

Xi'an Jiaotong-Liverpool University
西交利物浦大学

SAINT: The Next Generation Framework of Internet of Things



Nian Xue

ID: 1509957

Supervisor: Xin Huang

Department of Computer Science and Software Engineering

Xi'an Jiaotong-Liverpool University

A dissertation submitted for the degree of

Master of Research, Computer Science

December, 2016

Acknowledgement

First of all, I would like to take this chance to express my deepest gratitude to my supervisor – Dr. Xin Huang, for his patient guidance, support, encouragement and advice he offered throughout the time of studying for my master's degree. In the process of carrying out the study, his inspiration and personality also have won my highest respect and love.

In particular, I would like to thank Dr. Yong Yue, Dr. Paul Craig and Dr. Dawei Liu for the valuable suggestions and comments they made. I should also thank Jie Zhang and Kai Zheng for giving me lots of help.

My thanks also go to all the colleagues and students of CSSE department at Xi'an Jiaotong-Liverpool University for providing great resources and cooperation for this thesis.

Finally, I would like to thank my parents and friends who contribute to this thesis with support, encouragement, friendship and love during the past year.

Abstract

The recent surge in prosperity of the Internet of Things (IoT) has been attracting an increasing number of researchers and experts with great attentions due to its significant economic and social values. Along with the remarkable rise of IoT, an increasing number of IoT devices will be connected to the Internet in the near future, which also brings many challenges to current network architectures. The presence of Software Defined Networking (SDN) provides a potential solution for the next generation networks. Although several SDN-based schemes have been proposed theoretically, implementation and security mechanisms on this topic are still few at present. In this thesis, we have proposed a secure and intelligent framework for IoT named as SAINT, offering a comprehensive software defined based framework mode in order to simplify the data flow and function management operations of IoT devices. In addition, we have designed a suite of communication protocols which are used to securely exchange messages between the controller and IoT devices. Finally, a demo system using these protocols is implemented and evaluated regarding their security and performance. In doing so, we take an important step towards integrating SDN and IoT into one software defined control model.

Keywords: Security; Internet of things (IoT); Software Defined Networking (SDN); Software Defined Function (SDF); OpenFlow; OpenFunction.

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation.....	2
1.3	Current Mechanisms and Problems	3
1.4	Research Aim.....	4
1.5	Research Question	5
1.5.1	Date Flow in IoT	5
1.5.2	Function in IoT.....	6
1.6	Solution	7
1.7	Contribution	9
1.8	Publications.....	10
1.9	Thesis Organization	11
2	Literature Review and Definition of Preliminaries.....	12
2.1	SDN and OpenFlow	12
2.2	SNet.....	13
2.3	Secure SDN.....	15
2.4	Software Defined IoT	19
2.5	BAN and GNY logic.....	19
2.6	Casper/FDR.....	20
2.7	Notations	20
2.8	Definitions.....	21
3	SAINT Overview	24
3.1	Participants.....	25

3.2	Three-layer Structure	26
3.3	Networking	27
3.4	Threat Model.....	27
3.5	Desired Properties	28
3.6	Message Transmission	29
3.7	SAINT Agent	29
4	Secure Southbound API.....	31
4.1	Secure OpenFlow	31
4.1.1	OpenFlow Messages	32
4.1.2	Protocol I: Authenticated OpenFlow Association Protocol	33
4.1.3	Protocol II: Secure OpenFlow Message Issuing Protocol.....	35
4.1.4	OpenFlow Key Derivation	37
4.2	Secure OpenFunction.....	38
4.2.1	OpenFunction Messages	39
4.2.2	Protocol III: Authenticated OpenFunction Association Protocol	40
4.2.3	Protocol IV: Secure OpenFunction Message Issuing Protocol	42
4.2.4	OpenFunction Key Derivation	44
5	Security Analysis	46
5.1	Security Analysis of Protocol I	46
5.1.1	Security Under Impersonation Attacks	46
5.1.2	Security Under Man-in-the-middle Attack	47
5.1.3	Security of the Session Key	48
5.2	Security Analysis of Protocol II.....	49
5.3	Security Analysis of Protocol III	50
5.3.1	Security Under Impersonation Attacks	50
5.3.2	Security Under Man-in-the-middle Attacks	51

5.3.3 Security of the Session Key	52
5.4 Security Analysis of Protocol IV	53
5.5 GNY Logic Analysis.....	54
5.5.1 GNY Notions and Notation.....	54
5.5.2 Logical Postulates	55
5.5.3 GNY Analysis of Protocol I.....	57
5.5.4 GNY Analysis of Protocol II	63
5.5.5 GNY Analysis of Protocol III	69
5.5.6 GNY Analysis of Protocol IV	78
5.6 Formal Verification.....	84
5.6.1 Formal Checking of Protocol I.....	84
5.6.2 Formal Checking of Protocol III	85
6 Implementation & Performance	88
6.1 System Setup.....	88
6.2 Theoretical Performance Evaluation.....	90
6.3 Implementation of Protocol I & II	92
6.4 Implementation of Protocol III & IV	94
7 Conclusion.....	99
7.1 Summary	99
7.2 Limitations	99
7.3 Future Work	100
References.....	101

List of Figures

Figure 1 Typical Composition of Intelligent Building System.....	2
Figure 2 SNet Architecture	13
Figure 3 SAINT Architecture	24
Figure 4 Structure of IoT Device	26
Figure 5 SAINT Networking	27
Figure 6 OpenFlow Protocol.....	32
Figure 7 OpenFunction Protocol.....	40
Figure 8 Result of Model Checking for Protocol I	85
Figure 9 Result of Model Checking for Protocol III.....	87
Figure 10 SAINT Demo System.....	89
Figure 11 Implementation results of Protocol I	93
Figure 12 Implementation results of Protocol II.....	94
Figure 13 Implementation results of Protocol III & IV	96
Figure 14 Updating time comparison among different connection methods.....	98

List of Tables

Table 1 TLS Support in OpenFlow by Vendors. [46]	16
Table 2 Comparison of Different Solution for Secure South API.	17
Table 3 GNY Expression	55
Table 4 Features of Prototype Hardware	88
Table 5 Symbol in Performance Analysis	90
Table 6 Protocol I Evaluation	91
Table 7 Protocol II Evaluation	91
Table 8 Protocol III Evaluation.....	91
Table 9 Protocol IV Evaluation	92
Table 10 Average Runtime of Protocol I and Protocol II.....	94
Table 11 Average Runtime of Protocol III and Protocol IV	97
Table 12 Comparison Between OpenFunction and OpenFlow	97

List of Acronyms

ACS	Access Control System
AES	Advanced Encryption Standard
BAN	Burrows–Abadi–Needham
BAS	Building Automation System
CBC	Cipher Block Chaining
CSP	Communicating Sequential Processes
FDR	Failures-Divergences Refinement
FLS	Fire Life Safety
GNY	Gong–Needham–Yahalom
HMAC	Hash Message Authentication Code
IB	intelligent building
IoT	Internet of Things
LCR	Lighting Control and Reduction
OAS	Office Automation System
PK	public key
RSA	Rivest-Shamir-Adleman
SAINT	Secure and Intelligent Networks and Terminals
SDF	Software Defined Function
SDN	Software Defined Networking
SHA	Secure Hash Algorithm
SNet	SDN-based IBNet
SK	secret key

Chapter

1 Introduction

1.1 Background

The Internet of Things (IoT) is a novel and promising paradigm [1], [2], representing the current and future state of the Internet [3]. The emergence of IoT results from the proliferation of smart devices with embedded sensors and actuators in a communicating-actuating network [2]. Nowadays, with wider fusion and further penetration of the Internet in various fields, especially in the household and office domains [4], the IoT technology has been naturally becoming an increasingly fascinating and popular topic [5], attracting more and more interests in commercial, industrial and research facilities around the world.

At present, the IoT is still in the initial stage of development and evolvement [6]. Nevertheless, it is no doubt that the IoT will not only play a key role in the next generation network and communication developing, but also impose important influence on people's daily lives in the predictable future, just like the Internet does today. Recently, an analysis report envisions that the connected IoT will consist of almost 50 billion objects by 2020 [7], which is much more than the number of world population.

As a representative use case of a series of IoT, an intelligent building (IB) has also drawn an extensive public attention [64], [65], [66], [67], [68]. Research into IBs has a long history. For instance, as early as 1988, Carlini [8] has given the original definition of IBs. Wong, et al. summarize a systematic review on IBs, and gave some future research directions based on the comprehensive literature review [9]. Intelligent buildings are the inevitable result of social informationization and highly developed market economy. It is supposed to provide people with an efficient, comfortable, convenient and safe living and working environment [10]. Besides, these advanced technologies related to IBs including Building Automation System (BAS), Access Control System (ACS), Office Automation System (OAS), Lighting Control and

Reduction (LCR), Fire Life Safety (FLS) and some other smart systems or devices have also been greatly applied to much boarder fields, such as smart homes, smart hotels and smart cities. Figure 1 below shows a typical composition of the IB. Along with the remarkable rise of IoT, an increasing number of smart IoT devices will be connected to IB networks in the near future, which brings many challenges to current IB network architectures.

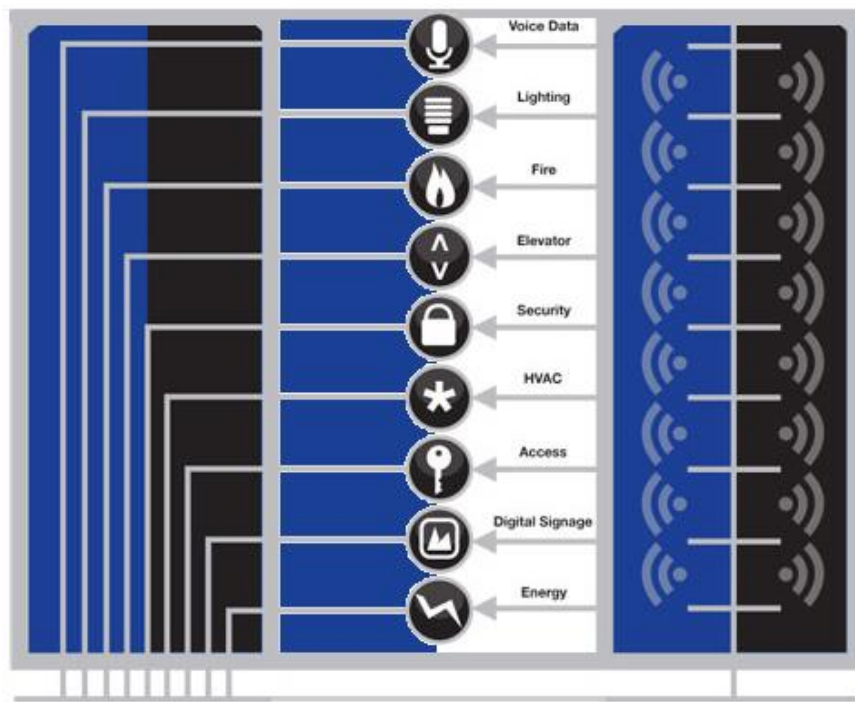


Figure 1 Typical Composition of Intelligent Building System

1.2 Motivation

Accompanied by gaining huge benefits from deploying IoT, a few problems arise gradually as well. As we all know, IoT has wide applications that cover many respects of modern life, technology and scientific research, such as intelligent buildings, smart home, environment monitoring and so forth. Applications of the IoT are often implemented on a complicated IoT system that involves varieties of devices and disparate kinds of networks for these devices. These IoT devices are different in computing ability, memory, storage, power supply and function. Additionally, assorted

types of networks for IoT devices are established on different protocols and standard, e.g., TCP/IP, Blue Tooth 2.0, BLE, nRF, Zigbee and so on.

Thus, two most significant concerns in IoT are:

- 1) The management of various IoT devices.
- 2) The performance and security of the IoT.

Specifically, there are two types of challenges in the aforementioned concerns: Data flow in IoT and function in IoT. These two challenges are not superficial problems but are inherent to the existing network and deep-rooted in the architecture. What's more, security is also an essential issue for IoT, since the computation capability, memory and energy of IoT devices are always limited. Hence, it means that lots of effective but complex security mechanisms are not applicable. More details related to the concerns are depicted in the following paragraphs.

1.3 Current Mechanisms and Problems

The previous challenges could reduce the efficiency and practicability of IoT, and sometimes they might even bring tremendous risks. Yet, these problems might be resolved by incorporating SDN technology. For example, many researchers recently have considered software defined method as a promising solution, and have already done some explorations. The term of "software defined" is originally proposed in Software Defined Networking (SDN). In SDN, the software defined approach is used to change the complicated situation of traditional networks by separating the networking into three planes: the application layer, the control layer and the data layer with interface (e.g. OpenFlow) among them. By doing this, the software defined method can bring lots of benefits to networking users, developers and administrators. As a result, enterprises and carriers can obtain programmability, automation and network control, which enables them to construct highly scalable, flexible networks adapting to the ever-changing requirements.

Meanwhile, to introduce the concept of Software Defined Networking (SDN) is indeed a feasible scheme. The benefits that software defined approach brings to networks let some researchers think about using a similar method to redesign the software defined IoT systems, since the IoT are also complex due to the various types of IoT devices and

networks. Such integration of software defined concept and IoT are referred to SDIoT. Available research achievements primarily focus on two respects:

- 1) Apply SDN for networking in traditional IoT system.
- 2) Utilize the software defined idea to reconstruct the traditional IoT system.

Specifically, the first aspect of the study is SDN-based IoT; however, the second is the accurate definition for SDIoT. Most available research focuses on SDN-based IoT. Only a few works have begun to study into the SDIoT and proposed several frameworks of SDIoT system. Yet, to the best of our knowledge, a complete SDIoT and with implementation and security mechanism has not been put forward by now.

1.4 Research Aim

This thesis aims to find an efficient framework to deal with the new challenges due to deploying IoT. This thesis also plans to design a new framework, which is derived from standard SDN architecture, to prove its practicability. After deploying a new prototype based on the new framework, some simple application tests can be done. For example, we can change the data flow or we can alter the function of a specified devices. Also, its performance will be analyzed and discussed. Finally, the limitations and conclusion for the newly designed system will be summarized, and we could point out some useful information for further research and optimization.

The primary objectives are summarized as follows:

1. Design a framework for IoT using the core concept of SDN.
2. Realize a demo system based on the proposed solution.
3. Design a lightweight security mechanism suitable for IoT devices.
4. Verify the feasibility of the SDN-based IoT model.
5. Test the basic function of the prototype and implement performance.
6. Analyze the test results and discuss the outcome theoretically.
7. Summarize the findings from the experimental results.
8. Review some related works and stat some advantages and disadvantages compared with the proposed system.

9. Point out some possible future research direction or improvement for the proposed system.

1.5 Research Question

1.5.1 Data Flow in IoT

Firstly, the rapid growth of the IoT inevitably generates huge amounts of data and information collected by sensors and data flow among different nodes, resulting in network congestion, which indicates that conventional network architecture is ill-fitted to meet the needs of the next generation Internet.

For instance, the core of an intelligent building is its communication network [11]. In tandem with the expansive connectivity of smart devices and dynamic multi-media services in intelligent buildings, its network is facing the following three questions.

1. The first question involves that the volume of smart devices in IB network, having divergent capabilities and requirements, is getting larger and larger. Thus, data flow among nodes becomes unprecedentedly huge.
2. The second question relates to the complexity of the intelligent building network of which scale is growing significantly than ever before [12]. Managing such huge and complicated networks becomes quite different.
3. The third question is that network security in IBs still has not been paid enough attentions at present [13], [14], which might lead to potential risks or even disasters.

In order to address these problems mentioned above, a Software defined IB Network (SNet) is proposed recently [15]. Using Software Defined Networking (SDN) technologies, it is believed that a network can obtain flexibility, expansibility, mobility, controllability and security [16], [17], [18]. However, how to design a security framework for SNet is still an open research question. This will involve the following sub-questions:

- 1) What security mechanisms (especially security protocols) are needed to avoid cyberattacks in SNet?
- 2) Whether smart IoT devices (e.g. temperature sensor nodes) can support these protocols?

3) Will these protocols have acceptable performance?

1.5.2 Function in IoT

Another challenge with extant networks is that IoT devices are rigid to function changes. In practice, various IoT smart devices are generally preprogrammed and deployed specifically in proper place to fulfill corresponding functions according to divergent requirements. Thanks to the noticeable progress of technology and science, the smart IoT devices have become increasingly intelligent, powerful and versatile. Consequently, people expect more capabilities and features from one smart device.

On the one hand, there is research indicating a tendency that IoT sensor nodes will receive software updates more frequently [19]. On the other hand, such devices are usually preprogrammed and deployed in place for some special applications, such as smoke sensor, noise monitoring and temperature monitoring in intelligent building [20, 21, 22]. In fact, these devices often have several functions (e.g. a single IoT device can embed a few different sensor); on the contrary, they only use a part of functions owing to limited processor and memory resource. They indeed can be refreshed and own new functions through software update.

The difficulty is that on site configuration or upgrade of each device is time-consuming and inefficient [15]. Current network model cannot make the various IoT devices deal with the dynamic and flexible requirements and services from users and environment efficiently and securely [23]. If there is a new software update for all the IoT devices, users have to remove and send back these devices to developers for upgrading, or replace them with new ones. Obviously, the former choice is considerably time-consuming and inconvenient owing to the large volume of smart devices and far distance between controller and smart terminals. Besides, the latter option is expensive and wasteful, which contradicts to today's ecological and environmental protection concept.

Inspired by SDN, [24] proposed a new framework: Software Defined Function (SDF), which adopts SDN principles. It is generally known that the OpenFlow is only designed for controllers and switches in general network [34], unsuitable for IoT devices. Hence, a newly designed OpenFunction which is stemmed from OpenFlow is presented to support upgrading and reprogramming IoT devices. SDF allows administrators to

manage IoT function services through abstraction of lower-level functionality.

In the aspect of function reprogramming, Thus, there are still some challenging research questions:

- 4) Can SDN architecture and its concept initially proposed for general networks be applied to IoT?
- 5) How can the researchers extend the OpenFlow protocol to provide IoT devices with upgrading and reprogramming?
- 6) How to guarantee the security and minimize the impact on performance at the same time in IoT?

1.6 Solution

We mainly aim to offer security mechanisms, especially key establishment protocols, in IoT. Thus, we have designed and implemented a secure and intelligent framework for IoT network and terminals, which is named as SAINT, aiming at providing an easily manageable networks and terminals for IoT. There are mainly three concerns that need to be solved.

The first point is the new architecture. As mentioned previously, our proposed solution incorporates the fundamental idea of SDN. Therefore, the proposed system involves both software defined networks and software defined function for the new IoT system. By introducing the core idea of SDN – decoupling the control logic from the underlying systems, it is expected that the IoT will also benefit from the major advantages of SDN, purporting to be manageable, cost-effective, centralized, programmable and adaptable [32]. In particular, OpenFlow is an embodiment of the SDN concept in reality [33]. We deploy a powerful smart device called network station, to support SDN OpenFlow mechanisms. Then the data flow from end IoT devices that connects to the network station directly can be easily regulated and configured via network station according to the instructions from the SDN controller. As a result, the newly designed protocols can support IoT devices.

Furthermore, this framework enables IoT smart devices to be upgraded or reprogrammed securely and remotely. We design a function station located in traditional SDN data layer, usually being a more powerful smart device (e.g., Raspberry

Pi) which can be placed quite far away from the controller; but nearby the IoT end devices (e.g., Arduino R3). In control layer, an SDF controller is dedicated to managing and controlling the function of end devices via the function station. To support IoT devices, we present OpenFunction protocol, which is detailed stated in the following chapter.

The third vital part of the design is a proper security mechanism. To solve this, a suite of protocols is developed and deployed with this system to meet security requirements including integrity and authentication for sensitive data if necessary. Besides, the adopted security mechanisms in SAINT system are lightweight since many IoT devices are not very powerful enough. Moreover, we use model checking method to verify its authenticity.

Particularly, four security protocols are designed in SAINT system as the lightweight mechanism to secure communication. Hence, the new proposed protocols mainly include:

- **Protocol I:** Authenticated OpenFlow Association Protocol. It is used to securely initialize the communication between a controller and a smart device called network station. This protocol is based on the connection setup procedure of OpenFlow.
- **Protocol II:** Secure OpenFlow Message Issuing Protocol. It is used to securely issue an OpenFlow instruction from a controller to a network station.
- **Protocol III:** Authenticated OpenFunction Association Protocol. It is used to securely initialize the communication between the controller and the smart device called function station after the TCP connection. It is also used for authentication and key agreement.
- **Protocol IV:** Secure OpenFunction Message Issuing Protocol. It is used for controller securely to issue a reprogramming command to the specific function station. The specific function station will automatically renew the corresponding IoT end devices after receiving the confidential reprogramming command from authenticated controller.

Last, to realize our proposal, we have designed a prototypical demo. We use common IoT devices for example, Raspberry Pi and Arduino to simulate the IoT devices and use

edited a standard SDN controller (e.g. POX) as our system controller. Through the experiment result and test performance, it is expected to validate the feasibility of the proposal. The entire code is in the attachment of the zip file.

1.7 Contribution

In summary, the primary contributions of this thesis are briefly summarized below:

- This work proposes a secure and intelligent software defined Internet of Things framework: SAINT. SAINT involves two aspects: (1) SDN for networking in an IoT system and (2) SDF for reprogramming IoT devices.
- This work designs and implements an authenticated OpenFlow association protocol, i.e. Protocol I in this thesis. Compared with existing schemes of OpenFlow for switches, this protocol realizes that an SDN controller and a computational limited IoT devices can authenticate each other and negotiate session keys using Protocol I.
- This work presents a secure OpenFlow message issuing protocol, i.e. Protocol II in this thesis. The SDN controller can issue an instruction or other messages to IoT devices securely using Protocol II without worrying about eavesdropping, replay attack, and session hijacking. Different keys are used for different cryptographic primitives and different transmission directions.
- This work puts forward an extension to OpenFlow called the OpenFunction, i.e. Protocol III, used for communication between the SDF controller and IoT devices when devices needed reprogramming. Protocol III in this thesis realizes authenticated handshake without increasing interaction between the two communicating parties compared to the original OpenFlow handshake protocol.
- This work implements a secure OpenFunction reprogramming protocol, i.e. Protocol IV. This protocol is used for SDF controller to securely issue a reprogramming command to a specified function station. By doing so, the controller can dynamically upgrade or reprogram the function prestored in IoT devices via a function station.
- This work also builds a demo SAINT system. Security of these four protocols is analyzed and formally checked. Performance of these protocols is analyzed

theoretically, and confirmed by experiments in SAINT. The experimental results also suggest that it incurs a very small overhead.

1.8 Publications

Conference:

1. Nian Xue, Xin Huang, Jie Zhang. S²Net: A Security Framework for Software Defined Intelligent Building Networks. The 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-16), Tianjin, China, 23-26 August, 2016.
2. Nian Xue, Lulu Liang, Xin Huang, Jie Zhang. POSTER: A Framework for IoT Reprogramming. 12th EAI International Conference on Security and Privacy in Communication Networks (SecureComm 2016), Guangzhou, China, 10-12 October, 2016.
3. Nian Xue, Lulu Liang, Jie Zhang, Xin Huang. An Access Control System for Intelligent Buildings. Accepted by The 9th EAI International Conference on Mobile Multimedia Communications (MOBIMEDIA 2016), Xi'an, China, 18–19 June, 2016.
4. Jiaren Cai, Nian Xue, Jie Zhang, Xin Huang. Evaluating performances of practically available hash functions in SDI message authentication. 1st Conference on Emerging Topics in Interactive Systems (ETIS 2016), Suzhou, China, 26-28 August, 2016.
5. Mi Li, Andi Xu, Nian Xue, Xin Huang, Jie Zhang, Qiankun Sheng. Enable Bitcoin Transaction in Public Transport Ticketing System. 1st Conference on Emerging Topics in Interactive Systems (ETIS 2016), Suzhou, China, 26-28 August, 2016.
6. Andi Xu, Mi Li, Xin Huang, Nian Xue, Jie Zhang, Qiankun Sheng. A Blockchain Based Micro Payment System for Smart Devices. 1st Conference on Emerging Topics in Interactive Systems (ETIS 2016), Suzhou, China, 26-28 August, 2016.
7. Andi Xu, Mi Li, Jiaren Cai, Nian Xue, Jie Zhang, Dawei Liu, Paul Craig, Xin Huang. Improving Efficiency of Authenticated OpenFlow Handshake using Coprocessors. The 8th International Conference on IT in Medicine and Education (ITME 2016), Fuzhou, China, 23-25 December 2016.

8. Jiehao Zhang, Xianbin Hong, Sheng-Uei Guan, Xuan Zhao, Xin Huang, Nian Xue. Maximum Gaussian Mixture Model for Classification. The 8th International Conference on IT in Medicine and Education (ITME 2016), Fuzhou, China, 23-25 December 2016.
9. Xianbin Hong, Jiehao Zhang, Sheng-Uei Guan, Di Yao, Nian Xue, Xin Huang. Incremental Maximum Gaussian Mixture Partition for Classification. Accepted by 2007 2nd Joint International Mechanical, Electronic and Information Technology Conference(JIMET 2017), Chongqing, China, 24-26 March, 2017.

Journal:

10. Jie Zhang, Nian Xue and Xin Huang. A Secure System For Pervasive Social Network-based Healthcare, IEEE ACCESS, 2017.

1.9 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, introductive literatures and underlying knowledge are briefly reviewed. Chapter 3 introduces the details of the SAINT system. Their security properties and involved protocols are discussed in Chapter 4. Security analyses are provided in Chapter 5. Chapter 6 includes a decryption of the prototype. The demo system is tested and the performance is studied. Finally, Chapter 7 draws the conclusion; and limitations and future work are proposed.

Chapter

2 Literature Review and Definition of Preliminaries

In this Chapter, we review relevant professional literatures, preliminaries and definitions.

2.1 SDN and OpenFlow

SDN, regarding as the future network, is an innovative way of virtualizing network. It allows network administrators to manage a whole networking system at an abstract level so that it will centralize and simplify control of network management. A standard SDN architecture is divided into three layers: application layer, control layer and data layer [25]. In addition to these three layers, two interfaces (Southbound API and Northbound API) between these layers are also crucial components.

OpenFlow protocol is a typical realization of SDN Southbound API [26], which gives access to the forwarding plane of a network switch or router over the network. Typically, a standard OpenFlow process is represented below:

Step 1. After launching controller and switch, a controller listens on a specific port (usually 6633).

Step 2. A switch sends a TCP connect request to the controller. After three-way handshake, there exists a TCP link between the controller and switch.

Step 3. After connection setup, the switch and controller send OFTP_HELLO message to each other for version negotiation.

Step 4. The controller sends OFPT_FEATURES_REQUEST message to switch in order to request the features of switch.

Step 5. After receiving the message from the controller, the switch sends its features to the controller with OFPT_FEATURES_REPLY message.

Step 6. The switch begins to forward data or execute other OpenFlow instructions from the controller.

A tsunami of network switch and router vendors have been involved in manufacturing OpenFlow supporting switches, such as IBM, Cisco, NEC, Pica8 and Huawei. OpenFlow nowadays, is still on its way of evolution. According to Pereini et al. [27], future development on OpenFlow would be more precise and concise.

2.2 SNet

SNet, derived from SDN, is a new network framework for IoT proposed in [15]. This network architecture is shown in Figure 2. Based on SDN technology, SNet also decouples network control and forwarding functions, making network control dynamic and programmable. The structure of SNet is divided into three layers: smart device layer, control layer and application layer.

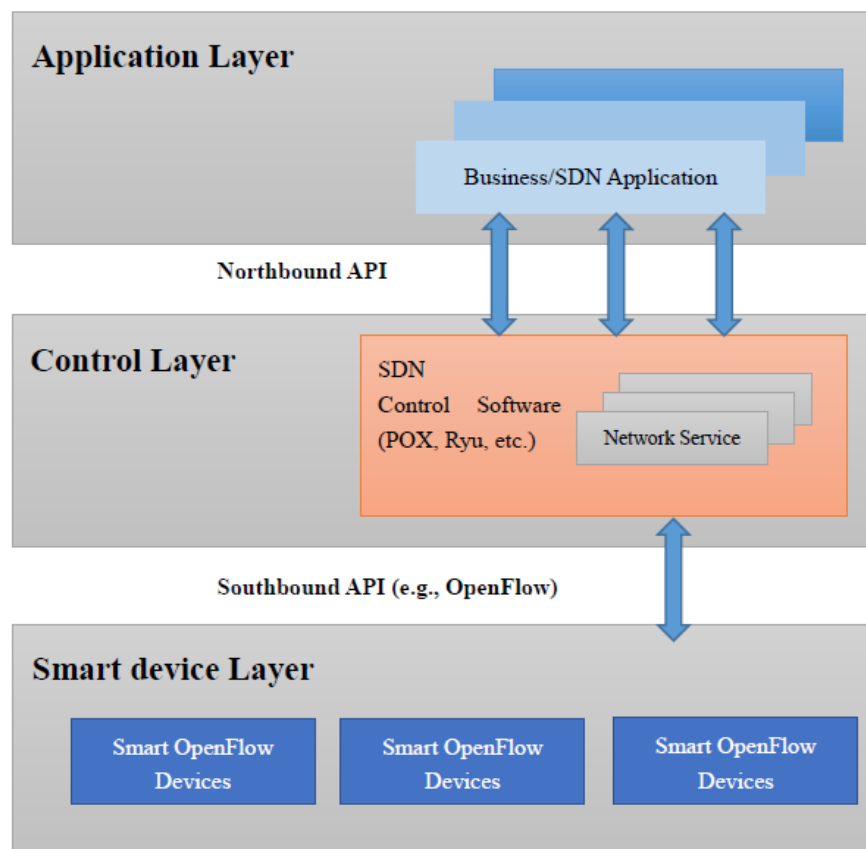


Figure 2 SNet Architecture

Each SNet architectural component is briefly defined and explained in the following paragraphs.

- **Application layer:** The top layer is the application layer, consisting of a variety of services, such as business applications, standard SDN applications. Applications in this layer can communicate with an SDN controller and define network policies provided by the controller through Northbound API.
- **Control layer:** The middle layer – the control layer, is the logic center of SNet, where the SDN controller can manage network services, and provide an abstract view of the whole network to the application layer. Meanwhile, it can integrate the business requirements, issue the switch instructions and control the forwarding on smart device layer. Some examples of open-source controller are OpenDayLight, Ryu, POX, FloodLight. Also, there are commercial SDN controllers such as XNC, SOX and Big Network Controller.
- **Smart device layer:** The lowermost layer is smart device layer, including many various network devices. Each smart OpenFlow device should obey the policies defined by the controller or network applications. However, in SNet architecture, every smart OpenFlow device can not only execute instructions from controller or application, but also can forward message to a specific destination. Even it can process extended OpenFlow instructions due to its programmable and flexible design.
- **Northbound API:** Northbound API is the interface between application layer and control layer. Its aim is to enable business applications to conveniently invoke the underlying network resources and capabilities.
- **Southbound API:** Southbound API is the interface between control layer and data layer. The controller manages network mainly through the southbound API by sending flow entry to switches. The most famous Southbound API is OpenFlow advocated by ONF. Some other protocols could be XMPP, PCEP, I2RS and OpFlex.

The key difference between SNet and SDN is the lowest layer. SDN aims at controlling switches in networks, however, SNet is trying to control smart IoT devices, for example

temperature sensors, gas sensors and fire alarms. So far, there is still few available Southbound API realizations that can support communications between the SDN controller and smart IoT devices. In [15], protocol stacks in smart devices that can support OpenFlow are designed. These smart devices are named as smart OpenFlow devices.

2.3 Secure SDN

SDN architecture can improve or enable network-related security [36], because the controller has a global view of the entire network. Thus, it can make a better policy to deal with the dynamics through reprogramming the data plane timely. While SDN architecture itself still remains some potential security problems that have generated enormous interest from researchers and vendors.

So far, large body of research has focused on enhancing the security between the application layer and the control layer or protecting the controller from malicious attacks. For instance, a large number of security mechanisms and frameworks have been developed for a single SDN controller to avoid malicious attacks from hackers [37], [38], [39], [40]. Furthermore, Flauzac et al. [41] propose a more complex SDN framework, including wireless network, wired network and Ad-Hoc networks. It is based on previous study, offering higher security level by using a single or multiple domains and distributed system. In a word, their studies have only emphasized the significance of security of controller in SDN architecture. As a result, these solutions are mainly designed to protect controllers from being exposed to the illegal users and hackers.

Schemes to solve the challenges of security between disparate layers in SDN architecture also get concerned gradually. Wen et al. [42] put forward a fine-grained access control system to improve the security between the control layer and the application layer by minimizing the privilege of OpenFlow apps after analyzing and summarizing a series of enforced permissions. Additionally, a new proposal presented by Kleadtke et al. [43], is safer, more comprehensive and as close as possible to the data layer. Besides, Banse and Rangarajan [44] constructed a secure northbound interface for an SDN controller offering network resources, via a REST-like API to register SDN applications. However, all the solutions mentioned in previous paragraph

focus on security between application and controller. The secure communication channel between the switch and controller is often neglected and receives relatively fewer attention, leaving a potential loophole for hackers.

What's more, the latest OpenFlow Specification ver. 1.4.0 has recommended of using the Transport Layer Security (TLS) to ensure the secure communication channel between a switch and controller [45]. Although the TLS usage is introduced in the OpenFlow specification, it is not a must-have demand. From Table 1, we can see that many OpenFlow equipment providers have not fully supported for TLS in current SDN switch and controller products.

Table 1 TLS Support in OpenFlow by Vendors. [46]

<i>Vendor</i>	<i>TLS Support</i>
HP switch	No
Brocade switch	Controller port only
Dell switch	No
NEC switch Partial	Partial
Indigo switch	No
Pica8 switch	Only new versions
Open vSwitch	Yes
NOX controller	No
Brocade Vyatta controller	Yes
POX controller	No
Beacon controller	No

Floodlight controller	No
MuL controller	No
FlowVisor	No
Big Network controller	Yes
Open Source controllers (i.e. Ryu, OpenDaylight)	Yes

Although this deployment without TLS is feasible in some cases considering the cost when the access to physical devices is difficult, it will become vulnerable if the network is less restricted. Recently, a few researchers who study on SDN have already realized this security flaw, and put forward several solutions for secure channel between control layer and data layer. Security of communication between SDN controller and the network devices has been studied gradually since the concepts of SDN and OpenFlow were introduced. Here we briefly summarize some typical works among them. In addition, we also compared these solutions with our security mechanism in Table 2.

Table 2 Comparison of Different Solution for Secure South API.

Solutions	Cryptographic Foundation	Number of Communications	Freshness of Key
[45]	Asymmetric cryptography	increased	Yes
[46]	Asymmetric cryptography	increased	Yes
[47]	Asymmetric cryptography	increased	Yes
[15]	Symmetric	increased	No

	cryptography		
[48]	Symmetric cryptography	increased	Yes
[49]	Asymmetric cryptography	not increased	Yes
Secure OpenFlow in this thesis	Symmetric cryptography	not increased	Yes
Secure OpenFunction in this thesis	Asymmetric cryptography	not increased	Yes

For example, Samociuk [46] proposed three different solutions for secure channel between controllers and network devices in SDN communicating with OpenFlow. The solutions include modified TLS, Secure Shell and IPsec. However, Samociuk's approaches are designed for general network, which might not be suitable for IoT devices because of the restricted resources of sensor nodes [50] and diversity. Moreover, this security architecture is inappropriate for IoT network due to the energy consumption and complexity.

In [47], authors use identity based method to realize authentication between the controller and devices. All of these works are based on asymmetric cryptography. Besides, compared with standard OpenFlow protocol without security mechanism, the solutions of these works increase the number of communications in each session.

Another solution, similar to SDN architecture, which is named as SNet, is a prototype proposed by Xu et al. [15], who also turn their focus on the communication security between the controller and the switch. However, it fails to generate a fresh session key and increases the handshake steps. Another similar security solution based on SNet using symmetric cryptography is proposed in [48], which is named as S²Net in that paper. [48] succeeds in creating a fresh key. Yet, the Extended OpenFlow Initialization uses seven steps of communication to complete authenticated OpenFlow handshake process, while a standard OpenFlow handshake protocol without security mechanism only needs four communications. Finally, almost all of these solutions focused on data

flow and routing rules, not reprogramming or upgrading function for the IoT devices.

2.4 Software Defined IoT

During recent years, a growing number of researchers have tried to introduce SDN key concept into IoT, and tentatively proposed various architecture for IoT, so-called SDIoT (Software Defined IoT) [60, 61, 62]. For example, Jararweh et al. [3] propose a software defined based framework for IoT, containing SDStore, SDSec and SDN. El-Mougy et al. [51] study and analyze some promising solutions derived from SDN for the IoT, aiming to manage resources of different kinds of networks such as WSN. In [23], the authors provide a brief up-to-date SDN-based IoT framework with NFV. Nevertheless, most researchers do not implement their design or demonstrate the feasibility. On the other hand, some researchers give their experiment results to support their conception. In Liu et al.'s paper [6], the authors present a SDIoT architecture for smart urban sensing to customize its data transmission, acquisition, and processing on-demand by physical devices. Another example is afforded by Qin et al. [52]. In that paper, the authors incorporate the SDN idea and design a software defined method for IoT in order to dynamically realize differentiated quality levels to diverse IoT tasks. Their results indicate that the SDN is meaningful and practicable for IoT.

Besides, the software defined things (SDT) tool [53] of Carnegie Mellon University enables users to build IoT system with virtual sensors. This tool would ease the implementation of SDF for IoT since it can manage abstract sensors that may encompass hierarchy of physical devices.

2.5 BAN and GNY logic

Security analysis using logics was first proposed in 1989 by the BAN logic (Burrows, Abadi and Needham logic) [55], which is a modal logic to reason about authentication protocols. It brings significant influence on the analysis methods for cryptographic protocols. However, Nessett [58] and Snekkenes [59] pointed out that BAN logic had several flaws. As a result, many researchers have refined BAN logic and made some extensions. GNY [56], as a successor of BAN, is considered to be one of the most popular formal analysis tools [57]. In this thesis, in order to verify the security of the

protocols, we utilize GNY to analyze the assumptions, goal and message transmission through formal analysis process.

2.6 Casper/FDR

In order to validate our proposed handshake protocols further, model checking method is used to verify the authenticity. In this thesis, we use communicating sequential processes (CSP, a formal language for describing patterns of interact in concurrent systems) [28], with its model refinement checker Failures-Divergences Refinement (FDR), which is proved to be an efficient way in analyzing the security of protocols [29]. The formal methods in this thesis, i.e. Casper/FDR [30], a program developed by University of Oxford, allows the user to write a description of a security protocol in a simple and abstract language, and produces a CSP description of the same protocol. It is very suitable for checking using FDR. And it is able to be employed either to discover attacks upon protocols, or to demonstrate that no such attack exists, subject to the assumptions of the Dolev-Yao Model (i.e., the intruder might eavesdrop or intercept messages, decrypt or encrypt messages with keys which he knows, and fake messages, but not execute any cryptological attacks) [31], [63].

2.7 Notations

We use the following notation to describe security protocols and cryptographic algorithms in this thesis:

- C, D, N and F are principals. C denotes the SDN/SDF controller. D denotes the IoT device. N denotes the network station and F denotes the function station.
- R_C, R_D, R_N and R_F are random numbers generated by C, D, N and F respectively (a number is unpredictable, often used to achieve freshness).
- PK_C and PK_F are public keys of the controller and function station respectively.
- SK_C and SK_F are private key of the controller and function station respectively.
- M denotes the message in the communication within a protocol run.
- $\{M\}_k$ denotes the ciphertext of message M encrypted by k .

- K_{CN} represents the key used for encrypting and decrypting messages sent by the SDN controller.
- K_{NC} represents the key used for encrypting and decrypting messages sent by the network station.
- K'_{CN} represents the key used for computing MAC for messages sent by the SDN controller.
- K'_{NC} represents the key used for computing MAC for messages sent by the network station.
- K_{CF} represents the key used for encrypting and decrypting messages sent by the SDF controller.
- K_{FC} represents the key used for encrypting and decrypting messages sent by the function station.
- K'_{CF} represents the key used for computing MAC for messages sent by the SDF controller.
- K'_{FC} represents the key used for computing MAC for messages sent by the function station.
- C_C represents the value of counter in the controller.
- C_N represents the value of counter in the network station.
- C_F represents the value of counter in the function station.

2.8 Definitions

Definition 1. Message Authentication Code (MAC). *A MAC is a short piece of information used to authenticate a message. It aims to prevent an adversary from modifying a message sent by one party to another, or from injecting a new message, without the receiver detecting that the message did not originate from the intended party [35].*

In this thesis, a MAC algorithm that takes input a secret key K to generate a MAC mac for a message M is denoted by the following equation:

$$mac = MAC(K, M)$$

Definition 2. Hash-and-MAC (HMAC). *There are many different ways to generate MACs. Among them, cryptographic hash function is a standardized and widely used method. Such construction is called HMAC [35].*

The HMAC algorithm to generate a MAC is denoted as following

$$mac = HMAC (K, M)$$

Definition 3. Digital Signature. *A signature scheme consists of three probabilistic polynomial-time algorithms (GEN, SIG, VER) such that:*

1. *The key-generation algorithm GEN takes as input a security parameter 1^n and outputs a pair of keys public key and private key (PK, SK) .*
2. *The signing algorithm SIG takes as input a private key SK and a message M from some message space (that may depend on PK). It outputs a signature σ written as $\sigma \leftarrow SIG (SK, M)$.*
3. *The deterministic verification algorithm VER takes as input a public key PK , a message M , and a signature σ . It outputs a bit b , with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := VER (M, \sigma)$.*

In this thesis, we denote the function to compute digital signature for a message M under the private key SK as follows:

$$\sigma = SIG (SK, H (M))$$

where $H ()$ is a hash function to hash the message down to a fixed-length digest.

In addition to the above definitions, the symmetric/asymmetric encryption and decryption algorithms are listed below.

- Symmetric encryption. The symmetric encryption algorithm that encrypts a message M under the secret key K is specified as follows:

$$\{M\}_K = SENC (K, M)$$

where $\{M\}_K$ is the ciphertext of M .

- Symmetric decryption. Below is the corresponding decryption algorithm of $SENC$:

$$M = SDEC (K, \{M\}_K)$$

- Asymmetric encryption. The asymmetric encryption algorithm uses the public key of an entity that only the entity can decrypt the ciphertext with its private key. Below is the asymmetric encryption under the public key PK :

$$\{M\}_{PK} = ASEC (PK, M)$$

where $\{M\}_{PK}$ denotes the ciphertext of M .

- Asymmetric decryption. The asymmetric decryption for $\{M\}_{PK}$ takes input
- the corresponding secret key SK is described as follows:

$$M = ASDEC (SK, \{M\}_{PK}).$$

Chapter

3 SAINT Overview

In this chapter we first describe the architecture of SAINT (secure and intelligent network and terminals). Then we briefly introduce the threat model, desired properties and message transmission. Figure 3 shows the logic view of our proposed architecture. As is shown in the figure, the framework has three logic layers: physical infrastructure layer, control layer and application layer. At first, we roughly explain the participants, three layers and networking, then we describe the overall process.

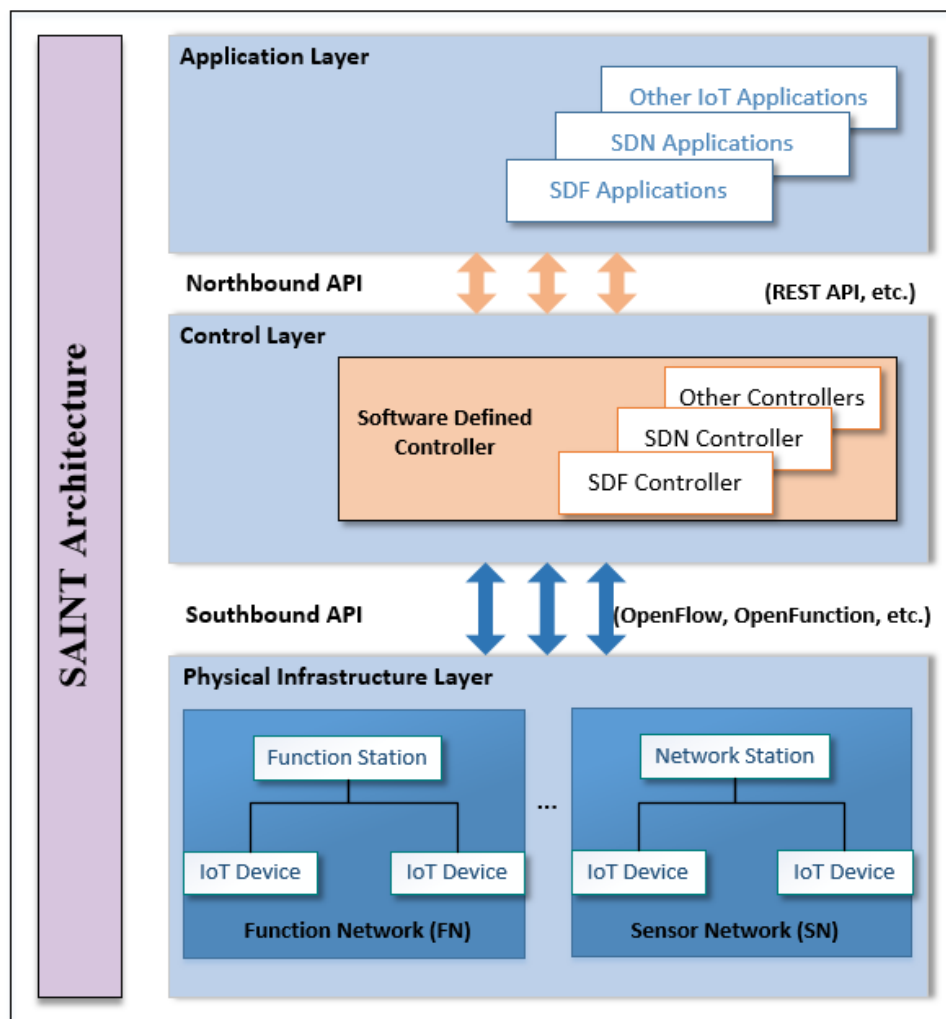


Figure 3 SAINT Architecture

3.1 Participants

In our proposed framework, it comprises four kinds of participants: IoT device, function station, network station and SDN/SDF controller. Below is the detailed description of these participants.

- **IoT device.** The IoT devices in our framework are inexpensive and low energy devices (e.g., Arduino R3, Arduino Nano and etc.). It consists of heterogeneous smart entities, for example, temperature sensor, noise sensor and PM 2.5 sensor. Figure 4 shows the abstract structure of a typical IoT device. The basic responsibility of the IoT devices is receiving data from the real world (the sensor) or causing events in the world (the actuator). In practice, the devices with certain functions are deployed in different place to complete special tasks for IoT applications.
- **Function station.** The function station in the function network (FN) works like a base station in sensor network, which is a more powerful IoT device (e.g., Raspberry Pi) than the IoT devices. It connects to IoT devices directly, and is responsible for upgrading or reprogramming the IoT devices. The function station and the IoT devices are generally placed nearby.
- **Network station.** The network station in the sensor network (SN) works like a switch. Its responsibility is to forward data from the subordinate IoT devices to the destination. Also, it can support SDN OpenFlow mechanisms and can adapt the forward rule according to the controller.
- **SDN/SDF controller.** The SDN/SDF controller can be integrated to a powerful and multifunctional server that connects with one or more function station or network station. It plays a pivotal role in the framework, similar to the "brain" of the human being. Since it can remotely control and manages the function station and network station according to the dynamic requirements from application layer, it is unnecessary to be deployed near the IoT end devices.

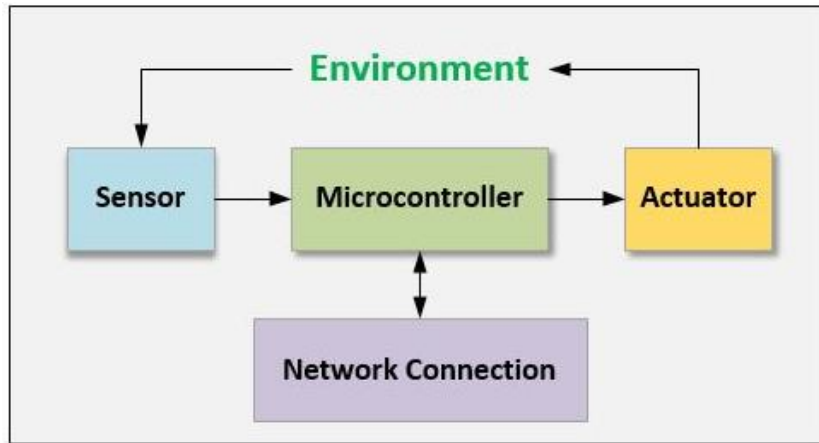


Figure 4 Structure of IoT Device

3.2 Three-layer Structure

From Figure 3, we can see that SAINT is logically divided into three layers from top to bottom: the application layer, the control layer and the physical infrastructure layer. The responsibilities of each layer are described below.

- **Application Layer.** The application layer is on the top of SDIoT. It comprises a series of applications: SDN applications, SDF applications and other IoT applications. They provide users or administrators with various services with the help of the controllers in control layer via a northbound interface (NBI).
- **Control Layer.** The middle layer of SAINT is the control layer in which there are divergent kinds of controllers for corresponding applications. For instance, the SDN controller is responsible for network traffic control, forwarding functions, etc.; and the SDF controller is allocated to reprogram or upgrade the IoT end devices according to new demands from application layer.
- **Physical Infrastructure Layer.** The physical infrastructure layer is the lowermost layer in SAINT architecture. This layer is composed of assorted physical devices that can be mainly categorized into two types: station and IoT device. The station is in charge of executing command from upper controller and managing its subordinate end terminals. There are various types of stations, for instance, network station controls network; data station manages data and function station renews function, etc. What's more, a powerful IoT device can serve as multiple functions. IoT terminal devices are often less powerful with limited resources, energy supply

and storage that have heterogeneous capabilities according to divergent requirements.

3.3 Networking

The networks in SAINT framework include both sensor network and function network. The network connection can be wired or wireless. Particularly, the controller and the network/function station is usually connected by the Internet or LAN, while the IoT end devices (e.g., PM 2.5 sensor, moisture sensor and etc.) are connected to the function station or network station through short range wireless or wire manners such as WiFi, Bluetooth, Zigbee, Serial line and so on. The networking is depicted in Figure 5 below.

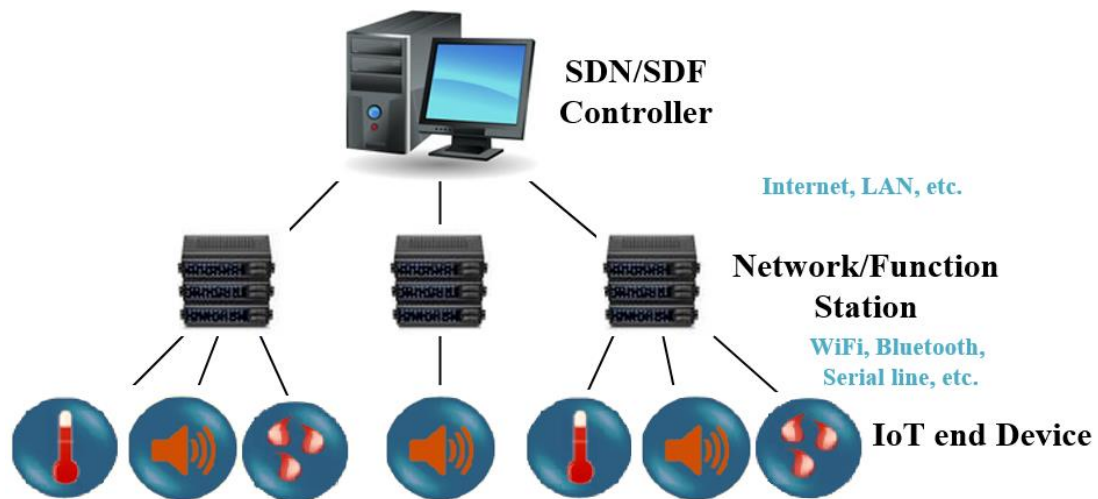


Figure 5 SAINT Networking

3.4 Threat Model

Security of SDN has already been studied by researchers, with many solutions and protocols being proposed. Therefore, in SAINT, the key part which is closely related with security of the whole system. The goal of our SAINT model is for an SDF controllers to legally and correctly reprogram IoT devices and for an SDN controller to control data flow flexibly. Each process involves two phases: connection setup phase and messages transmission phase. Consequently, we define the threat model according to the two phases as follows:

1. **Connection setup phase:** In the setup phase, we consider an attacker who tries to pretend to be a legal node and join the conversation (impersonation). Besides, the attacker also attempts to initiate a man-in-the-middle attack: (s)he secretly relays and possibly alters the communication between two parties who believe they are directly communicating with each other. Meanwhile, the attacker is trying to get the master key or session key.
2. **Message transmission phase:** After the connection setup phase, we consider an attacker is able to send fake messages, or maliciously repeat or delay valid messages. Besides, the attacker is trying to get some secret or sensitive information from controller-to-device instructions or messages. The adversary is also assumed to be able to observe and tamper with the traffic.

In addition to the above two-phase threat model, we also assume the adversary can obtain all session keys used in previous protocol runs in either of the two stages.

3.5 Desired Properties

Under the appearance of attackers specified in above threat mode, SAINT is designed to achieve the following security properties:

1. **Confidentiality.** All of the secret information (including the session keys, the master keys, the private keys and sensitive message) should not be known by attackers; also, sensitive instructions or messages should have encrypted to avoid the eavesdropping attack.
2. **Integrity.** Messages transmitted among the participants should be protected from unauthorized deletion, modification or fabrication during transmission.
3. **Authentication.** Proposed protocols need to resist impersonation attack. All of the participants should have the ability to identify whether the received messages are sent by the real source or an impersonation attacker.
4. **Lightweight.** The security mechanism of SAINT system should not bring too much communication and commutating burden to the plain system. (1) Many smart devices do not have enough computing resource to run asymmetric cryptographic algorithms. (2) Energy is a big concern in smart devices and thus expensive computations should be avoided to prolong its lifetime.

5. **Fresh key.** The rule and function might change over time, so we also must ensure the key freshness in communication phase for each session, and it ensure that no adversary replay old key. SAINT should establish fresh session keys in communication connection phase for each session. The session key is known only to the participants in the session.

3.6 Message Transmission

After the second stage, a channel is established between the controller and station. The following step is to transmit messages on the OpenFlow channel. There are three types of message: controller-to-device, asynchronous, and symmetric message.

- **Controller-to-device messages.** This type of messages is initiated by the controller and used to directly manage or inspect the state of the device. Besides, the controller-to-device messages may or may not require a response from the device.
- **Asynchronous messages.** This kind of messages is initiated by the device to update the controller of network events and changes to the device state.
- **Symmetric messages.** Symmetric messages can be initiated by either the device or the controller. This kind of messages is sent without solicitation.

3.7 SAINT Agent

We extend the standard OpenFlow protocol in order to support IoT devices. It is named as SAINT agent in this thesis. It is responsible for the communication between the controller and station. That is, it can parse messages from controllers and deal with these messages. According to the content of the head, the agent identifies the message type and handles parsed message to the controller or network/function station. The algorithm is described in Algorithm 1.

Algorithm 1 SAINT Agent Procedure

- 1: initialize TCP/IP connection
- 2: while (connection established)

```
3:  {
4:      parse the received msg
5:      if (msg is an OpenFlow msg)
6:          call OpenFlow procedure
7:          execute OpenFlow instruction
8:      else if (msg is an OpenFunction msg)
9:          call OpenFunction procedure
10:         execute OpenFunction instruction
11:     else
12:         send feedback msg
13: }
14: terminate TCP/IP connection
```

Chapter

4 Secure Southbound API

In this chapter, we introduce the secure application programming interface between controllers and stations. Typically, the Southbound API is used for the communication between the control layer and the physical infrastructure layer. One of the implementation of the Southbound API is OpenFlow in SDN. Addition to OpenFlow, we present OpenFunction for reprogramming IoT devices. Then we will introduce the two protocol suites respectively.

4.1 Secure OpenFlow

In the SAINT framework, we realize the secure OpenFlow between the SDN controller in the control layer and the network station in the physical infrastructure layer. The sensor network of SAINT primarily consists of security protocols between two entities: a network station (e.g., Raspberry Pi) and an SDN controller. There are two protocols, i.e. Protocol I: Authenticated OpenFlow Association Protocol and Protocol II: Secure OpenFlow Message Issuing Protocol. The whole process is shown in Figure 6.

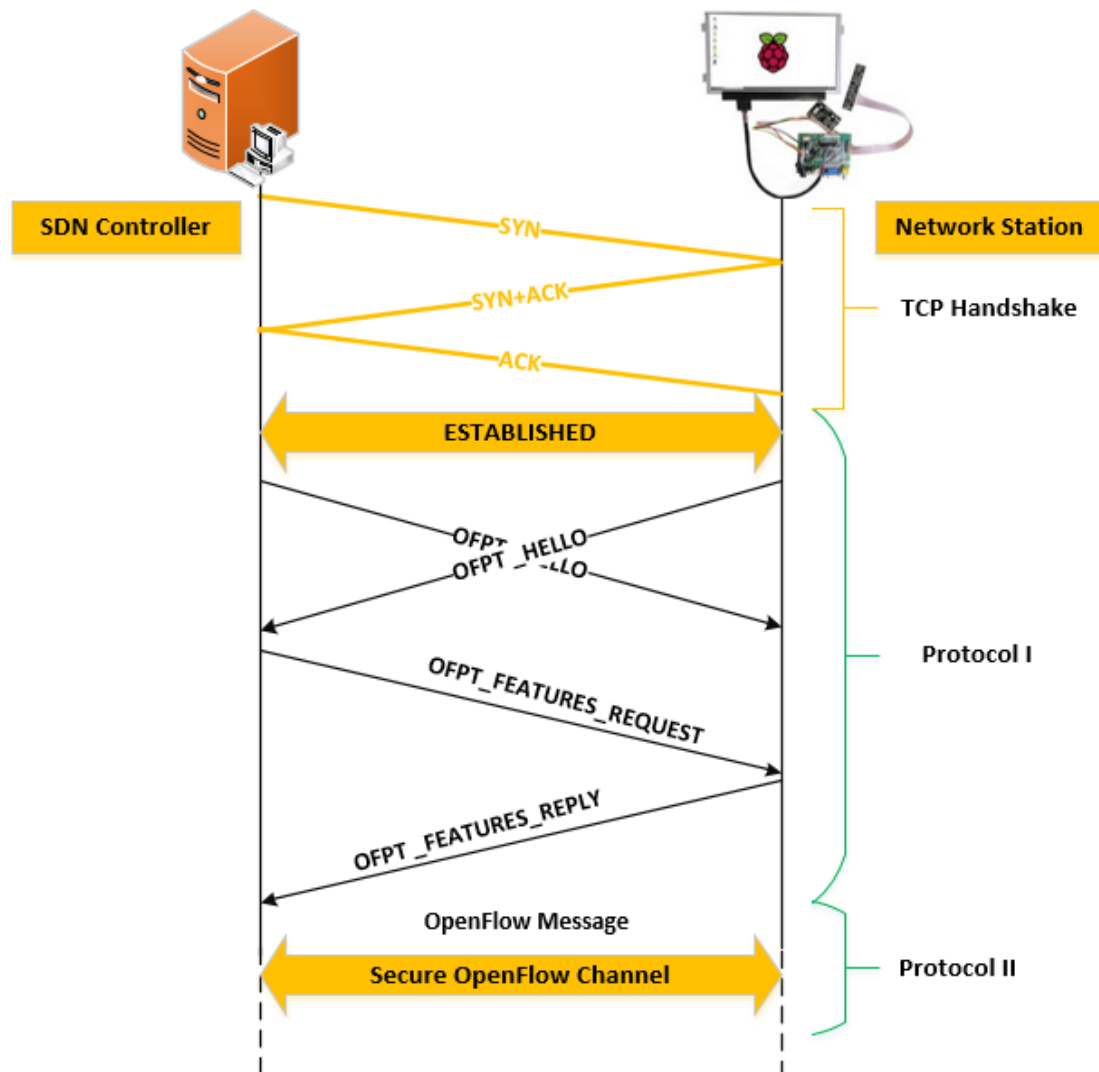


Figure 6 OpenFlow Protocol

4.1.1 OpenFlow Messages

Involved messages in secure OpenFlow are introduced below.

- **OFPT_HELLO:** It is used for version negotiation when a connection is established.
- **OFPT_FEATURES_REQUEST:** The feature request is always handled by the controller when version negotiation between two nodes is successful. This message is used for feature negotiation and belongs to controller-to-switch message.
- **OFPT_FEATURES REPLY:** Once the feature request is received, OpenFlow

device will report its capabilities to controller via feature reply message.

- **OFPT_ECHO_REQUEST/REPLY**: The two messages are symmetric messages that can be sent by either the device and the controller. The receiver of an echo request must return an echo reply to the sender. These messages are mainly used to verify the liveness of the connection between the device and the controller. Therefore, echo messages are very useful and common OpenFlow messages.

4.1.2 Protocol I: Authenticated OpenFlow Association Protocol

After the Connection Setup phase, the TCP connection between the network function and the SDN controller is established. Now each side of the TCP connection must immediately send a hello message to negotiate the OpenFlow protocol version. After the network station and the SDN controller have exchanged hello messages and successfully negotiated a common version number, the SDN controller will send a feature request to the network station and the device needs to reply its features to the controller.

The Authenticated OpenFlow Association Protocol is used to securely initialize the communication between the SDN controller and the network station. The protocol is described as follows.

1. A network function sends the hello message *OFPT_HELLO*, a nonce R_N and its identity N to an SDN controller.

$$msg1 = \langle OFPT_HELLO, N, R_N \rangle$$

2. The SDN controller also sends the *OFPT_HELLO* message, a nonce R_C and its identity C to the network station.

$$msg2 = \langle OFPT_HELLO, C, R_C \rangle$$

3. After receiving $msg1$, the SDN controller sends the *OFPT_FEATURES_REQUEST* with a MAC $mac1$ to the network station.

$$mac1 = HMAC(MK, msg1 || msg2 || OFPT_FEATURES_REQUEST)$$

$$msg3 = \langle OFPT_FEATURES_REQUEST, mac1 \rangle$$

4. After receiving msg_3 , the network station verifies mac_1 . If the verification succeeds, it will first compute the shared key K , and then replies the SDN controller with feature reply and a MAC mac_2 .

$$K = HMAC (MK, Rc || R_N)$$

$$mac_2 = HMAC (MK, ||msg_1||msg_2|| OFPT_FEATURES_REQUEST || \\ OFPT_FEATURES_REPLY)$$

$$msg_4 = < OFPT_FEATURES_REPLY, mac_2 >$$

5. After receiving msg_4 , the SDN controller verifies mac_2 . If the verification succeeds, the controller will forward the feature reply and compute the shared key K through the following equation.

$$K = HMAC (MK, Rc || R_N)$$

After the authenticated handshake, a secure OpenFlow channel between the SDN controller and the network station is established.

The aims of this protocol include: (1) to negotiate OpenFlow version; (2) to generate a shared session key between the controller and the device; (3) to realize mutual authentication for the controller and the network station.

Compared with the handshake of the original OpenFlow protocol, the authenticated OpenFlow association protocol has the following good features:

- **Integrity of handshake messages.** The authenticated handshake uses mac_1 and mac_2 to guarantee integrity of the hello messages, the feature request, and the feature reply, while in the original handshake established on TCP channels, these handshake messages could be modified during transmission without being detected.
- **Authentication.** In the authenticated handshake, the network station needs to take in the nonce Rc from the SDN controller to compute mac_2 , and the SDN controller needs to take in the nonce, i.e. R_N of the network station to compute mac_2 . This guarantees the authentication of the SDN controller and the network station.
- **Establishment of a session key.** A new shared key K is generated and kept secretly by the SDN controller and the network station both.

Corresponding to Chapter II, we review Protocol I as follows.

- Network Model. Communication parties of Protocol I are the SDN controller and the network station. It is used to establish the OpenFlow channel between the two participants.
- Threat Model. This protocol is corresponding to the connection setup phase, therefore, attackers of this protocol include impersonation attackers and man-in-the-middle attackers.
- Framework of Security Mechanism. This protocol guarantees confidentiality, authentication and integrity of all messages transmitted in the handshake phase. It also guarantees authentication between the SDF controller and the network station. Fresh session keys are established by this protocol, and it is light weight.
- Challenges. Since only symmetric cryptographic algorithms are used and the communications are not increased, this protocol does not increase much computational and communication cost for the two communication parties. Both of the network station and the SDF controller can afford the increased cost.

4.1.3 Protocol II: Secure OpenFlow Message Issuing Protocol

After Protocol I, a session key is established between the network station and the SDN controller. The session key can be used to protect OpenFlow commands. With the session key, we propose the Secure OpenFlow Message Issuing Protocol.

Here we take the transmission of echo request and echo reply as an example to show how our Protocol II protects the OpenFlow messages transmission. The whole process is described below.

1. The SDN controller encrypts the echo request and computes a MAC for the ciphertext. Then it sends the network station with $msg1$ as following

$$msg1 = < \{OFPT_ECHO_REQUEST\}_{K_{CN}}, mac1 >$$

where

$$\{OFPT_ECHO_REQUEST\}_{K_{CN}} = SENC(K_{CN}, OFPT_ECHO_REQUEST),$$

$$mac1 = HMAC(K'_{CN}, \{OFPT_ECHO_REQUEST\}_{K_{CN}} || CC)$$

2. The network station verifies $mac1$. If the verification succeeds, it will decrypt the ciphertext and get the echo request. According to the request, it replies the

controller with the following $msg2$

$$msg2 = \langle \{OFPT_ECHO_REPLY\}_{K_{NC}}, mac2 \rangle$$

where

$$\{OFPT_ECHO_REPLY\}_{K_{NC}} = SENC(K_{NC}, OFPT_ECHO_REPLY),$$

$$mac2 = HMAC(K'_{NC}, \{OFPT_ECHO_REPLY\})$$

Compared with the original OpenFlow protocol, Protocol II has the following security features:

- **Authentication.** Authentication in Protocol II means the receiver B can identify whether a message msg is sent by the legal participant A or by an attacker. It is guaranteed by the MAC mac . To compute and verify mac , A and B need to input the security key K'_{AB} and value of synchronized counter. Attackers are unable to compute mac since K'_{AB} is only kept by A and B . In addition, attackers cannot replay old macs since the counter changes in each run of the protocol.
- **Integrity.** Integrity means the receiver B can identify whether $\{M\}_{K_{AB}}$ has been modified during transmission. It is also guaranteed by mac . To compute and verify mac , $\{M\}_{K_{AB}}$, K'_{AB} and the value of counter are necessary inputs. If $\{M\}_{K_{AB}}$ is modified by an attacker, verification of mac will fail. Thus B can detect this modification.
- **Confidentiality.** Confidentiality in Protocol II means only A and B know M . It is guaranteed by the encryption algorithm. Before transmission, M is encrypted by A . To acquire M , one should use K_{AB} to decrypt the ciphertext. Since K_{AB} is only held by A and B , M is only known by A and B .

Corresponding to Chapter II, we review Protocol II as follows.

- **Network Model.** Communication parties of Protocol II are the SDN controller and the network station. Communications between the two parties take place over the OpenFlow channel.
- **Threat Model.** Protocol II is used in message transmission phase, which means that an adversary can observe and tamper with the traffic.

- **Framework of Security Mechanism.** This protocol is used to securing communication between SDN controller and the network station, which is the main concern of our design. Confidentiality, authentication and integrity of messages are guaranteed.
- **Challenges.** This protocol is also light weight since only symmetric cryptographic algorithms are applied. It does not increase computational and communication cost too much to the two participants, therefore, it is acceptable for the network station and the SDN controller.

4.1.4 OpenFlow Key Derivation

A good security design should not reuse the same cryptographic key for different cryptographic primitives. Thus, it would prevent any potential interaction between the primitives that might lead to a weakness. Therefore, in Protocol II we derive four independent keys for the encryption/decryption and HMAC operations. The keys are derived from the following function

$$F(K, x) = \text{HMAC}(K, x)$$

where F is another one-way function and we use HMAC algorithm here, K is the session key generated in the first stage by the Authenticated OpenFlow Handshake Protocol, and the domain of the variable x is the set of four integers $\{1, 2, 3, 4\}$. The key derivation algorithm is shown in Algorithm 2.

Algorithm 2 secure OpenFlow Key Derivation

- 1: initialization
- 2: $K = \text{HMAC}(MK, \text{Nonce}_C + \text{Nonce}_N)$
- 3: $K_{NC} = \text{HMAC}(K, '1')$
- 4: $K_{CN} = \text{HMAC}(K, '2')$
- 5: $K'_{NC} = \text{HMAC}(K, '3')$
- 6: $K'_{CN} = \text{HMAC}(K, '4')$

```

7:   if is SDN controller
8:   {
9:        $msg_{en} = \text{SENC}(msg, K_{CN})$ 
10:       $mac = \text{HMAC}(K'_{CN}, C_C + msg)$ 
11:  }
12:  if is network station
13:  {
14:       $msg_{en} = \text{SENC}(msg, K_{NC})$ 
15:       $mac = \text{HMAC}(K'_{NC} + msg)$ 
16:  }
17:  end key derivation

```

4.2 Secure OpenFunction

Similarly, we realize the secure OpenFunction between the SDF controller in the control layer and the function station in the physical infrastructure layer. Two security protocols are proposed, i.e. Protocol III Authenticated OpenFunction Association Protocol and Protocol IV Secure OpenFunction Message Issuing Protocol. Figure 7 shows the whole process.

The OpenFunction works as follows:

1. The user in the application layer submits his/her new requirement of an SDF application to the SDF controller.
2. In terms of the requirement, the SDF controller figures out the IoT device to be reprogrammed and the function station it connected to.
3. The SDF controller establishes a secure connection with the local function station and transmits the reprogramming messages to the function station.

4. According to the reprogramming messages from the authenticated SDF controller, the function station reprograms the IoT device with specified function stored in function station.

4.2.1 OpenFunction Messages

Involved messages in secure OpenFunction are introduced below.

- **OFCT_HELLO:** The OFCT_HELLO message can be sent by both of the controller and the function station. It consists of a set of hello element, a nonce and the identity of the sender. The hello element is used to negotiate the version and the type of OpenFunction, while the nonce and the identity are used for security purpose.
- **OFCT_FEATURES_REQUEST:** The OFCT_FEATURES_REQUEST message is sent by the controller to the function station. It consists of a set of device query element (such as the identity of the device to be reprogrammed) plus with the ciphertext of a secret and a digital signature. This message is used to query the state of a specified device connected to the function station.
- **OFCT_FEATURES_REPLY:** The OFCT_FEATURES_REPLY message is sent by the function station. It consists of a set of device reply element (such as the state of the device) and a signature. The function station stores a device table that specifies all of the states of devices connected to it. Once the function station receives an OpenFunction device query, it searches the state of the device and responses an OpenFunction device reply to the controller.
- **OFCT_REPROGRAMMING/CONFIRM:** The OFCT_REPROGRAMMING message is sent by the controller to the function station. OFCT_CONFIRM message is used to inform that the function station has confirmed the instruction. This kind of messages is transmitted in a secure channel established by OpenFunction handshake phase (explained later).

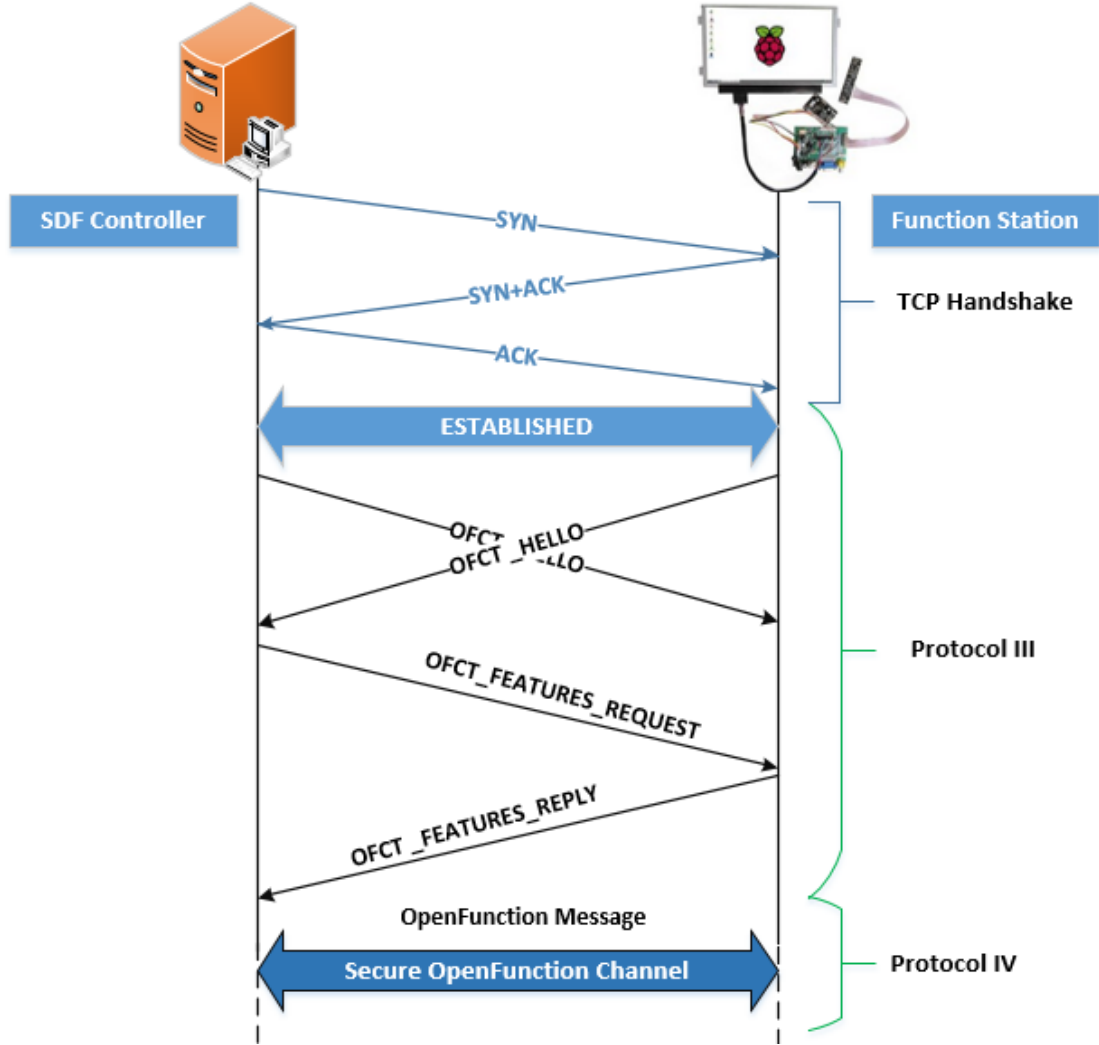


Figure 7 OpenFunction Protocol

4.2.2 Protocol III: Authenticated OpenFunction Association Protocol

The Authenticated OpenFunction Association Protocol is used to securely initialize the communication between the SDF controller and the smart device, which is inspired by [1], [7]. The aims of this protocol include: (1) to negotiate OpenFunction version; (2) to generate a shared session key between the SDF controller and the function station; (2) to realize mutual authentication for the SDF controller and the function station.

Assume an SDF controller has the public key PK_F of a function station, and the function station has the public key PK_C of the controller. The Authenticated OpenFunction Association Protocol is described as follows:

1. An SDF controller generates a nonce R_C and sends a function station with the following message $msg1$:

$$msg1 = \langle OFCT_HELLO, C, R_C \rangle.$$

2. The function station generates a nonce R_F and sends the controller with $msg2$ as follows:

$$msg2 = \langle OFCT_HELLO, F, R_F \rangle.$$

3. After the controller receives $msg2$, it chooses a random secret PMK and uses PK_F to encrypt it. Then the controller uses SK_C to generate a signature σ_C and replies the function station with the following $msg3$:

$$msg3 = \langle OFCT_FEATURES_REQUEST, \{PMK\}_{PK_F}, \sigma_C \rangle,$$

where

$$\sigma_C = SIG(SK_C, H(msg1 || msg2 || OFCT_FEATURES_REQUEST || \{PMK\}_{PK_F})).$$

4. When the function station receives $msg3$, it first checks σ_C . If σ_C is valid, the controller will decrypt $\{PMK\}_{PK_F}$ and compute the session key K . Then it generates a signature σ_F and replies the controller with $msg4$ as follows:

$$K = HMAC(PMK, R_C || R_F),$$

$$\sigma_F = SIG(SK_F, H(msg1 || msg2 || OFCT_FEATURES_REQUEST ||$$

$$\{PMK\}_{PK_F} || OFPT_FEATURES_REPLY))$$

$$msg4 = \langle OFPT_FEATURES_REPLY, \sigma_F \rangle.$$

5. After receiving $msg4$, the controller checks σ_F and computes the session key K as follows:

$$K = HMAC(PMK, R_C || R_F).$$

Protocol III has the following good features:

- **Authentication.** Authentication of messages and participants is guaranteed by digital signatures σ_C and σ_F . The two participants use their private key to sign all the transmitted messages.
- **Integrity of handshake message.** Integrity of handshake messages is also provided by digital signatures.

- **Establishment of a session key.** A session key is shared between the two participants. It can be used to protect OpenFunction reprogramming messages in the next phase.

Corresponding to Chapter II, we review Protocol III as follows.

- **Network model.** Communication parties of this protocol are the remote controller and the local controller. It establishes secure links and a session key between the two participants.
- **Threat model.** This protocol corresponds to the connection setup phase. Attackers to this protocol include impersonation attackers and man-in-the-middle attackers.
- **Framework of security mechanism.** Protocol III provides authentication and integrity for all messages transmitted in the handshake phase. It also provides confidentiality of secret values.
- **Challenges.** This protocol does not increase communication of OpenFunction handshake. Therefore, it will not increase too much cost for the two participants.

4.2.3 Protocol IV: Secure OpenFunction Message Issuing Protocol

After Authenticated OpenFunction Association process, a session key K is distributed between the controller and the function station. Protocol IV is used to transmit the OpenFunction reprogramming messages. The protocol is described as follows:

1. The SDF controller uses symmetric encryption algorithm to encrypt OpenFunction reprogramming messages $OFCT_REPROGRAMMING$ under the key K_{CF} . Then it uses another key K'_{CF} to generate a MAC mac for the ciphertext and the value of counter, i.e.

$$C \rightarrow F : msg = \langle \{OFCT_REPROGRAMMING\}_{K_{CF}}, mac \rangle,$$

where

$$\{M\}_{K_{CF}} = SENC(K_{CF}, OFCT_REPROGRAMMING)$$

and

$$mac = HMAC(K'_{CF}, \{OFCT_REPROGRAMMING\}_{K_{CF}} || Cc).$$

2. When the function station receives msg , it first checks the MAC mac . If the mac is valid, the function station will decrypt $\{OFCT_REPROGRAMMING\}_{K_{CF}}$ to acquire $OFCT_REPROGRAMMING$.

After the above process, the function station can reprogram the device according the content of $OFCT_REPROGRAMMING$. Then it replies the controller with the following $msg2$

$$msg2 = \langle \{OFCT_CONFIRM\}_{K_{FC}}, mac2 \rangle$$

where

$$\{OFCT_CONFIRM\}_{K_{FC}} = SENC(K_{FC}, OFCT_CONFIRM),$$

$$mac2 = HMAC(K'_{FC}, \{OFCT_CONFIRM\})$$

Protocol IV has the following security features:

- **Authentication.** Authentication is guaranteed by the MAC mac , which is computed by inputting the secret key K'_{CF} and the value of counter. In addition, attackers cannot replay old $macs$ since the counter changes in each run of the protocol.
- **Integrity of OpenFunction messages.** Integrity of OpenFunction messages is also guaranteed by mac .
- **Confidentiality.** The OpenFunction reprogramming message is encrypted under K_{CF} before transmitted. Therefore, attackers cannot decrypt and get the message without K_{CF} . That is, this protocol provides confidentiality for reprogramming messages.

Corresponding to Chapter II, we review Protocol IV as follows.

- **Network model.** Communication parties of this protocol are the remote controller and the local controller. It transmits OpenFunction reprogramming messages between the two participants.
- **Threat model.** This protocol corresponds to the message transmission phase. Attackers in this phase intend to send fake messages, repeat or delay messages and acquire sensitive messages.
- **Framework of security mechanism.** Protocol IV provides authentication,

integrity and confidentiality for all OpenFunction reprogramming messages.

- **Challenges.** This protocol uses symmetric encryption, decryption and HMAC. These algorithms are lightweight. Therefore, it will not increase too much cost.

4.2.4 OpenFunction Key Derivation

It is more secure to avoid reusing the same cryptographic key for different cryptographic primitives, since potential interaction between the primitives might lead to a weakness. Therefore, in Protocol IV, instead of use the session key K directly, we derive four independent keys from K for different operations.

The following function is used for key derivation:

$$F(K, x) = \text{HMAC}(K, x)$$

where F is a one-way function such as cryptographic hash functions, the domain of the variable x is the set of four integers $\{1, 2, 3, 4\}$. Algorithm 3 below illustrates the detail of the key derivation process and the use of the derived keys.

Algorithm 3 secure OpenFunction Key Derivation

```

1:  initialization
2:   $K = \text{HMAC}(MK, \text{Nonce}_C + \text{Nonce}_F)$ 
3:   $K_{CF} = \text{HMAC}(K, '1')$ 
4:   $K_{FC} = \text{HMAC}(K, '2')$ 
5:   $K'_{CF} = \text{HMAC}(K, '3')$ 
6:   $K'_{FC} = \text{HMAC}(K, '4')$ 
7:  if is SDF controller
8:  {
9:       $\text{msg}_{en} = \text{SENC}(\text{msg}, K_{CF})$ 
10:      $\text{mac} = \text{HMAC}(K'_{CF}, Cc + \text{msg})$ 
11:  }
```

```
12:  if is function station
13:  {
14:       $msg_{en} = \text{SENC}(msg, K_{FC})$ 
15:       $mac = \text{HMAC}(K'_{FC} + msg)$ 
16:  }
17:  end key derivation
```

Chapter

5 Security Analysis

In this chapter, security properties of four protocols in this thesis, i.e. Protocol I, II, III and IV, will be analyzed. In addition, the formal method to validate a protocol, i.e. Model Checking, is also used to verify Protocol I and Protocol III.

As we mentioned in Chapter 3.4, an impersonation attacker and a man-in-the-middle attacker are assumed to exist in the handshake phase and in the message transmission phase, the adversary is assumed to be able to observe and tamper with the traffic. In the following subsections, we will prove the security of our protocols under the above attacks.

5.1 Security Analysis of Protocol I

5.1.1 Security Under Impersonation Attacks

Theorem 1: *Suppose the network station and SDN controller have shared their master key, an impersonation attacker is not able to complete the handshake with any other of the SDN controller and network station.*

Proof: (1) Assume A_I is an attacker who attempts to impersonate the SDN controller and initialize Protocol I with the network station. The first two steps are executed as follows:

1. $A_I \rightarrow N: \overline{msg_1} = \langle \overline{OFPT_HELLO}, C, R_{A_I} \rangle$
2. $N \rightarrow A_I: \overline{msg_2} = \langle \overline{OFPT_HELLO}, N, R_N \rangle$

At the third step, A_I sends the network station with a feature request message $\overline{OFPT_FEATURES_REQUEST}$ and a MAC $\overline{mac_1}$, where the MAC can only be computed from $\overline{mac_1} = HMAC(MK, msg_1 || msg_2 || \overline{OFPT_FEATURES_REQUEST})$. The MAC $\overline{mac_1}$ will be verified in Step 4. Only the verification of $\overline{mac_1}$ succeeds, the network station will execute the remainder steps of Protocol I; else the protocol will terminate. However, since the adversary A_I does not have the pre-shared master key

MK , A_1 is not able to compute a valid MAC $\overline{mac1}$ in Step 3 so that $\overline{mac1}$ can pass the verification at the beginning of Step 4. Therefore, this impersonation attack will fail at the beginning of Step 4 of the protocol. The adversary fails to complete a handshake with the network station.

(2) On the other hand, assume A_2 is an adversary who intends to impersonate the network station and initialize Protocol I with the SDN controller. The first three steps execute as follows:

1. $A_2 \rightarrow C: \overline{msg1} = \langle \overline{OFPT_HELLO}, N, R_{A_2} \rangle$
2. $C \rightarrow A_2: msg2 = \langle OFPT_HELLO, C, R_C \rangle$
3. $C \rightarrow A_2: msg3 = \langle OFPT_FEATURES_REQUEST, \overline{mac1} \rangle$

where

$$\overline{mac1} = HMAC(MK, \overline{msg1} || msg2)$$

In the fourth step, A_2 first verifies $mac1$ and then computes the session key K from $\overline{K} = HMAC(MK, R_C || R_{A_2})$. However, the adversary does not have MK , therefore A_2 can not acquire \overline{K} . Moreover, in this step, A_2 should reply the SDN controller with $\overline{OFPT_FEATURES_REPLY}$ and $\overline{mac2}$, where $\overline{mac2} = HMAC(MK, \overline{msg1} || msg2 || OFPT_FEATURES_REQUEST || \overline{OFPT_FEATURES_REPLY})$. Then $\overline{mac2}$ will be verified when the SDN controller received this message. Only if the verification of $\overline{mac2}$ succeeds, the SDN controller will compute \overline{K} from $\overline{K} = HMAC(MK, R_C || R_{A_2})$. Since the adversary does not have MK , he/she are not able to compute a $\overline{mac2}$ that will pass the verification. Therefore, the SDN controller will not compute \overline{K} . In a word, this impersonation attack fails and A_2 does not establish a session key with the SDN controller.

From the above analysis (1) and (2), we can see that the impersonation attacker fails to complete the handshake with any other of the SDN controller and the network station.

5.1.2 Security Under Man-in-the-middle Attack

Theorem 2: Suppose the SDN controller and the network station have successfully shared a master key MK , a man-in-the-middle attacker is not able to complete the

authenticated OpenFlow handshake process between the SDN controller and network station without being detected.

Proof: Assume A_3 is a man-in-the-middle attacker between the SDN controller and network function. A_3 participants Protocol I as follows:

1. $C \rightarrow A_3: msg_1 = \langle OFPT_HELLO, C, R_C \rangle$
- $\bar{1}$. $A_3 \rightarrow N: \overline{msg_1} = \langle \overline{OFPT_HELLO}, C, R_{A_3} \rangle$
2. $N \rightarrow A_3: msg_2 = \langle OFPT_HELLO, N, R_N \rangle$
- $\bar{2}$. $A_3 \rightarrow C: \overline{msg_2} = \langle \overline{OFPT_HELLO}, N, R_{A_3} \rangle$
3. $C \rightarrow A_3: msg_3 = \langle OFPT_FEATURES_REQUEST, mac_1 \rangle$

where

$$mac_1 = HMAC(MK, msg_1 || \overline{msg_2} || OFPT_FEATURES_REQUEST)$$

- $\bar{3}$. At this stage, A_3 should send network station with the following message:

$$A_3 \rightarrow N: \overline{msg_3} = \langle \overline{OFPT_FEATURES_REQUEST}, \overline{mac_1} \rangle$$

where $\overline{mac_1}$ will be verified at the beginning of Step 4 through the following equation:

$$\overline{mac_1} = HMAC(MK, \overline{msg_1} || msg_2 || \overline{OFPT_FEATURES_REQUEST})$$

However, since the adversary does not have MK , he/she is not able to compute a valid $\overline{mac_1}$ to pass the verification. Therefore, the protocol terminates at the beginning of Step 4. The man-in-the-middle attack fails.

5.1.3 Security of the Session Key

The session key is agreed during Protocol I. According to Theorem 1 and Theorem 2 we can see Protocol I can resist impersonation attacks and man-in-the-middle attacks, which means messages transmitted during a completed run of Protocol I are sent from the true source and kept integrity when received by the true receiver, this can guarantee the authentication and integrity of the session key.

Here we analyze the security features of the session key as follows.

- **Integrity.** Integrity means the session keys computed by the SDN controller and the network station is equal. The reason is that the session key is computed through the following function:

$$K = HMAC (MK, R_C || R_N)$$

Since all the messages transmitted during Protocol I keep integrity, the MK , R_C , R_N held by the SDN controller equal with those values held by the network station. Therefore, the two participants compute an equal session key by the end of Protocol I.

- **Confidentiality.** Confidentiality of the session key means only the two participants of Protocol I know the session key. This feature is guaranteed by the MK , which is a necessary input to compute the session key. Since MK is pre-shared before the protocol, only the SDN controller and the network station know MK . Others without MK are not able to compute the session key from $K = HMAC (MK, R_C || R_N)$.
- **Forward Secrecy.** Forward secrecy is also an important feature of a session key. In Chapter 3.4 we assume the adversary can obtain all session keys used in previous protocol runs. Suppose the adversary gets a set of previous session keys $\{K_1, K_2 \dots, K_{i-1}\}$, it is difficult for him/her to derive K_i from these known values. This is because the two inputs R_C , R_N that used to compute K are randomly generated in each run of the protocol, which means these values are independent in different runs of the protocol.
- **Freshness.** Since $K = HMAC (MK, R_C || R_N)$ and MK , $R_C || R_N$ are newly generated in each run of the protocol, the session key K is fresh after every current run of the protocol.

5.2 Security Analysis of Protocol II

Theorem 3: *Suppose the SDN controller and the network station have successfully handshake with each other and have established a secret session key K , an adversary who attempts to alter the OpenFlow messages will be discovered by the receiver.*

Proof: Assume A_4 is an attacker who intends to tamper the traffic between the SDN controller and the function station. A_4 intercepts the following message:

$$C \rightarrow N : msg = \langle \{M\}_{K_{CN}}, mac \rangle$$

where

$$\{M\}_{K_{CN}} = SENC(K_{CN}, M)$$

and

$$mac = HMAC(K'_{CN}, \{M\}_{K_{CN}} || CC)$$

If the A_4 directly alters msg , the receiver will discover through verifying mac .

Additionally, A_4 may replace the message msg with a message $\overline{msg} = \langle \{\overline{M}\}_{K_{CN}}, \overline{mac} \rangle$ obtained from a previous run the Protocol II. In this case, the \overline{mac} can not pass the verification of the receiver. This is because the computation of the MAC involves the value of a counter, which is not repeated in each run of the protocol.

From the above analysis, we can see A_4 who wants to tamper the traffic will fail.

In addition to Theorem 3, Protocol II also guarantees the confidentiality of the OpenFlow messages, since the message is encrypted before transmission.

5.3 Security Analysis of Protocol III

5.3.1 Security Under Impersonation Attacks

Theorem 4. *Suppose the SDF controller and the function station have exchanged their public keys, an impersonation attacker is not able to complete the handshake with any other of the SDF controller and the function station.*

Proof. (1) Assume A_1 is an attacker who attempts to impersonate the SDF controller and initialize Protocol III with the function station. The first two steps are executed as follows:

1. $A_1 \rightarrow F : \overline{msg1} = \langle \overline{OFCT_HELLO}, C, R_{A1} \rangle$
2. $F \rightarrow A_1 : \overline{msg2} = \langle OFCT_HELLO, F, R_F \rangle$

At the third step, A_1 generates a random secret \overline{PMK} and a feature request message $\overline{OFCT_FEATURES_REQUEST}$. Then A_1 should forge a signature $\overline{\sigma C}$ of the SDF controller for the messages $\langle msg1, msg2, \overline{OFCT_FEATURES_REQUEST}, PMK_{A1} \rangle$.

The signature $\overline{\sigma_C}$ will be verified in Step 4. Only the verification of $\overline{\sigma_C}$ succeeds, the function device will execute the remainder steps of Protocol III; else the protocol will terminate. However, since the adversary A_1 does not have the private key of the SDF controller, A_1 is not able to generate a valid signature $\overline{\sigma_C}$ in Step 3 so that $\overline{\sigma_C}$ can pass the verification at the beginning of Step 4. Therefore, this impersonation attack will fail at the beginning of Step 4 of the protocol. The adversary fails to complete a handshake with the function station.

(2) On the other hand, assume A_2 is an adversary who intends to impersonate the function station and initialize Protocol III with the SDF controller. The first three steps execute as follows:

1. $C \rightarrow A_2: \overline{msg_1} = \langle \overline{OFCT_HELLO}, C, R_c \rangle$
2. $A_2 \rightarrow C: msg_2 = \langle OFCT_HELLO, F, R_{A_2} \rangle$
3. $C \rightarrow A_2: msg_3 = \langle OFCT_FEATURES_REQUEST, \{PMK\}_{PK_F}, \sigma_C \rangle$

In the fourth step, A_2 should decrypt $\{PMK\}_{PK_F}$ and use PMK to compute the session key. However, without the private key of the function station, A_2 is not able to decrypt $\{PMK\}_{PK_F}$, which means A_2 can not compute the session key.

Additionally, in Step 4, A_2 needs to generate $\overline{OFCT_FEATURES_REPLY}$ and forge a signature $\overline{\sigma_F}$ of the function station for the messages $\langle \overline{msg_1}, msg_2, OFCT_FEATURES_REQUEST, PMK_{PK_F}, \overline{OFCT_FEATURES_REPLY} \rangle$. However, without the private key of the function station, A_2 is not able to generate a $\overline{\sigma_F}$ that can pass the verification in Step 5. This means A_2 fails to complete the handshake process with the function station.

In a word, this impersonation attack fails and A_2 does not establish a session key with the function station.

From the above analysis (1) and (2), we can see that the impersonation attacker fails to complete the handshake with any other of the SDF controller and the function station.

5.3.2 Security Under Man-in-the-middle Attacks

Theorem 5. *Suppose the SDF controller and the function station have successfully*

exchanged their public keys, a man-in-the-middle attacker is not able to complete the OpenFunction handshake process between the SDF controller and the function station without being detected.

Proof. Assume A_3 is a man-in-the-middle attacker between the SDF controller and the function station. A_3 participants Protocol III as follows:

1. $C \rightarrow A_3: msg_1 = \langle OFCT_HELLO, C, N_C \rangle$

$\bar{1}$. $A_3 \rightarrow F: \overline{msg_1} = \langle \overline{OFCT_HELLO}, C, N_{A_3} \rangle$

2. $F \rightarrow A_3: msg_2 = \langle OFCT_HELLO, F, N_F \rangle$

$\bar{2}$. $A_3 \rightarrow C: \overline{msg_2} = \langle \overline{OFCT_HELLO}, F, N_{A_3} \rangle$

3. $C \rightarrow A_3: msg_3 = \langle OFCT_FEATURES_REQUEST, \{PMK\}_{PK_F}, \sigma_C \rangle$

where

$$\sigma_C = SIG(SK_C, H(msg_1 || \overline{msg_2} || OFCT_FEATURES_REQUEST || \{PMK\}_{PK_F}))$$

$\bar{3}$. At this stage, A_3 should send the function station with the following message:

$$A_3 \rightarrow F: \overline{msg_3} = \langle \overline{OFCT_FEATURES_REQUEST}, \{PMK\}_{PK_F}, \overline{\sigma_C} \rangle$$

where

$$\overline{\sigma_C} = SIG(SK_C, H(\overline{msg_1} || msg_2 || \overline{OFCT_FEATURES_REQUEST} || \{PMK\}_{PK_F}))$$

At the beginning of Step 4, the function station first verifies $\overline{\sigma_C}$ using PK_C . One condition for the verification being succeeded is that the message is signed by SK_C . However, since SK_C is only held by the SDF controller, in Step $\bar{3}$ A_3 is not able to compute a valid $\overline{\sigma_C}$ to pass the verification. The protocol terminates at the beginning of Step 4. The man-in-the-middle attack fails.

5.3.3 Security of the Session Key

The session key is agreed during Protocol III. According to Theorem 3 and Theorem 4 we can know Protocol III can resist impersonation attacks and man-in-the-middle attacks, which means messages transmitted during a completed run of Protocol III are sent from the true source and kept integrity when received by the true receiver, this can

guarantee the authentication and integrity of the session key.

Here we analyze the security features of the session key as follows.

- **Integrity.** Integrity means the session keys computed by the SDF controller and the function station is equal. The reason is that the session key is computed through the following function:

$$K = HMAC (PMK, R_C || R_F)$$

Since all the messages transmitted during Protocol III keep integrity, the PMK , R_C , R_F held by the SDF controller equal with those values held by the function station. Therefore, the two participants compute an equal session key by the end of Protocol III.

- **Confidentiality.** Confidentiality of the session key means only the two participants of Protocol III know the session key. This feature is guaranteed by the PMK , which is a necessary input to compute the session key. Since PMK is generated by the SDF controller and encrypted by the private key of the function station, only the SDF controller and the function station know PMK . Others without SK_F are not able to compute the session key from

$$K = HMAC (PMK, R_C || R_F).$$

- **Forward Security.** Forward security is also an important feature of a session key. In Chapter 3.4 we assume the adversary can obtain all session keys used in previous protocol runs. Suppose the adversary gets a set of previous session keys $\{K_1, K_2 \dots, K_{i-1}\}$, it is difficult for him/her to derive K_i from these known values. This is because all of the inputs PMK , N_C , N_F that used to compute K are randomly generated in each run of the protocol, which means these values are independent in different runs of the protocol.
- **Freshness.** Since $K = HMAC (PMK, R_C || R_F)$ and PMK , $R_C || R_F$ are newly generated in each run of the protocol, the session key K is fresh after every current run of the protocol.

5.4 Security Analysis of Protocol IV

Theorem 6. Suppose the SDF controller and the function station have successfully

handshake with each other and established a secret session key K , an adversary who attempts to alter the OpenFunction messages will be discovered by the receiver.

Proof. Assume A_4 is an attacker who intends to tamper the traffic between the SDF controller and the function station. A_4 intercepts the following message:

$$C \rightarrow F : msg = < \{M\}_{K_{CF}}, mac >$$

where

$$\{M\}_{K_{CF}} = SENC(K_{CF}, M)$$

and

$$mac = HMAC(K'_{CF}, \{M\}_{K_{CF}} || Cc)$$

If the A_4 directly alters msg , the receiver will discover through verifying mac .

Additionally, A_4 may replace the message msg with a message $\overline{msg} = < \{\overline{M}\}_{K_{CF}}, \overline{mac} >$ obtained from a previous run the Protocol IV. In this case, the mac can not pass the verification of the receiver. This is because the computation of the MAC involves the value of a counter, which is not repeated in each run of the protocol.

From the above analysis, we can see A_4 who wants to tamper the traffic will fail.

In addition to Theorem 3, Protocol IV also guarantees the confidentiality of the OpenFunction messages, since the message is encrypted before transmission.

5.5 GNY Logic Analysis

Here, we use GNY logic to analyze the security of our proposed four protocols.

5.5.1 GNY Notions and Notation

Before the verification, we first introduce the basic notions and their corresponding notation. A formula refers to a bit string, which would have a particular value in a run, like the name of a variable. Denote two formulae (i.e., variables) by X and Y respectively. Shared secrets and encryption keys are demoted as s and k respectively. A statement is some property of a formula. Denote two principals by P and Q . Denote a statement by C . The used expressions and their meanings are illustrated in Table 3.

Table 3 GNY Expression

Symbol	Meaning
(X, Y)	Conjunction of X and Y
$\{X\}_k, \{X\}_k^{-l}$	Symmetrical encryption and decryption
$\{X\}_{+k}, \{X\}_{-k}$	Asymmetrical encryption and decryption
$H(X)$	H is a one-way function of X . (e.g., <i>hash()</i>)
$F(X_1, \dots, X_n)$	F is a many-to-one computationally feasible function for any X_i
$*X$	X is a not-originated-here.
$X \rightsquigarrow C$	The precondition of X being conveyed is C . C is the message extension of X .
$P \text{ istold } X$	P is told X .
$P \text{ poss } X$	P possesses, or is capable of possessing X .
$P \text{ said } X$	P once conveyed X .
$P \text{ believes fresh } (X)$	P believes that X is fresh.
$P \text{ believes reco } (X)$	P believes that X is recognizable.
$P \text{ believes } P \xleftrightarrow{s} Q$	P believes that s is a suitable secret for P and Q .
$P \text{ believes } \xrightarrow{k^+} Q$	P believes that $+k$ is a suitable public key for Q .
$\langle s \rangle$	S is a secret message.

5.5.2 Logical Postulates

There are seven categories of postulates in GNY logic. In this section, we list the logical postulates and their description used in the proofs described later.

1) Being-Told Rules

$$\begin{aligned}
\mathbf{T1.} \quad & \frac{P \text{ istold } *X}{P \text{ istold } X} \\
\mathbf{T2.} \quad & \frac{P \text{ istold } (X, Y)}{P \text{ istold } X} \\
\mathbf{T4.} \quad & \frac{P \text{ istold } \{X\}^{+k}, P \text{ poss } -k}{P \text{ istold } X} \\
\mathbf{T6.} \quad & \frac{P \text{ istold } \{X\}^{-k}, P \text{ poss } +k}{P \text{ istold } X}
\end{aligned}$$

2) Possession Rules

$$\begin{aligned}
\mathbf{P1.} \quad & \frac{P \text{ istold } X}{P \text{ poss } X} \\
\mathbf{P2.} \quad & \frac{P \text{ poss } X, P \text{ poss } Y}{P \text{ poss } (X, Y), P \text{ poss } F(X, Y)} \\
\mathbf{P3.} \quad & \frac{P \text{ poss } (X, Y)}{P \text{ poss } X} \\
\mathbf{P4.} \quad & \frac{P \text{ poss } X}{P \text{ poss } H(X)} \\
\mathbf{P6.} \quad & \frac{P \text{ poss } k, P \text{ poss } X}{P \text{ poss } \{X\}_k, P \text{ poss } \{X\}_k^{-l}} \\
\mathbf{P8.} \quad & \frac{P \text{ poss } -k, P \text{ poss } X}{P \text{ poss } \{X\}^{-k}}
\end{aligned}$$

3) Freshness Rules

$$\begin{aligned}
\mathbf{F1.} \quad & \frac{P \text{ believes fresh } (X)}{P \text{ believes fresh } (X, Y), P \text{ believes fresh } (F(X))} \\
\mathbf{F10.} \quad & \frac{P \text{ believes fresh } (X), P \text{ poss } X}{P \text{ believes fresh } (H(X))}
\end{aligned}$$

4) Recognizability Rules

$$\mathbf{R1.} \quad \frac{P \text{ believes reco } (X)}{P \text{ believes reco } (X, Y), P \text{ believes reco } (F(X))}$$

$$\mathbf{R5.} \frac{P \text{ believes reco } (X), P \text{ poss } X}{P \text{ believes reco } (H(X))}$$

5) Message Interpretation Rules

$$\mathbf{I1.} \frac{P \text{ istold } * \{X\}_k, P \text{ poss } k, P \text{ believes } P \stackrel{k}{\leftrightarrow} Q, P \text{ believes reco } (X), P \text{ believes fresh } (X, k)}{P \text{ believes } Q \text{ said } X, P \text{ believes } Q \text{ said } \{X\}_k, P \text{ believes } Q \text{ poss } k}$$

$$\mathbf{I3.} \frac{P \text{ istold } * H(X, \langle s \rangle), P \text{ poss } (X, s), P \text{ believes } P \stackrel{s}{\leftrightarrow} Q, P \text{ believes fresh } (X, s)}{P \text{ believes } Q \text{ said } (X, \langle s \rangle), P \text{ believes } Q \text{ said } H(X)}$$

$$\mathbf{I6.} \frac{P \text{ believes } Q \text{ said } (X), P \text{ believes fresh } (X)}{P \text{ believes } Q \text{ poss } X}$$

$$\mathbf{I7.} \frac{P \text{ believes } Q \text{ said } (X, Y)}{P \text{ believes } Q \text{ said } X}$$

6) Jurisdiction Rules

$$\mathbf{J2.} \frac{P \text{ believes } Q \text{ controls } Q \text{ believes } *, P \text{ believes } Q \text{ said } (X \rightsquigarrow C), P \text{ believes fresh } (X)}{P \text{ believes } Q \text{ believes } C}$$

7) Rationality Rules

$$\text{If } \frac{C_1}{C_2} \text{ is a postulate, then for any principal } P, \text{ so is } \frac{P \text{ believes } C_1}{P \text{ believes } C_2}.$$

5.5.3 GNY Analysis of Protocol I

1) protocol paraphrase

The Protocol I consists of following three messages between SDN controller and network station. In order to simplify the analysis process, we re-write it as follows:

1. $N \rightarrow C: m1, N, n_n;$
2. $C \rightarrow N: m2, C, n_c, m3, H(\langle k \rangle, N, n_n, m1, C, n_c, m2, m3);$
3. $N \rightarrow C: m4, H(\langle k \rangle, N, n_n, m1, C, n_c, m2, m3, m4).$

2) protocol description

The parser algorithm would produce the following description of the protocol.

M1. C istold $*m_1, *N, *n_n$;

M2. N istold $*m_2, *C, *n_c, *m_3, *H(<k>, N, n_n, m_1, C, n_c, m_2, m_3)$;

M3. C istold $*m_4, *H(<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)$.

3) goal

In Protocol I, the secure channel for transferring OpenFlow message is constructed. Thus, we need to prove that N and C believe that each other possesses the fresh new session key (i.e., $K = \text{HMAC}(MK, R_C || R_N)$). Meanwhile, the session key should be recognizable and fresh.

For network stataion, we need to prove:

(a) N believes C poss $H(<k>, n_c, n_n)$;

(b) N believes fresh $(H(<k>, n_c, n_n))$;

(c) N believes reco $(H(<k>, n_c, n_n))$;

(d) C believes N poss $H(<k>, n_c, n_n)$;

(e) C believes fresh $(H(<k>, n_c, n_n))$;

(f) C believes reco $(H(<k>, n_c, n_n))$.

4) protocol initial assumption

We use **S1...Si** to represent initial assumption in proof. We assure that the following holds at the beginning of every run of the protocol:

S1. N poss k, m_1, m_4, N, n_n ;

S2. N believes $N \stackrel{k}{\leftrightarrow} C$;

S3. N believes fresh (n_n) ;

S4. N believes reco (n_n) ;

S5. C poss k, m_2, m_3, C, n_c ;

S6. C believes $N \stackrel{k}{\leftrightarrow} C$;

S7. C believes fresh (n_c) ;

S8. C believes reco (n_c).

5) protocol proof

(a) N believes C poss $H (<k>, n_c, n_n)$

From **M1**, applying **T1** and **P1** we obtain

$$\frac{\frac{C \text{ istold } *m_l}{C \text{ istold } m_l} \text{ (T1)}}{C \text{ poss } m_l} \text{ (P1)}$$

Similarly,

$$\frac{\frac{C \text{ istold } *N}{C \text{ istold } N} \text{ (T1)}}{C \text{ poss } N} \text{ (P1)}$$

$$\frac{\frac{C \text{ istold } *n_n}{C \text{ istold } n_n} \text{ (T1)}}{C \text{ poss } n_n} \text{ (P1)}$$

Thus, we can also obtain C poss N and C poss n_n .

Similarly, from **M2**, applying **T1** and **P1**

$$\frac{\frac{N \text{ istold } *m_2}{N \text{ istold } m_2} \text{ (T1)}}{N \text{ poss } m_2} \text{ (P1)}$$

$$\frac{\frac{N \text{ istold } *C}{N \text{ istold } C} \text{ (T1)}}{N \text{ poss } C} \text{ (P1)}$$

$$\frac{\frac{N \text{ istold } *n_c}{N \text{ istold } n_c} \text{ (T1)}}{N \text{ poss } n_c} \text{ (P1)}$$

$$\frac{\frac{N \text{ istold } *m_3}{N \text{ istold } m_3} \text{ (T1)}}{N \text{ poss } m_3} \text{ (P1)}$$

$$\frac{\frac{N \text{ istold } *H (<k>, N, n_n, m_l, C, n_c, m_2, m_3)}{N \text{ istold } H (<k>, N, n_n, m_l, C, n_c, m_2, m_3)} \text{ (T1)}}{N \text{ poss } H (<k>, N, n_n, m_l, C, n_c, m_2, m_3)} \text{ (P1)}$$

we can also obtain N poss m_2 , N poss C , N poss n_c , N poss m_3 and N poss $H (<k>, N, n_n, m_l, C, n_c, m_2, m_3)$

Applying **P2** and **S1** we obtain

$$\begin{array}{c}
\frac{N \text{ poss } k, N \text{ poss } N}{N \text{ poss } (k, N), N \text{ poss } (n_n)} \text{ (P2)} \\
\frac{N \text{ poss } (k, N, n_n), N \text{ poss } (m_1)}{N \text{ poss } (k, N, n_n, m_1), N \text{ poss } (C)} \text{ (P2)} \\
\frac{N \text{ poss } (k, N, n_n, m_1, C), N \text{ poss } (n_c)}{N \text{ poss } (k, N, n_n, m_1, C, n_c), N \text{ poss } (m_2)} \text{ (P2)} \\
\frac{N \text{ poss } (k, N, n_n, m_1, C, n_c, m_2), N \text{ poss } (m_3)}{N \text{ poss } (k, N, n_n, m_1, C, n_c, m_2, m_3)} \text{ (P2)}
\end{array}$$

Applying **F1** and **S3** we obtain

$$\frac{N \text{ believes fresh } (n_n)}{N \text{ believes fresh } (k, N, n_n, m_1, C, n_c, m_2, m_3)} \text{ (F1)}$$

Applying **I3**, **S2**, and **M2** we obtain

$$\begin{array}{c}
N \text{ istold } *H (<k>, N, n_n, m_1, C, n_c, m_2, m_3), N \text{ poss } (k, N, n_n, m_1, C, n_c, m_2, m_3), \\
\frac{N \text{ believes } N \stackrel{k}{\leftrightarrow} C, N \text{ believes fresh } (k, N, n_n, m_1, C, n_c, m_2, m_3)}{N \text{ believes } C \text{ said } (<k>, N, n_n, m_1, C, n_c, m_2, m_3)} \text{ (I3)}
\end{array}$$

Applying **I7** we obtain

$$\frac{N \text{ believes } C \text{ said } (<k>, N, n_n, m_1, C, n_c, m_2, m_3)}{N \text{ believes } C \text{ said } (<k>, n_c, n_n)} \text{ (I7)}$$

Applying **S3** and **F1** we obtain

$$\frac{N \text{ believes fresh } (n_n)}{N \text{ believes fresh } (<k>, n_c, n_n)} \text{ (F1)}$$

Applying **I6** we obtain

$$\frac{N \text{ believes } C \text{ said } (<k>, n_c, n_n), N \text{ believes fresh } (<k>, n_c, n_n)}{N \text{ believes } C \text{ poss } (<k>, n_c, n_n)} \text{ (I6)}$$

Applying Rationality Rules and **P4**, because

$$\frac{C \text{ poss } (<k>, n_c, n_n)}{C \text{ poss } H (<k>, n_c, n_n)} \text{ (P4)}$$

is a postulate. Additionally, $N \text{ believes } C \text{ poss } (<k>, n_c, n_n)$.

So we can obtain $N \text{ believes } C \text{ poss } H (<k>, n_c, n_n)$.

Thus goal **(a)** is proved.

(b) $N \text{ believes fresh } (H (<k>, n_c, n_n))$

Applying **P2** and **S1** we obtain

$$\frac{\frac{N \text{ poss } (<k>), N \text{ poss } (n_c)}{N \text{ poss } (<k>, n_c), N \text{ poss } (n_n)} \text{ (P2)}}{N \text{ poss } (<k>, n_c, n_n)} \text{ (P2)}$$

Then, applying **F10**

$$\frac{N \text{ believes fresh } (<k>, n_c, n_n), N \text{ poss } (<k>, n_c, n_n)}{N \text{ believes fresh } (H (<k>, n_c, n_n))} \text{ (F10)}$$

Thus goal **(b)** is proved.

(c) N believes reco $(H (<k>, n_c, n_n))$

Applying **R1** and **S4** we obtain

$$\frac{N \text{ believes reco } (n_n)}{N \text{ believes reco } (<k>, n_c, n_n)} \text{ (R1)}$$

Applying **R5** we obtain

$$\frac{N \text{ believes reco } (<k>, n_c, n_n) N \text{ poss } (<k>, n_c, n_n)}{N \text{ believes reco } (H (<k>, n_c, n_n))} \text{ (R1)}$$

Thus goal **(c)** is proved.

(d) C believes N poss $H (<k>, n_c, n_n)$

From **M3**, applying **T1** and **P1** we obtain

$$\frac{\frac{C \text{ istold } *m_4}{C \text{ istold } m_4} \text{ (T1)}}{C \text{ poss } m_4} \text{ (P1)}$$

Similarly,

$$\frac{\frac{C \text{ istold } *H (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)}{C \text{ istold } H (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)} \text{ (T1)}}{C \text{ poss } H (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)} \text{ (P1)}$$

we can also obtain

$C \text{ poss } H (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4).$

Applying **P2** and **S5** we obtain

$$\begin{array}{c}
\frac{C \text{ poss } (k), C \text{ poss } (N)}{C \text{ poss } (k, N), C \text{ poss } (n_n)} \text{ (P2)} \\
\frac{C \text{ poss } (k, N, n_n), C \text{ poss } (m_1)}{C \text{ poss } (k, N, n_n, m_1), C \text{ poss } (C)} \text{ (P2)} \\
\frac{C \text{ poss } (k, N, n_n, m_1, C), C \text{ poss } (n_c)}{C \text{ poss } (k, N, n_n, m_1, C, n_c), C \text{ poss } (m_2)} \text{ (P2)} \\
\frac{C \text{ poss } (k, N, n_n, m_1, C, n_c), C \text{ poss } (m_2)}{C \text{ poss } (k, N, n_n, m_1, C, n_c, m_2), C \text{ poss } (m_3)} \text{ (P2)} \\
\frac{C \text{ poss } (k, N, n_n, m_1, C, n_c, m_2), C \text{ poss } (m_3)}{C \text{ poss } (k, N, n_n, m_1, C, n_c, m_2, m_3), C \text{ poss } (m_4)} \text{ (P2)} \\
\hline
C \text{ poss } (k, N, n_n, m_1, C, n_c, m_2, m_3, m_4)
\end{array}$$

Applying **F1** and **S7** we obtain

$$\frac{C \text{ believes fresh } (n_c)}{C \text{ believes fresh } (k, N, n_n, m_1, C, n_c, m_2, m_3, m_4)} \text{ (F1)}$$

Applying **I3**, **S6** and **M3** we obtain

$$\begin{array}{c}
C \text{ istold } *H (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4), C \text{ poss } (k, N, n_n, m_1, C, n_c, m_2, m_3, m_4), \\
\frac{C \text{ believes } N \stackrel{k}{\leftrightarrow} C, C \text{ believes fresh } (k, N, n_n, m_1, C, n_c, m_2, m_3, m_4)}{C \text{ believes } N \text{ said } (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)} \text{ (I3)}
\end{array}$$

Applying **I7** we obtain

$$\frac{C \text{ believes } N \text{ said } (<k>, N, n_n, m_1, C, n_c, m_2, m_3, m_4)}{C \text{ believes } N \text{ said } (<k>, n_c, n_n)} \text{ (I7)}$$

Applying **S7** and **F1** we obtain

$$\frac{C \text{ believes fresh } (n_c)}{C \text{ believes fresh } (<k>, n_c, n_n)} \text{ (F1)}$$

Applying **I6** we obtain

$$\frac{C \text{ believes } N \text{ said } (<k>, n_c, n_n), C \text{ believes fresh } (<k>, n_c, n_n)}{C \text{ believes } N \text{ poss } (<k>, n_c, n_n)} \text{ (I6)}$$

Applying Rationality Rules and **P4**, because

$$\frac{N \text{ poss } (<k>, n_c, n_n)}{N \text{ poss } H (<k>, n_c, n_n)} \text{ (P4)}$$

is a postulate. Additionally, $C \text{ believes } N \text{ poss } (<k>, n_c, n_n)$.

So we can obtain $C \text{ believes } N \text{ poss } H (<k>, n_c, n_n)$.

Thus goal **(d)** is proved.

(e) $C \text{ believes fresh } (H (<k>, n_c, n_n))$

Applying **P2** and **S1** we obtain

$$\frac{\frac{C \text{ poss } (<k>), C \text{ poss } (n_c)}{C \text{ poss } (<k>, n_c), C \text{ poss } (n_n)} \text{ (P2)}}{C \text{ poss } (<k>, n_c, n_n)} \text{ (P2)}$$

Then, applying **F10**

$$\frac{C \text{ believes fresh } (<k>, n_c, n_n), C \text{ poss } (<k>, n_c, n_n)}{C \text{ believes fresh } (H (<k>, n_c, n_n))} \text{ (F10)}$$

Thus goal (e) is proved.

(f) C believes reco (H (<k>, n_c, n_n))

Applying **R1** and **S8** we obtain

$$\frac{C \text{ believes reco } (n_n)}{C \text{ believes reco } (<k>, n_c, n_n)} \text{ (R1)}$$

Applying **R5** we obtain

$$\frac{C \text{ believes reco } (<k>, n_c, n_n) C \text{ poss } (<k>, n_c, n_n)}{C \text{ believes reco } (H (<k>, n_c, n_n))} \text{ (R5)}$$

Thus goal (f) is proved.

5.5.4 GNY Analysis of Protocol II

After running Protocol I, the controller and network station obtain a new session key.

Denote $H (<k>, n_c, n_n)$ by K in protocol II.

1) protocol paraphrase

The Protocol II consists of following two messages between SDN controller and network station. Denote $H (<K>, 2)$ by k_{cn} , $H (<K>, 4)$ by k'_{cn} , Denote $H (<K>, 1)$ by k_{nc} , $H (<K>, 3)$ by k'_{nc} . In order to simplify the analysis process, we re-write it as follows:

1. $C \rightarrow N: \{m5\}_{k_{cn}}, H (<k'_{cn}>, \{m5\}_{k_{cn}}, Cc);$
2. $N \rightarrow C: \{m6\}_{k_{nc}}, H (<k'_{nc}>, \{m6\}_{k_{nc}}).$

2) protocol description

The parser algorithm would produce the following description of the protocol.

M1. $N \text{ istold } *(\{m5\}_{k_{cn}}, H (<k'_{cn}>, \{m5\}_{k_{cn}}, Cc);$

M2. $C \text{ istold } *(\{m6\}_{knc}, H(<k'_{nc}>, \{m6\}_{knc})).$

3) goal

In protocol II, the network station and SDN controller will use the session key generated in Protocol I to create four new keys. Then the network station and SDN controller will use the four keys to transfer OpenFlow messages securely. Hence, we need to prove the following goals:

- (a) $N \text{ poss } k_{cn}, N \text{ poss } k'_{cn}, N \text{ poss } k_{nc}, N \text{ poss } k'_{nc}, N \text{ poss } m5;$
- (b) $N \text{ believes } C \text{ poss } k_{cn}, N \text{ believes } C \text{ poss } k'_{cn}, N \text{ believes } C \text{ poss } k_{nc}, N \text{ believes } C \text{ poss } k'_{nc};$
- (c) $N \text{ believes fresh } (k_{cn}), N \text{ believes fresh } (k'_{cn}), N \text{ believes fresh } (k_{nc}), N \text{ believes fresh } (k'_{nc});$
- (d) $N \text{ believes } C \text{ said } m5;$
- (e) $C \text{ poss } k_{cn}, C \text{ poss } k'_{cn}, C \text{ poss } k_{nc}, C \text{ poss } k'_{nc}, C \text{ poss } m6;$
- (f) $C \text{ believes } N \text{ poss } k_{cn}, C \text{ believes } N \text{ poss } k'_{cn}, C \text{ believes } N \text{ poss } k_{nc}, C \text{ believes } N \text{ poss } k'_{nc};$
- (g) $C \text{ believes fresh } (k_{cn}), C \text{ believes fresh } (k'_{cn}), C \text{ believes fresh } (k_{nc}), C \text{ believes fresh } (k'_{nc});$
- (h) $C \text{ believes } N \text{ said } m6.$

4) protocol initial assumption

We use **S1...Si** to represent initial assumption in proof. We assure that the following holds at the beginning of every run of the protocol:

- S1.** $N \text{ poss } K;$
- S2.** $N \text{ poss } m6;$
- S3.** $N \text{ believes reco } (K);$
- S4.** $N \text{ believes reco } (m5);$
- S5.** $N \text{ believes fresh } (K);$
- S6.** $N \text{ believes } N \stackrel{k_{cn}}{\leftrightarrow} C;$

S7. N believes $N \xleftrightarrow{k'_{cn}} C$;

S8. C poss K ;

S9. C poss m_5 ;

S10. C believes reco (K);

S11. C believes reco (m_6);

S12. C believes fresh (K);

S13. C believes $N \xleftrightarrow{k_{nc}} C$;

S14. C believes $N \xleftrightarrow{k'_{nc}} C$.

5) protocol proof

(a) N poss k_{cn} , N poss k'_{cn} , N poss k_{nc} , N poss k'_{nc} , N poss m_5

As $k_{cn} = H(<K>, 2)$ and N poss 2, applying **P2** and **S1** we can obtain

$$\frac{N \text{ poss } <K>, N \text{ poss } 2}{N \text{ poss } (<K>, 2)} \text{ (P2)}$$

Applying **P4** we can obtain

$$\frac{N \text{ poss } (<K>, 2)}{N \text{ poss } H(<K>, 2)} \text{ (P4)}$$

Thus, N poss k_{cn} is proved.

Similarly,

$$\frac{N \text{ poss } (<K>, 4)}{N \text{ poss } H(<K>, 4)} \text{ (P4)}$$

Thus, N poss k'_{cn} is proved.

$$\frac{N \text{ poss } (<K>, 1)}{N \text{ poss } H(<K>, 1)} \text{ (P4)}$$

Thus, N poss k_{nc} is proved.

$$\frac{N \text{ poss } (<K>, 3)}{N \text{ poss } H(<K>, 3)} \text{ (P4)}$$

Thus, N poss k'_{nc} is proved.

Applying **M1**, **T1** and **T2** we obtain

$$\frac{\frac{N \text{ istold } *(\{m5\}_{kcn}, H(<k'_{cn}>, \{m5\}_{kcn}, CC)}{N \text{ istold } (\{m5\}_{kcn}, H(<k'_{cn}>, \{m5\}_{kcn}))} \text{ (T1)}}{N \text{ istold } \{m5\}_{kcn}} \text{ (T2)}$$

Applying **P1** we obtain

$$\frac{N \text{ istold } \{m5\}_{kcn}}{N \text{ poss } \{m5\}_{kcn}} \text{ (P1)}$$

Applying **P6** we obtain

$$\frac{N \text{ poss } kcn, N \text{ poss } \{m5\}_{kcn}}{N \text{ poss } \{\{m5\}_{kcn}\}_{kcn}^{-I}} \text{ (P6)}$$

Hence, we obtain **$N \text{ poss } m5$** .

Therefore, goal **(a)** is proved.

(b) $N \text{ believes } C \text{ poss } kcn, N \text{ believes } C \text{ poss } k'_{cn}, N \text{ believes } C \text{ poss } k_{nc}, N \text{ believes } C \text{ poss } k'_{nc}$

Applying Rationality Rules and **P4**, because

$$\frac{C \text{ poss } (<K>, 2)}{C \text{ poss } H(<K>, 2)} \text{ (P4)}$$

is a postulate. And $kcn = H(<K>, 2)$. Additionally, according to Protocol I we obtain $N \text{ believes } C \text{ poss } K$.

So we can obtain **$N \text{ believes } C \text{ poss } kcn$** .

Similarly, we can prove **$N \text{ believes } C \text{ poss } k'_{cn}, N \text{ believes } C \text{ poss } k_{nc}, N \text{ believes } C \text{ poss } k'_{nc}$** .

Therefore, goal **(b)** is proved.

(c) $N \text{ believes fresh } (kcn), N \text{ believes fresh } (k'_{cn}), N \text{ believes fresh } (k_{nc}), N \text{ believes fresh } (k'_{nc})$

Applying **S5** and **F1** we obtain

$$\frac{N \text{ believes fresh } (<K>)}{N \text{ believes fresh } (<K>, 2)} \text{ (F1)}$$

Applying **F10** we obtain

$$\frac{N \text{ believes fresh } (<K>, 2), N \text{ poss } (<K>, 2)}{N \text{ believes fresh } (H (<K>, 2))} \text{ (F10)}$$

Because $k_{cn} = H (<K>, 2)$, thus, **N believes fresh (k_{cn})** is proved.

Similarly, we can prove **N believes fresh (k'_{cn})**, **N believes fresh (k_{nc})** and **N believes fresh (k'_{nc})**.

Therefore, goal (c) is proved.

(d) N believes C said m_5

Applying **M1** and **T2** we obtain

$$\frac{N \text{ istold } *(\{m_5\}_{k_{cn}}, H (<k'_{cn}>, \{m_5\}_{k_{cn}}, CC)}{N \text{ istold } * \{m_5\}_{k_{cn}}} \text{ (T2)}$$

Applying **F1** we obtain

$$\frac{N \text{ believes fresh } (k_{cn})}{N \text{ believes fresh } (m_5, k_{cn})} \text{ (F1)}$$

Applying **I1**, **S4**, **S6** and **M1** we obtain

$$\frac{N \text{ istold } * \{m_5\}_{k_{cn}}, N \text{ poss } k_{cn}, N \text{ believes } N \stackrel{k_{cn}}{\leftrightarrow} C}{\frac{N \text{ believes reco } (m_5), N \text{ believes fresh } (m_5, k_{cn})}{N \text{ believes } C \text{ said } m_5, N \text{ believes } C \text{ said } \{m_5\}_{k_{cn}},} \text{ (I1)}}{N \text{ believes } C \text{ poss } k_{cn}}$$

Therefore, goal (d) is proved.

(e) C poss k_{cn} , C poss k'_{cn} , C poss k_{nc} , C poss k'_{nc} , C poss m_6

As $k_{cn} = H (<K>, 2)$ and C poss 2, applying **P2** and **S8** we can obtain

$$\frac{C \text{ poss } <K>, C \text{ poss } 2}{C \text{ poss } (<K>, 2)} \text{ (P2)}$$

Applying **P4** we can obtain

$$\frac{C \text{ poss } (<K>, 2)}{C \text{ poss } H (<K>, 2)} \text{ (P4)}$$

Thus, **C poss k_{cn}** is proved.

Similarly, we can prove **C poss k'_{cn}** , **C poss k_{nc}** and **C poss k'_{nc}** .

Applying **M2**, **T1** and **T2** we obtain

$$\frac{\frac{C \text{ istold } *(\{m_6\}_{k_{nc}}, H(<k'_{nc}>, \{m_6\}_{k_{nc}}))}{C \text{ istold } (\{m_6\}_{k_{nc}}, H(<k'_{nc}>, \{m_6\}_{k_{nc}}))} \quad (\text{T1})}{C \text{ istold } \{m_6\}_{k_{nc}}} \quad (\text{T2})$$

Applying **P1** we obtain

$$\frac{C \text{ istold } \{m_6\}_{k_{nc}}}{C \text{ poss } \{m_6\}_{k_{nc}}} \quad (\text{P1})$$

Applying **P6** we obtain

$$\frac{C \text{ poss } k_{nc}, C \text{ poss } \{m_6\}_{k_{nc}}}{C \text{ poss } \{\{m_6\}_{k_{nc}}\}_{k_{nc}}^{-I}} \quad (\text{P6})$$

Hence, we obtain **C poss m_6** .

Therefore, goal (e) is proved.

(f) C believes N poss k_{cn} , C believes N poss k'_{cn} , C believes N poss k_{nc} , C believes N poss k'_{nc}

Applying Rationality Rules and **P4**, because

$$\frac{N \text{ poss } (<K>, 2)}{N \text{ poss } H(<K>, 2)} \quad (\text{P4})$$

is a postulate. And $k_{cn} = H(<K>, 2)$. Additionally, according to Protocol I we obtain **C believes N poss K** .

So we can obtain **C believes N poss k_{cn}** .

Similarly, we can prove **C believes N poss k'_{cn} , C believes N poss k_{nc} , C believes N poss k'_{nc}** .

Therefore, goal (f) is proved.

(g) C believes fresh (k_{cn}), C believes fresh (k'_{cn}), C believes fresh (k_{nc}), C believes fresh (k'_{nc})

Applying **S12** and **F1** we obtain

$$\frac{C \text{ believes fresh } (<K>)}{C \text{ believes fresh } (<K>, 2)} \quad (\text{F1})$$

Applying **F10** we obtain

$$\frac{C \text{ believes fresh } (<K>, 2), C \text{ poss } (<K>, 2)}{C \text{ believes fresh } (H (<K>, 2))} \text{ (F10)}$$

Because $k_{cn} = H (<K>, 2)$, thus, **C believes fresh (k_{cn})** is proved.

Similarly, we can prove **C believes fresh (k'_{cn})**, **C believes fresh (k_{nc})** and **C believes fresh (k'_{nc})**.

Therefore, goal (g) is proved.

(h) C believes N said m_6

Applying **M2** and **T2** we obtain

$$\frac{C \text{ istold } *(\{m_6\}_{k_{nc}}, H (<k'_{nc}>, \{m_6\}_{k_{nc}})}{C \text{ istold } * \{m_6\}_{k_{nc}}} \text{ (T2)}$$

Applying **F1** we obtain

$$\frac{C \text{ believes fresh } (k_{nc})}{C \text{ believes fresh } (m_6, k_{nc})} \text{ (F1)}$$

Applying **I1**, **S11**, **S13** and **M2** we obtain

$$\frac{C \text{ istold } * \{m_6\}_{k_{nc}}, C \text{ poss } k_{nc}, C \text{ believes } N \stackrel{k_{nc}}{\leftrightarrow} C}{C \text{ believes reco } (m_6), C \text{ believes fresh } (m_6, k_{nc})} \text{ (I1)}$$

$$\frac{C \text{ believes } N \text{ said } m_6, C \text{ believes } N \text{ said } \{m_6\}_{k_{nc}}}{C \text{ believes } N \text{ poss } k_{nc}}$$

Therefore, goal (h) is proved.

5.5.5 GNY Analysis of Protocol III

1) protocol paraphrase

The Protocol III consists of following three messages between SDF controller and function station. In order to simplify the analysis process, we re-write it as follows:

1. $C \rightarrow F: m_1, C, n_c;$
2. $F \rightarrow C: m_2, F, n_f;$
3. $C \rightarrow F: m_3, \{s\}_{+k_f}, \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f})\}_{-k_c};$
4. $F \rightarrow C: m_4, \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4)\}_{-k_f}.$

2) protocol description

The parser algorithm would produce the following description of the protocol.

M1. F istold $*m_1, *C, *n_c$;

M2. C istold $*m_2, *F, *n_f$;

M3. F istold $*m_3, *\{s\}_{+k_f}, *\{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)\}_{-k_c}$;

M4. C istold $*m_4, *\{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)\}_{-k_f}$.

3) goal

In Protocol III, the secure channel for transferring OpenFunction message is constructed. Thus, we need to prove that F and C believe that each other possesses the fresh new session key (i.e., $K = \text{HMAC}(PMK, R_C || R_F)$). Meanwhile, the session key should be recognizable and fresh.

For network stataion, we need to prove:

(a) F poss $H(\langle s \rangle, n_c, n_f)$, F believes C poss $H(\langle s \rangle, n_c, n_f)$;

(b) F believes fresh $(H(\langle s \rangle, n_c, n_f))$;

(c) F believes reco $(H(\langle s \rangle, n_c, n_f))$;

(d) C poss $H(\langle s \rangle, n_c, n_f)$, C believes F poss $H(\langle s \rangle, n_c, n_f)$;

(e) C believes fresh $(H(\langle s \rangle, n_c, n_f))$;

(f) C believes reco $(H(\langle s \rangle, n_c, n_f))$.

4) protocol initial assumption

We use **S1...Si** to represent initial assumption in proof. We assure that the following holds at the beginning of every run of the protocol:

S1. F poss $m_2, m_4, N, n_f, +k_c, +k_f, -k_f, h$;

S2. F believes $\xrightarrow{+k_c} C$;

S3. F believes fresh (n_f) ;

S4. F believes reco (n_f) ;

S5. F believes $F \xleftrightarrow{h} C$;

S6. $C \text{ poss } s, m_1, m_3, C, n_c, +k_f, +k_c, -k_c, h;$

S7. $C \text{ believes } \xrightarrow{+k_f} F;$

S8. $C \text{ believes fresh } (n_c);$

S9. $C \text{ believes reco } (n_c);$

S10. $C \text{ believes } C \xleftrightarrow{h} F;$

S11. $C \text{ believes fresh } (s);$

S12. $F \text{ believes } C \text{ poss } \langle s \rangle;$

S13. $C \text{ believes } F \text{ poss } -k_f.$

5) protocol proof

(a) $F \text{ believes } C \text{ poss } H (\langle s \rangle, n_c, n_f)$

From **M1**, applying **T1** and **P1** we obtain

$$\frac{\frac{F \text{ istold } *m_1}{F \text{ istold } m_1} \text{ (T1)}}{F \text{ poss } m_1} \text{ (P1)}$$

Similarly,

$$\frac{\frac{F \text{ istold } *C}{F \text{ istold } C} \text{ (T1)}}{F \text{ poss } C} \text{ (P1)}$$

$$\frac{\frac{F \text{ istold } *n_c}{F \text{ istold } n_c} \text{ (T1)}}{F \text{ poss } n_c} \text{ (P1)}$$

we can also obtain $F \text{ poss } C$ and $F \text{ poss } n_c$.

Similarly, from **M3**, applying **T1** and **P1** we can also obtain

$F \text{ poss } m_3, F \text{ poss } \{s\}_{+k_f}, F \text{ poss } \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}), \langle h \rangle\}_{-k_c}.$

Applying **T4** we obtain

$$\frac{F \text{ istold } \{s\}_{+k_f}, F \text{ poss } -k_f}{F \text{ istold } s} \text{ (T4)}$$

Applying **P1** we obtain

$$\frac{F \text{ istold } s}{F \text{ poss } s} \text{ (P1)}$$

Applying **P2** we obtain

$$\frac{\frac{F \text{ poss } \langle s \rangle, F \text{ poss } n_c}{F \text{ poss } (\langle s \rangle, n_c), F \text{ poss } n_c} \text{ (P2)}}{F \text{ poss } (\langle s \rangle, n_c, n_f)} \text{ (P2)}$$

Applying **P4** we obtain

$$\frac{F \text{ poss } (\langle s \rangle, n_c, n_f)}{F \text{ poss } H (\langle s \rangle, n_c, n_f)} \text{ (P4)}$$

Thus, $F \text{ poss } H (\langle s \rangle, n_c, n_f)$ is proved.

From **M1**, applying **T1** and **P1** we obtain

$$\frac{\frac{F \text{ istold } *m_1}{F \text{ istold } m_1} \text{ (T1)}}{F \text{ poss } m_1} \text{ (P1)}$$

Similarly, we can also obtain $F \text{ poss } C$ and $F \text{ poss } n_c$.

Similarly, from **M3**, applying **T1** and **P1**,

$$\frac{\frac{F \text{ istold } *m_3}{F \text{ istold } m_3} \text{ (T1)}}{F \text{ poss } m_3} \text{ (P1)}$$

$$\frac{\frac{F \text{ istold } * \{s\}_{+k_f}}{F \text{ istold } \{s\}_{+k_f}} \text{ (T1)}}{F \text{ poss } \{s\}_{+k_f}} \text{ (P1)}$$

$$\frac{\frac{F \text{ istold } * \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)\}_{-k_c}}{F \text{ istold } \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)\}_{-k_c}} \text{ (T1)}}{F \text{ poss } \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)\}_{-k_c}} \text{ (P1)}$$

we can also obtain

$$F \text{ poss } m_3, F \text{ poss } \{s\}_{+k_f}, F \text{ poss } \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)\}_{-k_c}.$$

Applying **T4** we obtain

$$\frac{F \text{ istold } \{s\}_{+k_f}, P \text{ poss } -k_f}{F \text{ istold } s} \text{ (T4)}$$

Applying **P1** we obtain

$$\frac{F_{\text{istold } s}}{F_{\text{poss } s}} \quad (\mathbf{P1})$$

Applying **P2** and **S1** we obtain

$$\begin{array}{l}
\frac{F \text{ poss } m_1, F \text{ poss } C}{F \text{ poss } (m_1, C), F \text{ poss } n_c} \text{ (P2)} \\
\frac{F \text{ poss } (m_1, C, n_c), F \text{ poss } m_2}{F \text{ poss } (m_1, C, n_c, m_2), F \text{ poss } F} \text{ (P2)} \\
\frac{F \text{ poss } (m_1, C, n_c, m_2, F), F \text{ poss } n_f}{F \text{ poss } (m_1, C, n_c, m_2, F, n_f), F \text{ poss } m_3} \text{ (P2)} \\
\frac{F \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3), F \text{ poss } \{s\}_{+k_f}}{F \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}), F \text{ poss } h} \text{ (P2)} \\
\frac{F \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, h)}{} \text{ (P2)}
\end{array}$$

Applying **F1** and **S3** we obtain

$$\frac{N \text{ believes fresh } (nf)}{N \text{ believes fresh } (m1, C, nc, m2, F, nf, m3, \{s\}_{+kf}, h)} \quad (\mathbf{F1})$$

Applying **T6** we obtain

$$\frac{F \text{ istold } * (\{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}), \langle h \rangle\}_{-k_c}), F \text{ poss } +k_c}{F \text{ istold } * H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)} \quad (\text{T6})$$

Applying **I3** and **S5** we obtain

$$\frac{F \text{ istold } *H(m_1, C, nc, m_2, F, nf, m_3, \{s\}_{+k_f}, <h>), F \text{ poss } (m_1, C, nc, m_2, F, nf, m_3, \{s\}_{+k_f}, h), \quad F \text{ believes } F \xleftrightarrow{h} C, N \text{ believes fresh } (m_1, C, nc, m_2, F, nf, m_3, \{s\}_{+k_f}, <h>)}{F \text{ believes } C \text{ said } (m_1, C, nc, m_2, F, nf, m_3, \{s\}_{+k_f}, <h>)} \quad (I3)$$

Applying **I7** we obtain

$$\frac{F \text{ believes } C \text{ said } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, \langle h \rangle)}{F \text{ believes } C \text{ said } (n_c, n_f)} \quad (17)$$

Applying **S3** and **F1** we obtain

$$\frac{F \text{ believes fresh } (n_f)}{F \text{ believes fresh } (n_c, n_f)} \quad (\mathbf{F1})$$

Applying **I6** we obtain

$$\frac{F \text{ believes } C \text{ said } (nc, nf), F \text{ believes fresh } (nc, nf)}{F \text{ believes } C \text{ poss } (nc, nf)} \quad (I6)$$

Applying Rationality Rules and **P3**, because

$$\frac{C \text{ poss } (n_c, n_f)}{C \text{ poss } n_c, C \text{ poss } n_f} \text{ (P3)}$$

is a postulate. Additionally, F believes $C \text{ poss}(n_c, n_f)$.

So we can obtain F believes $C \text{ poss } n_c$ and F believes $C \text{ poss } n_f$.

Applying **S6** and **P2** we obtain

$$\frac{\frac{C \text{ poss } \langle s \rangle, C \text{ poss } n_c}{C \text{ poss } (\langle s \rangle, n_c), C \text{ poss } n_f} \text{ (P2)}}{C \text{ poss } (\langle s \rangle, n_c, n_f)} \text{ (P2)}$$

is a postulate. Additionally, F believes $C \text{ poss } (\langle s \rangle)$, F believes $C \text{ poss } n_c$ and F believes $C \text{ poss } n_f$.

So we can obtain F believes $C \text{ poss } (\langle s \rangle, n_c, n_f)$.

Applying Rationality Rules and **P4**, because

$$\frac{C \text{ poss } (\langle s \rangle, n_c, n_f)}{C \text{ poss } H (\langle s \rangle, n_c, n_f)} \text{ (P4)}$$

is a postulate. Additionally, F believes $C \text{ poss } (\langle s \rangle, n_c, n_f)$.

So we can obtain F believes $C \text{ poss } H (\langle s \rangle, n_c, n_f)$.

Thus goal **(a)** is proved.

(b) F believes fresh ($H (\langle s \rangle, n_c, n_f)$)

Applying **F1** we obtain

$$\frac{F \text{ believes fresh } (n_f)}{F \text{ believes fresh } (\langle s \rangle, n_c, n_f)} \text{ (F1)}$$

Applying **P2** and **S1** we obtain

$$\frac{\frac{F \text{ poss } (\langle s \rangle), N \text{ poss } (n_c)}{F \text{ poss } (\langle s \rangle, n_c), N \text{ poss } (n_f)} \text{ (P2)}}{F \text{ poss } (\langle s \rangle, n_c, n_f)} \text{ (P2)}$$

Then, applying **F10**

$$\frac{F \text{ believes fresh } (\langle s \rangle, n_c, n_f), F \text{ poss } (\langle s \rangle, n_c, n_f)}{F \text{ believes fresh } (H (\langle s \rangle, n_c, n_f))} \text{ (F10)}$$

Thus goal **(b)** is proved.

(c) ***F* believes reco (*H* (<*s*>, *n_c*, *n_f*)**

Applying **R1** and **S4** we obtain

$$\frac{F \text{ believes reco } (n_f)}{F \text{ believes reco } (<s>, n_c, n_f)} \text{ (R1)}$$

Applying **R5** we obtain

$$\frac{F \text{ believes reco } (<s>, n_c, n_f) \quad F \text{ poss } (<s>, n_c, n_f)}{F \text{ believes reco } (H (<s>, n_c, n_f))} \text{ (R1)}$$

Thus goal (c) is proved.

(d) ***C* poss *H* (<*s*>, *n_c*, *n_f*)**

From **M2**, applying **T1** and **P1** we obtain

$$\frac{\frac{C \text{ istold } *m_2}{C \text{ istold } m_2} \text{ (T1)}}{C \text{ poss } m_2} \text{ (P1)}$$

Similarly, we can also obtain *C* poss *F* and *C* poss *n_f*.

Similarly, from **M4**, applying **T1** and **P1** we can also obtain

C poss *m₄*, *C* poss {*H* (*m₁*, *C*, *n_c*, *m₂*, *F*, *n_f*, *m₃*, {*s*}_{+k_f}, *m₄*)}_{-k_f}.

Applying **P2** we obtain

$$\frac{\frac{C \text{ poss } <s>, C \text{ poss } n_c}{C \text{ poss } (<s>, n_c), C \text{ poss } n_f} \text{ (P2)}}{C \text{ poss } (<s>, n_c, n_f)} \text{ (P2)}$$

Applying **P4** we obtain

$$\frac{C \text{ poss } (<s>, n_c, n_f)}{C \text{ poss } H (<s>, n_c, n_f)} \text{ (P4)}$$

Thus, *C* poss *H* (<*s*>, *n_c*, *n_f*) is proved.

From **M2**, applying **T1** and **P1** we obtain

$$\frac{\frac{C \text{ istold } *m_2}{C \text{ istold } m_2} \text{ (T1)}}{C \text{ poss } m_2} \text{ (P1)}$$

Similarly, we can also obtain *C* poss *F* and *C* poss *n_f*.

Similarly, from **M4**, applying **T1** and **P1** we can also obtain

$C \text{ poss } m_4, C \text{ poss } \{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)\}_{-k_f}$

Applying **S6**, **P2** and **S1** we obtain

$$\begin{array}{c}
\frac{C \text{ poss } m_1, C \text{ poss } C}{C \text{ poss } (m_1, C), C \text{ poss } n_c} \text{ (P2)} \\
\frac{C \text{ poss } (m_1, C, n_c), C \text{ poss } m_2}{C \text{ poss } (m_1, C, n_c, m_2), C \text{ poss } F} \text{ (P2)} \\
\frac{C \text{ poss } (m_1, C, n_c, m_2, F), C \text{ poss } n_f}{C \text{ poss } (m_1, C, n_c, m_2, F, n_f), C \text{ poss } m_3} \text{ (P2)} \\
\frac{C \text{ poss } (m_1, C, n_c, m_2, F, n_f), C \text{ poss } m_3}{C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3), C \text{ poss } \{s\}_{+k_f}} \text{ (P2)} \\
\frac{C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}), C \text{ poss } m_4}{C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4), C \text{ poss } h} \text{ (P2)} \\
\frac{C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, h)}{C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, h)} \text{ (P2)}
\end{array}$$

Applying **F1** and **S8** we obtain

$$\frac{C \text{ believes fresh } (n_c)}{N \text{ believes fresh } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, h)} \text{ (F1)}$$

Applying **T6** we obtain

$$\frac{C \text{ istold } *(\{H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}), m_4, \langle h \rangle\}_{-k_f}, C \text{ poss } +k_f)}{C \text{ istold } *H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)} \text{ (T6)}$$

Applying **I3** and **S5** we obtain

$$\frac{C \text{ istold } *H(m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle), C \text{ poss } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, h),}{C \text{ believes } C \xrightarrow{h} F, C \text{ believes fresh } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)} \text{ (I3)} \\
\frac{C \text{ believes } C \xrightarrow{h} F, C \text{ believes fresh } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)}{C \text{ believes } F \text{ said } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)} \text{ (I3)}$$

Applying **I7** we obtain

$$\frac{C \text{ believes } F \text{ said } (m_1, C, n_c, m_2, F, n_f, m_3, \{s\}_{+k_f}, m_4, \langle h \rangle)}{C \text{ believes } F \text{ said } (\{s\}_{+k_f}, n_c, n_f)} \text{ (I7)}$$

Applying **S3** and **F1** we obtain

$$\frac{C \text{ believes fresh } (n_c)}{C \text{ believes fresh } (\{s\}_{+k_f}, n_c, n_f)} \text{ (F1)}$$

Applying **I6** we obtain

$$\frac{C \text{ believes } F \text{ said } (\{s\}_{+k_f}, n_c, n_f), C \text{ believes fresh } (\{s\}_{+k_f}, n_c, n_f)}{C \text{ believes } F \text{ poss } (\{s\}_{+k_f}, n_c, n_f)} \text{ (I6)}$$

Applying Rationality Rules and **P3**, because

$$\frac{F \text{ poss } (\{s\}_{+kf}, n_c, n_f)}{F \text{ poss } \{s\}_{+kf}, F \text{ poss } n_c, F \text{ poss } n_f} \text{ (P3)}$$

is a postulate. Additionally, C believes $F \text{ poss } (\{s\}_{+kf}, n_c, n_f)$.

So we can obtain C believes $F \text{ poss } \{s\}_{+kf}$, C believes $F \text{ poss } n_c$ and C believes $F \text{ poss } n_f$.

Applying Rationality Rules and **P8**, because

$$\frac{F \text{ poss } -kf, F \text{ poss } \{s\}_{+kf}}{F \{ \{s\}_{+kf} \} -kf}$$

That is $F \text{ poss } s$. Additionally, C believe $F \text{ poss } -kf$.

So we can obtain C believes $F \text{ poss } \langle s \rangle$.

Applying **P2** we obtain

$$\frac{\frac{F \text{ poss } \langle s \rangle, F \text{ poss } n_c}{F \text{ poss } (\langle s \rangle, n_c), F \text{ poss } n_f} \text{ (P2)}}{F \text{ poss } (\langle s \rangle, n_c, n_f)} \text{ (P2)}$$

is a postulate. Additionally, C believes $F \text{ poss } (\langle s \rangle)$, C believes $F \text{ poss } n_c$ and C believes $F \text{ poss } n_f$.

So we can obtain C believes $F \text{ poss } (\langle s \rangle, n_c, n_f)$.

Applying Rationality Rules and **P4**, because

$$\frac{F \text{ poss } (\langle s \rangle, n_c, n_f)}{F \text{ poss } H (\langle s \rangle, n_c, n_f)} \text{ (P4)}$$

is a postulate. Additionally, C believes $F \text{ poss } (\langle s \rangle, n_c, n_f)$.

So we can obtain C believes $F \text{ poss } H (\langle s \rangle, n_c, n_f)$.

Thus goal **(d)** is proved.

(e) C believes fresh ($H (\langle s \rangle, n_c, n_f)$)

Applying **F1** we obtain

$$\frac{C \text{ believes fresh } (n_c)}{C \text{ believes fresh } (\langle s \rangle, n_c, n_f)} \text{ (F1)}$$

Applying **P2** and **S1** we obtain

$$\frac{\frac{C \text{ poss } (<s>), C \text{ poss } (n_f)}{C \text{ poss } (<s>, n_c), C \text{ poss } (n_f)} \text{ (P2)}}{C \text{ poss } (<s>, n_c, n_f)} \text{ (P2)}$$

Then, applying **F10**

$$\frac{C \text{ believes fresh } (<s>, n_c, n_f), C \text{ poss } (<s>, n_c, n_f)}{C \text{ believes fresh } (H (<s>, n_c, n_f))} \text{ (F10)}$$

Thus goal (e) is proved.

(f) C believes reco (H (<s>, n_c, n_f))

Applying **R1** and **S8** we obtain

$$\frac{C \text{ believes reco } (n_c)}{C \text{ believes reco } (<s>, n_c, n_f)} \text{ (R1)}$$

Applying **R5** we obtain

$$\frac{C \text{ believes reco } (<s>, n_c, n_f) \ C \text{ poss } (<s>, n_c, n_f)}{C \text{ believes reco } (H (<s>, n_c, n_f))} \text{ (R5)}$$

Thus goal (f) is proved.

5.5.6 GNY Analysis of Protocol IV

After running Protocol III, the controller and function station obtain a new session key.

Denote $H (<s>, n_c, n_f)$ by K in protocol IV.

1) protocol paraphrase

The Protocol IV consists of following two messages between SDF controller and network station. Denote $H (<K>, 2)$ by k_{cf} , $H (<K>, 4)$ by k'_{cf} , Denote $H (<K>, 1)$ by k_{fc} , $H (<K>, 3)$ by k'_{fc} . In order to simplify the analysis process, we re-write it as follows:

3. $C \rightarrow F: \{m5\}_{k_{cf}}, H (<k'_{cf}>, \{m5\}_{k_{cf}}, Cc);$

4. $N \rightarrow C: \{m6\}_{k_{fc}}, H (<k'_{fc}>, \{m6\}_{k_{fc}}).$

2) protocol description

The parser algorithm would produce the following description of the protocol.

M1. $N \text{ istold } *(\{m5\}_{k_{cf}}, H (<k'_{cf}>, \{m5\}_{k_{cf}}), Cc);$

M2. $C \text{ istold } *(\{m6\}_{k_{fc}}, H (<k'_{fc}>, \{m6\}_{k_{fc}}).$

3) goal

In protocol IV, the network station and SDF controller will use the session key generated in Protocol III to create four new keys. Then the network station and SDF controller will use the four keys to transfer OpenFunction messages securely. Hence, we need to prove the following goals:

- (a) $F \text{ poss } k_{cf}, F \text{ poss } k'_{cf}, F \text{ poss } k_{fc}, F \text{ poss } k'_{fc}, F \text{ poss } m_5;$
- (b) $F \text{ believes } C \text{ poss } k_{cf}, F \text{ believes } C \text{ poss } k'_{cf}, F \text{ believes } C \text{ poss } k_{fc}, F \text{ believes } C \text{ poss } k'_{fc};$
- (c) $F \text{ believes fresh } (k_{cf}), F \text{ believes fresh } (k'_{cf}), F \text{ believes fresh } (k_{fc}), F \text{ believes fresh } (k'_{fc});$
- (d) $F \text{ believes } C \text{ said } m_5;$
- (e) $C \text{ poss } k_{cf}, C \text{ poss } k'_{cf}, C \text{ poss } k_{fc}, C \text{ poss } k'_{fc}, C \text{ poss } m_6;$
- (f) $C \text{ believes } F \text{ poss } k_{cf}, C \text{ believes } F \text{ poss } k'_{cf}, C \text{ believes } F \text{ poss } k_{fc}, C \text{ believes } F \text{ poss } k'_{fc};$
- (g) $C \text{ believes fresh } (k_{cf}), C \text{ believes fresh } (k'_{cf}), C \text{ believes fresh } (k_{fc}), C \text{ believes fresh } (k'_{fc});$
- (h) $C \text{ believes } F \text{ said } m_6.$

4) protocol initial assumption

We use **S1...Si** to represent initial assumption in proof. We assure that the following holds at the beginning of every run of the protocol:

- S1.** $F \text{ poss } K;$
- S2.** $F \text{ poss } m_6;$
- S3.** $F \text{ believes reco } (K);$
- S4.** $F \text{ believes reco } (m_5);$
- S5.** $F \text{ believes fresh } (K);$
- S6.** $F \text{ believes } N \xleftrightarrow{k_{cf}} C;$
- S7.** $F \text{ believes } F \xleftrightarrow{k'_{cf}} C;$

S8. $C \text{ poss } K$;

S9. $C \text{ poss } m5$;

S10. $C \text{ believes reco } (K)$;

S11. $C \text{ believes reco } (m6)$;

S12. $C \text{ believes fresh } (K)$;

S13. $C \text{ believes } F \stackrel{k_{fc}}{\leftrightarrow} C$;

S14. $C \text{ believes } F \stackrel{k'_{fc}}{\leftrightarrow} C$.

5) protocol proof

(a) $F \text{ poss } k_{cf}, F \text{ poss } k'_{cf}, F \text{ poss } k_{fc}, F \text{ poss } k'_{fc}, F \text{ poss } m5$

As $k_{cf} = H(<K>, 2)$ and $F \text{ poss } 2$, applying **P2** and **S1** we can obtain

$$\frac{F \text{ poss } <K>, N \text{ poss } 2}{F \text{ poss } (<K>, 2)} \text{ (P2)}$$

Applying **P4** we can obtain

$$\frac{F \text{ poss } (<K>, 1)}{F \text{ poss } H(<K>, 1)} \text{ (P4)}$$

Thus, $F \text{ poss } k_{cf}$ is proved.

Similarly,

$$\frac{F \text{ poss } (<K>, 3)}{F \text{ poss } H(<K>, 3)} \text{ (P4)}$$

Thus, $F \text{ poss } k'_{cf}$ is proved.

$$\frac{F \text{ poss } (<K>, 2)}{F \text{ poss } H(<K>, 2)} \text{ (P4)}$$

Thus, $F \text{ poss } k_{fc}$ is proved.

$$\frac{F \text{ poss } (<K>, 4)}{F \text{ poss } H(<K>, 4)} \text{ (P4)}$$

Thus, $F \text{ poss } k'_{fc}$ is proved.

we can prove $F \text{ poss } k'_{cf}, F \text{ poss } k_{fc}$ and $F \text{ poss } k'_{fc}$.

Applying **M1**, **T1** and **T2** we obtain

$$\frac{\frac{F \text{ istold } *(\{ms\}_{kcf}, H(<k'_{cf}>, \{ms\}_{kcf}, CC)}{F \text{ istold } (\{ms\}_{kcf}, H(<k'_{cf}>, \{ms\}_{kcf}))} \text{ (T1)}}{F \text{ istold } \{ms\}_{kcf}} \text{ (T2)}$$

Applying **P1** we obtain

$$\frac{F \text{ istold } \{ms\}_{kcf}}{F \text{ poss } \{ms\}_{kcf}} \text{ (P1)}$$

Applying **P6** we obtain

$$\frac{F \text{ poss } k_{cf}, F \text{ poss } \{ms\}_{kcf}}{F \text{ poss } \{\{ms\}_{kcf}\}_{k_{cf}}^{-I}} \text{ (P6)}$$

Hence, we obtain ***F poss ms***.

Therefore, goal **(a)** is proved.

(b) *N believes C poss k_{cf}, N believes C poss k'_{cf}, N believes C poss k_{fc}, N believes C poss k'_{fc}*

Applying Rationality Rules and **P4**, because

$$\frac{C \text{ poss } (<K>, 2)}{C \text{ poss } H(<K>, 2)} \text{ (P4)}$$

is a postulate. And $k_{cf} = H(<K>, 2)$. Additionally, according to Protocol III we obtain *F believes C poss K*.

So we can obtain *F believes C poss k_{cf}*.

Similarly, we can prove *F believes C poss k'_{cf}, F believes C poss k_{fc}, F believes C poss k'_{fc}*.

Therefore, goal **(b)** is proved.

(c) *F believes fresh (k_{cf}), F believes fresh (k'_{cf}), F believes fresh (k_{fc}), F believes fresh (k'_{fc})*

Applying **S5** and **F1** we obtain

$$\frac{F \text{ believes fresh } (<K>)}{F \text{ believes fresh } (<K>, 2)} \text{ (F1)}$$

Applying **F10** we obtain

$$\frac{F \text{ believes fresh } (<K>, 2), F \text{ poss } (<K>, 2)}{F \text{ believes fresh } (H (<K>, 2))} \text{ (F10)}$$

Because $k_{cf} = H (<K>, 2)$, thus, **F believes fresh (k_{cf})** is proved.

Similarly, we can prove **F believes fresh (k'_{cf})**, **F believes fresh (k_{fc})** and **F believes fresh (k'_{fc})**.

Therefore, goal (c) is proved.

(d) F believes C said m_5

Applying **M1** and **T2** we obtain

$$\frac{F \text{ istold } *(\{m_5\}_{k_{cf}}, H (<k'_{cf}>, \{m_5\}_{k_{cf}}, Cc)}{N \text{ istold } * \{m_5\}_{k_{cf}}} \text{ (T2)}$$

Applying **F1** we obtain

$$\frac{F \text{ believes fresh } (k_{cf})}{F \text{ believes fresh } (m_5, k_{cf})} \text{ (F1)}$$

Applying **I1**, **S4**, **S6** and **M1** we obtain

$$\frac{\begin{array}{l} F \text{ istold } * \{m_5\}_{k_{cf}}, F \text{ poss } k_{cf}, F \text{ believes } F \stackrel{k_{cf}}{\leftrightarrow} C \\ F \text{ believes reco } (m_5), F \text{ believes fresh } (m_5, k_{cf}) \end{array}}{F \text{ believes } C \text{ said } m_5, F \text{ believes } C \text{ said } \{m_5\}_{k_{cf}}, F \text{ believes } C \text{ poss } k_{cf}} \text{ (I1)}$$

Therefore, goal (d) is proved.

(e) C poss k_{cf} , C poss k'_{fc} , C poss k_{fc} , C poss k'_{fc} , C poss m_6

As $k_{cf} = H (<K>, 2)$ and $C \text{ poss } 2$, applying **P2** and **S8** we can obtain

$$\frac{C \text{ poss } <K>, C \text{ poss } 2}{C \text{ poss } (<K>, 2)} \text{ (P2)}$$

Applying **P4** we can obtain

$$\frac{C \text{ poss } (<K>, 2)}{C \text{ poss } H (<K>, 2)} \text{ (P4)}$$

Thus, **C poss k_{cf}** is proved.

Similarly, we can prove **C poss k'_{cf}** , **C poss k_{fc}** and **C poss k'_{fc}** .

Applying **M2**, **T1** and **T2** we obtain

$$\frac{\frac{C \text{ istold } *(\{m_6\}_{k_{fc}}, H(<k'_{fc}>, \{m_6\}_{k_{fc}}))}{C \text{ istold } (\{m_6\}_{k_{fc}}, H(<k'_{fc}>, \{m_6\}_{k_{fc}}))} \text{ (T1)}}{C \text{ istold } \{m_6\}_{k_{fc}}} \text{ (T2)}$$

Applying **P1** we obtain

$$\frac{C \text{ istold } \{m_6\}_{k_{fc}}}{C \text{ poss } \{m_6\}_{k_{fc}}} \text{ (P1)}$$

Applying **P6** we obtain

$$\frac{C \text{ poss } k_{fc}, C \text{ poss } \{m_6\}_{k_{fc}}}{C \text{ poss } \{\{m_6\}_{k_{fc}}\}_{k_{fc}}^{-1}} \text{ (P6)}$$

Hence, we obtain **C poss m_6** .

Therefore, goal **(e)** is proved.

(f) C believes F poss k_{cf} , C believes F poss k'_{cf} , C believes F poss k_{fc} , C believes F poss k'_{fc}

Applying Rationality Rules and **P4**, because

$$\frac{F \text{ poss } (<K>, 2)}{F \text{ poss } H(<K>, 2)} \text{ (P4)}$$

is a postulate. And $k_{cn} = H(<K>, 2)$. Additionally, according to Protocol I we obtain **C believes F poss K**.

So we can obtain **C believes F poss k_{cf}** .

Similarly, we can prove **C believes F poss k'_{cf} , C believes F poss k_{fc} , C believes F poss k'_{fc}**

Therefore, goal **(f)** is proved.

(g) C believes fresh (k_{cf}), C believes fresh (k'_{cf}), C believes fresh (k_{fc}), C believes fresh (k'_{fc})

Applying **S12** and **F1** we obtain

$$\frac{C \text{ believes fresh } (<K>)}{C \text{ believes fresh } (<K>, 2)} \text{ (F1)}$$

Applying **F10** we obtain

$$\frac{C \text{ believes fresh } (<K>, 2), C \text{ poss } (<K>, 2)}{C \text{ believes fresh } (H (<K>, 2))} \text{ (F10)}$$

Because $k_{cf} = H (<K>, 2)$, thus, **C believes fresh** (k_{cf}) is proved.

Similarly, we can prove **C believes fresh** (k'_{cf}), **C believes fresh** (k_{fc}) and **C believes fresh** (k'_{fc}).

Therefore, goal (g) is proved.

(h) C believes F said m_6

Applying **M2** and **T2** we obtain

$$\frac{C \text{ istold } * (\{m_6\}_{k_{fc}}, H (<k'_{fc}>, \{m_6\}_{k_{fc}})}{C \text{ istold } * \{m_6\}_{k_{fc}}} \text{ (T2)}$$

Applying **F1** we obtain

$$\frac{C \text{ believes fresh } (k_{fc})}{C \text{ believes fresh } (m_6, k_{fc})} \text{ (F1)}$$

Applying **I1**, **S11**, **S13** and **M2** we obtain

$$\frac{C \text{ istold } * \{m_6\}_{k_{fc}}, C \text{ poss } k_{fc}, C \text{ believes } F \stackrel{k_{fc}}{\leftrightarrow} C}{\frac{C \text{ believes reco } (m_6), C \text{ believes fresh } (m_6, k_{fc})}{C \text{ believes } F \text{ said } m_6, C \text{ believes } F \text{ said } \{m_6\}_{k_{fc}}}, C \text{ believes } F \text{ poss } k_{fc}} \text{ (I1)}$$

Therefore, goal (h) is proved.

5.6 Formal Verification

In this section, we use Casper/FDR to verify the Protocol I and Protocol III. However, we have not formally verified the other two protocols, because the analysis is quite straightforward.

5.6.1 Formal Checking of Protocol I

In order to validate the authenticated OpenFlow handshake protocol, we use model checking method to verify its authenticity. We re-write it as follows.

1. This step corresponds to Step 1 in the theoretical Protocol I.

$$N \rightarrow C: N, nn, mI$$

where $m1$ corresponds to $msg1$ and nn corresponds to nonce R_N of Protocol I.

2. This step corresponds to Step 2-3 in the theoretical Protocol I.

$$C \rightarrow N: C, nc, m23, \text{hash}(mk, N, nn, m1, C, nc, m23)$$

where $m1$ corresponds to $msg1$, and $m23$ corresponds to $msg2||msg3$ of Protocol I.

3. This step corresponds to Step 4 in the theoretical Protocol I.

$$N \rightarrow C: m4, \text{hash}(mk, N, nn, m1, C, nc, m23, m4)$$

where $m1$ corresponds to $msg1$, and $m23$ and $m4$ correspond to $msg2||msg3$ and $msg4$ in Protocol I respectively.

The verification result is shown in Figure 8. No attack was found.

```

19 #Protocol description
20
21 0. -> N : C
22 [N != C]
23 1. N -> C : N, nn, m1
24 2. C -> N : C, nc, m23, hash(mk, N, nn, m1, C, nc, m23)
25 3. N -> C : m4, hash(mk, N, nn, m1, C, nc, m23, m4)
26
27
28 #Specification
29
30 Agreement(C, N, [nc, m23])
31
32 Agreement(N, C, [nn, m1, m4])
33

```

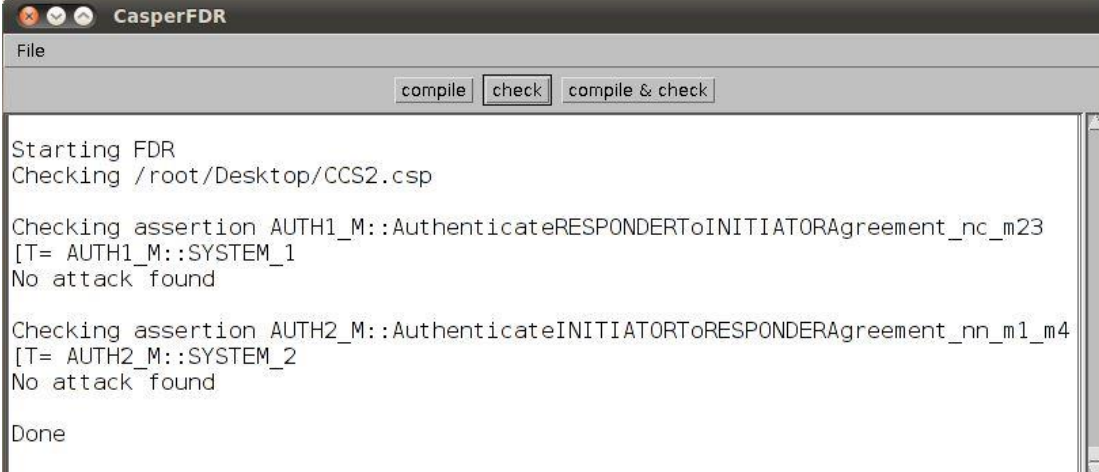


Figure 8 Result of Model Checking for Protocol I

5.6.2 Formal Checking of Protocol III

In order to validate the authenticated OpenFunction handshake protocol, we use model checking method to verify its authenticity. We re-write it as follows.

1. This step corresponds to Step 1 in the theoretical Protocol III.

$$C \rightarrow F: C, NCHC$$

where C corresponds to identity of controller and $NCHC$ corresponds to the nonce R_C || $OFCT_HELLO$ of Protocol III.

2. This step corresponds to Step u2 in the theoretical Protocol III.

$$F \rightarrow C: F, NFHF$$

where F corresponds to identity of the function station and $NFHF$ corresponds to the nonce R_F || $OFCT_HELLO$ of Protocol III.

3. This step corresponds to Step 3 in the theoretical Protocol III.

$$C \rightarrow F: RC, \{PMK\} \{PK(F)\}, \{C, NCHC, F, NFHF, RC\} \{SK(C)\}$$

where RC , $\{PMK\} \{PK(F)\}$ and $\{C, NCHC, F, NFHF, RC\} \{SK(C)\}$ correspond to $OFCT_FEATURES_REQUEST$, $\{PMK\}_{PK_F}$, and σ_C in Protocol III respectively.

4. This step corresponds to Step 4 in the theoretical Protocol III.

$$F \rightarrow C: RF, \{C, NCHC, F, NFHF, RC\} \{PMK\}$$

where RF and $\{C, NCHC, F, NFHF, RC\} \{PMK\}$ correspond to $OFCT_FEATURES_REPLY$ and σ_F in Protocol III respectively.

The verification result is shown in Figure 9. No attack was found.

#Protocol description

```
0. -> C : F
[F != C]
1. C -> F : C, NCHC
2. F -> C : F, NFHF
3. C -> F : RC, {PMK}{PK(F)}, {C, NCHC, F, NFHF, RC}{SK(C)}
4. F -> C : RF, {C, NCHC, F, NFHF, RC, RF}{PMK}
```

#Specification

Agreement(C, F, [PMK])

Secret(C, PMK, [F])

Agreement(C, F, [NCHC, RC])

Agreement(F, C, [NFHF, RF])



Figure 9 Result of Model Checking for Protocol III

Chapter

6 Implementation & Performance

In this chapter, we will introduce a demo system of SAINT first and analyze the performance of these new proposed protocols theoretically. In addition, we also use a set of experiments to verify the feasibility and real performance.

6.1 System Setup

To demonstrate the practicality of our SAINT framework, we have implemented a demo system. Our current prototype primarily consists of four components: SDN/SDF controller, router, switch, network/function station and, IoT end devices. Besides, we also could have an option to choose a remote cloud server or mobile controller. Table 4 summarized the features of this demo system.

- Hardware information

Table 4 Features of Prototype Hardware

Communication parties	devices	Details	
Transmission	Switch	TP-link Switch	
Transmission	Router	Feixun Router	
SDN/SDF controller	Laptop	CPU	2.67 GHz (i5 M560)
		Memory	4 G
		Hard Disk	500 G
Network/Function Station	Raspberry Pi	CPU	1.2GMHz ARM v8
		Memory	1G (LPDDR2)
		Hard Disk	NOOBS Micro SD 16G
IoT end device	Arduino Nano	Microcontroller	16 MHz, 8 bit (ATmega328)
		SRAM	2 KB
		EEPROM	1 KB
		Flash Memory	32 KB (bootloader 0.5 K)

- 1) SDN/SDF controller is the headquarters of the system, which is used to manage and maintain the IoT devices and network directly. In addition, mobile controller and cloud controller can send control commands via the network/function station as to manage the data flow and function in IoT remotely.
- 2) Raspberry Pi is designed as a network/function device because of its more powerful capacity. Its responsibilities are managing data flow and function of corresponding IoT devices.
- 3) IoT end devices such as smart light, monitor, smart socket, PM2.5 smoke sensor and etc. with intelligent chip, which is able to receive and response messages from controller, and execute instructions automatically, can also be regarded as smart devices. In our experiments, we use Arduino Nano to simulate IoT end devices due to its programmability, expandability, inexpensiveness and low-power. As a result, the computational and communicational capacity is restricted.
- 4) Switch/router transfer and forward messages to the destination. It works as a bridge between the controller and station.

The test environment is shown in Figure 10 below.

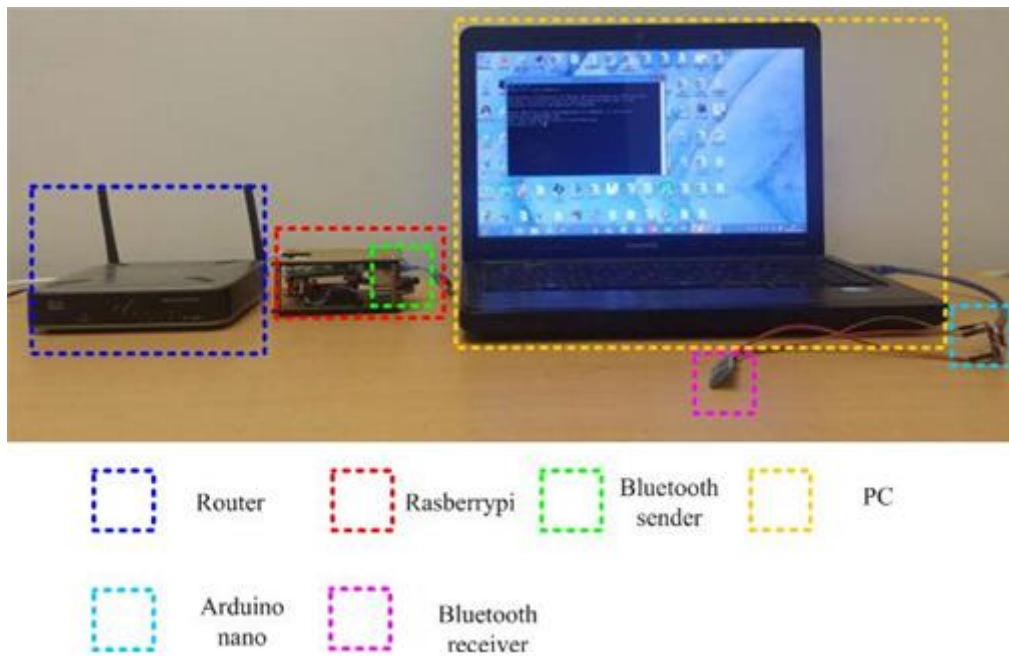


Figure 10 SAINT Demo System

- Software environment

In this experiment, we utilize the open-source controller – POX. We edit POX as our SDN/SDF controller to support IoT, running on Windows 7 professional 32-bit system. It is launched by CML with Python Version 2.7. The OpenFlow version is 1.0. A set of smart devices can make up different network topologies. By design, these smart terminals should be cheap and low-energy. Taken together, Arduino Nano, being an open source hardware, might be an appropriate selection. Arduino SDK is to develop and debug code. The router could embed a firewall to resist some normal attacks. And the IoT smart devices and network/function station together with the controller are in the same local network segment. The asymmetric algorithm in our experiment is RSA, and the symmetric encryption algorithm is AES, and the HMAC function is SHA. Finally, we use Avrdude, a program to burn a hexadecimal coding into a firmware, to reprogram IoT devices.

6.2 Theoretical Performance Evaluation

In order to observe the performance of these protocols, we evaluate computation and communication cost theoretically and compare them with plain OpenFlow protocol without security mechanism. The symbols used in performance analysis are explained in Table 5 below.

Table 5 Symbol in Performance Analysis

Symbol	Meaning
H	HMAC algorithm
msg	message designation
AE	AES encryption algorithm
AD	AES decryption algorithm
RE	RSA encryption algorithm
RD	RSA decryption algorithm
S	RSA signature generation
V	RSA signature verification

The analytical results are shown in Table 6, Table 7, Table 8 and Table 9. The length of payloads is listed. Also, the number of cryptographic computations is listed respectively.

Table 6 Protocol I Evaluation

Parameters	Protocol I	Original OpenFlow
$msg1$	24 bytes	8 bytes
$msg2$	24 bytes	8 bytes
$msg3$	40 bytes	8 bytes
$msg4$	64 bytes	32 bytes
Computation cost in N	$3H$	0 bytes
Computation cost in C	$3H$	0 bytes

Table 7 Protocol II Evaluation

Parameters	Protocol II	Original OpenFlow
msg	40 bytes	8 bytes
Computation cost in N	$3H + AE$	0 bytes
Computation cost in C	$3H + AD$	0 bytes

Table 8 Protocol III Evaluation

Parameters	Protocol III	Original OpenFlow
$msg1$	24 bytes	8 bytes
$msg2$	24 bytes	8 bytes
$msg3$	40 bytes	8 bytes
$msg4$	64 bytes	32 bytes
Computation cost in F	$H + RE + RD + S + V$	0 bytes
Computation cost in C	$H + RE + RD + S + V$	0 bytes

Table 9 Protocol IV Evaluation

Parameters	Protocol II	Original OpenFlow
msg	40 bytes	8 bytes
Computation cost in F	$3H + AE$	0 bytes
Computation cost in C	$3H + AD$	0 bytes

From these tables, we can see that our proposed protocols do not increase the number of communications. Furthermore, the additional burden caused by our protocols is not heavy.

6.3 Implementation of Protocol I & II

Experiment 1 and 2 are used to test newly designed OpenFlow. These first two protocols are implemented using AES (Mode: CBC) and SHA-256.

Experiment 1: Authenticated OpenFlow Association Protocol. This experiment establishes a secure OpenFlow channel between the SDN controller and the network station. Necessary messages are exchanged and a fresh session key is generated to protect the OpenFlow channel. A successful running result of Protocol I is shown in Figure 11.

```

Q:\poxccs>python pox.py openflow.of_01_p1 --address=192.168.1.102 py
POX 0.2.0 <carp> / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.2.0 <carp> is up.
Ready.
POX> OFPT_HELLO
OFPT_FEATURES_REQUEST
MAC1
1dde093b30b454f98d71679da16e9c72c2016aac11150cdcdc5d166dc4a2e448
MAC2
4820560393160a37cbdec79df82047b35cc036623022ccda68e6fdcb141e8965

```

(a) Experiment result of Protocol I on SDN controller

```

pi@raspberrypi:~/xvswitch $ python xvsp1.py
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH
WARNING: No route found for IPv6 destination :: (no default route?)
begin connect
OFPT_HELLO
OFPT_FEATURES_REPLY
MAC1
1dde093b30b454f98d71679da16e9c72c2016aac11150cdc5d166dc4a2e448
MAC2
4820560393160a37cbdec79df82047b35cc036623022ccda68e6fdbcb141e8965

```

(b) Experiment result of Protocol I on network station

Figure 11 Implementation results of Protocol I

Experiment 2: Secure OpenFlow Message Issuing Protocol. This experiment realizes the echo message transmission between the SDN controller and the network station. Protocol II is applied to protect the echo message. The SDN controller encrypts echo request and generates the MAC for the ciphertext, and then sends the ciphertext with the MAC to the network station; the network station verifies the MAC and decrypts the ciphertext. A successful running result of Protocol II is shown in Figure 12. From Figure 12(b) we can see the network station successfully decrypted the ciphertext and got the content of the echo request ("12080001" in the figure).

```

OFPT_HELLO
OFPT_FEATURES_REQUEST
MAC1
39e7dab255682f94386f7b34e2b8ed04bd6690ac183058306d9212fe2413cd1b
MAC2
9872c9adbfe37e1d67205283d29fcd0b4490e299714beb0c0d53eba2e477aa65
OFPT_ECHO
The message is:
12080001
ECHO MAC
655f223a959f11eb5ec68e396c568525717607165cae9195e1d4c69e4c7d45f7

```

(a) Experiment result of Protocol II on SDN controller

```

pi@raspberrypi:~/xvswitch $ python xvsp2.py
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH
WARNING: No route found for IPv6 destination :: (no default route?)
server ip at: 192.168.1.102
begin connect
OFPT_HELLO
OFPT_FEATURES_REPLY
MAC1
39e7dab255682f94386f7b34e2b8ed04bd6690ac183058306d9212fe2413cd1b
MAC2
9872c9adbfe37e1d67205283d29fcd0b4490e299714beb0c0d53eba2e477aa65
OFPT_ECHO
ECHO MAC
655f223a959f11eb5ec68e396c568525717607165cae9195e1d4c69e4c7d45f7
Result of decryption:
12080001

```

(b) Experiment result of Protocol II on network station

Figure 12 Implementation results of Protocol II

We run Protocol I and Protocol II for 10 times. Results of the experiments are illustrated in Table 10. From Table 10 we can see the average run time of our Protocol I and Protocol II is acceptable.

Table 10 Average Runtime of Protocol I and Protocol II

Protocols	Average time	Number of rounds
Protocol I	79.6 ms	10
Protocol II	92.57 ms	10

6.4 Implementation of Protocol III & IV

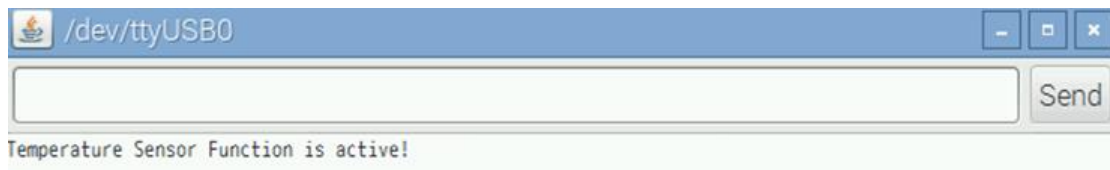
To test performance and feasibility of the newly designed OpenFunction, we perform experiment 3. The public key algorithm in this experiment is RSA (512-bit key), and the symmetric encryption algorithm is AES (Mode: CBC) and the HMAC function is SHA-256.

Experiment 3: Authenticated OpenFunction Association Protocol and Secure OpenFunction Message Issuing Protocol. A successful running result of complete experiment 3 is shown in Figure 13. The results of Protocol I are marked with a red rectangle in 13(c) and 13(d). From the figure, we can see that new session key

calculated by controller and function station respectively was equal to each other. Following Protocol III, the purpose of Protocol IV is about to reprogram the IoT device via a function station. The running result of Protocol 2 is marked with a yellow rectangle in 13(c) and 13(d).

The function station has stored beforehand a series of functions like a function warehouse. According to the reprogramming command received from the authenticated controller, it will renew its specified subordinate smart device with specific function through Avrdude, a program used to burn a hexadecimal coding into a firmware. The IoT device is a preprogrammed originally as a temperature sensor which can be seen from 13(a). After experiment, its function is rewritten to as a smoke sensor. Result of this experiment is shown in 13(b) and highlighted with blue rectangle in 13(d). From the figures, we can find that the function has been changed.

Additionally, we measure the average time of running Protocol III and Protocol IV. We gauge the accurate execution time of each experiment with help of a timer that is embedded in the code. We run each protocol for ten times. When running on Protocol II, the bulk of the overhead time comes from the program update time that is primarily determined by the size of code. As a result, we only measure the running time just before the rewriting. The average runtime for Protocol III and IV is shown in Table 11. The test results indicate that it does not require high communication and computation overhead, and the efficiency is acceptable.



(a) IoT Function before experiment



(b) IoT Function after experiment

Table 11 Average Runtime of Protocol III and Protocol IV

Protocols	Average time	Number of rounds
Protocol III	35.01 ms	10
Protocol IV	31.33 ms	10

In addition, we compared the secure OpenFunction and standard OpenFlow. The results in Table 12 indicate that the proposed protocols do not require high communication and computation overhead, and the efficiency is acceptable.

Table 12 Comparison Between OpenFunction and OpenFlow

Parameters	OpenFunction	OpenFlow
Handshake stage time	0.15297s	0.0339s
Message transmission time	0.01236s	0.0339s
Voltage range	4.97 ~ 5.02 V	0.00295s
Electric current range	0.27 ~ 0.41 A	4.96 ~ 4.99 V
Controller footprint	32,658 bytes	30,054 bytes
Function station footprint	7,579 bytes	4,925 bytes

We have also tested and compared the updating speed with Arduino Nano through serial line versus BlueTooth 2.0 (BT 2.0) communication modules. The results in Figure 14 are provided with the refresh time concerning two different hex file size (10,879 bytes and 14,492 bytes). From Figure 14, we can see that the serial line method has the highest efficiency, it is, however, less convenient than wireless plan. Furthermore, the reprogramming time extended with the increase of the code size and the length of the distance between the function station and the IoT devices.

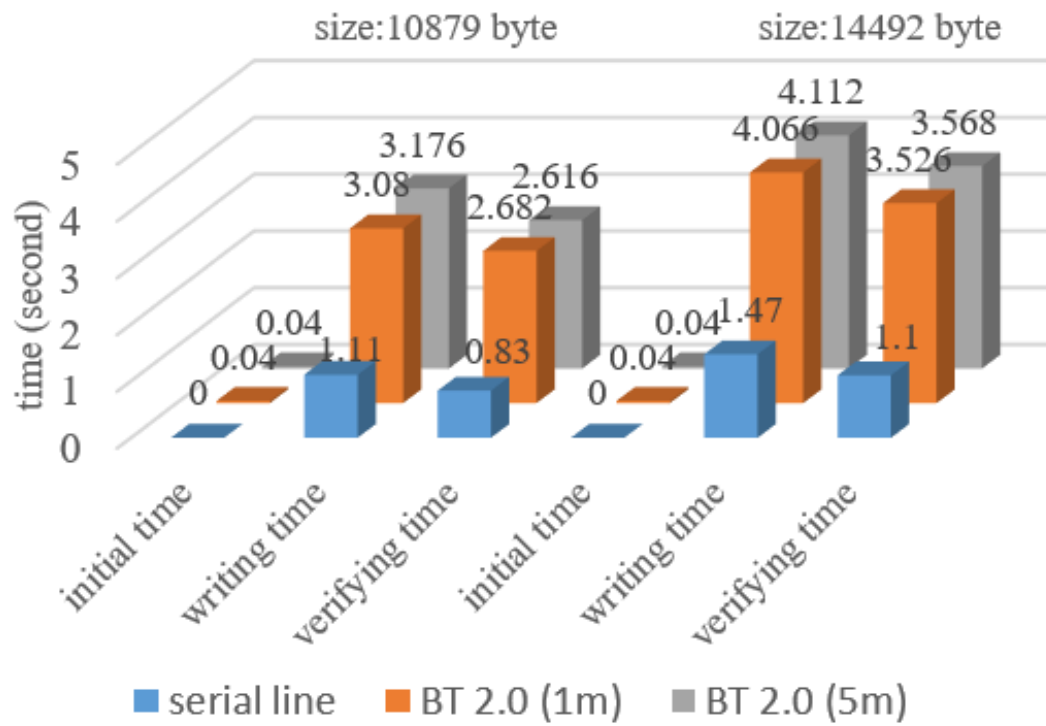


Figure 14 Updating time comparison among different connection methods

Chapter

7 Conclusion

7.1 Summary

Today, with high-speed progress of information technology and globalization, a giant tide of IoT is irresistibly spreading all over the world. After decades of development, IBs, as a representative use case of IoT, have become to gradually replace the traditional buildings, which is obviously necessary and inevitable. Nonetheless, the pivot of the IB, i.e. the management of IoT, has run into the bottleneck.

In this thesis, we focus on the design of a pragmatic, flexible and secure framework for dynamic requirements in IoT. Specifically, we have proposed SAINT that decouples the control layer from the underlying physical infrastructure layer, allowing the controller to customize and reprogram the data flow rules in IoT and functions embedded in remote smart devices easily, also depending on the basic design of the smart devices and their capabilities. In addition, we have extended standard OpenFlow and redesigned OpenFunction protocol for the IoT, aiming to extend the original OpenFlow to support IoT devices with light-weight security mechanism. The security of the four protocols is analyzed and validated. Finally, we have presented a prototypical implementation of SAINT, and evaluated its performance. Achieved results demonstrate that the SAINT framework was feasible, flexible, secure and efficient for the IoT. And IB network can benefit security after deploying SAINT with acceptable performance loss.

7.2 Limitations

Here, we merely discuss SAINT framework between control layer and physical infrastructure layer because of a few conditional constraints. This project still has several limitations.

- In this dissertation, we only focus on the feasibility and security. Nevertheless, the security between the function/network station and IoT end devices should also be

considered. In practice, The IoT devices only can be reprogramed by the function station that has the privilege. And the data flow should be transferred to the legal network station. Security is important for IoT too.

- The use case of SAINT has not been studied. In this thesis, we only simulate a handshake and message issuing process. How to apply this demo system to practice should be seriously considered.
- We do not test the performance of our protocols in different types of networks. By contrast, we only test a single IoT device within a simple network structure. However, in reality, the network is much more complex and various.

7.3 Future Work

The state-of-the-art technologies of IoT and SDN have highly commercial and academic value for the next-generation network. Fusing IoT and SDN technologies will fundamentally change the future network infrastructure, IT industry, as well as numerous high-tech markets, such as in industrial automation, smart energy, smart car, smart home and smart city. However, there are still tremendous challenges on the way, for instance, dynamics, diversities, security, scalability and etc.

Some experts even estimate that the IoT will grow extraordinarily to cover all the things in our environment, stepping into a new era called the Internet of Everything (IoE) [54]; perhaps one of the crucial keys to open this door is SDN. As a result, a term – Software-Defined Everything (SDE) is proposed, toward promoting a pivotal role for software system in controlling divergent types of hardware systems. With SDE, the computing infrastructure can be virtualized and delivered as a service. Under a Software-Defined Everything environment, control and management of the function, networking, storage, data and etc. are automated by the intelligent software rather than by the hardware parts of the infrastructure. It allows software to control a great range of IoT devices, surely raising new and significant changes around the world in the future.

As for future works, firstly we plan to perform a set of experiments in varieties of network conditions for our SAINT framework. Then we intend to further investigate and extend this framework. To construct a systematic, comprehensive and secure Software Defined Internet of Things architecture is our future work's destination.

References

- [1] I. Silva, R. Leandro, D. Macedo, and L. A. Guedes, "A dependability evaluation tool for the Internet of Things," *Computers & Electrical Engineering*, vol. 39(7), pp. 2005-2018, 2013.
- [2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol.29(7), pp. 1645-1660, 2012.
- [3] Y. Jararweh, M. Al-Ayyoub, A. Darabseh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDIoT: a software defined based internet of things framework," *Journal of Ambient Intelligence and Humanized Computing*, vol.6(4), pp. 453-461, 2015.
- [4] Y. Jian, Z. Yu, and S. Chen, "Design of Intelligent Power Management System Based on Internet of Things," *Applied Mechanics & Materials*, vol. 340(340), pp. 979-983, 2013.
- [5] H. Ning, and Z. Wang, "Future Internet of Things Architecture: Like Mankind Neural System or Social Organization Framework?," *IEEE Communications Letters*, vol. 15(4), pp. 461-463, 2011.
- [6] J. Liu, Y. Li, M. Chen, and W. Dong, "Software-defined internet of things for smart urban sensing," *Communications Magazine IEEE*, vol.53(9), pp.55-63, 2015.
- [7] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, pp. 1-11, 2011.
- [8] J. Carlini, "The Intelligent Building Definition Handbook," *IBI*, Washington, DC, 1988.
- [9] J. K. W. Wong, H. Li, and S. Wang, "Intelligent building research: a review," *Automation in construction*, vol. 14(1), pp. 143-159, 2005.
- [10] L. C. Fu, and T. J. Shih, "Holonc supervisory control and data acquisition kernel for 21st century intelligent building system," *IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2641-2646, 2000.

- [11] Z. Sun, "Intelligent building and communication network," *Shanxi Architecture*, 2001.
- [12] D. M. Han, and J. H. Lim, "Design and implementation of smart home energy management systems based on zigbee," *IEEE Transactions on Consumer Electronics*, vol. 56(3), pp. 1417-1425, 2010.
- [13] X. Huang, P. Craig, H. Lin, and Z. Yan, "Seciot: a security framework for the internet of things," *Security & Communication Networks*, vol. 9, 2015.
- [14] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "Spins: Security protocols for sensor networks," *Wireless Networks*, vol. 8(5), pp. 521-534, 2002.
- [15] R. Xu, X. Huang, J. Zhang, Y. Lu, G. Wu, and Z. Yan, "Software Defined Intelligent Building," *International Journal of Information Security and Privacy (IJISP)*, vol. 9(3), pp. 84-99, 2015.
- [16] A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and Thierry Turetletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16(3), pp. 1617–1634, 2014.
- [17] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig, "Software defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103(1), pp. 10–13, 2014.
- [18] D. Dratsky, E. Keller, and J. Rexford, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17(2), pp. 20–27, 2013.
- [19] C. Huth, P. Duplys, and T. Güneysu, "Secure software update and IP protection for untrusted devices in the Internet of Things via physically unclonable functions," *IEEE International Conference on Pervasive Computing and Communication Workshops Workshops*, pp. 1-6, 2016.
- [20] B. Xu, L. Xu, H. Cai, C. Xie, J. Hu, and F. Bu, "Ubiquitous data accessing method in IoT-based information system for emergency medical services," *IEEE Transactions on Industrial Informatics*, vol. 10(2), pp. 1578-1586, 2014.

- [21] W. Zhao, C. Wang, and Y. Nakahira , "Medical application on Internet of Things," *Iet International Conference on Communication Technology and Application*, pp. 660-665, 2011.
- [22] S. D. T. Kelly, N. K. Suryadevara and S. C. Mukhopadhyay, "Towards the Implementation of IoT for Environmental Condition Monitoring in Homes," *IEEE Sensors Journal*, vol. 13(13), pp.3846-3853, 2013.
- [23] J. Li, E. Altman, and C. Touati, "A General SDN-Based IoT Framework with NVF Implementation," *ZTE Communications*, vol. (3), pp. 42-45, 2015.
- [24] N. Xue, L. Liang, X. Huang, and J. Zhang. "POSTER: A Framework for IoT Reprogramming," *12th EAI International Conference on Security and Privacy in Communication Networks (SecureComm 2016)*, 2016.
- [25] Open Networking Foundation, "Software-Defined Networking: The new norm for networks," <http://www.opennetworking.org>, 2012.
- [26] T. Huang, S. Yan, F. Yang, T. Pan, and J. Liu, "Building SDN-Based Agricultural Vehicular Sensor Networks Based on Extended Open vSwitch," *Sensors*, vol. 16(1), pp. 108, 2016.
- [27] P. Pereñi, M. Kuzniar, and D. Kostic, "OpenFlow needs you! a call for a discussion about a cleaner OpenFlow API," *European Workshop on Software Defined Networks*, IEEE Computer Society, pp. 44-49, 2013.
- [28] C.A.R Hoare, "Communicating Sequential Processes," Prentice-Hall, vol. 21, pp. 666-677, 1985.
- [29] P. Y. A. Ryan, S. A. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe, "Modelling and analysis of security protocols," Addison-Wesley Longman, 2001.
- [30] G. Lowe, "Casper: a compiler for the analysis of security protocols," *Journal of Computer Security*, vol. 6(1), pp.53-84, 2000.
- [31] A. W. Roscoe, "Modelling and verifying key-exchange protocols using CSP and FDR," *Computer Security Foundations Workshop, 1995. Proceedings. Eighth IEEE*, pp. 98-107, 1995.

- [32] R. Sathya, and R. Thangarajan, "Efficient anomaly detection and mitigation in software defined networking environment," *International Conference on Electronics and Communication Systems*, pp.391-3, 2015.
- [33] S. Dotcenko, A. Vladyko, and I. Letenko, "A fuzzy logic-based information security management for software-defined networks," *International Conference on Advanced Communication Technology*, pp. 167 – 171, 2014.
- [34] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," *Acm Sigcomm Computer Communication Review*, vol. 38(2), pp. 69-74, 2008.
- [35] J. Katz, "Introduction to modern cryptography," Chapman & Hall/CRC, 2000.
- [36] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: a survey," *Future Networks and Services*, pp. 1-7, 2013.
- [37] R. Braga, E. Mota, and A. Passito, "Lightweight ddos flooding attack detection using nox/openflow," *Jouranal of Nuclear Cardiology*, vol. 8(1Suppl1), pp. 408–415, 2010.
- [38] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," *The Workshop on Hot Topics in Software Defined Networks*, pp. 127–132, 2012.
- [39] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," *IEEE International Conference on Network Protocols*, pp. 1-6, 2012.
- [40] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular composable security services for softwaredefined networks," *Network & Distributed Security Symposium*, 2013.
- [41] O. Flauzac, C. Gonzalez, A. Hachani, and F. Nolot, "Sdn based architecture for iot and improvement of the security," *IEEE International Conference on Advanced Information NETWORKING and Applications Workshops*, pp. 688–693, 2015.
- [42] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," *ACM SIGCOMM Workshop on Hot Topics in Software Defined NETWORKING*, pp. 171–172, 2013.

- [43] F. Klaedtke, G. O. Karame, R. Bifulco, and H. Cui, "Access control for SDN controllers," *Ecosystems*, vol. 16(7), pp. 1325–1335, 2014.
- [44] C. Banse, and S. Rangarajan, "A secure northbound interface for sdn applications," *The IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 834-839, 2015.
- [45] OpenFlow Switch Consortium et al. "Openflow switch specification version 1.4.0," 2015.
- [46] D. Samociuk. "Secure communication between openflow switches and controllers," *The Seventh International Conference on Advances in Future Internet (AFIN 2015)*, pp. 32-37, 2015.
- [47] H. Wang, "Authentic and Confidential Policy Distribution in software defined wireless network," *Wireless Communications and Mobile Computing Conference (IWCMC 2014)*, pp. 1167-1171, 2014.
- [48] N. Xue, X. Huang and J. Zhang, "S²Net: A Security Framework for Software Defined Intelligent Building Networks," *The IEEE International Conference on Trust, Security and Privacy in Computing and Communications (Trustcom 2016)*, Tianjin, China, 2016.
- [49] N. Xue, J. Zhang and X. Huang, "POSTER: A Framework for IoT Reprogramming," *Eai International Conference on Security and Privacy in Communication Networks (Securecomm 2016)*, Guangzhou, China, 2016.
- [50] Z. Liu, H. Seo, Z Hu, and X. Hunag, "Efficient Implementation of ECDH Key Exchange for MSP430-Based Wireless Sensor Networks," *ACM ASIACCS*, pp.145-153, 2015.
- [51] A. El-Mougy, M. Ibnkahla, and L. Hegazy, "Software-defined wireless network architectures for the Internet-of-Things," *Local Computer Networks Conference Workshops*, pp. 804-811, 2015.
- [52] Z. Qin, G. Denker, C. Giannelli, and P. Bellavista, "A Software Defined Networking architecture for the Internet-of-Things," *Network Operations and Management Symposium*, pp. 1-9, 2014.

- [53] U Carnegie Mellon University, "Software defined Internet of Things (SD-IoT)," <http://www.cmu.edu/silicon-valley/research/techshowcase/pdfs/2014/showcase-paper41-poster.pdf>, 2014.
- [54] S. V. Vandebroek, "1.2 Three pillars enabling the Internet of Everything: Smart everyday objects, information-centric networks, and automated real-time insights," *IEEE International Solid-State Circuits Conference*, 2016.
- [55] M. Abadi, M. Burrows, and R. Needham, "A Logic of Authentication, " *Proceedings of the Royal Society A Mathematical Physical & Engineering Sciences*, vol. 426(1871), pp. 1-13, 1989.
- [56] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols," *1990 IEEE Computer Society Symposium on IEEE*, pp. 234-248, 1990.
- [57] E. Saul, and A. C. M. Hutchison, "A Graphical Environment for the Facilitation of Logic-Based Security Protocol Analysis," *South African Computer Journal*, vol. 26, pp. 196-200, 2000.
- [58] D. M. Nessett, "A critique of the Burrows, Abadi and Needham logic," *Acm Sigops Operating Systems Review*, vol. 24(2), pp. 35-38, 1990.
- [59] E. Sneekenes, "Exploring the BAN approach to protocol analysis," *1991 IEEE Computer Society Symposium on IEEE*, pp. 171-181, 1991.
- [60] A. D. Gante, M. Aslan, and A. Matrawy, "Smart wireless sensor network management based on software-defined-networking," *Communications (QBSC), 2014 27th Biennial Symposium on*, pp. 71-75, 2014.
- [61] H. Huang, J. Zhu, and L. Zhang, "An sdn based management framework for iot devices," *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET.*, pp. 175-179, 2013.
- [62] Á. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. Garcia Villalba, "SDN: Evolution and Opportunities in the Development IoT Applications," *International Journal of Distributed Sensor Networks*, vol. 2014, pp. 1-10, 2014.

- [63] D. Dolev, and A. Yao, "On the Security of Public Key Protocols," *IEEE Transactions on Information Theory*, vol. 29(2), pp. 198-208, 1983.
- [64] W. Bo, Y. Zhang, X. Hong, H. Sun, and X. Huang, "Usable Security Mechanisms in Smart Building," *IEEE, International Conference on Computational Science and Engineering. IEEE*, pp. 748-753, 2014.
- [65] X. Huang, W. Bo, Y. Zhang, and N. Gong, "I-lock: A phone-based access control system," *Computatinal Conference on Computing and Technology Inovation (CTI2015)*, 2015.
- [66] J. Li, Y. Zhang, and F. Kuang, "Intelligent Building Automation and Control Based on IndasIBMS," pp. 266-270, 2013.
- [67] L. Zhang, B. Liu, Q. Tang, and L. Wu, "The development and technological research of intelligent electrical building," *China International Conference on Electricity Distribution. IEEE*, pp. 88-92, 2014.
- [68] N. Xue, L. Liang, J. Zhang, X. Huang, "An Access Control System for Intelligent Buildings," *Eai International Conference on Mobile Multimedia Communications ICST (Mobimedia 2016)*, pp.11-17, 2016.