IoTivity Packet Parser for Encrypted Messages in Internet of Things

Hyeonah Jung, Hyungjoon Koo, and Jaehoon (Paul) Jeong
Department of Computer Science and Engineering
Sungkyunkwan University, Suwon, Republic of Korea
Email: {hyeonah214, kevin.koo, pauljeong}@skku.edu

***Abstract*—The Internet of Things (IoT) market has been ever-growing because both the demand of smart lives and the number of mobile users keep increasing. On the other hand, IoT device manufacturers tend to employ proprietary operating systems and network protocols, which may lead device interoperability issues. The Open Connectivity Foundation (OCF) has established a standard protocol for seamless IoT communication. IoTivity is one of reference implementations that conforms to the OCF specification. IoTivity utilizes both Datagram Transport Layer Security (DTLS) and Constrained Application Protocol (CoAP) to support a lightweight and secure communication. Although a packet analysis tool like Wireshark offers a feature to decrypt messages over TLS or DTLS by feeding a session key that a Web browser records, it cannot be directly applied to IoTivity because it lacks such a key-tracing functionality. In this paper, we present an IoTivity Packet Parser (IPP) for encrypted CoAP messages tailored to IoTivity. To this end, we modify IoTivity source code to extract required keys, and leverage them to parse each field automatically for further protocol analysis in a handy manner.**

***Index Terms*—IoTivity, IoT, DTLS, CoAP, Packet Parser**

I. Introduction

The Internet of Things (IoT) represents a collection of physical entities that can be connected over the communication networks, which are capable of carrying out varying tasks with embedded sensors and software under limited computation power. IoT devices offer a wide spectrum of services including remote control, real time monitoring, and automatic data collection. With ever-increasing mobile users and the popularity of smart life (e.g., smart phones, smart home appliances, smart speakers), the IoT market has been surged accordingly. GlobeNewswire [1] foresees that the number of IoT devices reaches up to 25 billion by 2025. However, IoT device makers employ proprietary operating systems and network protocols, which often lack interoperability between heterogeneous devices.

In response to addressing the interoperability issue, the Open Connectivity Foundation (OCF) [2], a de facto standards developing organization (SDO) that consists of a group of IoT industry leaders, has enacted a standard protocol to communicate with each other on the Internet. OCF provides a unifying specification of a core framework (i.e., architectures, features, resources, protocols) that enables a seamless device-to-device connectivity without worrying out incompatibility problems for both IoT businesses and end users. IoTivity [3] is one of OCF reference implementations based on the OCF specification [1]. Its design supports a large number of software and hardware for the IoT ecosystem, ranging from an authentication and permission configuration to an access control service for the owner of a device.

The IoTivity protocol leverages both Datagram Transport Layer Security (DTLS) [4], [5] and Constrained Application Protocol (CoAP) [6] standards to provide a lightweight and secure interaction (i.e., request and response) model over UDP (User Datagram Protocol) between application endpoints. Akin to Transport Layer Security (TLS), DTLS affords an analogous cryptographic service on top of a stateless UDP protocol such as key generation, encryption, and decryption. CoAP protocol defines functionalities that a node (i.e., IoT device) can communicate with others in a constrained environment efficiently. In a nutshell, once a user owns an IoT device for the first time, the owner registers the device through an on-boarding process (i.e., ownership transfer) with CoAP messages, followed by processing a proper configuration (i.e., credential provisioning and access control) for further interactions. It is noteworthy mentioning that all CoAP messages are encrypted over DTLS after owner transfer is complete.

CoAP messages follow a relatively simple structure and well-defined [7], thus they can be parsed by a protocol analyzer such as Wireshark [8] in a handy manner. As it is often required to analyze encrypted CoAP messages on the fly, a protocol analysis tool offers a functionality to parse encrypted messages with TLS or DTLS once a key is provided. Obtaining a session key [9] is trivial with the feature that a Web browser offers, which keeps track of a set of appropriate keys during a key negotiation process. Specifically, this can be done by setting up a certain environment variable (i.e., `SSLKEYLOGFILE`) that pinpoints a file to store the keys so that they can be tracked during SSL/TLS sessions. However, this is not a viable option for the IoTivity protocol in general because one cannot obtain a key with the absence of that feature.

In this paper, we introduce an IoTivity packet parser that can deal with encrypted CoAP messages. To this end, we modify IoTivity source code [10] to be able to drop necessary keys for decryption, obeying the original format that can be fed into a packet analyzer (e.g., WireShark). Finally, we implement an

[1]The IoTitity project has started since 2015, supported by Samsung and Linux Foundation.

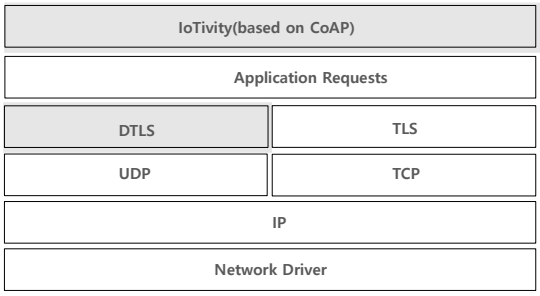

Fig. 1. Networking stack for IoTivity. IoTivity leverages CoAP and DTLS for mutual communications across heterogeneous devices.

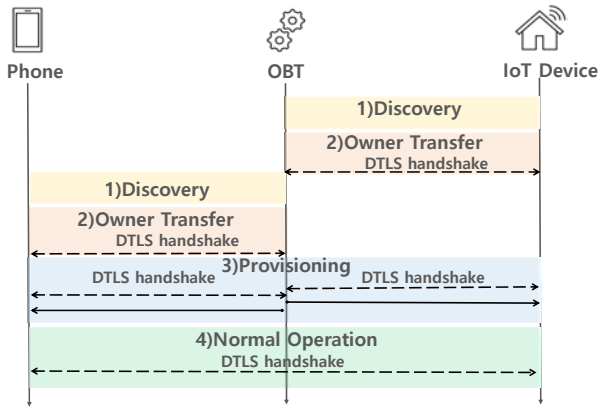

Fig. 2. Simplified steps for on-boarding (1,2), provisioning (3) and ordinary communications (4) afterwards. OBT offers three options for authentication: `JustWorks`, `Random PIN`, and `Certificate`.

automated CoAP parser dubbed IPP with the provided key that parses each field of a CoAP message on demand for further protocol analysis.

II. Background

This section describes a handful of concepts to better understand the behavior of the IoTivity protocol. Figure 1 briefly illustrates networking stack for various protocols.

*a) **Datagram TLS (DTLS)**:* Similar to TLS, the DTLS protocol [4] offers confidentiality, integrity and authentication (against packet eavesdropping and message tampering) over UDP with its simple design. IoTivity adopts UDP in the underlying network stack for performance, hence it takes advantage of DTLS for encryption, decryption, and integrity of messages. The main difference between TLS and DTLS is that DTLS implements the mechanism of packet retransmission by assigning sequence numbers within a handshaking phase, which resolves two major problems that are out of UDP's concerns, packet loss, and reordering. This design allows DTLS to provide the identical security level with TLS. For example, DTLS handshaking establishes a secure session channel dynamically via an automatic key negotiation (i.e., `ChangeCipherSpec` phase) between the client and the server, followed by sending and receiving encrypted messages for application data.

*b) **Constrained Application Protocol (CoAP)**:* CoAP [6] is an application protocol that has been specialized for devices under resource-hungry (i.e., literally constrained) environments such as wireless sensor networks. It allows each device (i.e., node) to speak to another irrespective of its manufacturer or a built-in communication mechanism within. Considering the nature of IoT devices, the design of CoAP requires simplicity [2] for imposing a quite low overhead over UDP. In particular, it adopts a simple design that can be easily converted to HTTP using a REST (Representational State Transfer)-based (i.e., RESTful) protocol. It handles asynchronous communications of events that occur between nodes in the transport and application layers of TCP and UDP. Besides, it offers DTLS binding for a secure communication. In this regard, IoTivity largely utilizes both DTLS and CoAP for both efficiency and security.

*c) **IoTivity Protocol**:* IoTivity [3] is one of the representative implementations that obeys the OCF standard [2]. It aims to build a de facto standard system that encompasses every feature (e.g., device discovery, device management, and data transmission) so that billions of IoT devices can communicate with each other seamlessly in a backward compatibile and interoperable manner. An IoTivity device begins with enrolling itself to an owner. On-boarding is a process of discovering and registering the IoT device by the owner, where its status would be changed to "owned" from "unowned". An on-boarding tool (OBT) is a helper for this registration process as a logical entity (Figure 4). To exemplify, OBT can be a built-in tool in a mobile phone for connecting (i.e., pairing) a device over Wi-Fi or Bluetooth. Figure 2 shows 1) discovery phase and 2) owner transfer phase. OBT assists a mutual authentication between a device and an owner with three different methods: `JustWorks`, `Random PIN`, and `Certificate`. First, `JustWorks` omits an authentication procedure, which may be vulnerable to an attack (e.g., connecting to an attacker's device). IoTivity recommends this option only when a device can be authenticated physically. Second, `Random PIN` confirms that OBT and a device have the same PIN by generating a random number, which is often transferred to the device by entering the created PIN from OBT. Third, `Certificate` employs a signed certificate (by a manufacturer) for authenticating the device. The device does not authenticate OBT due to the lack of a resource constraint. Once owner transfer with OBT is complete, OBT establishes a DTLS session with a pre-shared key (PSK) that has been generated during the owner transfer. Provisioning phase involves with security information such as credentials

[2]The mandatory fields of a CoAP message [7] are solely four bytes, including a version, type, token length, method code, and two bytes of a message identifier. Other fields are optional if needed.

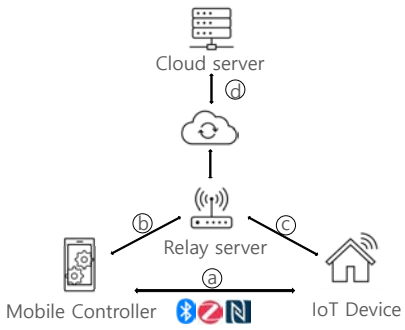

Fig. 3. The overall IoT architecture for communication between an IoT device, mobile controller, relay server, and cloud server. As an initial step, the IoT device can be registered with the controller via either the relay server (ⓑ and ⓒ) or a direct communication over short-range networks (ⓐ), followed by further communication with the cloud server (ⓒ and ⓓ).

and access control policy [3] (Phase 3 in Figure 2). All further communications are encrypted over DTLS without OBT intervention (Phase 4 in Figure 2).

*d) **Packet Analysis Tool**:* A packet analysis tool allows one to analyze packets that have been either captured on the fly or that are recorded on the disk as a single file. Wireshark [8] is one of the most popular open-source packet analysis tools, which supports hundreds of different network protocols. It automatically parses each field of well-defined protocols including CoAP. Besides, it is feasible to decrypt even encrypted packets if an appropriate session key would be provided to Wireshark [9].

III. DESIGN AND IMPLEMENTATION

In this section, we describe the design and implementation of our packet parser, IPP in detail. Recall that IPP aims to assist the *analysis of encrypted IoTivity packets*. We use an IoTitivty-lite version [10] for our implementation.

*a) **Environmental Setup for IoT Devices**:* Figure 3 depicts the IoT architecture in general, which consists of four components: an IoT device, mobile controller, cloud server, and relay server. The IoT device and its controller (e.g., mobile phone) are under the same broadcast domain for mutual communication (ⓑ and ⓒ) through the relay server like onboarding and provisioning processes (see Section II). It can be done with a direct communication over short-range networks (ⓐ) such as Bluetooth [12], Zigbee [13] and Radio Access Network (RAN) [14]. Once an initial phase (e.g., device registration) is complete, an IoT device is ready for varying operations to the cloud server directly. Note that our experimental setup excludes the cloud server because its setup has been unavailable [15]. Hence, this work targets captured packets at the range of ⓑ and ⓒ for further analysis.

[3]The OCF model supports CRUDN (Create, Read, Update, Delete and Notify) operations for resource management.

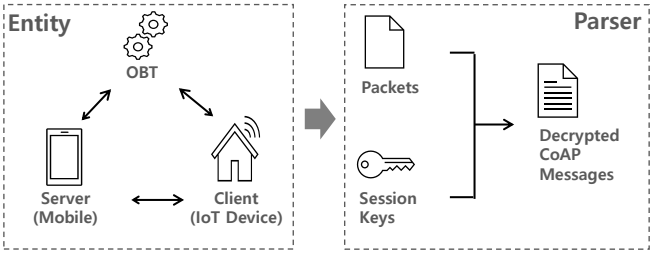

Fig. 4. Overview of IPP. There are three entities to control an IoT device (left): on-boarding tool (OBT), client and server. OBT is a logical unit that is often embedded into the server. IPP collects both packets and session keys from communications between a pair of entities, and generates decrypted CoAP messages (right).

*b) **DTLS Decryption with a Network Security Services (NSS) Format**:* Similar to TLS decryption [9] using an external program like Wireshark [8], session keys must be provided for decryption beforehand. NSS [16] defines a set of libraries to support security-enabled client and server applications in multiple platforms. One of NSS features is capable of logging a key for further decryption of encrypted payloads during TLS/DTLS connections with a specified format [11]. The key logging functionality can be enabled by setting the environment variable, named `SSLKEYLOGFILE`, which pinpoints a file path, allowing a Web browser (e.g., Chrome and Firefox) to record all keys in that file. Each log line follows the format of `<Label><space><ClientRandom><space><Secret>` where a label is summarized in Table I. Note that an RSA key list has been deprecated. Besides, IoTivity does not support RSA for performance.

*c) **IoTivity Source Modification**:* IoTivity does not provide a feature of logging with the aforementioned environment variable, hence, we modified an IoTivity-lite implementation version [10] to be able to emit all random values from a client and master secrets during DTLS handshaking phases in Figure 2. As IoTivity-lite employs TLSv1.2, it is sufficient to bring the `CLIENT_RANDOM` label with a 32-Byte random value from the `Client Hello` message (corresponding to `<ClientRandom>`) and a 48-Byte master secret (corresponding to `<Secret>`).

*d) **Implementation for Parsing Encrypted IoTivity Packets**:* Note that we narrow down the scope of IPP from a generic IoT architecture in Figure 3 due to the unavailability of a cloud environment [15]. Figure 4 depicts an overview of IPP (The current version supports ⓐ in Figure 3). We developed IPP that takes both raw packets (e.g., PCAP format) captured by Wireshark [8] and session keys acquired from the modified IoTivity source as inputs, decrypting and parsing CoAP payloads on demand. We utilize `pyshark` 0.4.3 [17] for Python packet parsing with Wireshark dissectors, and `txThings` [18] for CoAP interpretation. Note that IPP can facilitate further analysis by field-wise parsing from decrypted messages, which the naïve Wireshark cannot offer.

TABLE I
DEFINED LABELS FOR THE NSS LOG FORMAT [11]. THE IOTIVITY-LITE [10] FOR OUR IMPLEMENTATION SUPPORTS TLS V1.2, NECESSITATING THE CLIENT_RANDOM (*) LABEL ALONE. NOTE THAT TLS V1.3 REQUIRES SEVEN DIFFERENT LABELS INCLUDING CLIENT AND SERVER SECRETS.

Label Name	Description	Note
RSA	48 Bytes for the premaster secret, encoded as 96 hexadecimal characters	Removed in NSS 3.34
CLIENT_RANDOM*	48 Bytes for the master secret, encoded as 96 hexadecimal characters	SSL 3.0, TLS 1.0, 1.1 and 1.2
CLIENT_EARLY_TRAFFIC_SECRET	Hex-encoded early traffic secret for the client side	TLS 1.3
CLIENT_HANDSHAKE_TRAFFIC_SECRET	Hex-encoded handshake traffic secret for the client side	TLS 1.3
SERVER_HANDSHAKE_TRAFFIC_SECRET	Hex-encoded handshake traffic secret for the server side	TLS 1.3
CLIENT_TRAFFIC_SECRET_0	First hex-encoded application traffic secret for the client side	TLS 1.3
SERVER_TRAFFIC_SECRET_0	First hex-encoded application traffic secret for the server side	TLS 1.3
EARLY_EXPORTER_SECRET	Hex-encoded early exporter secret	TLS 1.3
EXPORTER_SECRET	Hex-encoded exporter secret	TLS 1.3

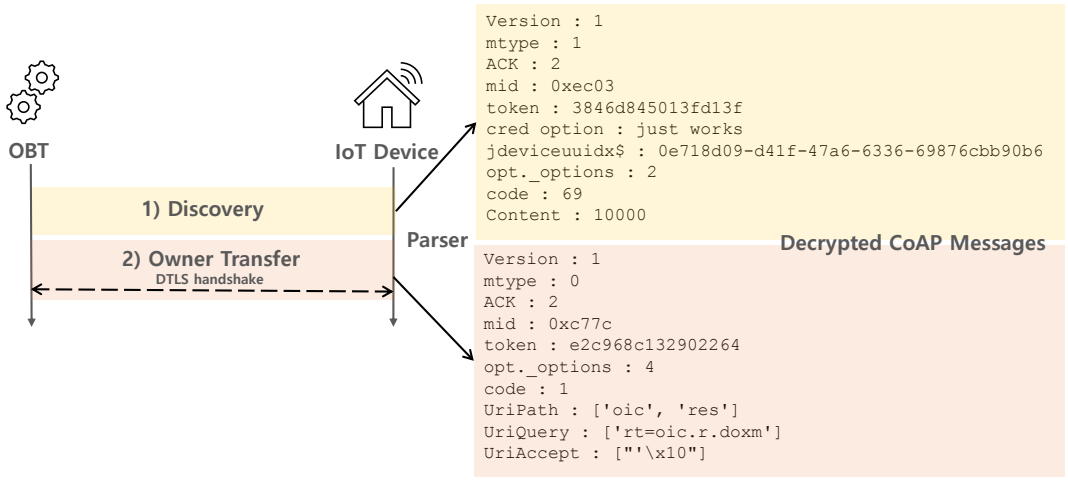

Fig. 5. Example of decrypted CoAP messages with IPP. This example shows important fields that have been successfully parsed during discovery and owner transfer steps.

TABLE II
IOTIVITY AUTHENTICATION OPTIONS. NOTE THAT RANDOM PIN IS THE MOST COMMON AUTHENTICATION OPTION.

Authentication Option	Number
JustWorks	00
Random PIN	01, 10
Certificate	11

*e) **Example**:* Figure 5 illustrates an example of decrypting and parsing messages with IPP. During a discovery phase, we can confirm that it uses the JustWorks option and vaild ID for authentication in Table II, which is determined by a particular value in a payload. In this example, the captured valid ID was 0e718d09-d41f-47a6-6336-69876cb90b6, indicating that the device has no ownership. In case of being owned, the device would not advertise itself during the discovery phase. At the second phase of owner transfer, IPP successfully parses a message type, token, code, and URI information (UriPath, UriQuery, and UriAccept). The URI information implies the delivery status from OBT, client or server. Additionally, we confirmed varying statuses for authentication options, device IDs, ownership transfers, onboarding, and provisioning tasks.

IV. DISCUSSION AND FUTURE WORK

In this section, we discuss several limitations and future work. First, as of writing, the IoTivity project does not support a cloud environment. We have contacted one of IoTivity developers and received an official answer that the cloud for testing is currently unavailable [15], and the version under development has been tested merely in a local environment. Likewise, we had to test IPP under the localhost. With the cloud setting, every normal operation (i.e., incoming and outgoing packets) after discovery and provisioning in Figure 2 should be through the cloud. Testing our parser with the cloud is part of our future work. Second, the current version of IoTivity supports DTLS v1.2. Thus, IPP must be updated if IoTivity employs a higher version of DTLS (v1.3) because the key format of v1.2 significantly differs from that of v1.3 (See Table I). We leave supporting the future version of DTLS as part of our future work. Third, we plan to expand the usage of IPP in finding a vulnerability for the design of the IoTivity protocol (e.g., fuzzing [19]).

V. Conclusion

IoT devices are all around us. One of key features is a seamless communication on the Internet across heterogeneous devices. To achieve this goal, OCF has specified a standard protocol, and IoTivity implements the OCF specification as a reference. For fast and secure communications, IoTivity adopts DTLS and CoAP. With an existing packet analysis tool like Wireshark, it is not possible to directly parse each field of a message from encrypted messages. In this paper, we introduce IPP that can interpret IoTivity packets particularly for encrypted CoAP messages. We demonstrate that IPP can successfully interpret all fields for a CoAP message by extracting session keys with our modified version of IoTivity.

Acknowledgments

This work was supported in part by Institute for Information & Communications Technology Planning & Evaluation (IITP) grant (No. 2019-0-01343, Regional strategic industry convergence security core talent training business) and in part by Information Technology Research Center (ITRC) support program (IITP-2021-2017-0-01633). We would like to thank Songhee Kwon for helping static analysis of IoTivity source code and automatically extracting required keys. Note that Hyungjoon Koo is the corresponding author.

References

[1] GlobeNewswire, "Worldwide Industry for IoT Middleware to 2025 - Manufacturing Expected to Have High Potential Growth," 2020, https://www.globenewswire.com/news-release/2020/12/21/2148872/0/en/Worldwide-Industry-for-IoT-Middleware-to-2025-Manufacturing-Expected-to-Have-High-Potential-Growth.html.

[2] OCF, "Open Connectivity Foundation: a Standard Communications Platform, Bridging Specification, and Open Source Implementation," 2021, https://openconnectivity.org/.

[3] I. Project, "Open Source Software Framework for Internet of Things," 2021, https://iotivity.org/.

[4] N. Modadugu and E. Rescorla, "The design and implementation of datagram tls," in *Proceedings of the 11th Network and Distributed System Security Symposium (NDSS '04)*. Internet Society, 2004.

[5] I. E. T. F. (IETF), "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," 2015, https://www.ietf.org/rfc/inline-errata/rfc7525.html.

[6] ——, "RFC7252: The Constrained Application Protocol (CoAP)," 2014, https://datatracker.ietf.org/doc/html/rfc7252.

[7] Markushx, "A Cheatsheet for the Constrained Application Protocol (CoAP)," 2017, https://github.com/markushx/coap-cheatsheet/blob/master/coap-cheatsheet.pdf.

[8] Wireshark, "Network Protocol Analyzer," 2021, https://wireshark.com/.

[9] A. Phillips, "How to Decrypt SSL with Wireshark – HTTPS Decryption Guide," 2021, https://www.comparitech.com/net-admin/decrypt-ssl-with-wireshark/.

[10] IoTivity-Lite, "Open-source, Reference Implementation of the Open Connectivity Foundation (OCF) Standards for the Internet of Things (IoT)," 2021, https://github.com/iotivity/iotivity-lite.

[11] Mozilla, "NSS Key Log Format," 2021, https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format.

[12] C. Bisdikian, "An overview of the bluetooth wireless technology," *IEEE Communications Magazine*, vol. 39, no. 12, pp. 86–94, 2001.

[13] S. C. Ergen, "ZigBee/IEEE 802.15. 4 Summary." Sep. 2004.

[14] Y. H. J. Wu, Z. Zhang and Y. Wen, "Cloud radio access network (C-RAN): a primer," pp. vol. 29, no. 1, pp. 35–41, Jan.-Feb. 2015.

[15] P. Hub, "Secure and Interoperable Internet of Things," 2021, https://github.com/plgd-dev/hub.

[16] Mozilla, "Network Security Services," 2021, https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[17] KimiNewt, "Python wrapper for tshark," 2021, https://github.com/KimiNewt/pyshark.

[18] M. Wasilak, "txThings: CoAP library for Twisted framework," 2021, https://github.com/mwasilak/txThings.

[19] B. Melo, "Fuzzing for Robustness and Security Testing of CoAP Servers," 2021, https://github.com/bsmelo/fuzzcoap.

Hyeonah Jung is a Ph.D student in the Department of Computer Science and Engineering, College of Computing and Informatics, Sungkyunkwan University, South Korea, since fall 2021. Her Ph.D advisor is Professor Jaehoon (Paul) Jeong. She got a BS degree from Mokpo National Maritime University. Her research interests is Internet of Things (IoT), Cloud Computing and Internet Protocols Standardization.

Hyungjoon Koo is an assistant professor in the Department of Computer Science and Engineering, College of Computing at Sungkyunkwan University (SKKU). Before joining SKKU, he was a post-doctoral researcher at SSLab in Georgia Institute of Technology. He earned his Ph.D. in Computer Science from Stony Brook University in 2019, and received the M.Sc. degree in Information Security from Korea University in 2010. He is leading the SecAI (Security with AI) Lab. His research interests lie in software security, network security, binary analysis and protection, and security leveraging artificial intelligence.

Jaehoon (Paul) Jeong is an associate professor in the Department of Computer Science and Engineering at Sungkyunkwan University in Korea. He received his Ph.D. degree in the Department of Computer Science and Engineering at the University of Minnesota in 2009. He received his B.S. degree in the Department of Information Engineering at Sungkyunkwan University and his M.S. degree from the School of Computer Science and Engineering at Seoul National University in Korea in 1999 and 2001, respectively. His research areas are Internet of Things, Software-Defined Networking, Network Functions Virtualization, security, and vehicular networks. Dr. Jeong is a member of ACM, IEEE and the IEEE Computer Society.