery system provides a function to notify the head-end system that Rich OS has stopped working and is rebooting the system by sending a message through the network even if the operating system is

modified or the control of the operating system is taken over; resulting network function is disabled by an attacker or the operating system is completely destroyed in the worst case. The Notification module has the role of sending a message. Although Rich OS has a network connectivity function, such as TCP/IP stack, Tracker Application cannot use the function since there is a case where it is not working when sending a message. Thus, Tracker Application supports the network connectivity function including the network application, the network protocol stack and the network driver to notify the error situation to the system administrator through the network. Obviously, it is possible to send a head-end system a message whenever Tracker Application is executed to notify that the system works correctly.

4.2.3 Prototype implementation

We used ARM C/C++ Compiler 5.01 to build Monitor and Tracker Application. We used gcc 4.4.1 to build Linux 3.6.1 as Rich OS. We chose Motherboard Express uATX with the CoreTile Express A9x4 processor that supports TrustZone as an execution environment.

Regarding a memory map, from 0x48000000 through 0x4A000000 is assigned for SRAM, and from 0x60000000 through 0xE0000000 is assigned for DRAM. Table 4.3 shows the memory map with the access control policy of the memory. In Table 4.3, Rich OS (code) indicates the Linux kernel code. Rich OS (data) includes the Linux data, the application code and the application data. For clarification, full access is applied from the non-secure world for an area not described in Table 4.3.

Table 4.3: Memory map of the recovery system

Data	***Start address***	***Size***	***Security permission (From non-secure world)***
Vector tables + Initialization code + Monitor + Tracker Application	0x48000000	0x01B00000	Access denied
Rich OS (code)	0x60000000	0x002FE000	Read-only
Rich OS (data)	0x602FE000	0x3EF02000	Full access
Shared memory	0x9F200000	0x00C00000	Full access

For the Policy Manager in Monitor to install an access control policy on TZASC, the start address and the size of each memory region are predefined. After the boot loader loads Linux at the predefined value, Monitor installs the access control policy on TZASC. As shown in Table 4.3, the access to the memory regions allocated to Monitor, Tracker Application and the code segment of Rich OS is restricted for the Rich OS running in the non-secure world, whereas the access to the region allocated to the data segment of Rich OS and shared memory is not. For clarification, Monitor and Tracker Application running in the secure world can access all regions. Furthermore, since Monitor sets the configuration registers of TZASC to prohibit Rich OS from accessing them, Rich OS cannot change this configuration.

Table 4.4 shows the configuration of TZASC. In Table 4.4, the meaning of the value of the security permissions field is as follows: 0b1111 indicates full access from both the secure world and the non-secure world, 0b1100 indicates secure read/write is permitted but non-secure read/write is restricted (access denied), and 0b1110 indicates secure read/write and non-secure read are permitted but non-secure write is restricted (read-only). An entry with larger entry number is accorded higher priority than one with smaller entry number. Therefore, we first set all regions with a policy of full access as entry number 0, and then set access control policies from entry number 1 through 7. The size of a region to which access control is applied is discrete, such as 32[KB], 64[KB], …, 1[MB], 2[MB], 4[MB], …, 2[GB], 4[GB]. Therefore, to set policy for Monitor and Tracker Application whose size is 0x01B00000 (27[MB]), we used four entries: entry number 1 (16[MB]), entry number 2 (8[MB]), entry number 3 (2[MB]), and entry number 4 (1[MB]). In contrast to the size of Monitor and Tracker Application, the size of Rich OS (code) is a fraction (32[MB] – 8[KB]), and TZASC has restrictions such that it is impossible to define an entry whose size is smaller than 32[KB]. Instead, it is possible to define a subregion to equally divide a region into eight with the access control policy, and enable the policy for each subregion. For example, when the size of a region is 32[KB], it is possible to enable a policy for each 4[KB] subregion. An 8 bits subregion disable field controls enabling and disabling the policy. Each bit in a subregion disable field enables the corresponding subregion to be disabled. For example, when zero is set to the value of the highest bit in a subregion disable field, the policy for subregion 0 (the subregion having the highest address) is enabled. To set the policy for a Rich OS (code) region, we first defined two regions, 2[MB] (entry number 5) and 1[MB] (entry number 6)

and set the read-only policy. Then, we defined the region with a size of 64[KB] (entry number 7) that overlaps the last portion of entry number 6, equally divides the region into eight, sets the policy of full access, and enables the policy for the last subregion only. As a result, the policy of full access is set to the subregion having the highest address only, and the policy of read-only remains for the rest of the subregions.

As shown in Table 4.3 and Table 4.4, the policies can be clearly defined and there is no overlapped region. Thus, no policy conflict exists in the proposed recovery system.

Table 4.4: Configuration of TZASC

Entry number	*Start address*	*Size*	*Subregion disable*	*Security permission*
0	--	--	--	0b1111
1	0x48000000	0x17(16MB)	0x0	0b1100
2	0x49000000	0x16(8MB)	0x0	0b1100
3	0x49800000	0x14(2MB)	0x0	0b1100
4	0x49A00000	0x13(1MB)	0x0	0b1100
5	0x60000000	0x14(2MB)	0x0	0b1110
6	0x60200000	0x13(1MB)	0x0	0b1110
7	0x602F0000	0xF(64KB)	0x7F	0b1111

Figure. 4.12 shows the assignment of the timer interrupt. We allocated a timer interrupt caused by a timer (timer 1) to Fast Interrupt Request (FIQ) and the timer interval was set to 1[s]. The FIQ interrupt is handled by the hardware interrupt handler in Monitor, then it calls Tracker Application and, as a result, Tracker Application is periodically called. We used another timer (timer 2) and allocated it to Interrupt Request (IRQ), and the timer interval was set to 4[ms]. The IRQ interrupt is handled by the interrupt handler in Linux. Since Linux assumes the timer interrupt is allocated to IRQ, modification of the Linux source code to adopt Monitor is unnecessary.

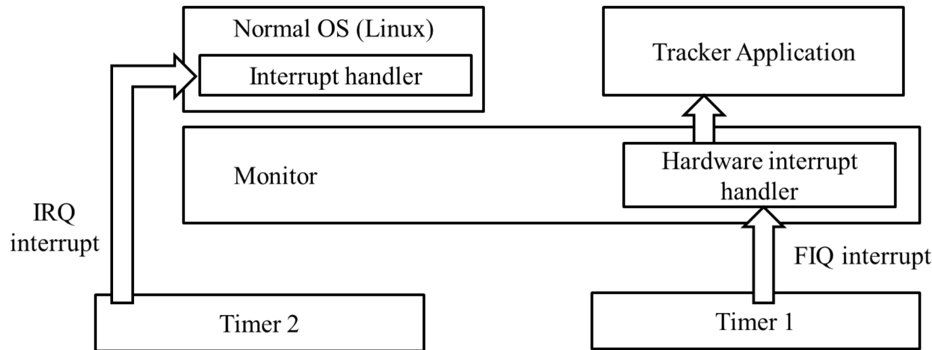

Figure 4.12: Assignment of the timer interrupt.

Table 4.5 shows a configuration of hardware interrupt. We configured Secure Configuration Register (SCR) and Current Program Status Register (CPSR) so that the FIQ handler of Monitor is called when the FIQ interrupt occurs, whereas the IRQ handler in Linux is called when the IRQ interrupt occurs during executing Linux. Table 4.6 shows the register setting to achieve the configuration of Table 4.5. CPSR.I indicates the Interrupt disable bit and is used to mask the IRQ interrupt. CPSR.F indicates the Fast interrupt disable bit and is used to mask the FIQ interrupt. CPSR.A indicates the asynchronous abort disable bit and is used to mask asynchronous abort. SCR.FIQ controls which mode the processor enters when the FIQ interrupt occurs. If one is set, it enters monitor mode, otherwise it enters FIQ mode. SCR.IRQ controls which mode the processor enters when the IRQ interrupt occurs. If one is set, it enters monitor mode, otherwise it enters IRQ mode. SCR.FW controls whether the F bit in the CPSR can be modified in the non-secure world. SCR.EA controls which mode the processor enters when external abort including the one generated by TZASC. If one is set, it enters monitor mode, otherwise it enters abort mode. SCR.AW controls whether the A bit in the CPSR can be modified in the non-secure world. If zero is set, CPSR.A can be modified only in the secure world, otherwise it can be modified in both worlds.

Table 4.5: Configuration of hardware interrupt

World when interrupt occurs	***Interrupt***	***Jumps to***
Non-secure world	FIQ	Hardware interrupt handler (FIQ handler) in Monitor
	IRQ	IRQ handler in Rich OS (Linux)
Secure world	FIQ	Pending FIQ
	IRQ	Pending IRQ

As shown in Table 4.6, when a processor is in the non-secure world and the FIQ interrupt assigned for timer 1 occurs, the FIQ handler in Monitor is called since one is set to SCR.FIQ. The FIQ handler in Monitor switches from the non-secure world to the secure world and calls the FIQ handler in Tracker Application. Finally, the FIQ handler in Tracker Application calls the Watcher module. The entry point to Tracker Application from Monitor is only the FIQ handler in Tracker Application and it never returns to Tracker Application after the Watcher module calls SMC instruction under the current implementation. When considering returning to the original location in Tracker Application when entering the secure world next time as future extension, the FIQ handler in monitor mode sets the instruction located in the address next to the address of the instruction just after calling the SMC instruction in the previous time to r14 before calling the FIQ handler of Tracker Application. On the other hand, when the IRQ interrupt occurs, the IRQ handler in Rich OS is called. Furthermore, Rich OS cannot change the configuration of CPSR.F since zero is set to SCR.FW. Therefore, the FIQ interrupt is always enabled and the timer interrupt is input to the monitor.

Table 4.6: CPSR and SCR register configuration

		Non-secure world	***Secure world (Tracker Application)***	***Secure world (Monitor)***
CPSR	I	0/1 (depending on the configuration of Rich OS)	1 (IRQ disabled)	1 (IRQ disabled)
	F	0 (FIQ enabled)	1 (FIQ disabled)	1 (FIQ disabled)
	A	0 (Asynchronous abort enabled)	0 (Asynchronous abort enabled)	1 (Asynchronous abort disabled)
SCR	FIQ	1 (enter monitor mode)	0 (enter FIQ mode)	0/1 (depending on which world transiting to)
	IRQ	0 (enter IRQ mode)	0 (enter IRQ mode)	0 (enter IRQ mode)
	FW	0 (can be modified CPSR.F only in secure)	0 (can be modified CPSR.F only in secure)	0 (can be modified CPRS.F only in secure)
	EA	1 (enter monitor mode)	0 (enter abort mode)	0/1 (depending on which world transiting to)
	AW	0 (can be modified CPSR.A only in secure)	0 (can be modified CPSR.A only in secure)	0 (can be modified CPSR.A only in secure)

When a processor is in the secure world and FIQ or IRQ interrupt occurs, the interrupt is pending since zero is set to CPSR.F and CPSR.I. For future extension, Monitor changes SCR.FIQ setting during context switching so that Tracker Application handles the FIQ interrupt directly without Monitor when the FIQ interrupt occurs in the secure world. That is, zero is set to SCR.FIQ when it transits from the non-secure world to the secure world to jump to the FIQ handler in Tracker Application when the FIQ interrupt occurs in the secure world. On the other hand, one is set when it transits from the secure world to the non-secure world to enter monitor mode when the FIQ interrupt occurs in the non-secure world.

When a processor is in monitor mode, FIQ and IRQ interrupt are disabled to avoid occurrence of multiple interrupt.

In order to determine whether Rich OS is working or not, we made a small application program, which runs on Linux and communicates with Tracker Application. Shared memory is used to exchange data between

Tracker Application and Rich OS. The application program writes a counter value into the shared memory periodically. Then Tracker Application reads the counter value from the shared memory. When Rich OS is crashed, the application program cannot update the counter value. If the counter value is not updated in a certain amount of time or the counter value is not an expected value, Tracker Application determines that Rich OS is not working. Another method of checking the status of Rich OS is to monitor the status of a specific field, such as a task structure or page tables in Rich OS, but we have not implemented it. Thanks to the memory protection function, it is impossible for Rich OS to analyze the checking process running in Tracker Application. Since it is possible to maintain secrecy of Rich OS as to which memory area of Rich OS Tracker Application monitors or how often Tracker Application checks it, it is difficult for an attacker to plan a countermeasure to circumvent the checking.

The proposed recovery system provides a method to continue working even if a memory access violation caused by TZASC occurs. Figure 4.13 shows the flowchart of how Tracker Application and Monitor recover from the error status to the normal status when an access violation caused by TZASC occurs. When booting the system, Monitor configures SCR.EA so that external aborts including the ones TZASC generates are handled in Monitor mode, instead of by the abort handler in Rich OS. Furthermore, it is prohibited to mask external abort from the non-secure world to configure SCR.AW. Therefore, when an access violation occurs in user mode in the non-secure world, for example, a processor jumps to the abort handler in Monitor. At this time, the values of r14 (lr) and spsr are the values of PC (Program Counter) and spsr of the mode just before the access violation occurs, respectively. The abort handler in Monitor saves registers including r14 and spsr of original mode in the non-secure world on working memory, context switches from the non-secure world to secure world, and calls the abort handler in Tracker Application. The abort handler in Tracker Application checks the status of Rich OS. For example, Tracker Application checks which process running in Rich OS triggers access violation or checks memory address where an access violation is triggered to investigate the reason for the access violation later. After Tracker Application checks the status, it calls the SMC instruction and jumps to Monitor. While Tracker Application works in the background when an access violation occurs, the proposed recovery system behaves as if data abort occurs from the viewpoint of Rich OS. When data abort occurs, a processor automatically stores PC and cpsr of the mode just

before data abort occurs to r14 and spsr respectively. Monitor carries out a similar operation with the processor when an access violation occurs. Monitor switches from the secure world to the non-secure world, restores the saved values including setting the saved value of r14 and spsr just before the access violation occurs to banked registers for abort mode in order to be able to return to the original location after exiting abort mode, and calls the abort handler of Rich OS. Therefore, when Rich OS restarts a process, the data abort handler is executed.

When Tracker Application determines that Rich OS is not working, it sends the head-end system a message. In order to send a message to the head-end system when Tracker Application detects that Rich OS is not working, we ported a network driver and UDP/IP stack to Tracker Application. We defined a proprietary protocol and data format over UDP to notify the head-end system that Tracker Application starts reboot of the system. An application data size of UDP packet is 32[B], and it consists of 4[B] of device ID, 1[B] of flag indicating the status of the device, and 27[B] of reserved area.

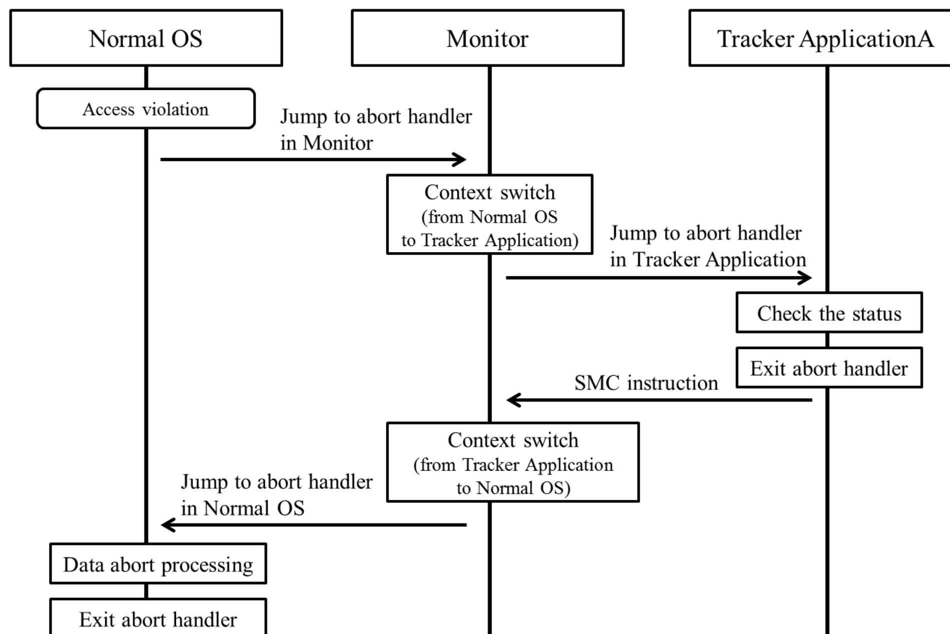

Figure 4.13: Flowchart of the access violation handling.

4.2.4 Evaluation

In this section, we describe the result of the evaluation in terms of security functions to verify the problems of the legacy system defined in Chapter 2 can be solved. Performance and cost analysis of the propose system is also described below.

Functional analysis

1) Surveillance and recovery

   The proposed recovery system can recover from a failure to reboot the system even if Rich OS crashes. The reason for the crash could be a software bug or a cyber-attack, including a zero-day attack prompted by unknown vulnerabilities. In either case, since the hardware timer interrupt continues working regardless of the state of Rich OS, Tracker Application is always periodically called and can detect a failure of Rich OS. At the next level, it is desirable to detect the failure as soon as possible.

   Detection time depends on how frequently Tracker Application checks the status of Rich OS. Since the execution time of Tracker Application and context switching by Monitor is very short, the proposed recovery system can detect the crash of Rich OS very quickly. Some attackers may continue to attack just after rebooting the system. One possible approach to a countermeasure for the attack is to let Tracker Application have a minimum function like the "safe mode", but we have not implemented that.

2) Attack prevention

   The proposed recovery system provides two levels of attack prevention mechanism. The first level is to prevent Rich OS from illegitimate modification. When an attacker gains full control of Rich OS to misuse the vulnerability, the attacker may overwrite the code segment of Rich OS to directly overwrite the memory. In fact, many vulnerabilities (e.g., CVE-2013-4342, CVE-2013-1969, and CVE-2008-1673) allowing a remote attacker to execute arbitrary code are reported [8]. In the case of Linux, for example, once arbitrary code is executed with an administrator privilege by an attacker, it is possible for the attacker to overwrite an arbitrary area of code segment through /dev/mem, resulting in system crash or misbehavior. Although overwriting the code segment in memory is generally difficult, it is

relatively easy in the case of end-point devices since the software is uniform and the hardware configuration is fixed. As a result, the system may go down. However, since Monitor sets the access control of the memory region for the code segment of Rich OS as read-only, and its configuration can be changed only from the secure world, it is impossible for attackers to overwrite the code segment of Rich OS.

An advantage is that the protection does not cause any side effects. Since a data segment is used to store the state of the program, Rich OS updates the content of the data segment frequently during its execution. In contrast to the data segment, since a code segment is used to store program code, it is not expected to update its content after booting the system. In particular because devices such as smart meters or concentrators are not expected to change their core function after being deployed, the dynamic update function to working memory is not required. Thus, this protection mechanism can protect Rich OS from illegitimate modification without side effects.

Moreover, the feature of read-only memory is very useful for the data, whose value is only changed by Tracker Application and to which Rich OS only refers. The typical application is a secure clock. In a legacy system, it is very difficult to provide a secure clock on an operating system without network connectivity or dedicated hardware if illegitimate modification of the operating system is premised. However, Tracker Application can provide a local secure clock function by software. Since Tracker Application is executed periodically and it knows the frequency of the execution, it is possible for Tracker Application to update a counter value written in a read-only memory in a certain amount of time periodically. Because the counter value is read-only from Rich OS, Rich OS cannot rewind the counter value.

The second level is to protect Monitor and Tracker Application from illegitimate modification and suspension. Since the first level of protection is effective only for a code segment of Rich OS, an attack that overwrites a data segment cannot be prevented. Thus, there are still possibilities that control of Rich OS is gained by an attacker. Even in such cases, thanks to TZASC, since Rich OS is prohibited from overwriting the content of memory where Tracker Application and Monitor are allocated, illegitimate modification is prevented. Since communication interface between Rich OS and Tracker Application is limited, it is impossible to compromise Tracker Application by an attack. Moreover, since the interrupt configuration

register is accessible only from the secure world, there is no way for Rich OS to stop the timer interrupt.

Furthermore, the proposed recovery system provides a mechanism to protect against shutdown attack. Since it is impossible to prevent Rich OS from executing a shutdown procedure with a privileged instruction in the non-secure world, when a process running in the non-secure world tries to shutdown the system, Tracker Application can detect it and discard the shutdown request. Since end-point devices usually keep working all the time, devices could be implemented without having a shutdown or reboot function. However, it is necessary to have a shutdown function in some cases. For example, the system may need to reboot when updating firmware. Another example is that a service engineer may need to reboot the system when inspecting the status of the end-point devices and fixing problems on site for maintenance purposes. Although it has not been implemented, it is possible to endow Tracker Application with a function to determine whether it should shutdown or not based on the status of the system. For example, when Tracker Application detects an access to the memory region mapped to the registers corresponding to power management and determines that the system is under a particular status, such as a maintenance mode, it may allow executing a shutdown procedure. Similarly, when Tracker Application detects the access, it sends a head-end system a message to inquire whether the shutdown request is accepted or not by using the message notification function. Based on a response to the inquiry, it can determine whether or not a shutdown procedure can be executed without interference of Rich OS.

3) System reliability

In a legacy system, one single bug could affect the entire system, causing a critical failure. Ideally, from a defensive viewpoint, the entire system including the operating system should be bug-free to achieve high availability. However, it is impracticable to build a complicated system without bugs. Linux 3.6.1 consists of over 15 million lines of code and many new bugs that cause critical crash are reported frequently (e.g., CVE-2013-4563, CVE-2013-4387, and CVE-2012-2127) even though it is carefully reviewed by many professionals [8]. Thus, the smaller the critical component that has to be robust within a system, the better. In the case of the proposed recovery system, the critical components correspond to Tracker

Application and Monitor. In contrast to Linux, the code size of Monitor and Tracker Application is relatively small. The volume of source code for Monitor is about 700 lines and its code and data size are 2.1[KB] and 1.6[KB], respectively. Similarly, the volume of source code of Tracker Application is about 41200 lines and its code and data size are 1.09[MB]. Compared to the volume of source code of Linux, the risk of Monitor and Tracker Application including bugs is small.

4) Response to failure

The Notification module in Tracker Application sends a message to the head-end system just before rebooting the system. The message, which notifies that particular devices are about to reboot, is sometimes useful information for administrators. For example, if messages are sent by devices having a particular software version number, the reboot could be caused by an attack aimed at a vulnerability specific to the software. If messages are sent by devices located in one particular network, the reboot could be caused by a network worm distributed in that specific network. Although the proposed recovery system cannot prevent an attack in advance, the notification feature can help the administrator investigate the reason for the failure during or after the incident. For example, it is impossible for the proposed recovery system to prevent an attacker from compromising Rich OS and causing reboot frequently. However, the administrator can notice that frequent reboot occurs to the device through network since the Notification module sends a message each time when rebooting. The attackers may try to block sending of the message to circumvent the notification. However, Rich OS cannot interfere with the Notification module sending a message to the head-end system since the Notification module is executed inside Tracker Application. Moreover, since Tracker Application is processed in an environment isolated from Rich OS, security processes, such as encrypting a message, are easy to implement in Tracker Application. Therefore, once an encryption key and an encryption process are implemented Tracker Application, it is possible to keep them secret from Rich OS. In the next step, it is possible to include a firmware update feature to implement functions receiving data from the head-end system and writing the data into the file system to extend the function of the Notification module. In combination with the “safe

mode" described above, this function is effective against a continuous attack that occurs just after the system recovers.

Performance analysis

As well as the implementation environment, we used Motherboard Express uATX that contains the ARM Cortex-A9x4 processor running at 400 MHz as an experimental environment. Level 1 instruction cache, level 1 data cache, and level 2 cache are 32[KB], 32[KB], and 512[KB], respectively. It contains 1[GB] DRAM as the main memory and we assigned the same memory map as that of previously described.

First, we measured the execution time of Tracker Application during execution of Rich OS; to be precise, the time period from the beginning of the hardware interrupt handler in Monitor through the execution of the SMC instruction. Without calling the Notification module, the average time is 1.7[ms] over 10,000 trials. However, if the Notification module is called, the average time is 4.1[ms] over 10,000 trials. Note that the Notification module is called when rebooting the system, which rarely occurs. Thus, this performance overhead poses no problem.

Next, we measured the performance degradation of Rich OS. Since the execution of Rich OS is suspended during execution of Tracker Application, the performance of Rich OS degrades in any case. The total of Rich OS suspension time depends on the frequency of calling Tracker Application. There is a tradeoff between the performance degradation of Rich OS and the delay in detecting the crash of Rich OS. When the frequency is increased, the performance degradation of Rich OS is also increased. On the other hand, when the frequency is decreased, the delay for detecting the crash of Rich OS becomes larger. Since a general application is assumed to be executed on Rich OS, we used dhrystone as a benchmark program to measure the performance degradation [53].

Figure 4.14 shows the result of the experiment. The bar graph shows the dhrystone score and the line graph shows the performance degradation. The higher the score, the better the performance is. Each bar shows the timer interval of calling Tracker Application and its value is default (never called), 5[s], 3[s], 1[s], 0.2[s] and 0.04[s] respectively. When the timer interval was set to 5[s], the performance degradation was suppressed within 0.001 %. Even if the interval was set to 0.04[s], the performance degradation was less than 0.2 %. The result shows that although there is a tradeoff between performance degradation of Rich OS and detection rate logically, the performance degradation can be ignored in practice even if

the frequency of calling Tracker Application is increased.

Figure 4.15 shows another result of the experiment. In the case of Figure 4.14, it is assumed that the Notification module sends a head-end system a message only when Rich OS stops working and the system is rebooting. Therefore, the result does not include processing time of the Notification module. On the other hand, Figure 4.15 assumes that the Notification module sends a head-end system a 32[B] message whenever Tracker Application is executed even if Rich OS is working correctly. This experiment assumes that the Notification module sends the head-end system a message periodically even if Rich OS keeps working so that an administrator can monitor the status of each device. Although the result of the experiment shows that the performance slightly degrades compared with the experiment without message transmission, it can still be ignored in practice. Note that the score was better for the experiment with message transmission than for the experiment without message transmission when the interval was set to 5[s], 3[s], and 1[s]. When the timer interval is long, the execution times of Tracker Application and Monitor are negligible compared with the execution time of Rich OS since the task is too small to measure accurately. Thus, this can be regarded as an error.

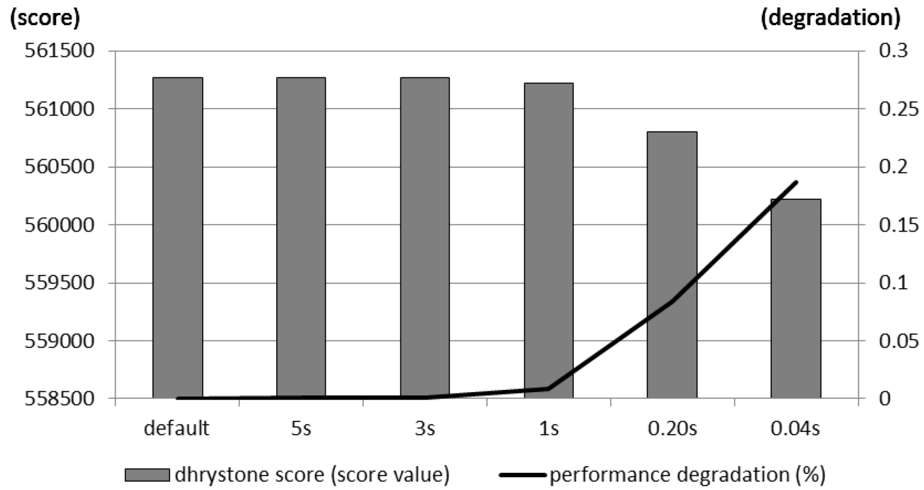

Figure 4.14: Result of the performance degradation.

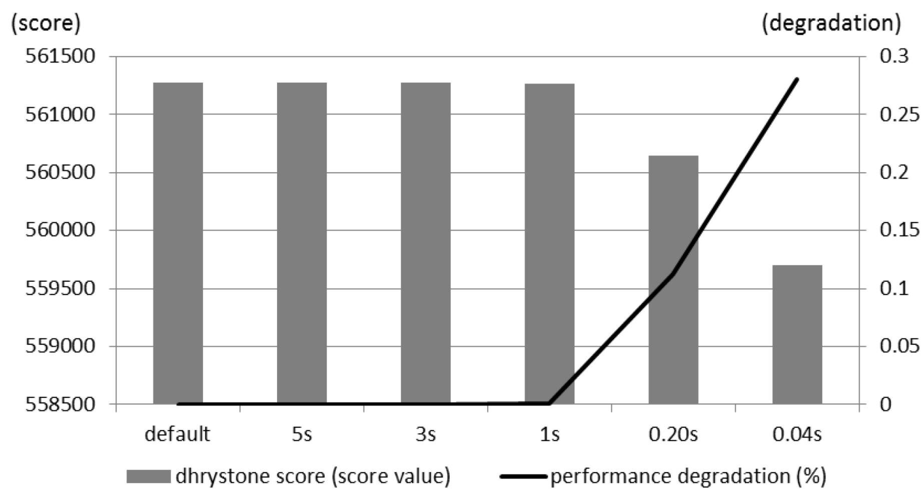

Figure 4.15: Result of the performance degradation with message notification.

Cost analysis

1) Development cost

   The proposed recovery system does not require any modification to Linux in order to run it as Rich OS on Monitor. Thus, in terms of application developer's cost, since developers can reuse all existing programs including libraries, middleware, and applications running on Linux, no additional development cost is necessary. In terms of device developer's cost, configuration, such as network address setting of the Notification module, and memory address setting and security permission setting of TZASC is necessary to integrate the proposed system into a device. In addition to the development cost, verification cost in order to check that the configuration is correct is necessary. For embedded devices in a smart grid, there are cases where the performance requirement is specified. For example, in the case of a smart meter, it is reported that an acceptable delay in responding to a management server is in the range of 50-300[ms] under a specific condition [54]. As described in the performance analysis, since performance degradation is insignificant when introducing our proposed method, the cases requiring performance tuning are limited. Therefore, the development cost can be controlled.

2) Production cost

   The proposed recovery system is software-based technology and no additional hardware except a TrustZone-capable ARM processor and an address space controller is required. TrustZone-capable processors are widely available. In fact, all ARM Cortex A series processors support TrustZone. Therefore, the additional cost is mitigated. As a result, development cost per device can be minimized.

3) Maintenance cost

   It is assumed that a tremendous number of devices will be deployed in the field for smart grids. In the case of a cyber-attack, since many devices could be a target of the attack and the attack could be done in a very short period of time through the network, it is impracticable in terms of both cost and time for field service engineers to physically visit each site and reboot them. The auto-recovery feature of the proposed recovery system mitigates this problem. Moreover, the

report is sent to the head-end system once the device reboots. This function contributes to reduction of the cost of troubleshooting. Thus, the proposed recovery system provides an opportunity to reduce maintenance cost compared with legacy systems.

Chapter 5

Secure mobile agent system

In this chapter, we present a secure mobile agent system, utilizing the proposed method described in the previous chapter as underlying technology, and apply it to Field Area Network (FAN) in smart grids in order to achieve autonomous distributed smart grid architecture. First, we present an overview of a mobile agent system and its security threats. Although many mobile agent systems have been proposed, few studies address the problem of keeping secrecy and integrity of mobile agents, as most previous research endeavored to prevent attacks from agent to platform or from agent to agent. We propose a secure mobile agent system in order to keep secrecy and integrity of mobile agents. The method enables mobile agents to execute their processes on untrusted mobile agent platforms. We demonstrate a full implementation of the proposed secure mobile agent system. We also present experimental results of the proposed system. Furthermore, we propose to apply the secure mobile agent system to smart grids, in particular, to FAN in smart grids. Finally, we present some new application examples that were previously difficult to achieve.

5.1 Security threat to mobile agent system

5.1.1 Mobile agent system

A mobile agent system is a distributed system where a program called a mobile agent autonomously moves from one host to another connected through a network [55]. The mobile agent working on a mobile agent platform performs various tasks by using resources on the platform or

communicating with other agents, and achieves its goal on behalf of its owner. The key feature of a mobile agent system is that the execution state of the mobile agent is saved when leaving a host, transmitted with its execution code, and reused on another host.

Therefore, the mobile agent can get information depending on each host, process it, and use the result of the process on a different host by traversing hosts.

In a smart grid, a tremendous number of connected embedded devices will be deployed and many kinds of applications will work on the devices. If a mobile agent system is applied to a smart grid, it will be very useful and contribute to cost reduction since a mobile agent system offers several advantages, including reduced communication costs, asynchronous task execution, dynamic software deployment and ease of development [56]. However, it is difficult to use existing mobile agent systems because many security problems remain although many mobile agent architectures and implementations have been proposed. In particular, there is a high risk of illegitimate interception of data and code managed by the mobile agent whose secrecy and integrity need to be kept or of corruption of content of working memory used by the mobile agent when a target host platform and network are untrustworthy. To treat those risks, cryptography may be useful. For example, it is possible to keep secrecy and integrity of data managed by the mobile agent through a communication channel for encrypting and signing the data when migrating to hosts. However, since the mobile agent executes the decryption process on the platform, the data could be intercepted or modified if the platform were malicious or illegitimately modified. To solve those problems, some approaches have been proposed, such as obfuscating the mobile agent execution code, or attesting to the integrity of a target host platform with dedicated hardware. However, those approaches do not tackle the root of the problem. In the following section, we will look the security threats in depth.

5.1.2 Security threat

In order to identify the possible source and target of an attack, we summarized security threats to a mobile agent system corresponding to the following four based on the NIST classification from the viewpoint of the components of the mobile agent system [57].

1) Agent-to-Platform

   The mobile agent runs on the mobile agent platform. When the mobile agent is malicious, it may attack the platform. The attacks include masquerading as an authorized agent to gain access to services and resources to which it is not entitled, unauthorized access to services and resources by bypassing access control mechanisms, modifying and damaging the platform to exploit security faults on the platform, and denial of service by consuming a tremendous amount of platform resources.

2) Agent-to-Agent

   In a multi-agent system, the mobile agents communicate and exchange data with one another to accomplish their tasks. When the mobile agent is malicious, it may masquerade to deceive other agents to gain unauthorized information from them or to cause them to misbehave by sending false messages. Moreover, the malicious agent may launch a denial-of-service attack by sending spammed messages or tremendous amount of messages.

3) Platform-to-Agent

   When the mobile agent migrates from host to host, the code and its context are transmitted through an unprotected network. Similarly, the agent may exchange messages with other agents or remote platforms through an unprotected network. When the network intermediate device is malicious, there is a risk of those messages being illegitimately modified or eavesdropped, resulting in overwriting or theft of the information. It is possible to prevent the attack by establishing secure channels between platforms. However, if the mobile agent platform is malicious, the problems are more serious. Since the code of mobile agents and their context must be in plaintext on the mobile agent platform to execute the mobile agents, the platform can monitor any instructions and data of the mobile agents, modify the mobile agent code and its context, or disturb the execution of the mobile agents without being noticed by the mobile agents; posing the same threats on the network.

4) Other-to-Agent Platform

   Since the mobile agent platform is on the network, there is a risk that a remote attacker may try to penetrate the system and modify the

mobile agent system or gain control of resources by exploiting security faults. Furthermore, a system administrator could be malicious. Therefore, even if the mobile agent platform itself is not malicious, if a system below the platform, such as an operating system is modified or the system administrator is malicious, the same problem described in 3) occurs.

Many agent architectures and implementations with security functions have been proposed and some of them try to address the challenges described above.

Agent Tcl supports secure communication and agent transfer [58]. Each mobile agent platform of Agent Tcl manages its own public and private key. When the mobile agent is transferred, it is encrypted and signed by the mobile agent platform so that the secrecy of the mobile agent can be kept through the network and a remote mobile agent platform can confirm that the mobile agent is not modified.

Voyager is a Java-based mobile agent platform. Because Voyager uses a Java virtual machine (Java VM) as a mobile agent platform, a mobile agent runs in a sandbox, which is designed to protect a host from misbehaving or malicious mobile agent code [59]. Aglets is also a Java-based mobile agent platform [60]. Aglets provides a platform authentication mechanism where mobile agent platforms mutually authenticate one another before the mobile agent moves to a target remote host. Since transfer of the mobile agent to an untrusted platform can be prevented, it is possible to keep the secrecy of the mobile agent. Moreover, Aglets provides an access control mechanism to protect the mobile agent from receiving unauthorized messages. It defines a security policy that describes a sender of the message and approved actions so that the mobile agent can determine whether the received message should be accepted or not.

In this manner, most previous research endeavored to remove threats 1) and 2) whereas very few attempts were made to remove threats 3) and 4). If threats 3) and 4) cannot be removed, an application of a mobile agent system will be severely limited. For example, it is difficult for the owner of the mobile agent to keep data which the mobile agent gets on one host secret even if it migrates to another host, or to keep the algorithm implemented inside the mobile agent secret without removing threats 3) and 4). Furthermore, the previous research assumes that the mobile agent platform works correctly. However, it is impossible to prevent all possible attacks and remove all security faults in practice. In fact, much

vulnerability is frequently reported [8] and middleware such as a Java VM or an operating system such as Linux is no exception. If the mobile agent platform is modified or tampered with by the malicious owner of the platform, methods proposed in previous research are invalid. Moreover, in most previous research, mobile agent platforms are built on virtual machines or middleware that typically works on an operating system and the mobile agents are under the control of the mobile agent platform. Therefore, it is difficult for the mobile agent to detect that the mobile agent platform is malicious and is attempting an attack. Moreover, even if the integrity of the mobile agent platform is verified, if the owner of the host is untrustworthy or the operating system is modified, there is a high risk that the secrecy and integrity of the mobile agent platform cannot be kept.

5.1.3 Apply secure mobile agent system to smart grids

In smart grids, a communication network can be represented by a hierarchical multi-layer architecture [61]. Figure 5.1 shows an example of network architecture of a smart grid. Typically, it comprises Wide Area Network (WAN), Field Area Network (FAN), and Home Area Network (HAN). Since requirements are different for each network, different communication technologies are used for each network. In particular, since a tremendous number of connected embedded devices will be deployed and many kinds of applications will work on the devices in FAN, several network architectures are considered in order to operate and manage devices efficiently. For example, Tokyo Electric Power Company lists candidate communication systems for application in FAN [62]: RF mesh network where data are transmitted via other wireless terminals, wireless star network where data are transmitted between a base station and a wireless terminal directly, and Power Line Communications (PLC) where electrical power lines are used as communication lines. Each of them has advantages and disadvantages in terms of efficiency, robustness, and ease of maintenance.

Gungor surveys network architecture for electrical systems and indicates that a wireless sensor network can enhance the performance of electric utility operations for automatic meter reading and reliable, real-time monitoring [63][64]. As well as Gungor, Gharavi presents mesh network architecture for a smart grid [65]. In a mesh network, only Internet gateway devices connect with a head-end system, relaying messages and data, and other devices directly communicate with one another. The model is very similar to a mobile agent system. In a mobile agent system, an application program

autonomously moves from one host to another connected through a network. Therefore, if a smart meter is regarded as a mobile agent platform, the topology is well suited to mobile agent systems. Furthermore, it is very useful since the advantages of the mobile agent system, including reduced communication costs, asynchronous task execution, dynamic software deployment and ease of development, are applicable to the mesh network. Particularly, Zhabelova proposes to introduce multi-agent model in a smart grid and the simulation result indicates its efficiency [66]. Similarly, Hernandez presents a multi-agent system model for virtual power plants by utilizing intelligence of agent in order to improve the precision of the prediction of future energy demand [67]. Pipattanasomporn presents that mobile agent system can enhance management capability in the distributed smart grids by utilizing its flexible and updatable feature [68].

Although it has great advantages, the lack of security may restrict the network architecture since there is a great risk that intermediate nodes will eavesdrop or modify data, prompting utilities to hesitate to introduce a particular network architecture. Therefore, if we can provide a method to prevent the threats, it would create a good opportunity to enhance the performance of electric utility operations.

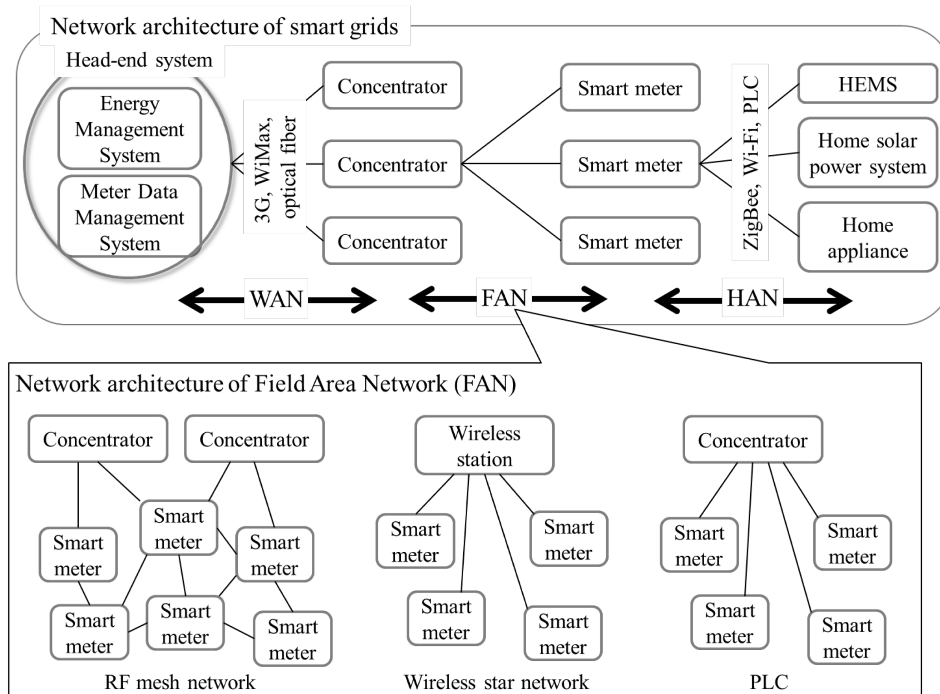

Figure 5.1: Network architecture of a smart grid.

5.2 Architecture of the secure mobile agent system

The proposed secure mobile agent system provides a secure execution environment on which a part of a mobile agent that needs to be protected executes securely. Hence, even if a mobile agent platform or an operating system on which the mobile agent platform runs is modified by an attacker, the part of the mobile agent is still securely executed without illegitimate modification and eavesdropping. Figure 5.2 shows the entire architecture of the proposed secure mobile agent system. It consists of four major components: Mobile Agent, Mobile Agent Platform, Secure Execution Environment (SEE), and Monitor.

- Mobile Agent: Mobile Agent is an autonomous program whose migration from one host to another is under its control. In the proposed secure mobile agent system, Mobile Agent consists of the Basic module, the Secure Mobile Agent (SMA) module, and context. The Basic module is a program setting up an initialization process to execute the SMA module, accessing file and network resources and communicating with other Mobile Agent by using functions provided by Mobile Agent Platform, and executing various processes whose secrecy and integrity do not need to be kept. The SMA module is a program independent from the Basic module to execute processes whose secrecy and integrity need to be kept. The Basic module and Secure Mobile Agent cooperate with each other to exchange data via shared memory. The Basic module is executed on Mobile Agent Platform whereas the SMA module is executed on SEE. Context is an execution state of the Mobile Agent. When Mobile Agent migrates, all three elements are transferred to a remote host.

- Mobile Agent Platform: Mobile Agent Platform provides an execution environment to run Mobile Agents. It also provides migration of Mobile Agent. It saves the context of Mobile Agent, transports the code of Mobile Agent and saved context to the target remote host, and resumes execution from the saved context. Most of the mobile agent platforms proposed in previous research are built on a middleware, such as Java VM, which is executed on an operating system. Since the proposed secure mobile agent system supports Linux as the operating system, any mobile agent platform running on Linux is able to run on the proposed secure mobile agent system. Besides Mobile Agent Platform, the operating system executes general-purpose processes, such as storage access or network

communication, implemented as native applications or device drivers. The operating system, native applications, Mobile Agent Platform, and the Basic module of the Mobile Agent are all executed in the non-secure world.

- Secure Execution Environment (SEE): SEE includes two modules: The Common module and the Installer module. The Common module is an entry point of SEE. It initializes context of SEE, starts the Installer module, and executes it when booting a system. It also calls the SMC instruction to switch to the operating system executed in the non-secure world when necessary. The Installer module reads the SMA module from shared region and executes it on SEE. If the SMA module is encrypted and signed, the Installer module verifies and decrypts it before executing it.

- Monitor: A program running in the monitor mode. It initializes configurations of TrustZone-related hardware when booting the system. It also provides a context switching function between worlds in the SMC handler. Moreover, Monitor contains Policy Manager which manages the access control policy and installs the policy on TZASC when booting.

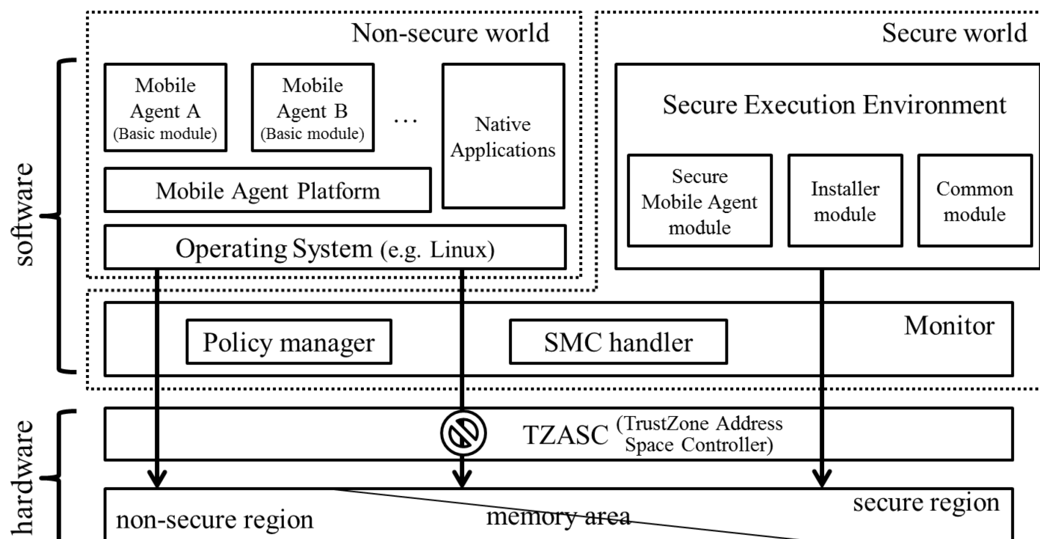

Figure 5.2: Architecture of the secure mobile agent system.

5.3 Functions of the secure mobile agent system

The primary feature of the proposed mobile agent system is provision of a method that gives mobile agents a secure environment, thus preventing them from being subject to modification or eavesdropping even if a mobile agent platform or an operating system is illegitimately modified.

Since the proposed mobile agent system is based on the proposed method described in the previous chapter, the architecture is similar with the ones depicted in Figure 4.1 and Figure 4.9. However, we will describe each component in detail for clarification in this section.

1) Memory access control

Monitor provides an access control function such that access of the operating system running in the non-secure world to the working memory, which SEE running in the secure world uses, is subject to restrictions. When booting the system, the processor is in the secure world and Monitor is firstly executed. Policy Manager in Monitor configures TZASC to install memory access policy. Table 5.1 shows how each policy works and how each policy is applied.

Table 5.1: Access control policy and its mapping

Policy	*From secure world process*	*From non-secure world process*	*Applied to*
Full access	OK	OK	Non-secure region Shared region
Access denied	OK	NG	Secure region

Policy Manager in Monitor manages two kinds of access control policies: full access and access denied. Full access indicates no restriction. A process running in both the non-secure world and the secure world can freely access the region configured according to this policy. Access denied indicates full restriction. A process running in the non-secure world can neither read nor write to a region configured according to this policy, whereas a process running in the secure world can read and write to the region. In the initial state before booting the system, all regions are allowed to be accessed from the non-secure world by default. When booting the system, Policy

Manager divides the working memory into three regions: non-secure region, shared region, and secure region. It applies full access policy to non-secure region and shared region whereas it applies access denied policy to secure region before starting the operating system. Note that although the policies applied to non-secure region and shared region are identical, we refer to the regions by different names for clarification. Secure region is divided into several sub-regions: code and data area for an initial code, Monitor, and SEE. The size of each sub-region is predefined.

2) Mobile agent migration

Mobile Agent Platform provides a function for Mobile Agent to migrate between hosts. Migration is invoked by the request of Mobile Agent. Three elements are transferred to the remote Mobile Agent Platform: code of the Basic module, code of the SMA module, and context. Since the code of the Basic module and the SMA module is a file whereas context is not, the context must be transformed into data format to be able to be transmitted. Mobile Agent Platform collects context of the Mobile Agent from the working memory, assembles the three elements into a form suitable for transmission, and transmits to the remote Mobile Agent Platform. The remote Mobile Agent Platform receives and disassembles them, puts the context on the working memory so that Mobile Agent can restart the process, and installs and executes the SMA module.

3) Context switch between worlds

To execute the operating system and SEE concurrently in the same host, Monitor provides a context switching function between worlds. General registers and Saved Program Status Register are not banked between worlds. Therefore, Monitor saves the contents of the registers belonging to the current world on working memory to prevent loss of the previous context, and then changes the setting of the world. Finally, it restores the contents of the registers belonging to the transition destination world and resumes the execution. In SEE, a device driver of the operating system and a native application implemented in Mobile Agent Platform are provided to call SMC instructions from Mobile Agent since the Basic module can neither directly access arbitrary memory regions nor call CPU native

instructions. For the initialization process, Monitor first initializes the status of the processor in both worlds, and executes SEE in the secure world. Then, it invokes context switching to transit from the secure world to the non-secure world and executes the boot loader program of the operating system in the non-secure world. Finally, the boot loader program executes the operating system and the operating system executes Mobile Agent Platform.

4) Installation and Execution of the SMA module

Mobile Agent consists of the Basic module and the SMA module and it processes various tasks by communicating one another. Since the processing details are different for each Mobile Agent, code of the SMA module is also different. It is not feasible to implement and install all security-sensitive functions of the SMA module in SEE in advance. Therefore, in the proposed mobile agent system, the SMA module is transferred with the Basic module attached as a file to the remote Mobile Agent Platform and enabled before the Basic module calls it each time Mobile Agent migrates. The Basic module requests Mobile Agent Platform to install and execute the SMA module with a file containing the transmitted SMA module. Mobile Agent Platform reads it from storage and writes it on the shared region. The Installer module in SEE reads it from the shared region and executes it in the secure world.

Figure 5.3 illustrates the protocol when the SMA module needs to be protected. After completion of developing the SMA module, an SMA module developer encrypts it with a program key (K_{prog}) that is generated by the developer and unique to each developer or each Mobile Agent. Then, the hash value of the encrypted SMA module is calculated and signed with a private key (K^{-1}_{sig}) in order to generate a signature. The program key is encrypted with an encryption key (K_{enc}). The encrypted SMA module, the encrypted program key, and the signature are transferred to Mobile Agent Platform. After they are received and written on the shared region, the Installer module reads them from the shared region. The Installer module manages a decryption key (K^{-1}_{enc}) corresponding to the encryption key (K_{enc}) inside. It decrypts the encrypted program key with the decryption key (K^{-1}_{enc}) and retrieves a plaintext program key. Then, it decrypts the encrypted SMA module with the plaintext program key. It verifies the

signature with the public key (K_{sig}) corresponding to the private key (K^{-1}_{sig}) and if it fails, it stops installing. Otherwise, it installs the plaintext SMA module on secure region. Because both decryption and verification processes are executed in the secure world, an attacker can neither get nor modify the plaintext SMA module even if the attacker modifies the operating system or Mobile Agent Platform. To make a trust chain of keys, Public Key Infrastructure (PKI) can be introduced to the encryption key (K_{enc}) and the public key (K_{sig}) when deploying in the market. For clarification, when the SMA module is encrypted and signed, it contains the encrypted SMA module, the signature and the encrypted program key as explained below.

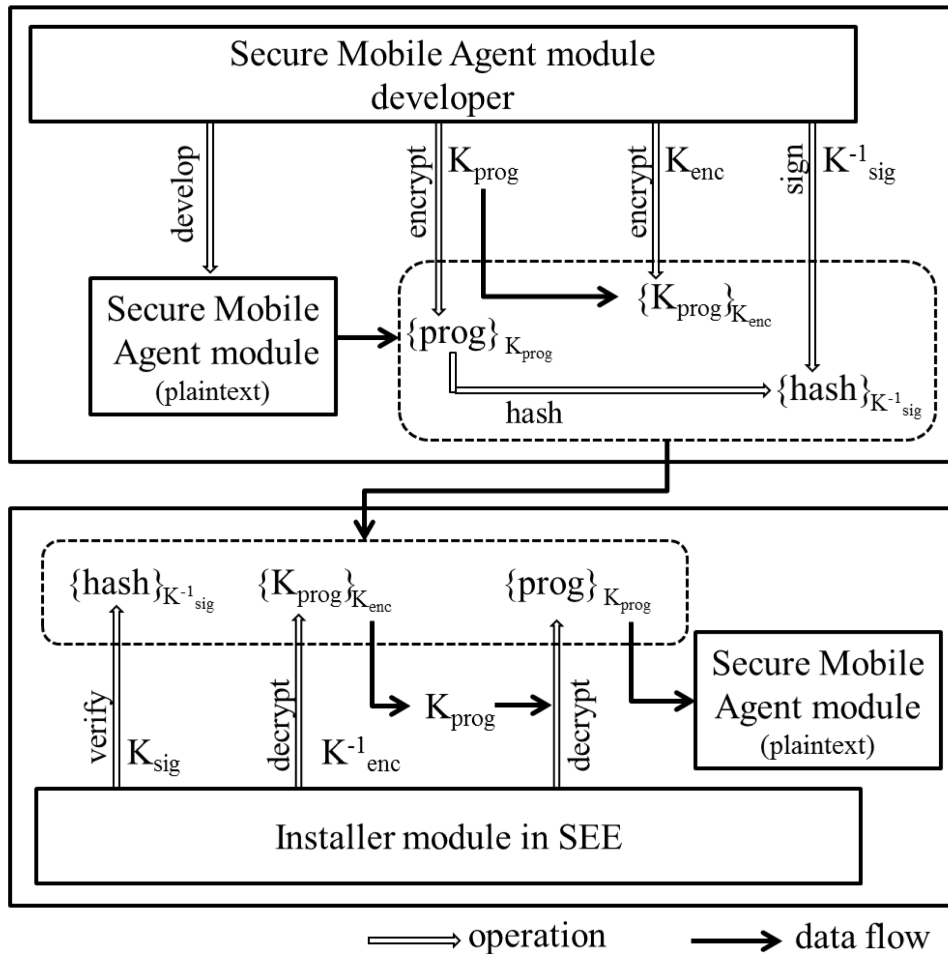

Figure 5.3: Process to develop and install the SMA module.

5) Mobile Agent internal interface

In the proposed mobile agent system, Mobile Agent is divided into the Basic module and the SMA module, and the modules communicate and exchange data with each other via the shared region. The proposed mobile agent system supports the data exchange function. Here, data include content data the SMA module uses, or an operation that is an instruction to the SMA module. The Basic module requests Mobile Agent Platform to send data. Mobile Agent Platform writes the data on the shared region. Then, Monitor context switches from the non-secure world to the secure world. Finally, the SMA module reads the data from the shared region.

When designing Mobile Agent, developers need to clarify which function belongs to which module and they need to define an internal interface between the Basic module and the SMA module. The internal interface includes data structure and coding rule of the operation. Furthermore, when the size of data is large or the operation consists of several steps, the Basic module requests context switch several times. Although it is easy for the developers to use a data exchange function since Mobile Agent Platform provides API, the data exchange, including context switch needs processing time. Consequently, in order to prevent performance degradation, the developers need to consider how often and how many times the Basic module calls the data exchange function.

6) Encryption key and host restriction

From the viewpoint of security architecture, the fact that the decryption key (K^{-1}_{enc}) is a shared key between SEE of different hosts poses no problem, since it is managed by the Installer module and an attacker cannot get the decryption key even if it successfully modifies Mobile Agent Platform or the operating system. However, it is vulnerable once the decryption key is leaked from one of the hosts for any reason, for example, mismanagement attributable to a human factor. Therefore, in view of the possibilities of security incidents, it is preferable to assign unique key.

From the viewpoint of an application provider, it is also preferable that a unique decryption key (K^{-1}_{enc}) be assigned to each host. Let us assume that the owner of Mobile Agent wants to restrict the host in terms of where the SMA module is allowed to be executed or restrict the number of hosts where the SMA module is allowed to be installed.

If a shared key is used, since all hosts have the same decryption key (K^{-1}_{enc}) and can decrypt the SMA module, it can neither restrict the host, nor identify each host. In contrast to a shared key, if a unique key with public key algorithm is used, it becomes possible to restrict a host allowed to execute the SMA module. Although it has not been implemented, if SEE provides a function to encrypt the program key (K_{prog}), it encrypts the program key (K_{prog}) with the encryption key (K_{enc}) corresponding to the decryption key (K^{-1}_{enc}) of a host allowed to execute the SMA module before Mobile Agent migrate to the host. Depending on the application, the encryption keys (K_{enc}) of the allowed hosts are either included in the SMA module when distributed Mobile Agent, or the SMA module dynamically collects and chooses the host.

5.4 Process flow

In this section, process flows of the proposed mobile agent system are described to summarize the functions described above.

1) Installation and uninstallation of the SMA module

Figure 5.4 depicts the process flow when installing the SMA module. The explanation below assumes that the SMA module is encrypted and signed. When receiving Mobile Agent from a remote host, Mobile Agent Platform also receives the SMA module, including the encrypted program key and the signature, as an encrypted file. The Basic module of Mobile Agent requests Mobile Agent Platform to install the SMA module (process (1)) and Mobile Agent Platform reads the encrypted SMA module from a disk, writing them on the shared region (process (2, 3)). Then, Mobile Agent Platform requests the operating system to switch to the secure world. The device driver of the operating system calls SMC instruction to switch to the non-secure world with the operation that this world transition is to install the SMA module (process (4)). SMC handler in Monitor context switches from the operating system to SEE (process (5)). The Common module identifies that the operation is installation of the SMA module and calls the Installer module (process (6)). The Installer module manages the public key (K_{sig}) and the decryption key (K^{-1}_{enc}). It first verifies the signature. Then, it decrypts the encrypted program key with the decryption key (K^{-1}_{enc}) and retrieves a plaintext

program key. It decrypts the encrypted SMA module to read it from the shared region with the plaintext program key.

Since the size of the encrypted SMA module is generally larger than the block size of decryption, it cannot read the entire module at once. Therefore, the Install module reads it by some block units, copies the block units to secure region, calculates a hash value, and decrypts them with the plaintext program key (process (7)). And it appends the plaintext block units to the temporary area of the secure region. The calculation and decryption continue until the entire module is processed. Then, it verifies the signature based on the hash value for the entire module with the public key (K_{sig}). If the verification fails, it stops installing and switches back to the operating system via SMC handler as an error status. Otherwise, it copies the plaintext module from the temporary area to the area where the code of the SMA module is placed in the secure region (process (8)). Finally, the Common module calls SMC instruction to switch back to the operating system (process (9)). SMC handler in Monitor context switches from SEE to the operating system (process (10)). In this manner, it is ready for the Basic module to use the SMA module.

Uninstall is a process to clear the memory area where the SMA module is placed. The uninstall process is executed when Mobile Agent explicitly requests, for example, when migrating to another host. Mobile Agent Platform writes the operation indicating that the operation is uninstallation and requests the operating system to switch to the secure world. The Common module identifies that the operation is uninstallation of the SMA module and clears the memory area that the old SMA module uses. The uninstall process is also executed just before being overwritten by the new SMA module. When the new SMA module is installed, the Installer module simply overwrites the old SMA module in the new SMA module. If the owner of the new SMA module is different from the old one and the size of the old SMA module is larger than the new one, the new SMA module can access the data of the area where it was not overwritten, resulting in information leakage. Therefore, it is necessary to uninstall the old SMA module before installing the new one. In this case, the Installer module clears the memory area that the old SMA module uses.

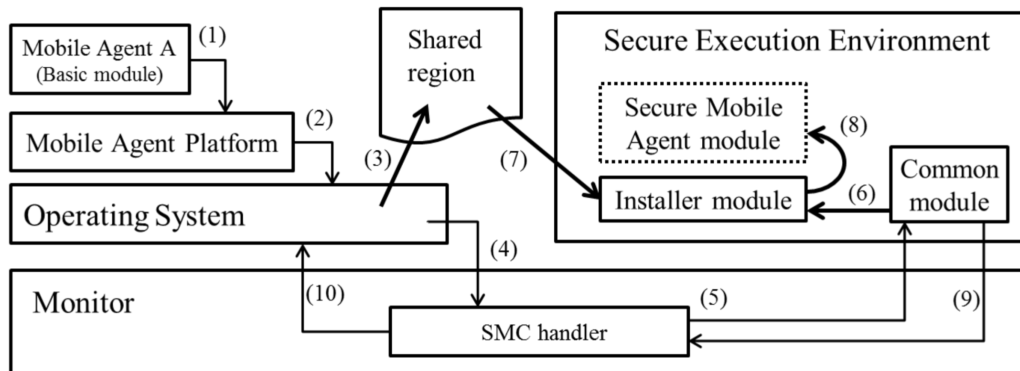

Figure 5.4: Execution flow of Secure Mobile Agent module installation.

2) Communication between the Basic module and the SMA module

The process flow when the Basic module communicates with the SMA module is very similar to the flow of installation of the SMA module. The explanation below shows an example in which the Basic module requests the SMA module to encrypt plaintext data and gets encrypted data processed in the secure world. First, the Basic module requests Mobile Agent Platform to execute the SMA module attached with a plaintext data. Mobile Agent Platform writes the plaintext data on the shared region. Then, Mobile Agent Platform requests the operating system to switch to the secure world. The device driver of the operating system calls SMC instruction to switch to the secure world attached with the operation that this world transition is to call the SMA module. SMC handler in Monitor context switches from the operating system to SEE. The Common module identifies the operation and calls the SMA module. The SMA module manages a key to encrypt data. It reads the plaintext data from the shared region, encrypts the plaintext data with the key. The SMA module executes the encryption process including intermediate data in the secure world. Then, it writes the encrypted data on the shared region. Finally, the Common module calls SMC instruction to switch back to the operating system. SMC handler in Monitor context switches from SEE to the operating system. The operating system resumes its processes, including the execution of Mobile Agent Platform.

The operating system stops working while the SMA module works. If the data size becomes large, the time necessary to encrypt the data becomes long. Consequently, the suspension time of the

operating system becomes longer. For end-point devices in a smart grid, there are cases where some tasks coexist and some of them are not required to be protected, but long suspension time is unacceptable. For example, when devices need to measure data from a sensor every period of time, or devices need to respond to a server within a certain amount of time after receiving data, they may miss measuring data or delay to respond if the suspension time is very long. Mobile Agent developers need to design the interface to avoid these situations. For example, it is necessary to design the SMA module so that it suspends in the middle of its task and switches back to the operating system. Since the upper limit time of suspension depends on an application, Mobile Agent developers also needs to design the data structure and the size of data that the Basic module writes on the shared region at one time to exchange data with the SMA module.

5.5 Prototype implementation

We used ARM C/C++ Compiler 5.01 to build Monitor and SEE. We used gcc 4.4.1 to build Linux 3.6.1 as the operating system. We used JDK 1.8 to build Mobile Agent Platform. We chose Motherboard Express uATX with the CoreTile Express A9x4 processor that supports TrustZone as an execution environment.

We developed very simple Mobile Agent Platform and the Basic module in the Java environment using Remote Method Invocation (RMI). Java RMI provides an infrastructure where the method of remote object executed on Java VM of a remote host can be invoked from an object executed on Java VM of different hosts [69]. Furthermore, Java supports object serialization that transforms an object into bytecode that can be transmitted over a network [70]. By utilizing these technologies, we implemented an RMI server that provides methods available to incoming objects and serializable objects that call remote methods on the RMI server. The RMI server and the serializable object are regarded as “Mobile Agent Platform” and “Basic module”, respectively. In order to share data via the shared region, a program needs to access memory managed by the operating system. Furthermore, SMC instruction can only be executed in privileged mode. However, Java applications can neither directly access memory, nor directly call CPU native instruction. Therefore, we implemented a device driver that executes SMC instruction and provides an interface that allows a native application to call SMC instruction. We also implemented Mobile Agent Platform with proxy methods that invoke

a native application program running on the operating system. The native application writes the encrypted SMA module and data to be processed in the secure world on the shared region, and calls the device driver to execute SMC instruction based on the request from the Basic module through Mobile Agent Platform.

Monitor supports Linux 3.6.1 as the operating system. Regarding a memory map, from 0x48000000 through 0x4A000000 is assigned for SRAM, and from 0x60000000 through 0xE0000000 is assigned for DRAM. Table 5.2 shows the memory map with the access control policy of the memory. In Table 5.2, the operating system indicates the Linux kernel code and data, and application code and data. For clarification, full access is applied for an area not described in Table 5.2.

Table 5.2: Memory map of the virtual security hardware module

Data	***Start address***	***Size***	***Security permission (From non-secure world)***
Vector tables + Initialization code	0x60000000	0x00008000	Access denied
Operating system	0x60008000	0x2FFF8000	Full access
Monitor + SEE	0x90000000	0x01000000	Access denied
SMA module	0x91000000	0x0E200000	Access denied
Shared region	0x9F200000	0x00E00000	Full access

For the Policy Manager in Monitor to install an access control policy on TZASC, the start address and the size of each memory region are predefined. After the boot loader loads Linux at the predefined value, Monitor installs the access control policy on TZASC. Since Monitor sets the configuration registers of TZASC to prohibit a program running in the non-secure world from accessing them, the operating system cannot change this configuration.

We implemented the Installer module with 128 bit AES algorithm in CBC mode to decrypt the encrypted SMA module and the encrypted program key, and with 1024 bit RSA algorithm to verify a signature of the SMA module. When the SMA module has already been installed, the Installer module clears and overwrites the memory area where the original SMA module is located when the new SMA module is installed. The Installer module installs the SMA module only when its size does not

exceed the predefined size of the area provided for the SMA module. Since the Installer module just copies the plaintext SMA module on the memory without distinguishing code and data, developers need to generate a binary image of the SMA module so that the binary image becomes identical to the image on the memory. Similarly, the Basic module or the SMA module can write data on the shared region, whose size is predefined, only when the size of the data does not exceed the size of the shared region. If they need to exchange data, whose size exceeds the size of the shared region, they need to divide data so that the size does not exceed it, and context switch each time they write the chunk of data on the shared memory until all data are sent.

We implemented the SMA module with an encryption function with 128 bit AES algorithm in ECB mode. The size of code and data is 8[KB] and 2.4[KB], respectively. Padding process has not been implemented.

We have not implemented a function whereby SEE automatically saves and restores the context of the SMA module at an arbitrary point. Instead, the SMA module sends its intermediate data to the Basic module via the shared region if Mobile Agent needs to restart its process on SEE of a remote host after migration. Although it is possible to execute several Basic modules on Mobile Agent Platform, only one SMA module can be installed in SEE at one time. Therefore, Mobile Agent needs to reinstall its own SMA module if another Mobile Agent installs its SMA module since the SMA module has been overwritten by another Mobile Agent.

When building SEE and Monitor, we configured them so that they are the same binary module.

Monitor does not support multi-core.

5.6 Evaluation

5.6.1 Security and cost analysis

1) Basic security property

   In the evaluation of a security system, a key aspect is the preservation of confidentiality, integrity and availability. Regarding confidentiality, the proposed mobile agent system can keep the secrecy of mobile agents, including that of data and code. Since memory access control is configured when booting the system and the protection-required module is executed in the secure world, it is impossible for attackers to intercept data and code even if they modify the mobile agent

system or the operating system. The mobile agent system consists of various subsystems and most of the subsystems do not require security-sensitive processes. For example, network access or the file access utility process itself does not contain secret information. It is almost impossible to exclude all vulnerability from an entire system, and especially so when the system is large. To make matters worse, users of mobile agents or administrators of mobile agent platforms cannot maintain the entire system by themselves because the middleware on which the mobile agent platforms run, such as Java VM or the operating systems are developed by someone else, or some parts of the components are proprietary software and their source codes are not disclosed. In those cases, it is difficult to exclude the vulnerability even if the components of the system are known to be vulnerable. Rather, it is practicable to divide a system between a small security core subsystem and general-purpose subsystems, and to focus on carefully checking the code of the security core subsystem only. In fact, the source code of Linux 3.6.1 comprises over 15 million lines whereas that of Monitor, SEE, and the SMA module comprises 1100, 3800, and 1300 lines, respectively. Therefore, the volume of the code of the security core subsystem is sufficiently small to enable careful review and testing in order to exclude vulnerability. Moreover, since the proposed mobile agent system provides a secure isolated execution environment for the SMA module by utilizing the memory access control function and the context switch function, attackers cannot get the code and data of the security-sensitive core subsystem consisting of Monitor, SEE and the SMA module even if they can take control of the operating system or the mobile agent platform. Furthermore, since the SMA module is transferred over the network in an encrypted manner, the attackers cannot eavesdrop on the plaintext code and data of the SMA module on the network. Similarly, because the key to decrypt the SMA module is managed and the decryption process is executed in the secure world only, the attackers cannot get the plaintext SMA module on the host.

Regarding integrity, the proposed mobile agent system can detect the illegitimate modification of mobile agents. Since the proposed mobile agent system provides a method to attach a signature with the SMA module, it is possible to detect illegitimate modification to verify the signature before executing it. In the same manner as the SMA module encryption process, since the verification process is

executed in the secure world only, attackers can neither modify the verification process nor skip it. Furthermore, since the SMA module is executed on SEE only when the signature verification succeeds, the risk of SEE or Monitor being attacked by the illegitimate SMA module is mitigated unless developers of the SMA module are malicious.

Regarding availability, the proposed mobile agent system does not provide a method to prevent Denial of Service (DoS) attacks. For example, although it can prevent attackers from stopping the SMA module during execution of the SMA module, the attackers can corrupt or delete a file of the SMA module on the operating system or even shut down the entire system. We describe the limitations in the following section.

2) Flexibility, extensibility, and mobile agent usage

In PC-based systems, it is possible to include many security features with the supports of powerful processing power and sufficient computational resources. However, it is unreasonable to have rich functions in embedded end-point devices in smart grids due to the limited computational resources. Furthermore, necessary features are different for each embedded end-point devices in smart grids. It is desirable to provide a method to selectively install functions after deployment, depending on the features of devices. In order to keep flexibility, our proposed method provides a function to dynamically add and update a module after deployment by Mobile Agent migrating and installing the SMA module in a target device.

Moreover, the encryption algorithm and the key length depend on the application, or a new encryption algorithm may be developed in the future. Therefore, it is not feasible to provide the mobile agent platform with every variety of algorithm and key length in advance; rather, it is reasonable for each Mobile Agent to have the security functions necessary to complete its task and to migrate to a host with the SMA module by exploiting the features of Mobile Agent. Therefore, our proposed method provides extensibility to end-point devices.

The proposed mobile agent system allows mobile agents to exchange data. Mobile Agent developers can freely define an interface to exchange data between Mobile Agents or define a set of Mobile Agent among which sharing of data is allowed. For example,

by giving plural Mobile Agents a shared key, Mobile Agents having the same shared key can decrypt data securely to exchange data. Or, by giving a Mobile Agent a set of public keys of other Mobile Agents, the Mobile Agent allows only other Mobile Agents having the corresponding private key for decrypting data to exchange data.

Developers of Mobile Agent can also limit Mobile Agent Platform where Mobile Agent works. For example, they can build Mobile Agents with the SMA module having a set of public keys of SEE so that only SEE having the corresponding private key can decrypt and run the SMA module. They can also limit the number of times the SMA module is used by building Mobile Agent with the upper limit of times of decryption of the SMA module. In the case of previous research, it was necessary to provide a trusted server that counts the number of times of use since there is a risk of the counter being modified by attackers. However, on the proposed mobile agent system, they can build Mobile Agent with the upper limit of times of execution of the SMA module without connecting to the network because the execution of the SMA module is protected and there is no concern that attackers may modify the counter.

3) Cost

In terms of hardware cost per device, since the proposed mobile agent system runs on general ARM processor and does not require any additional dedicated hardware, no additional cost is necessary.

In terms of software development cost, the additional cost is mitigated since developers can reuse most of their software assets. Since the proposed mobile agent system supports Linux as an operating system, they can reuse all software assets built on Linux. Furthermore, since we build our prototype implementation of the Mobile Agent Platform on Java VM, it is easy to port functions of Mobile Agent Platform to the existing mobile agent platform written in Java. The only additional requirements specific to the proposed mobile agent system are division of the mobile agent into two parts – general-purpose processes and security-sensitive processes, defining the interface between the processes, and implementing the security-sensitive processes in C language. Although developers of mobile agents need to carefully review the security-sensitive code to exclude vulnerability, the additional work could be minimized since the code volume is small.

4) Limitations

There are some limitations and attacks beyond the scope of the proposed system.

First, we have not yet implemented auto-saving and auto-restoring the context of the SMA module in SEE. If Mobile Agent needs to migrate to a host with the context of the SMA module, the SMA module itself needs to save its context as a file, send the file and restore the context from the file. As well as migrating, the SMA module needs to save its context and restore it when reinstalling and restarting the SMA module to avoid losing the context when other SMA modules overwrite it.

Second, the proposed mobile agent system cannot prevent DoS attacks. For example, attackers can disturb the execution of the SMA module. Since SEE does not inspect the origin of issuing an operation, any Basic module can send it the operation although the Basic module cannot modify or eavesdrop on SEE. For example, malicious Basic module may try to send an install operation with a corrupted SMA module. As a result, the execution of the SMA module is suspended during verifying the requested SMA module; causing extra time to be required to complete its task. Malicious Basic module may also try to send an install operation with the SMA module copied from a legitimate Mobile Agent during execution of another Mobile Agent. Since the SMA module has a valid signature, the installation succeeds and the SMA module is overwritten by the instruction of the malicious Mobile Agent. As a result, the Mobile Agent needs to reinstall its SMA module; causing extra time to be required to verity the SMA module again, or the intermediate result of the calculation may be destroyed and it may need to restart the calculation from the beginning.

Third, the proposed mobile agent system does not prevent attacks on the middleware or the operating system. For example, when there is vulnerability in the middleware or the operating system and a malicious program reboots the system during executing the SMA module or deleting system files to corrupt the system, the proposed mobile agent system does not prevent those attacks.

Finally, there are physical attacks, such as a power analysis attack or differential fault analysis using professional tools. There are some devices with hardware tamper resistance to resist physical attacks in the market. Since this dissertation focuses on the software system, physical attacks are beyond the scope of this dissertation.

5.6.2 Performance analysis

Whereas our proposed system greatly enhances security, the performance degradation is inevitable since it needs context switch to execute security-sensitive processes. We measured installation and execution time of the SMA module in order to evaluate the extent to which our proposed system degrades performance.

1) Installation time of the SMA module

Since the operating system is suspended while installing the SMA module, the shorter the installation time becomes, the more the performance of the operating system improves. We measured the processing time necessary to install the SMA module in three cases: the SMA module is in plaintext, the SMA module is encrypted, and the SMA module is encrypted with a signature.

When the SMA module is in plaintext, we measured the transaction time of the Installer module to copy the SMA module from the shared region to the secure region including context switch. In our prototype implementation, the Basic module launches a native application that is executed as an external process from Java VM, and the native application writes the SMA module on the shared region and triggers context switch. We measured the overall time from requesting Mobile Agent Platform and installing the SMA module to getting the response measured by the Basic module. Figure 5.5 shows the result of the measurement. When the size of the SMA module is 1[KB], 10[KB], 100[KB], and 1[MB], the time measured by the Basic module is 80[ms], 87[ms], 89[ms], and 92[ms], respectively. When the size of the SMA module is larger, the time measured by the Basic module tends to become longer, but the difference is negligible since the execution time of memory copy is very short. The measurement time consists of two processes: the time of launching an external application program by the Basic module, and the time of context switch by Monitor and the time of memory access to copy the SMA module by the Installer module. We also measured the execution time of Monitor and the Installer module only in order to break down the overall time measured by the native application. When the size of the SMA module is 100[KB], the context switch and memory access are 9[ms]. The result indicates that the execution time of Monitor and the Installer module is much smaller than the overall time; implying that launching an external program out of Java VM takes most of the

execution time.

Next, in the case where the SMA module is encrypted, we measured the transaction time of the Installer module to decrypt the SMA module in addition to memory copy and context switch. Figure 5.6 shows the result of the measurement. When the size of the SMA module is 1[KB], 10[KB], 100[KB], and 1[MB], the time measured by the Basic module is 81[ms], 91[ms], 135[ms], and 580[ms], respectively. In the same manner as the previous case, when the size of the SMA module is larger, the time measured by the Basic module becomes longer. Since AES decryption process is added to the previous case, the time becomes longer than the previous case for every size.

Finally, in the case where the SMA module is encrypted and a signature is attached, we measured the transaction time of the Installer module to calculate hash and verify the signature in addition to memory copy, decryption, and context switch. Figure 5.7 shows the result of the measurement. When the size of the SMA module is 1[KB], 10[KB], 100[KB], and 1[MB], the time measured by the Basic module is 18.97[s], 18.99[s], 19.03[s], and 19.53[s], respectively. It takes much more time than the previous two cases since asymmetric key calculation cost is higher than the memory copy or symmetric key calculation cost. Furthermore, we have not optimized the code of RSA algorithm. There is no significant difference between the sizes of the SMA module. We measured the RSA signature verification process only. When the size of the SMA module is 100[KB], the time is 18.89[s]. The result indicates that the RSA signature verification process dominates the overall time. For clarification, the RSA signature verification process does not depend on the size of the SMA module. Although it depends on an application, the suspension time of the second order may be too long for the operating system and mobile agents. Hence, we modified the Installer module to set a timer so that the installation process is divided into several parts and it context switches to the operating system periodically when the timer expires. We also modified the native application to continue to call the Installer module until it finishes the installation process. The measurement shows that when the size of the SMA module is 1[MB], the time measured by the Basic module is 19.56[s] and 19.94[s] when the timer is set to 1[ms] and 100[us], respectively. The result indicates that the additional overhead is very small even if we introduce the time-out mechanism. In this mechanism, the situation that the

operating system is suspended for a long time can be mitigated.

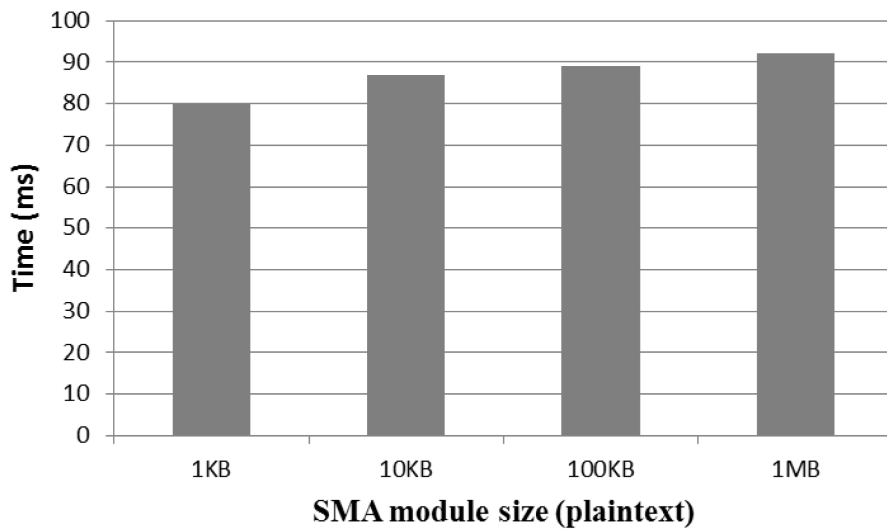

Figure 5.5: Performance result of SMA module installation (plaintext).

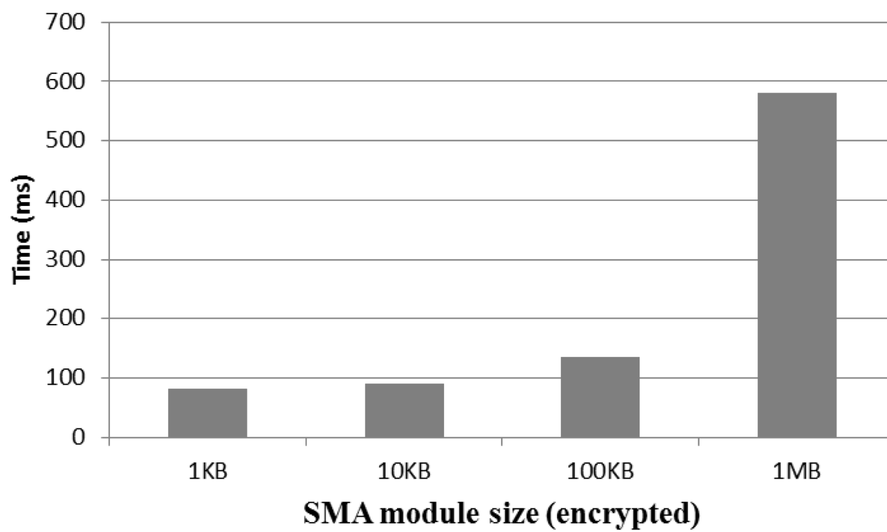

Figure 5.6: Performance result of SMA module installation (encrypted).

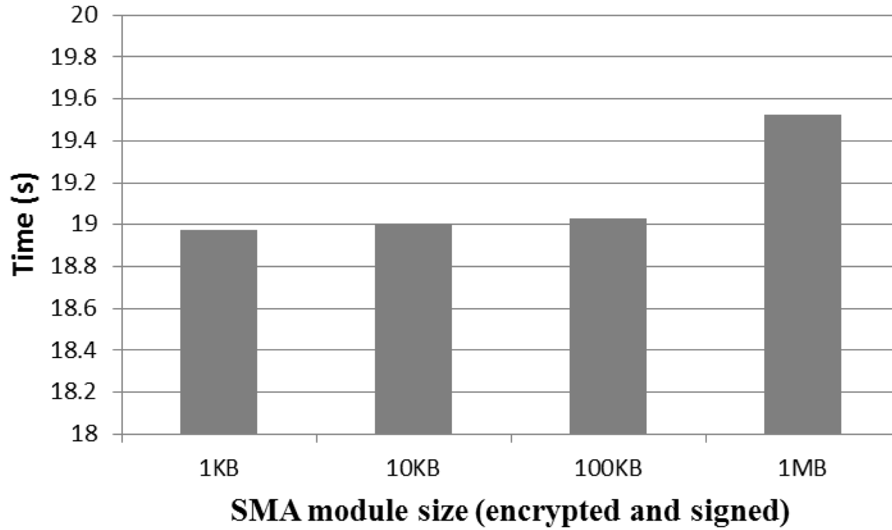

Figure 5.7: Performance result of SMA module installation (encrypted and signed).

2) Execution time of the SMA module

There is a relationship between the performance of the SMA module and the suspension time of the operating system. When we configure the SMA module to shorten the execution time at one time to divide processes into several chunks and to increase the number of times of context switch, the suspension time of the operating system decreases whereas the performance of the SMA module degrades since the overhead of context switch increases. It is difficult to set the target performance since the acceptable suspension time of the operating system and the target performance of the SMA module highly depend on the application. We measured the relationship to show developers of the Mobile Agent the guideline for designing it.

First, we developed the SMA module that encrypts data with 128 bit AES algorithm in ECB mode and a native application that communicates with the SMA module via the shared region. We also developed a test native application program with 128 bit AES algorithm in ECB mode for the evaluation purpose. We prepared 10[MB] random data and wrote them on the shared memory in advance, and measured the performance of the SMA module when

the data are divided into 16[B], 32[B], 64[B], 256[B], 1[KB], 4[KB], and 16[KB]. When the SMA module is executed, context switch occurs twice; one is from the non-secure world to the secure world, and the other is from the secure world to the non-secure world. Therefore, when block size is 16[B], context switch occurs 1310720 times (2×10×10242/16). Similarly, when block size is 32[B], 64[B], 256[B], 1[KB], 4[KB], and 16[KB], the number of context switch is 655360, 327680, 82920, 20480, 5120, and 1280 times, respectively. In this case, we implemented a test application program running on the operating system, which decrypts data with 128 bit AES algorithm in ECB mode without context switch, for evaluation purpose. Figure 5.8 shows the result of the measurement. When block size is 16[B], 32[B], 64[B], 256[B], 1[KB], 4[KB], and 16[KB], the throughput measured by the Basic module is 1.32[MB/s], 1.91[MB/S], 2.49[MB/s], 3.28[MB/s], 3.56[MB/s], 3.62[MB/s], and 3.59[MB/s], respectively. The throughput of the test application program is 3.79[MB/s]. The result indicates that when the block size is small, the overhead of context switch becomes large as expected. In fact, when the block size is 16[B], the performance degrades 64.3% compared with the performance of the test application program. However, when the block size is 256[B], 1[KB], and 16[KB], the degradation becomes 11.4%, 3.9%, and 2.9%, respectively, indicating almost no difference from the test native application. The result shows that choosing the appropriate block size mitigates the performance degradation.

Note that there is no data exchange between the Basic module and the native application in this measurement. It is not recommended to design Mobile Agent such that the Basic module calls the native application each time it requests to encrypt the block of the data since the overhead of launching the external process for Java VM is large. Rather, it is recommended that the Basic module calls the native application once and the native application calls context switch several times.

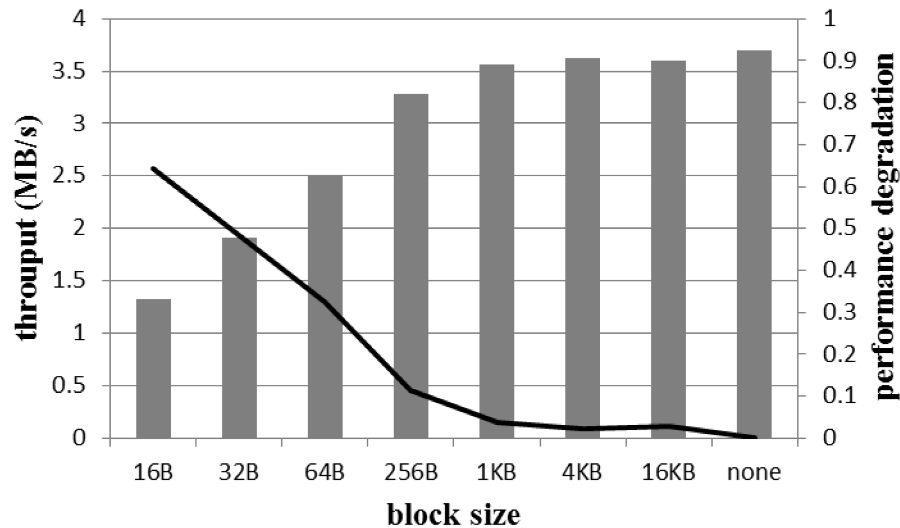

Figure 5.8: Performance ratio of the SMA module execution.

5.7 Application examples

1) Privacy information protection in a smart grid

   As shown in section 5.1.3, the proposed system can be applied to end-point devices in a smart grid. One of the primary services of a smart grid is demand response that dynamically changes the electricity tariff depending on the power consumption. To measure the electricity consumption, smart meters are deployed at each household. When we apply the proposed secure mobile agent system to smart meters, a mobile agent dispatched from a head-end system circulates among hosts deployed at each household while collecting power consumption data from smart meters, and returns to the server after collecting all the data. Besides collecting data, the mobile agent may execute calculation that the server performed in a legacy system, on residential devices, such as calculating average power consumption or forecasting the future trend of power consumption from historical data based on a particular statistical model. Although power consumption data of each household is not each resident's secret information, privacy is an issue concerning the data since it includes when and how much electricity the resident consumes, which

corresponds to the resident's activity. Therefore, the data collected at other residences must be kept secret whereas the data collected from the resident can be handled in plaintext. Otherwise, power consumption data may be disclosed for each neighborhood by analyzing the context of the mobile agent. The requirement is a suitable for the proposed mobile agent system. The Basic module collects plaintext data and the SMA module encrypts collected data and processes calculation inside SEE. Thus, end-users cannot eavesdrop any data except the data collected by the owner of the data, and only the owner of the mobile agent, which is an electric utility in this example, can get it.

2) Pay-per-use software charging and software activation

There are use cases in which end-users are charged to the extent that they used functions of a mobile agent whereas software is bundled and charged per device in the legacy business model. In a smart grid, since required functions of devices in smart grids vary, it is ideal if we can provide a method whereby a mobile agent brings only the functions required to a particular usage model and charges depending on use frequency, and returns to a head-end system to report the result of the usage. However, no such system has yet been deployed in the market because the devices are located on the user side, which makes it difficult to detect and prevent tampering. The proposed mobile agent system enables the SMA module to have the required core functions, and counts how many times and how long the SMA module is used. Since only the owner of a mobile agent can access the counter whereas end-users of the mobile agent and the administrator of mobile agent platform cannot, attackers cannot cheat the mobile agent out of its functions.

There is another use case in which a mobile agent is distributed with full functions and end-users are permitted to use partial functions only if they are successfully authenticated. For example, it is desirable for developers to have a maintenance mode in mobile agents to debug devices on site, which is available for field engineers only. Legacy systems need to have network connectivity to authenticate end-users since a mobile agent cannot have secret information, such as PIN code inside since there is a risk of tampering the mobile agent. For the proposed mobile agent system, it is easy to realize the maintenance mode to have the SMA module PIN code inside. If PIN code and PIN

code verification process is implemented in the SMA module and particular functions are activated only when the verification succeeds, end-users cannot use the functions. Only the field engineers who know the secret PIN code can activate full functions, including maintenance mode, of the mobile agent.

3) Safety vault

A smart grid consists of a tremendous number of sensors and actuators. Connected sensors and actuators exchange data and they are operated through a network. If attackers modify parameters input to the actuators, they may misbehave or it may lead to serious accidents in the worst case. Since the appropriate range of parameters varies depending on applications, it is difficult to preset them in actuators in advance. By utilizing the proposed mobile agent system, it is possible to build a system where mobile agents bring the parameters securely to each host and the SMA module checks the parameters before the parameters are sent to the actuator. Since attackers can neither modify the parameters nor invalidate the check process, the SMA module can be used as a safety valve.

Chapter 6

Related work

In this chapter, we refer to work related to this dissertation. The research field of this dissertation is trusted computing. There are many techniques to realize trusted computing. One approach is to utilize a dedicated hardware module, such as the Trusted Platform Module (TPM), in order to establish secure storage to protect confidential information, such as a key, or to attest the integrity of the target system. Another approach is to utilize virtualization technology in order to make an isolated secure environment. Furthermore, in terms of applications of the proposed methods, we introduce related work on fault-tolerant systems and secure mobile agent systems.

Trusted computing

The concept of trusted computing has a long history. In 1983, the U.S. Department of Defense published a computer security standard, referred to as the "Orange Book," which describes trusted computer system evaluation criteria [71]. The basic concept of trusted computing is introduced in the standard. For example, Trusted Computing Base (TCB) is defined as follows:

> *The heart of a trusted computer system is the Trusted Computing Base (TCB) which contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based.*

Based on this concept, various approaches have been proposed in order to implement TCB. In a modern computer system, TCB is realized as a small amount of built-in hardware in order to create a foundation of trust for software processes [72]. In regard to industry, an industrial consortium, Trusted Computing Platform Alliance (TCPA), published a specification that defines a subsystem containing an isolated computing engine in 2001 [73]. Followed by the specification, a non-profit organization, Trusted Computing Group (TCG), which is the successor of TCPA, publishes overall architectures of trusted computing and defines specifications [74].

Dedicated security hardware for trusted computing

TCG publishes many specifications that define functions of dedicated hardware modules, such as the Trusted Platform Module (TPM) or Trusted Storage, which are the root of trust. Those hardware modules and hardware devices have been implemented by various vendors and used as trust anchors to realize trusted computing.

McCune proposes a method that provides a secure execution environment to minimize TCB by utilizing TPM and a commodity processor for personal computers [75]. However, the current TPM assumes that a secure execution environment is provided only when booting a system, and does not assume context switching between the secure world and the non-secure world while the system is working. The result of the experiment shows that 10[s] is required for context switching between worlds. Since the operating system working in the non-secure environment is suspended during context switching, it is impracticable to introduce embedded end-point devices.

There are some proposals to implement functions equivalent to TPM by software. Strasser proposes emulating functions of TPM with software [76]. The results of his experiments show the performance degradation compared with hardware implementation. However, his objective is testing or debugging only. Thus he does not mention a method for using it in real applications to keep confidentiality, integrity, and availability. Liu proposes virtual TPM for cloud architecture [77]. Since his purpose is the provision of crypto functions for users so that they can move to a platform that they do not own, he neither provides an execution environment nor does he target embedded end-point devices.

Virtualization

In the field of general-purpose computer systems, processors supporting a virtualization function are widely available and there are a number of reports on attempts to execute two operating systems concurrently and efficiently on one processor. Some of the research results have led to commercial products widely deployed in the market [78][79]. However, the inclusion of vulnerability is inevitable even in a virtual machine [80]. Furthermore, most embedded processors do not yet support the virtualization function. Although it is technically possible to implement the virtualization function by software, it is impracticable for embedded end-point devices since many functions need to be implemented, such as memory management, resulting in large performance degradation.

To make a secure environment by utilizing TrustZone, various systems have been proposed. Santos proposes runtime execution for secure component for embedded devices by using TrustZone [81]. Yan-ling proposes a secure embedded system environment with a multi-policy access control mechanism and a secure reinforcement method based on TrustZone [82]. He assumes various applications and services run in the environment. Winter presents a method to implement Mobile Trusted Module which is defined in TCG specification on a software-only base by utilizing TrustZone [83]. Sangorrin proposes a software architecture on which a real-time operating system and a general-purpose operating system are executed concurrently on a single ARM processor with low overhead and reliability by utilizing TrustZone [84]. Sangorrin further proposes a method to minimize communication overhead while satisfying the strict reliability requirements of the real-time operating system [85]. In addition, Nakajima proposes using TrustZone to enable dependability and real-time capability [86]. Baseline common functions in our proposed systems use the same techniques as in the existing approaches. Our contribution is clarification of an overall architecture and functions that work in a secure environment with a full implementation to enable end-point devices to keep long-term security and automatically recover from an error status in a smart grid.

Fault-tolerant system

To recover from an operating system failure, various approaches have been proposed.

The simplest approach is that of including the recovery mechanism

within the operating system. One method is to use Non-maskable Interrupt (NMI) as a watchdog timer [87]. NMI is a processor interrupt that cannot be ignored. When NMI is generated, the NMI handler implemented within the operating system is called regardless of the status of the operating system. Since it is unnecessary to save and restore registers to execute a process implemented in the NMI handler, performance overhead is mitigated. Thus, NMI can be used as a surveillance and recovery process to implement the NMI handler so that it detects whether the operating system hangs or not. Dolev proposes a self-stabilizing operating system utilizing NMI [88]. Although NMI is easy to use as a watchdog timer because it has already been implemented in Linux, it is vulnerable because the NMI handler could be invalidated to overwrite the code segment of the operating system. Furthermore, since implementation of a rich application in an interrupt handler, such as a network communication function or a data encryption function, is not anticipated, it is difficult to realize the notification function.

Another approach to recover from failure is to check the status of the operating system from outside using virtualization technology. It is easy to realize an isolation environment by utilizing virtualization technology. Karfinkel developed the trusted virtual machine monitor (TVMM), on which a general-purpose platform and a special-purpose platform executing security-sensitive processes run separately and concurrently [89]. The libvirt project is developing a virtualization abstraction layer including a virtual hardware watchdog device [90]. To cooperate with the watchdog daemon installed in a guest OS, a virtual machine monitor can notice that the daemon is no longer working when periodically trying to communicate with it. Although virtualization technology is widely deployed in PC-based systems, it is difficult to implement it in embedded devices as fewer hardware devices support it. Moreover, since the volume of source code for a virtual machine monitor (VMM) tends to become large, the risk of VMM including bugs also becomes large. To overcome the restriction, Kanda developed SPUMONE, a lightweight virtual machine monitor designed to work on embedded processors [91]. It provides a function to reboot the guest OS. However, SPUMONE does not provide a memory protection mechanism between the virtual machine monitor and the guest OS (Rich OS). Thus, it is vulnerable to an attack on the virtual machine monitor from the guest OS.

Secure mobile agent system

There is little prior work on protection of mobile agents. Badger [92], Shah [93], and Balachandran [94] propose the application of code obfuscation techniques to mobile agents. They implemented their proposed techniques as Java bytecode transition tools that generate obfuscated platform-independent bytecode formats of mobile agents. The obfuscated code is functionally identical to the original one. Obfuscating a mobile agent code makes it difficult for an attacker to reverse engineer the code. As a means of supplementing code obfuscation, Hohl proposes a technique that defines the time necessary for analyzing the mobile agent code that was obfuscated as the lifetime [95]. After the lifetime expires, data of the mobile agent are invalid and the mobile agent cannot migrate or interact anymore. Although those techniques are useful and robust to some extent against automatic software analysis tools, such as decompilers, they basically aim to delay an attacker, thereby preventing the attacker from completing the analysis of the code and data. Furthermore, because techniques to bypass the effects of obfuscation are rapidly becoming more sophisticated [96], the time necessary to analyze the obfuscated code is becoming shorter. Moreover, since the code must be executable, the attackers can manually analyze the code if they take enough time. Besides, the execution time of the obfuscated code tends to become larger than the original one. Since programmers need to tune up the speed of the program in the case of embedded devices with poor processors, the development cost becomes high. Therefore, code obfuscation mitigates the threat but does not solve the problem.

To self-check the modification of the mobile agent, Vigna [97] and Holh [98] propose using execution trace. They trace the execution states of the mobile agent generated by a mobile agent platform, and after executing the mobile agent, compare them with those generated by an honest platform. If they do not match, occurrence of modification on a remote host is detected. Although it may be useful to find the trace of modification after an attack, it does not provide a method to protect against the modification itself.

Batarfi [99] and Sander [100] propose using homomorphic encryption. In homomorphic encryption, the computation is done on the encrypted data themselves without decrypting them. Therefore, once the mobile agent is encrypted on the owner's host, the mobile agent does not need to be in plaintext on the mobile agent platform when processing on a remote host, nor does the mobile agent platform need to have a secret key to decrypt the mobile agent. Although homomorphic encryption is a

powerful and useful schema to protect the mobile agent in that it is possible to keep the secrecy of the mobile agent, there are problems in that computational cost is high and a long length of the encrypted message is necessary under the currently proposed schema. Therefore, application of homomorphic encryption to a mobile agent platform with embedded devices remains impracticable.

In earlier work, Yee [101] briefly describes a trusted execution environment where mobile agents run within a secure coprocessor, allowing Java-based agents to run securely. Wilhelm [102] proposes introducing dedicated trusted and tamper-proof hardware on which a virtual machine serves as an execution environment for a mobile agent. The mobile agent is first encrypted by a public key installed in the hardware and then distributed. Since only the mobile agent platform having a private key corresponding to the public key can decrypt the mobile agent, the mobile agent's code and data are protected. Muñoz proposes a protocol for the secure migration of the mobile agent to introduce remote attestation with the Trusted Platform Module (TPM) in order to authenticate the integrity of the platform [103]. By utilizing dedicated hardware, their proposed method greatly improves robustness compared with a software solution, such as code obfuscation, since it is difficult to analyze or modify the hardware without deep knowledge and specialized tools. However, since the dedicated hardware is separated from the main processor, additional cost is incurred that poses an obstacle to commercialization. Besides, it is effective only for the mobile agent platform on which the hardware is installed. Furthermore, since there has been no implementation, it is unclear whether performance would be degraded.

Chapter 7

Future work

As in all research, the work presented in this dissertation is by no means complete. Rather, we believe that the ideas are just a starting point for establishing a security platform for embedded end-point devices in a smart grid.

With regard to the extension of the method proposed in this dissertation, issues to be addressed are the enhancement of performance and the extension of functions. The first issue is performance enhancement. Similar to processors for personal computer systems, there is a trend toward the use of multi-core processors in embedded end-point devices to enhance processing speed. Our proposed method uses one core only even if a processor supports multi-cores. In order to minimize the performance degradation of Rich OS, we must support multi-cores. However, it is not easy since we need to support an exclusive control mechanism in terms of security. The proposed method provides an isolated secure environment in which general-purpose processes and protection-required processes are concurrently executed. Therefore, the exclusive control mechanism is much more complex than the existing multi-core exclusive control. The second issue is the extension of functions of secure mobile agent systems. Ideally, it is desired to support auto-saving and auto-restoring the context of the Secure Mobile Agent (SMA) module in Secure Execution Environment (SEE). If a mobile agent needs to migrate to a host with the context of the SMA module, the SMA module itself needs to save its context as a file, send the file and restore the context from the file. As well as migrating, the SMA module needs to save its context and restore it when reinstalling and restarting the SMA module to avoid losing the context when other SMA modules overwrite it. The third issue is prevention of Denial of Service

(DoS) attacks. The proposed method cannot prevent DoS attacks. For example, attackers can disturb the execution of the SMA module. Since SEE does not inspect the origin of issuing an operation, any Basic module can send it the operation although the Basic module cannot modify or eavesdrop on SEE. For example, a malicious Basic module may try to send an install operation with a corrupted SMA module. As a result, the execution of the SMA module is suspended during verifying the requested SMA module, and consequently extra time is required to complete its task. The malicious Basic module may also try to send an install operation with the SMA module copied from a legitimate Mobile Agent during execution of another Mobile Agent. As a result, the Mobile Agent reinstalls its SMA module since the SMA module has a legitimate signature and its SMA module is overwritten by the instruction of the malicious Mobile Agent, and consequently extra time is required to verity the SMA module again, or the intermediate result of the calculation may be destroyed and it may need to restart the calculation from the beginning. In order to prevent those attacks, a mechanism for inspecting the origin of issuing an operation is necessary. However, it is not easy to solve the attack since it is essentially impossible to establish a trusted area in Rich OS. Even if we successfully identify a process issuing the operation, another process that controls the process might be the source of an attack. We need a method of finding the sources of attacks, which poses a problem that is very difficult to solve.

Another approach is to enhance usability for developers. The proposed method assumes that it is possible for developers to clearly divide a security-sensitive module from a general-purpose process. If a system is developed from scratch, it might not be a big issue since developers have a chance to consider how to divide modules when designing the system from an early stage in the development. However, if developers need to reuse existing modules in which security-sensitive modules and general-purpose modules are tightly bound, it is difficult to divide functions into those of security-sensitive modules and those of general-purpose modules, and extra cost is incurred. Therefore, it is desirable to provide methods to minimize the porting cost for developers. From a technical viewpoint, it is desirable to support functions that make it possible to flexibly define the security-sensitive module and protect modules initially loaded in the non-secure world.

Besides the technical improvement, a deployment issue of the result of the proposed method should be addressed. One approach is exploring new other application examples to motivate developers and end-users to

introduce our proposed methods. Security is frequently regarded as cost, hesitating to introduce security features. By showing new application examples that are attractive to service providers or end-users, penetration of introducing the security features will improve. Another approach is building a security requirement through standardization activity. Although existing technical standards and guidelines conceptually indicates security requirements that we proposed in this dissertation, such as upgradability and availability, no specific method has been defined. However, the importance of implementation method will become larger since most critical security incidents are caused by implementation faults. Therefore, there will be a good opportunity to discuss methods that provide a robust mechanism against attacks that exploit implementation vulnerability in standardization bodies. Such standardization activities will be an effective way in order to help momentum to introduce sophisticated security technologies, which we presented in this dissertation, in a smart grid society.

Chapter 8

Conclusion

In this dissertation, we proposed a security platform for embedded end-point devices in a smart grid. The proposed method addressed two critical problems that have not been solved by previous research: keeping long-term security and keeping availability. Furthermore, we proposed a secure mobile agent system that provides a secure execution environment for mobile agents in order to achieve autonomous distributed smart grid architecture. We also presented examples of new application that enhanced efficiency and reliability for Field Area Network in smart grids.

In Chapter 1, we outlined the methods proposed in this dissertation. We introduced the motivation for this dissertation together with a brief explanation of a smart grid. The principal concern of this dissertation is realization of a secure system to keep embedded end-point devices in a smart grid secure, in particular, provision of a robust mechanism against attacks that exploit implementation vulnerability, including illegitimate modification and eavesdropping, at reasonable cost in terms of development, deployment and maintenance. We summarized our contributions: keeping long-term security, satisfying the three pillars of information security, demonstrating feasibility with full implementation, and enabling new applications in a smart grid with a security platform.

In Chapter 2, we defined the problems. We provided an overview of the security problems in smart grids, including the clarification of differences between information and communication systems and control systems. As a result of the analysis, it is clarified that the major problems in embedded end-point devices in smart grids concern keeping long-term security and keeping availability. Then, we analyzed the reasons why the

problems are difficult to solve. Regarding keeping long-term security, the following reasons were clarified: it is difficult to prevent tampering of software, it is difficult to exclude vulnerability in a large system, it is difficult to eliminate the risk of compromising a crypto system, and it is difficult to attest a part of a software module only. Regarding keeping availability, the following reasons were clarified: it is difficult to keep a high level of surveillance continuity, it is difficult for an administrator to detect when an incident occurs, and it is difficult to minimize development cost and production cost.

In Chapter 3, we provided background information on the hardware technologies leveraged by the proposed method. We presented security functions of ARM processors. In addition, we presented functions of typical dedicated security hardware.

In Chapter 4, we proposed a method to keep long-term security and a method to keep availability. The proposed methods basically consist of three components: a secure module that executes security-sensitive processes and runs in the secure world, Rich OS that is an operating system executing general-purpose processes and running in the non-secure world, and Monitor that provides a context switch function. We clarified the functions of each component and process flows. The features of our proposed method to keep long-term security are virtual hardware with a dynamic loading function, a decryption function, a verification function, and an attestation function that works in the secure world. We demonstrated full implementation of the proposed method. The results of experiments showed that performance degradation of the Rich OS is less than 10% in a severe case whereas robustness was greatly improved compared with the existing vulnerable system implemented only with software. The features of our proposed method to keep availability are a surveillance function, a read-only memory function, and a notification function. We demonstrated full implementation of the proposed method. The results of experiments showed that the performance degradation is under 0.2% in a normal use case.

In Chapter 5, we proposed a secure mobile agent system. The method enabled mobile agents to execute their processes on an untrusted mobile agent system in order to keep secrecy and integrity of mobile agents. We demonstrated a full implementation of the proposed secure mobile agent system. The results of experiments showed that although installation of the secure module takes a long time, performance degradation of the Rich OS was mitigated by introducing context switch while executing installation. Furthermore, we proposed the application of the secure

mobile agent system to smart grids, in particular, Field Area Network in smart grids, and showed that it enables the introduction of new applications that are difficult to realize by previous methods.

In Chapter 6, we referred to work related to this dissertation. The research field of this dissertation is trusted computing. We referred to papers that realize trusted computing to utilize dedicated security hardware and virtualization. We also referred to papers related to fault-tolerant systems and secure mobile agent systems.

In Chapter 7, we discussed future work to be addressed in order to improve our proposed method by extending it and enhancing usability for developers.

Finally, we provide a brief overview of this dissertation. The proposed method isolates a protection-required process from general-purpose processes, including an operating system, and provides an execution environment to run both processes concurrently. Since the protection-required process is executed in the secure world whereas the general-purpose processes are executed in the non-secure world, the protection-required process can be securely executed without interference even if attackers completely take control of the general-purpose processes by exploiting the vulnerability. In order to keep long-term security, the proposed method enables dynamic loading and updating of the security-sensitive module only with sufficient robustness against tampering. Furthermore, it does not require rebooting the entire system including the operating system. In order to keep availability, the proposed method realizes a fault-tolerant system enabling the embedded end-point devices to monitor the status of the operating system and to recover even if they stop working owing to unexpected behavior or cyber-attacks. Furthermore, we proposed a secure mobile agent system to provide an isolated execution environment for a mobile agent by generalizing methods to keep long-term security and availability.

References

[1] K. C. Budka, J. G. Deshpande, and M. Thottan, "Smart Grid Applications," in Communication Networks for Smart Grids: Making Smart Grid Real, Springer-Verlag, London, UK, 2014, pp. 111-145.

[2] V. C. Gungor, D. Sahin, T. Kocak, S. Ergut, C. Buccella, C. Cecati, and G. P. Hancke, "A Survey on Smart Grid Potential Applications and Communication Requirements," in IEEE Transactions on Industrial Informatics, vol. 9, no. 1, pp. 28-42, 2013.

[3] Y. Mo, T. H. Kim, K. Brancik, D. Dickinson, H. Lee, A. Perrig, and B. Sinopoli, "Cyber-Physical Security of a Smart Grid Infrastructure," in Proceedings of the IEEE, vol. 100, no. 1, pp. 195-209, 2011.

[4] National Institute of Standard and Technology, NISTIR 7628 Guidelines for Smart Grid Cyber Security, 2010.

[5] CEN-CENELEC-ETSI, Smart Grid Information Security, 2014.

[6] H. Li, "Enabling Secure and Privacy Preserving Communications in Smart Grids," Springer International Publishing, 2014.

[7] E. Auchard, "Popular electricity smart meters in Spain can be hacked, researchers say", Reuters, Oct. 2014 [Online]. Available: http://www.reuters.com/article/2014/10/07/us-cybersecurity-spain-idUSKCN0HW15E20141007 [Accessed 30, Jul. 2015].

[8] MITRE. “Common vulnerabilities and exposures,” [Online]. Available: http://cve.mitre.org [Accessed 30 Jul. 2015].

[9] U.S. Department of Homeland Security. "ICS-CERT," [Online]. Available: https://ics-cert.us-cert.gov [Accessed 30 Jul. 2015].

[10] H. Isozaki and J. Kanai, "Embedded System with Long-term Security Utilizing Hardware Security Function," in Transactions of Information Processing Society of Japan (in Japanese and in press).

[11] H. Isozaki, J. Kanai, S. Sasaki, and S. Sano, "Keeping High Availability of Connected End-point Devices in Smart Grid," in Proceedings of the Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies, 2014, pp. 73-80.

[12] H. Isozaki, J. Kanai, S. Sasaki, and S. Sano, "Security system for connected end-point devices in a smart grid with commodity hardware," in International Journal on Advances in Intelligent Systems, vol. 7, no. 3&4, pp. 533-546, 2014.

[13] J. Zheng, D. W. Gao, and L. Lin, "Smart Meters in Smart Grid: An Overview," in Proceedings of 2013 IEEE Green Technologies Conference, 2013, pp. 57-64.

[14] European Network and Information Security Agency, Smart Grid Threat Landscape and Good Practice Guide, 2013.

[15] D. Dzung, M. Naedele, M., T. P. v. Hoff, and M. Crevatin, "Security for Industrial Communication Systems," in Proceedings of the IEEE, vol. 93, no. 6, pp. 1152-1177, 2005.

[16] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A Survey on Cyber Security for Smart Grid Communications," in IEEE Communications Surveys & Tutorials, vol. 14, no. 4, pp. 998-1010, 2012.

[17] A. R. Metke and R. L. Ekl, "Smart Grid Security Technology," in Proceedings of the Innovative Smart Grid Technologies, 2010, pp. 1-7.

[18] P. McDaniel and S. McLaughlin, "Security and Privacy Challenges in the Smart Grid," in IEEE Security & Privacy, vol. 7, no. 3, pp. 75-77, 2009.

[19] M. Davis, "SmartGrid Device Security: Adventures in a New Medium," in Black Hat USA 2009, 2009.

[20] T. Goda, S. Morozumi, "Smart Grid TEXTBOOK," Impress Japan, Tokyo, Japan, 2011 (in Japanese).

[21] National Institute of Standard and Technology, Guide to Industrial

Control Systems (ICS) Security, Special Publication 800-82, 2011.

[22] S. Clements, "Cyber-security Considerations for the Smart Grid," in Proceedings of 2010 IEEE Power and Energy Society General Meeting, 2010, pp. 1-5.

[23] W. Wang and Z. Lu, "Cyber security in the Smart Grid: Survey and challenges," in Computer Networks, vol. 57, no. 5, pp. 1344-1371, 2013.

[24] F. M. Cleveland, "Cyber Security Issues for Advanced Metering Infrasttructure (AMI)," in Proceedings of IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008, pp. 1-5.

[25] Idaho National Laboratory, Cyber Security Procurement Language for Control Systems Version 1.8, 2008.

[26] G. Gaubatz, J.-P. Kaps, and E. Ozturk, and B. Sunar, "State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks," in Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications Workshops, 2005, pp. 146-150.

[27] P. Ganesan, R. Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, and M. Sichitiu, "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes," in Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, 2003, pp. 151-159.

[28] Y. W. Law, J. Doumen, and P. Hartel, "Benchmarking Block Ciphers for Wireless Sensor Networks," in Proceedings of the 1st IEEE international conference on Mobile Ad-hoc and Sensor Systems, 2004, pp.447-456.

[29] H. Khurana, R. Bobba, T.Yardley, P. Agarwal, and E. Heine, "Design Principles for Power Grid Cyber-Infrastructure Authentication Protocols," in Proceedings of the 43rd Hawaii International Conference on System Sciences, 2010, pp. 1-10.

[30] H. Khurana, M. Hadley, N. Lu, and D. A. Frincke, "Smart-Grid Security Issues,", in IEEE Security & Privacy, vol. 8, no. 1, pp. 81-85, 2010.

[31] National Electrical Manufacturers Association, Requirements for

Smart Meter Upgradeability, 2009.

[32] K. Li, "Towards Security Vulnerability Detection by Source Code Model Checking," in Proceedings of the Third International Conference on Software Testing, Verification, and Validation Workshops, 2010, pp. 381-387.

[33] M. Rinard, C. Cadar, D. Dumitran, D. M. Roy, and T. Leu, "A Dynamic Technique for Eliminating Buffer Overflow Vulnerabilities (and Other Memory Errors)," in Proceedings of the 20th Annual Computer Security Applications Conference, 2004, pp. 82-90.

[34] Z. Huang and I. G. Harris, "Return-oriented Vulnerabilities in ARM Executables," in Proceedings of IEEE 2012 Conference on Technology for Homeland Security, 2012, pp. 1-6.

[35] R. Sailer, X. Zhang, T. Jaeger, and L. v. Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in Proceedings of the 13th conference on USENIX Security Symposium, 2004, pp. 223-238.

[36] K. D. Craemer and G. Deconinck, "Analysis of State-of-the-art Smart Metering Communication Standards," in Proceedings of the 5th young researchers symposium, 2010.

[37] W. Wang, Y. Xu, and M. Khanna, "A survey on the communication architectures in smart grid," in Computer Networks, vol. 55, no. 15, pp. 3604-3629, 2011.

[38] A. Liotta, D. Geelen, G. v. Kempen, and F. v. Hoogstraten, "A survey on networks for smart-metering systems," in International Journal of Pervasive Computing and Communications, vol. 8, no.1, pp. 23-52, 2012.

[39] S. M. Varghese and K. P. Jacob, "Anomaly Detection Using System Call Sequence Sets," in Journal of Software, vol. 2, no. 6, pp. 14-21, 2007.

[40] M. J. Pont and R. H. L. Ong, "Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study," in Proceedings of the First Nordic Conference on Pattern Languages of Programs, 2002, pp. 159-200.

[41] ARM, "ARM Security Technology," [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-

009492c/PRD29-GENC-
009492C_trustzone_security_whitepaper.pdf [Accessed 30 Jul. 2015].

[42] T. Alves and D. Felton, "TrustZone: Integrated Hardware and Software Security," in Information Quarterly, vol. 3, no. 4, pp. 18-24, 2004.

[43] Trusted Computing Group: TPM Main Specification Part 1 Design Principles, Specification Version 1.2 Revision 116, 2011.

[44] D. Kleidermacher and M. Kleidermacher, "Embedded Systems Security", Newnes, Oxford, UK, 2012.

[45] T. Otani, "A Primary Evaluation for Applicability of International Standard Protocol for Meter Reading to Next-generation Grids," CRIEPI Research Report, R09009, 2010 (in Japanese).

[46] M. Kanda, Y. Ohba, and Y. Tanaka, "AMSO (TM) Unified Key Management Mechanism Integrating Authentication and Encryption for Smart Meters," in Toshiba Review, vol. 65, no. 9, pp. 23-27, 2010 (in Japanese).

[47] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, and A. Yegin, "Protocol for Carrying Authentication for Network Access (PANA)," IETF RFC 5191.

[48] Coverity, "Coverity Scan: 2013 Open Source Report", Available: http://softwareintegrity.coverity.com/rs/coverity/images/2013-Coverity-Scan-Report.pdf [Accessed 30 Jul. 2015].

[49] J. Kanai, H. Sasaki, M. Kondo, H. Nakamura, and M. Namiki, "Energy-Eficient Scheduler by Statistical Analysis for Linux", in The Special Interest Group Technical Reports of IPSJ OS-106, pp. 9-16, 2007 (in Japanese).

[50] P. B. Ghewari, J. K. Pati, and A. B. Chougule, "Efficient Hardware Design and Implementation of AES Cryptosystem," in International Journal of Engineering Science and Technology, vol. 2, no. 3, pp. 213-219, 2010.

[51] A. Gielata, P. Russek, and K. Wiatr, "AES hardware implementation in FPGA for algorithm acceleration purpose," in Proceedings of the International Conference on Signals and Electronic Systems, 2008, pp. 137-140.

[52] F. Zhao, Y. Hanatani, Y. Komano, B. Smyth, S. Ito, and T. Kamibayashi, "Secure Authenticated Key Exchange with Revocation for Smart Grid," in Proceedings of the third IEEE PES Conference on Innovative Smart Grid Technologies, IEEE Power & Energy Society, 2012, pp. 1-8.

[53] ARM, "Dhrystone Benchmarking for ARM Cortex Processors," [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dai0273a/DAI0273A_dhrystone_benchmarking.pdf [Accessed 30 Jul. 2015].

[54] M. Miyashita and T. Ohtani, "Transmission Characteristics Evaluation of Demand-side Communication -Evaluation of Response Time Using International Standard Protocol for Meter Reading and Wireless LAN-," in CRIEPI Research Report, R10035, 2011 (in Japanese).

[55] I. Sato, “Mobile Agent,” in Handbook of Ambient Intelligence and Smart Environments. Springer US, 2010, pp. 771-791.

[56] D. P. Buse and Q. H. Wu, "Mobile Agents for Remote Control of Distributed Systems," in IEEE Transactions on Industrial Electronics, vol. 51, no. 6, pp. 1142-1149, 2004.

[57] W. Jansen and T. Karygiannis (1999, Oct.). Mobile Agent Security. NIST Special Publication 800-19.

[58] R. Gray, “Agent Tcl: A flexible and secure mobile-agent system,” in Proceedings of the Fourth Annual Tcl/Tk Workshop, 1996, pp. 9-23.

[59] G. Graham, “ObjectSpace voyager - The agent ORB for Java,” in Lecture Notes in Computer Science, vol. 1368, Springer-Verlag Berlin Heidelberg, 1998, pp. 38-55.

[60] G. Karjoth, D. B. Lange, and M. Oshima, “A Security Model for Aglets”, in Mobile Agents and Security. Springer-Verlag Berlin Heidelberg, 1998, pp. 188-205.

[61] M. Kuzlu, M. Pipattanasomporn, and S. Rahman, "Communication network requirements for major smart grid applications in HAN, NAN and WAN," in Computer Networks, vol. 67, pp. 74-88, 2014.

[62] Tokyo Electric Power Company, "Basic Concept for Smart Meter Specification based on RFC", [Online]. Available: http://www.tepco.co.jp/en/press/corp-

com/release/betu12_e/images/120712e0101.pdf [Accessed 30 Jul. 2015].

[63] V. C. Gungor and F. C. Lambert, "A survey on communication networks for electric system automation," in Computer Networks, vol. 50, no. 7, pp. 877-897, 2006.

[64] V. C. Gungor, B. Lu, and G. P. Hancke, "Opportunities and Challenges of Wireless Sensor Networks in Smart Grid", in IEEE Transactions on Industrial Electronics, vol. 57, no. 10, pp. 3557-3564, 2010.

[65] H. Gharavi and B. Hu, "Multigate Communication Network for Smart Grid," in Proceedings of the IEEE, 2011, vol. 99, pp. 1028-1045, no. 6.

[66] G. Zhabelova, V. Vyatkin, "Multiagent Smart Grid Automation Architecture Based on IEC 61850/61499 Intelligent Logical Nodes," in IEEE Transactions on Industrial Electronics, vol. 59, no. 5, pp. 2351-2362, 2012.

[67] L. Hernandez, C. Baladron, J. M. Aguiar, B. Carro, A. Sanchez-Esguevillas, J. Lloret, D. Chinarro, J. J. Gomez-Sanz, and D. Cook, "A Multi-Agent System Architecture for Smart Grid Management and Forecasting of Energy Demand in Virtual Power Plants," in IEEE Communications Magazine, vol. 51, no. 1, pp. 106-113, 2013.

[68] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-Agent Systems in a Distributed Smart Grid: Design and Implementation," in Proceedings of the IEEE Power Systems Conference and Exposition, 2009, pp.1-8.

[69] Oracle. Remote Method Invocation Home. [Online]. Available: http://www.oracle.com/technetwork/articles/javaee/index-jsp-136424.html [Accessed 30 Jul. 2015].

[70] Oracle. Java Object Serialization Specification version 6.0. [Online]. Available: http://docs.oracle.com/javase/6/docs/platform/serialization/spec/serial-arch.html [Accessed 30 Jul. 2015].

[71] U.S. Department of Defense, "Trusted Computer System Evaluation Criteria," Department of Defense Standard, 1983.

[72] S. Pearson, B. Balacheff, L. Chen, D. Plaquin, and G. Proudler,

"Trusted (Computing) Platforms: An Overview," in Trusted Computing Platforms: TCPA Technology in Context Prentice Hall, New Jersey, U.S, 2002, pp. 3-42.

[73] Trusted Computing Platform Alliance, "Trusted Computing Platform Specifications," 2001.

[74] Trusted Computing Group: TCG Specification Architecture Overview , Specification Revision 1.2, 2004.

[75] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An Execution Infrastructure for TCB Minimization," in Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2008, pp. 315-328.

[76] M. Strasser and H. Stamer, "A Software-Based Trusted Platform Module Emulator,"in Lecture Notes in Computer Science, vol. 4968, Springer-Verlag Berlin Heidelberg, 2008, pp. 33-47.

[77] D. Liu, J. Lee, J. Jang, S. Nepal, and J. Zic, "A New Cloud Architecture of Virtual Trusted Platform Modules," in IEICE Transactions on Information and Systems, vol. E95-D, no. 6, pp.1577-1589, 2012.

[78] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in Proceedings of the nineteenth ACM symposium on Operating Systems Principles, 2003, pp. 164-177.

[79] C. A. Waldspurger, "Memory Resource Management in VMware ESX Server", in Proceedings of the 5th symposium on Operating Systems Design and Implementation, vol. 36, issue SI, pp. 181-194, 2002.

[80] S. T. King and P. M. Chen, "SubVirt: Implementing malware with virtual machines," in Proceedings of the 2006 IEEE Symposium on Security and Privacy, 2006, pp. 314-327.

[81] N. Santos, R. Himanshu, S. Stefan, and W. Alec, "Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications," in Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, 2014, pp. 67-80.

[82] Z. Yan-ling and P. Wei, "Design and Implementation of Secure

Embedded Systems Based on Trustzone," In Proceedings of International Conference on Embedded Software and Systems, 2008, pp. 136-141.

[83] J. Winter, "Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms," in Proceedings of the 3rd ACM workshop on Scalable Trusted Computing, 2008, pp.21-30.

[84] D. Sangorrin, S. Honda, and H. Takada, "Dual Operating System Architecture for Real-Time Embedded Systems," in Proceedings of 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications, 2010, pp. 6-15.

[85] D. Sangorrin, S. Honda, and H. Takada, "Reliable and Efficient Dual-OS Communications for Real-Time Embedded Virtualization," in Japan Society for Software Science and Technology, vol.29, no.4, pp. 182-198, 2012.

[86] K. Nakajima, S. Honda, S. Teshima, and H. Takada, "Enhancing reliability in Hybrid OS system with security hardware,", in the IEICE Transactions on Information and Systems, vol. J93-D, no. 2, pp. 75-85, 2010.

[87] A. Kleen, "Machine check handling on linux," Technical report, SUSE Labs, Aug. 2004 [Online]. Available: http://halobates.de/mce.pdf [Accessed 30 Jul. 2015].

[88] S. Dolev and R. Yagel, "Towards Self-Stabilizing Operating Systems," IEEE Transaction on Software Engineering, vol. 34, no. 4, pp. 564-576, 2008.

[89] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A Virtual Machine-Based Platform for Trusted Computing," In Proceedings of the nineteenth ACM symposium on Operating Systems Principles, 2003, pp. 193-206.

[90] "libvirt - the virtualization API.," [Online]. Available: http://libvirt.org [Accessed 30 Jul. 2015].

[91] W. Kanda, Y. Yumura, Y. Kinebuchi, K. Makijima, and T. Nakajima, "SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems," In Proceedings of IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008, pp. 144-151.

[92] L. Badger, L. D'Anna, D. Kilpatrick, B. Matt, A. Reisse, and T. V. Vleck, "Self-Protecting Mobile Agents Obfuscation Techniques Evaluation Report," Network Associates Laboratories, Report, 01-036, 2002.

[93] S. W. Shah, P. Nixon, R. I. Ferguson, S. R. Hassnain, M. N. Arbab, and L. Khan, "Securing Java-Based Mobile Agents through Byte Code Obfuscation Techniques," in Proceedings of the IEEE Conference on Multitopic INMIC, 2006, pp. 305-308.

[94] V. Balachandran and S. Emmanuel, "Potent and Stealthy Control Flow Obfuscation by Stack Based Self-Modifying Code," in IEEE Transactions on Information Forensics and Security, vol. 8, no. 4, pp. 669-681, 2013.

[95] F. Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents from Malious Hosts," in Lecture Notes in Computer Science, vol. 1419, Springer-Verlag Berlin Heidelberg, 1998, pp. 92-113.

[96] S. K. Udupa, S. K. Debray, and M. Madou, "Deobfuscation: Reverse Engineering Obfuscated Code," in Proceedings of the IEEE 12th Working Conference on Reverse Engineering, 2005.

[97] G. Vigna, "Protecting Mobile Agents through Tracing," in Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, 1997.

[98] F. Hohl, "A Protcol to Detect Malicious Hosts Attacked by Using Reference States," Technical Report Nr. 09/99, Faculty of Informatics, University of Stuttgart, Germany, 1999.

[99] O. A. Batarfi and A. I. Metro, "Protecting Mobile Agents against Malicious Hosts Using Dynamic Programming Homomorphic Encryption," in International Journal of Science & Emerging Technologies, vol. 1, 2011.

[100] T. Sander and C. F. Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in Lecture Notes in Computer Science, vol. 1419, Springer-Verlag Berlin Heidelberg, 1998, pp. 44-60.

[101] B. Yee, "A Sanctuary for Mobile Agents," in Lecture Notes in Computer Science, vol. 1603, Springer-Verlag Berlin Heidelberg, 1999, pp. 261-273.

[102] U. G. Wilhelm, L. Buttyan, and S. Staamann, "On the Problem of

Trust in Mobile Agent Systems," in Symposium on Network and Distributed System Security, Internet Society, 1998, pp. 114-124.

[103] M. Antonio and M. Antonio, "A Hardware Based Infrastructure for Agent Protection," in Advances in Soft Computing, vol. 51, Springer Berlin Heidelberg, 2009, pp. 39-47.