Hindawi
Security and Communication Networks
Volume 2019, Article ID 4869167, 11 pages
https://doi.org/10.1155/2019/4869167

Research Article

SSPSoC: A Secure SDN-Based Protocol over MPSoC

Soultana Ellinidou, Gaurav Sharma, Théo Rigas, Tristan Vanspouwen, Olivier Markowitch, and Jean-Michel Dricot

Cybersecurity Research Center, Université Libre de Bruxelles, Belgium

Correspondence should be addressed to Soultana Ellinidou; soultana.ellinidou@ulb.ac.be

Received 7 July 2018; Revised 14 December 2018; Accepted 6 February 2019; Published 18 March 2019

Guest Editor: Daniel Schneider

In recent years, Multi-Processor System-on-Chips (MPSoCs) are widely deployed in safety-critical embedded systems. The Cloud-of-Chips (CoC) is a scalable MPSoC architecture comprised of a large number of interconnected Integrated Circuits (IC) and Processing Clusters (PC) destined for critical systems. While many researches have focused on addressing the hardware issues of MPSoCs, the communication over them has not been very well explored. Following the SDN concept, we propose a new protocol in order to secure the communication and efficiently manage the routing within the CoC. The SSPSoC includes a private key derivation phase, a group key agreement (GKA) phase, and a data exchange phase in order to ensure that basic security primitives are preserved and provide secure communication. Furthermore, a network of 1-30 nodes is set in order to validate the proposed protocol and measure the network performance and memory consumption of the proposed protocol.

1. Introduction

Since the late 1990s the rise of System-on-Chips (SoCs) has caused a huge technological progress with a dramatic involvement not only on the field of electronics but also on Internet of Things (IoT) and Internet of Everything (IoE) field [1]. IoT and IoE brought into the surface a wide variety of applications, able to satisfy our needs in transportation, health-care, manufacturing, and energy management with diverse requirements, which traditional SoCs are not always able to support. Hence, the creation of a flexible, technology aware SoC design is vital.

A new scalable MPSoC architecture, called Cloud-of-Chips (CoC) (Figure 1), could be a possible solution, which consists of a combination of multiple Integrated Circuits (ICs) and IC building blocks interconnected together with different communication speeds and hierarchy levels. The CoC design contains a Printed Circuit Board (PCB) of multiple ICs where each IC contains scalable Processing Clusters (PCs). Each PC comprises a combination of High Performances (HP) and Low Power (LP) cores and thus enables a heterogeneous system architecture [2]. The on-chip data communication is managed by Network on Chip (NoC) [3], which supports regular interconnected topologies, such as MESH, TORUS, etc. [4].

In order to manage the routing on CoC with multiple ICs, the size of routing tables will be large enough not to be accommodated on ordinary NoC routers. The memory overhead for routing tables will grow by $n^2 * k$ units where n^2 is the number of PCs on each IC and k is the number of IC on each PCB. Therefore, in order to achieve secure communication among PCs and ICs on CoC but also to manage the traffic of the network with an efficient manner, the Software Defined Networking (SDN) technology [5] could be adopted. Hence the design of a secure networking approach of SDN technology and its integration into hardware by moving network intelligence and services from complex and expensive physical switches and routers and placing them into software that runs on the top of the underlying hardware need to be addressed.

The SDN-based networking strategy enables the communication between any two PCs on CoC. There is no existing literature for CoC. However, there is limited literature available in NoC-based MPSoC which includes SDN as a packet routing approach. SDNoC is a NoC communication paradigm rather than a specific design and implementation,

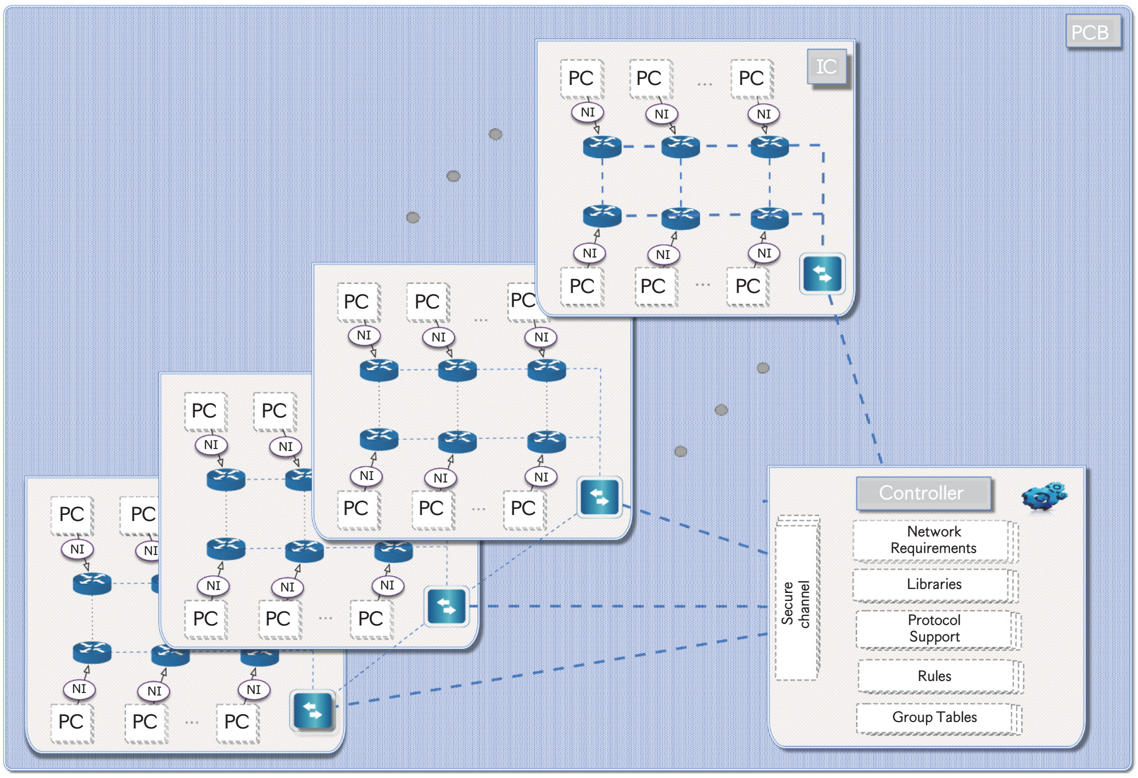


Figure 1: CoC architecture.

presented for first time in 2014 [6]. Specifically Cong et al. propose a SDNoC architecture where the control plane is deployed as a distributed unity at each router; however this is contrary to SDN philosophy because both planes are placed inside the router. Afterwards Sandoval et al. [7, 8] propose an SDNoC architecture which consists of three layers: Operating System, Network Operating System, and Infrastructure. In this research the authors made an assumption that the routers need configuration data by the SDN controller. However flows, that are not managed by the SDN controller, use the XY routing algorithm. Thereafter Scionti et al. [9] use the SDN architecture in order to explore dynamic changes in the network topology; each Processing Element (PE) has specific instructions to control the network topology by software, including switch off the links which are not used. Another interesting approach is proposed by Berestizshevsky et al. [10], in which an SDNoC strategy by implementing a network manager is introduced, acting as controller with global view of the network, which controls the network in adaptive manner. However in order to be deployed on chip, a detailed API implementation and standardization between control plane and data forwarding plane need to be considered.

In CoC, the network is separated into two main abstraction hardware levels: IC and PCB. On the IC level the hardware routers on Network on Chip (NoC) are functional as SDN switches and the routing on individual IC is managed by an IC level controller. For the inter IC communication, a SDN switch is placed at the boundary of each IC and it is connected to a PCB level controller [11] through physical links. The controller consists of a *Secure Channel*, where the private and public keys are stored for the communication between SDN switches, *Rules* where the applicable rules to the switches are included, *Protocol Support* where the supported communication protocols are included, and *Libraries* and *Network Requirements*, where the network interface (NI) informs the controller about the traffic of the network and according to that field the *Rules* is constructed (Figure 1).

There are several SDN-based communication protocols seen in literature but the most used is the OpenFlow (OF). OF establishes a unicast communication channel between each individual switch and the controller. It allows the controller to discover OF-compatible switches, create rules for the switching hardware, and also collects statistics [12]. This protocol has several security flaws that can be exploited to compromise the network. Unfortunately, the SDN paradigm is susceptible to several security breaches [13, 14]. However, in this paper, the main concern is the communication security among SDN switches (on each IC) and controller. In addition to the switch controller unicast communication, multicast communication is also needed to address the issue of secure transfer of routing updates to all/some of the switches [15, 16].

The existing security solution such as the Transport Layer Security (TLS) protocol is not well enforced in the current version of the OF standard [17]. The specification mentioned that "the switch initiates a standard TLS or TCP connection to the controller" which means that using TLS is completely optional. Moreover, the Public Key Infrastructure (PKI) overhead includes generation and signing of digital certificates for switches and controller. This makes PKI based solution less acceptable for and MPSoC architecture. Therefore, we propose a more suitable solution that fits to unique characteristics of the CoC architecture.

In SDN concept, multiple vulnerability analyses have been performed [3, 5, 7–9, 12, 13, 18–22]; however, none of them attempts to extensively analyze the security issues of the SDN architecture. In the latest version of OpenFlow, in order to provide authentication and encryption of the connection between switch and controller, TLS is suggested security mechanism [23]. However, the latest OpenFow Switch specification [17], presented by Open Networking Foundation (ONF), does not provide enough information about security features and the implementation of TLS. The lack of TLS use could lead to fraudulent data insertion and DoS attacks [24]. Under the umbrella of Public Key Cryptography (PKC), TLS protocol requires a Certificate Authority (CA) to generate the CA's key, certificates for the controllers, switches, and then the signing of these certificates with the CA's key. Afterwards the certificates and the keys of the network entities will be deployed to the respective devices.

1.1. Group Key Agreement. The existing literature on SDN security refers to point-to-point communication between switches and controller. However, running different applications on CoC architecture creates multiple routing paths among ICs and therefore, multiple switches interacting with SDN controller quite frequently. Generally, the application based logical subsystems will be created which may involve multiple ICs (with different components such as GPU, crypto processor, etc.). Our idea is to create a secure point-to-multipoint communication between controller and group of switches, i.e., secure multicast. Therefore, two group key agreement (GKA) protocols are chosen for our architecture: the Sharma et al. approach [25] and the Teng et al. approach [26]. The protocols that we chose are balanced and imbalanced group key agreement (GKA) protocols. In balanced GKA protocols, all the participating nodes share the same computation burden while, in imbalanced, the powerful node (in our case, the controller) is majorly responsible for expensive computations. Sharma's protocol comes with the benefit of achieving mutual authentication using signature and of course the assurance that every participant at the end is in possession of the valid group key. However, the Teng protocol does not provide mutual authentication, thus trading security for efficiency.

1.2. Contribution. The need of secure communication between different hierarchy layers on MPSoC architecture in conjunction with the necessity of low power computation, scalability, and flexibility not only between the network nodes but also in the network as a whole leads us to the structure of a new communication protocol. This protocol will provide authenticity to network entities, low power consumption for the system, and a new message stack in order to have a quick secure communication among switches and controller.

The contribution of this paper can be summarized as follows:

(i) The design of new SDN-based protocol, which has three main functionalities: the derivation of keys for every node in our network through private key generator (PKG), the establishment of a secure group of participants, and the secure communication between the participants.

(ii) The validation of the proposed protocol, which is based on the SDN concept (by using switches and controller as network entities).

(iii) The performance analysis of two GKA protocols in order to verify which is more suitable in order to cover the second functionality of the proposed protocol in the view of running time and memory consumption.

The rest of the paper is organized as follows: in Section 2 the security requirements are presented; based on them it follows Section 3, where the group key agreement approach is analyzed. Afterwards, in Section 4, the SSPSoC communication protocol for an SDN-CoC network is described, where the architecture, the packet format, the proposed network messages, and the three different phases of the SSPSoC protocol, Private Key Extraction, Group Formation, and Switch Controller Communication, are presented. It follows the validation and the performance analysis of the proposed protocol in Section 5. Last but not least our research is summarized in the last section.

2. Security Requirements

The CoC architecture contains multiple switches and a single controller to manage the overall communication among ICs. The infrastructure to implement identity based cryptography requires an on-board PKG. The overall communication security on this layer (switch controller) can be investigated from two viewpoints. The first view is to securely deliver the private keys to switches and controller. The second view covers the secure communication among all the switches and the controller. We achieve this security using an authenticated group key agreement protocol.

2.1. Phase 1. The foremost issue to address is to transport the private key and required security parameters to all the switches and the controller. The PKG generates the private key for all switches and the controller and delivers it securely. The literature refers to using a secure channel but they do not specify exactly what this channel could be and its security requirements. The possible threats and solutions are as follows:

(i) In order to ensure that the only legitimate nodes can receive identity ID and private key, node authentication must be performed by the *PKG*.

(ii) A counterfeit PKG with different master key generates private keys and IDs for the nodes. This *PKG* is able to decrypt all the traffic between nodes and controller.

In this case, authentication of the PKG by the nodes is also needed.

(iii) An attacker can eavesdrop on the response of the PKG and steal the private key of a node. A solution must be there to ensure the confidentiality of communication between a node and the PKG.

(iv) An attacker can sniff the packets exchanged between a node and the PKG and replay them later to obtain a private key.

(v) An attacker can manage to compromise the integrity of the packets between node and PKG.

2.2. Phase 2. This phase refers to switch controller group communication where we adopted a GKA protocol to secure it. The common threats are spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privileges. The authenticated GKA protocol provides authentication of all participants. As the session key is derived, rest of the communication is encrypted using AES-GCM with session key. Therefore, confidentiality, authenticity, integrity, and nonrepudiation are maintained. To address denial of service and authorization issues, separate precautions need to be enabled.

3. Group Key Agreement

3.1. Assumptions. Before the design of the protocol some assumptions were vital to be made:

(i) The network consists of multiple nodes, which can be either a controller or SDN switches or the PKG, which derives the private keys to the network entities.

(ii) The private ID-based keys are provided to the participants of the group by PKG, which is the private key generator. Suppose that *msk* is the master secret key of the PKG and $KPub_{PKG}$ is the public key.

(iii) Given that there are n entities in the network, $U_1, U_2, \ldots, U_n$ is a group of participants in a GKA session for establishing a group key. Assuming that U_n will be the controller of every group, there are $n-1$ nodes (SDN switches) in this group.

3.2. GKA Protocols. By taking into account the above assumptions and the security requirements, we present the two GKA protocols that we based on and we constructed our protocol. The two protocols consist of 3 major phases: The *Setup* phase (hash functions, group generators, and pairing), the *Key Extraction* phase (nodes and controller obtain their private keys from PKG), and the *Key Agreement* phase (establishment of a group session key for participants).

The Teng protocol [26]:

Setup

(i) The PKG chooses two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q, a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \longrightarrow \mathbb{G}_2$.

(ii) The PKG selects two random generators P and Q of $\mathbb{G}_1$.

(iii) The PKG selects $s \in \mathbb{Z}_q^*$ as the *msk* and sets $KPub_{PKG} = sP$.

Private Key Extraction. Defining as input parameters, *msk* and $ID_i \in \{0,1\}^*$ with ID_i being the ID of the node,

(i) The PKG computes $KPriv_i$ as $S_i = (q_i + s)^{-1}Q$ where $q_i = H(ID_i)$.

(ii) The PKG communicates secretly $KPriv_i$ to node i.

(iii) The public key of node i is $T_i = q_iP + KPub_{PKG} = (q_i + s)P$.

Key Agreement Round 1. Each participant

(i) U_i selects a random $r_i \in Z_q^*$.

(ii) U_i precomputes $P_i = r_iT_i$.

(iii) U_i $(1 \le i \le n)$ sends P_i to the controller S.

Key Agreement Round 2. Upon receiving P_i from all nodes, each participant

(i) U_i $(1 \le i \le n)$, S chooses random $r \in Z_q^*$.

(ii) U_i computes $Q_i = rP_i$.

(iii) U_i broadcasts Q_i $(1 \le i \le n)$, keeping r secret.

Key Computation. On receiving Q_j $(1 \le i \le n)$,

(i) U_i computes the final session key as

$$\begin{aligned} sk &= e(Q_i, S_i)^{r_i^{-1}} e(Q_1 + Q_2 + \cdots + Q_n, Q) \\ &= e(P, Q)^{r + rr_1(s+q_1) + \cdots + rr_n(s+q_n)}. \end{aligned} \tag{1}$$

Precomputation. The following tuples (r_i, r_i^{-1}, P_i) should be created and stored in the memory storage of the nodes before the execution of the GKA. This essentially reduces the computation cost of the first round for the nodes and also improves the speed of key computation phase.

The Sharma protocol [25]:

Assumption. Let *pid* be the set of the identities of the participants in one session of the protocol and *sid* the session identifier.

Setup

(i) The PKG selects an EC group G of prime order q. Let P be a generator of group G.

(ii) The PKG computes the system's public key as $KPub_{PKG} = sP$ by choosing a master secret $s \in \mathbb{Z}_q^*$.

(iii) The PKG chooses cryptographic hash functions $H_1 : \{0,1\}^* \times G \longrightarrow \mathbb{Z}_q^*$, $H_2 : \{0,1\} \times G \times G \longrightarrow \mathbb{Z}_q^*$ and $H : \{0,1\}^* \longrightarrow \{0,1\}^k$.

Key Extraction. Defying the system parameters $Params = \{G, q, H_1, H_2, H_3, H, KPub_{PKG}\}$ and by keeping the master key secret,

(i) The PKG selects $r_i \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $R_i = r_i P$.

(ii) The PKG computes the private key for the user U_i as $KPriv_i = r_i + sH_1(ID_i, R_i)$.

(iii) Each participant U_i can verify the private key as $KPriv_i P = R_i + H_1(ID_i, R_i) KPub_{PKG}$.

Key Agreement Round 1. Each participant

(i) U_i $(1 \le i \le n)$ chooses $eph_i \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $l_i = H_3(eph_i, KPriv_i)$ and $L_i = l_i P$.

(ii) U_i $(1 \le i \le n)$ selects a random string $k_i \in \{0,1\}^k$. Each user, except U_n computes $H(k_i)$. The user U_n masks the randomness as $\widetilde{k}_n = H(k_n, x_n)$ where x_n is the long-term secret of U_n.

(iii) U_i $(1 \le i \le n)$ computes $H(\widetilde{k}_n)$.

(iv) U_i $(1 \le i \le n)$ broadcasts the tuple $\langle L_i, H(k_i), H(\widetilde{k}_n), R_i \rangle$ to all $n-1$ members.

Key Agreement Round 2. Upon receiving the message $\langle L_j, H(k_j), H(\widetilde{k}_n), R_j \rangle$, each participant

(i) U_i computes $U_{ij} = l_i L_j$ and $L = L_1 \parallel L_2 \parallel .. \parallel L_n$.

(ii) U_i, except U_n, computes $K_{ij} = H(U_{ij}) \oplus k_i$. the user U_n computes $mask = H(U_{ij}) \oplus \widetilde{k}_n$.

(iii) U_i chooses another random number $t_i \in \mathbb{Z}_q^*$ and computes $T_i = t_i l_i P$. Also computes the signature on $\langle L, T_i \rangle$ as $\sigma_i = t_i l_i + KPriv_i H_2(ID_i, L, T_i, pid)$.

(iv) U_i broadcasts $\langle K_{ij}\ (1 \le j \le n, j \ne i), mask, \sigma_i, T_i \rangle$ to all $n-1$ members.

Key Computation. Upon receiving $\langle K_{ji}, mask, \sigma_i, T_i \rangle$, each participant

(i) U_i verifies the received signature as $\sigma_i P = T_i + (R_i + H_1(ID_i, R_i) KPub_{PKG}) H_2(ID_i, L, T_i, pid)$.

(ii) U_i computes $\widetilde{k}_j = H(U_{ji}) \oplus K_{ji}$. (Similarly, $\widetilde{k}_n$ can be computed using mask.)

(iii) Note that $U_{ij} = l_i L_j = l_i l_j P = l_j l_i P = l_j L_i = U_{ji}$.

(iv) U_i checks the k_i as $H(k_j) = H(\widetilde{k}_j)$ for $(1 \le j \le n, j \ne i)$.

(v) U_i computes the session identity $sid = H(k_i) \parallel H(k_2) \parallel \ldots \parallel H(\widetilde{k}_n)$.

(vi) The session key is computed as $sk = H(k_1 \parallel k_2 \parallel \ldots \parallel \widetilde{k}_n \parallel sid \parallel pid)$.

4. Communication Protocol

In this section, the network architecture following the packet format, the network messages that are broadcasted within our network, and the three phases of the proposed protocol are introduced.

4.1. Architecture. The proposed architecture of the full system is presented on Figure 1. However our network architecture consists of 3 main network entities:

(i) A PKG, which is considered as a trusted third party and generates the corresponding private key to the rest of the nodes (switches and controller).

(ii) A centralized controller with a broader network view to manage the routing of the packets within the network.

(iii) Multiple switches which are responsible to route the packets between the PC

The communication between controller and switches is managed through a virtual network running on the top of the CoC.

4.2. Packet Format. The packet format is the core of the protocol stack. Every packet consists of a header structure, which is 32-bit long (Figure 2) [11]. The header message format consists of three main fields. Firstly, the version field indicates the version of communication protocol to which this message belongs. Secondly, the length field indicates where this message will end in the byte stream starting from the first byte of the header. Thirdly, the xid, or transaction identifier, is a unique value used to match requests to responses. The type field which indicates what type of message is present and how to interpret the payload is version dependent and we can see the messages that it is including above. Furthermore every message that travels across the network consists of the same header of 32 bits. However the payload size depends on the length field that is provided through header message and it can vary according to the type of the message. Afterwards it included the source and destination ID following the addressing format that we previously presented. Another field is the type of the packet; for example, it could be opcodes for a given processor, followed by some padding and the data. As far as the messages of the type field, a specific message stack is designed, which is presented afterwards.

4.3. Network Messages. The different types of messages, which were designed and integrated into packet format, are depicted in Table 1. The SSPSoC protocol includes 8 types of messages with different content. These messages are flowing through the links between the network entities. In addition the type value of the messages is used to distinguish GKA protocol messages from other messages that might be circulating on the network and one byte is used to encode the message type.

4.4. SSPSoC Network Initialization

Phase 1 (obtain private key). During the first phase, the switches and the controller communicate with the PKG, in order to obtain their long-term private keys. One of the major issues of concern on this phase was the security level of the communication between the nodes and the *PKG*, by

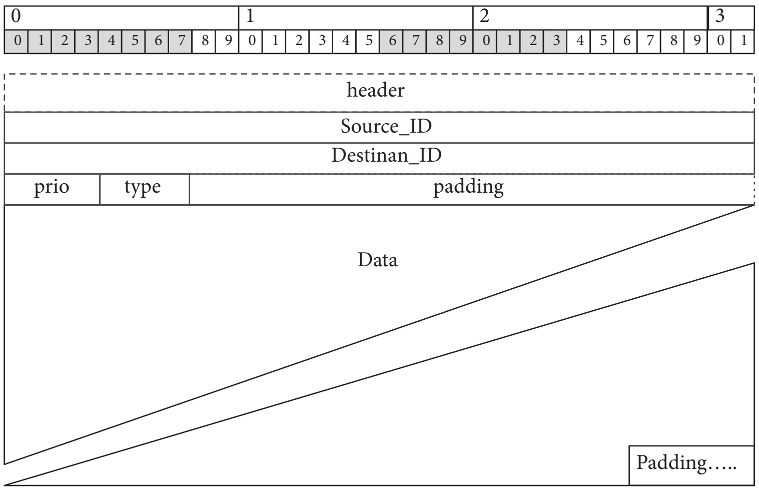


Figure 2: Packet format [11].

Table 1: Designed network messages.

Type	Type Value	Description	Contents
`KEY_REQUEST`	0x06	Sent by nodes to the PKG	Enc(timestamp), IV, tag
`KEY_REPLY`	0x07	Sent by PKG to nodes as a reply to `KEY_REQUEST` message	Enc(System Parameters, node ID, private key), IV, tag
`JOIN`	0x01	Broadcasted by nodes who wants to join a group	node ID, timestamp, JOIN token
`INVITE`	0x02	Broadcasted by controller for inviting nodes to form a group	(participant ID_1, node ID_1, . . ., participant ID_n, $Node\ ID_n$)
`READY`	0x03	Broadcasted by nodes as a reply to `INVITE` message	participant ID, timestamp, READY token
`ROUND_1`	0x04	Contains cryptographic material for the first round of the GKA	sender ID, Crypto R1
`ROUND_2`	0x05	Contains cryptographic material for the second round of the GKA	sender ID, receiver ID, Crypto R2
`DATA`	0x08	Contains encrypted data with the group key	sender ID, receiver ID, Enc(data), IV, tag

tackling the problem of establishing a secure channel for the private key transmission. However, keeping the private key confidential is not the only security consideration; we should also take into account the authentication of the nodes. The authentication will ensure that only legitimate nodes can obtain a private key from the *PKG*. For this reason, the implementation of authenticated `KEY_REQUEST` messages is mainly used.

A node first determines a timestamp (t_s) to prevent the replay attacks [27]. Afterwards it generates the random part of the Initialization Vector (IV) and it encrypts t_s using the Preshared Symmetric Secret Key (*PSK*) and AES in Galois Counter Mode (AES-GCM) [28]. AES-GCM outputs the ciphertext c and the authentication tag: $c, tag = AES_{PSK,IV}(t_s)$. Thereafter the node sends a `KEY_REQUEST` message to the PKG, which contains the IV, the ciphertext, and the authentication tag. It follows a process where the *PKG* decrypts the ciphertext and checks the authentication tag. If the tags are matching, it will check that the decrypted timestamp is within a given threshold. In case the timestamp is valid, it will generate a random node ID and it will extract the associated private key, $KPriv(i)$. Thereupon it generates a random IV and encrypts them using the *PSK* and AES-GCM and sends the IV, ciphertext, and the authentication tag to the node. The steps of the this procedure are depicted on Figure 3.

Phase 2 (form a group). On this phase each switch communicates with the controller in order to show interest to join a group. The controller decides upon the group members and invites them to join the group.

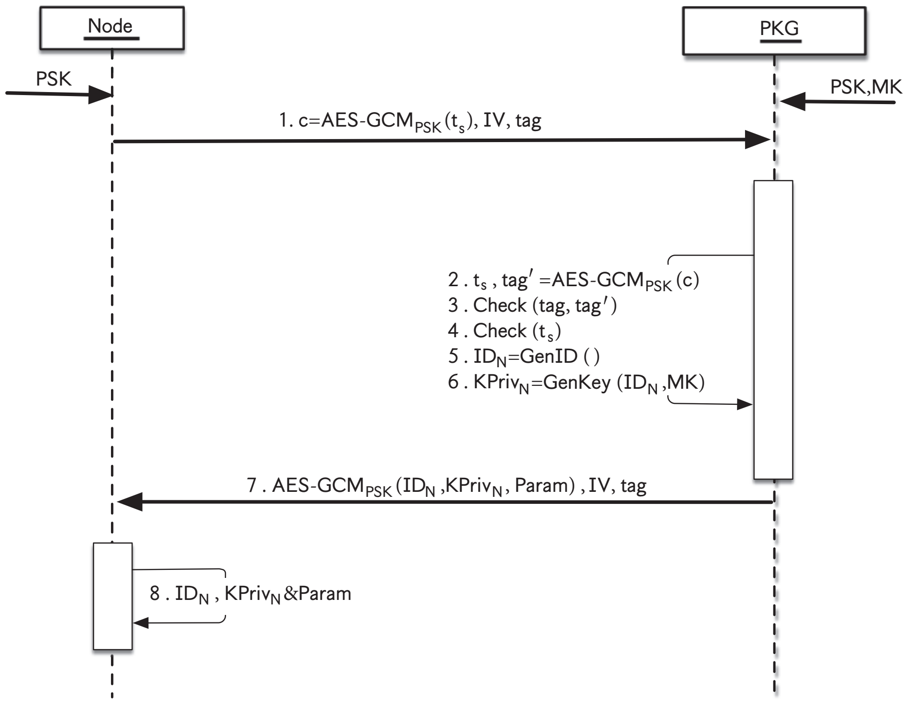

Node
PKG
PSK
PSK,MK
1. c=AES-GCM$_{PSK}$(t$_s$), IV, tag
2 . t$_s$, tag′ =AES-GCM$_{PSK}$ (c)
3 . Check (tag, tag′)
4 . Check (t$_s$)
5 . ID$_N$=GenID ( )
6 . KPriv$_N$=GenKey (ID$_N$,MK)
7 . AES-GCM$_{PSK}$ (ID$_N$,KPriv$_N$, Param) ,IV, tag
8 . ID$_N$, KPriv$_N$&Param

Figure 3: Private key exchange.

Firstly we are assuming that Phase 1 has been already performed and that the switches and controller have securely obtained their private keys. The controller has the power to decide which participants to invite to join a group, according the *Network Requirements* and *Rules* that includes (Figure 1). However when a switch wants to join an already existed group, it broadcasts a `JOIN` message with its node ID, without knowing the ID of controller. The controller is waiting for `JOIN` messages in order to start forming a group of switches. The behavior of the controller when receives `JOIN` message depends on the characteristics and requirements of the running applications which are translated and stored on the *Network Requirements* and *Rules* of the controller. As soon as the controller receives a number of `JOIN` messages and a group has been created, it broadcasts an `INVITE` message to all participated switches of the group, including a given session participant *ID* and node *ID* of all members of the group. The switches afterwards verify that they received the invitation and it follows the group key agreement process, where they are performing two rounds of messages. The two rounds are described in Section 3.

Phase 3 (switch controller communication). Once the group has been formed, we move into the last phase which is used for data exchange. In this phase the controller exchanges data messages with the groups of switches. Furthermore before the controller start exchanging any message with a group of switches, it checks the *Group Table* where all the IDs of group participants information that we described in the previous phase are stored. In case of data transmission between a group of switches and controller, the controller is using the *Secure Channel* where it encrypts the data. The controller encrypts the data using AES in GSM mode, the group key, and an *IV*.

5. Validation and Performance Analysis

As far as the performance analysis of SSPSoC protocol, we based on a simple scenario with three participants: PKG, Switch, and Controller. The messages that will be exchanged between three participants are depicted in Figure 4. As a first step the switch and controller will obtain a private key from PKG by establishing a TCP connection and transmitting the `KEY_REQUEST` message, the PKG will reply by `KEY_REPLY` message. While a TCP connection is needed in order to conduct a validation of our protocol, Layer 4 headers and protocols are not needed in the context on MPSoC, thus before its integration into an MPSoC platform some proper modifications should be performed. Afterwards we move into GKA process, where we implemented the Sharma protocol and Teng protocol, described in Section 3. In order to fit these two GKA protocols in our scenario we implement the following steps:

(1) A switch broadcasts a `JOIN` message, which contains its *ID*; the destination is always the controller and waits for an `INVITE` message.

(2) The controller receives the `JOIN` message, makes a decision about the participants of the group, and broadcasts an `INVITE` message to them, which contains the *IDs* of all the invited participants and waits for `READY` messages.

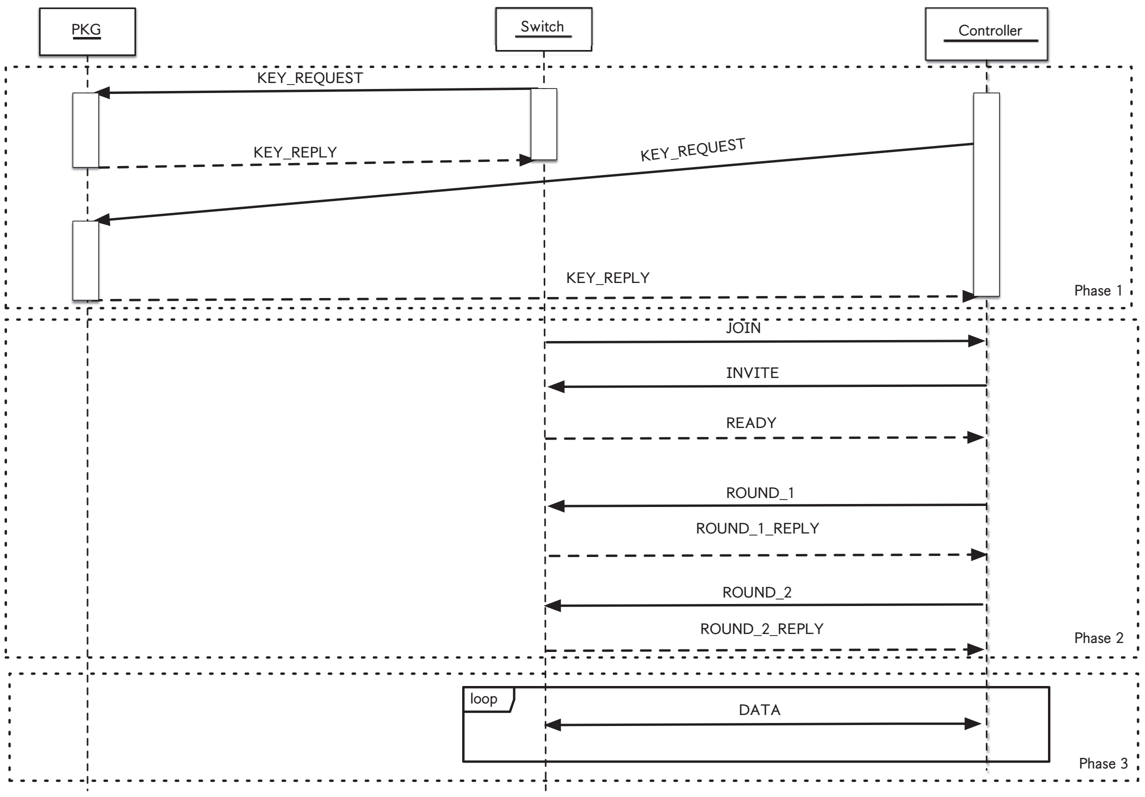


Figure 4: SSPSoC message layer.

(3) The switch receives the `INVITE` message and creates a list of participants. If the *ID* is valid, based on the list, it broadcasts a `READY` message.

(4) The controller remains in idle mode until it receives `READY` messages from the switches that are participants of the group for a specific time. Afterwards it sends a `ROUND_1` message and waits for `ROUND_1_REPLY` messages.

(5) As soon as the switch receives the `ROUND1` message, it broadcasts a `ROUND_1_REPLY` message by waiting for `ROUND_1` and `ROUND_2` messages.

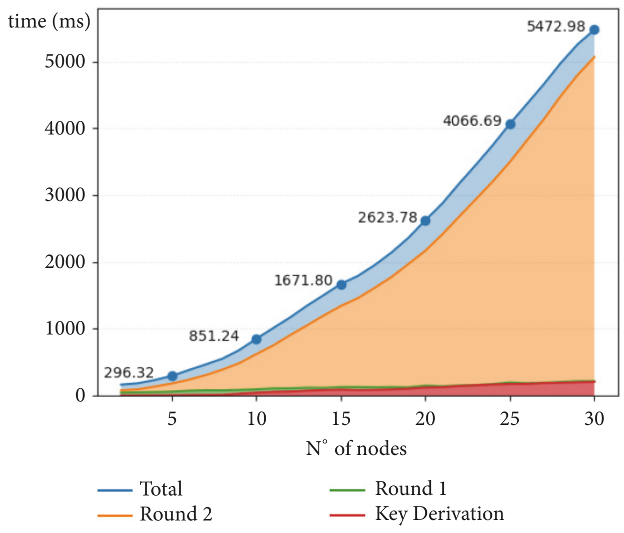
(6) When the controller receives the `ROUND_1_Reply` message from all the participants of the group, it will send `ROUND_2` messages by waiting for `ROUND_2_REPLY` messages.

(7) When the controller receives `ROUND_2_REPLY` messages from all switches, the key computation of group key is started.

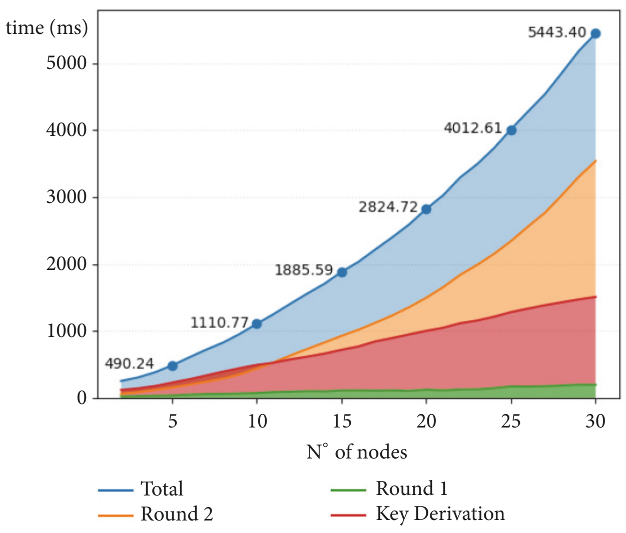
(8) As a last step the switches that belong already into a group can start exchanging `DATA` messages with controller by using OpenFlow protocol.

Following the SDN concept, we validate the SSPSoc protocol by using the emulator Mininet 3 [29], running on a computer. As far as the network entities: OpenVSwitches (OVS) [30] was used as SDN switches and a Ryu [31] was used as an SDN controller. The network hosts are emulated using lightweight OS-level virtualisation: each virtual host inside the mininet network corresponds to a container and it has a virtual network interface with a distinct IP address [21]. Applications, such as the PKG, controller, and node executables, can run directly inside virtual hosts. In our experiments, the hosts are interconnected using virtual Ethernet links and OVS switches running in kernel mode. In each emulated network instance, one virtual host was used to run the PKG, one host for the controller, and the rest of the hosts to run the nodes participating in the GKA. As far as the implementation of GKA protocols, we used the Pairing Based Cryptography (PBC) [32] cryptographic library, RIPEMD-160, SHA-256 hash functions, and AES-GCM cipher. Following the PBC library, Type A (based on symmetric pairing) and Type d159 (based on MNT curves [19]) parameters were used for the implementation of [26] and [25], respectively.

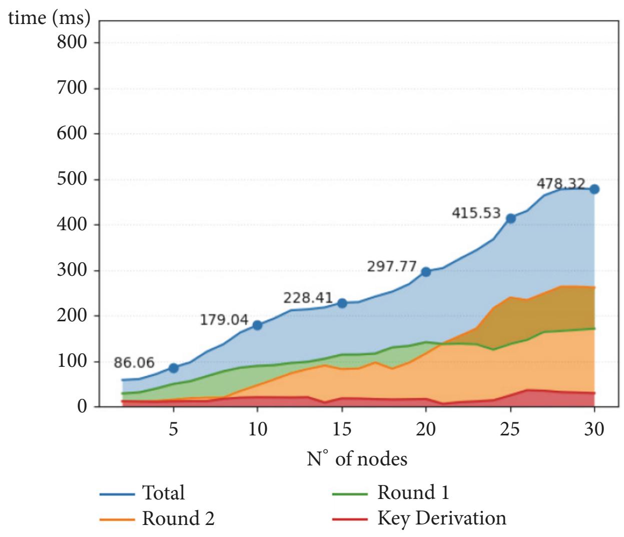
5.1. Network Performance. We perform the scenario for a sample of 1 up to 30 nodes (32 virtual hosts in total). Specifically, in order to test the performance of our protocol based on group key agreement, groups of 2 up to 30 nodes (switches) were created. In this work, the first concern was the evaluation of the performance of our SSPSoC by using two different GKA protocols in order to find out which is more appropriate in our case according to their total cost, the cost of `ROUND_1` and `ROUND_2`, and the key derivation cost. The total cost is referring to the time when a first `INVITE` message has been broadcasted until the time that the first `DATA` message has been formed and sent. The cost of two



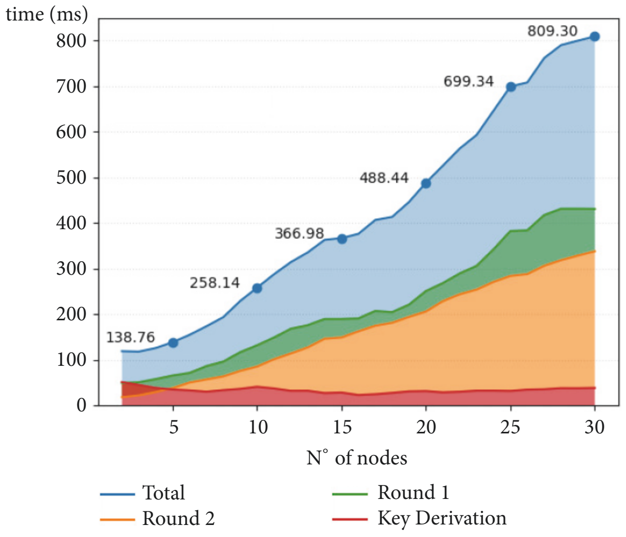
(a) Scalability: Controller delay according to Sharma protocol



(b) Scalability: Nodes delay according to Sharma protocol



(c) Scalability: Controller delay according to Teng protocol



(d) Scalability: Nodes delay according to Teng protocol

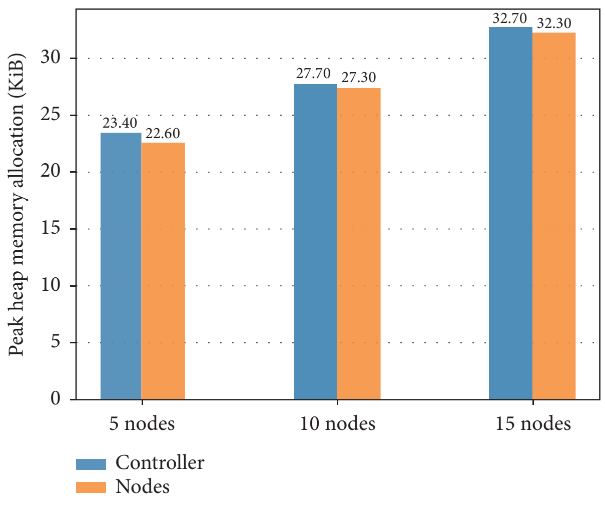
Figure 5: Performance results.

rounds is referring to the period from first ROUND_1 message or ROUND_2 message accordingly sent by controller until the period that the last ROUND_1_REPLY or ROUND_2_REPLY message received by the switch. The key derivation cost refers to time that we need in order to derive the group key (Figure 5).

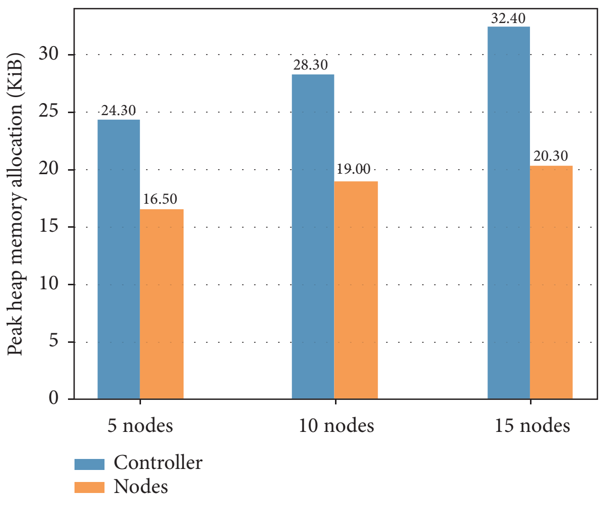
5.2. Memory Consumption. Following the MPSoC concept another important factor we should take into account is the memory consumption, since both GKA protocols are promising low power consumption. We perform the above scenario for 5, 10, and 15 nodes. The total amount of heap memory allocated during the execution of the SSPSoC protocol by using the two GKA schemes was measured with Valgrind tool Suite [20], which perform a dynamic binary analysis and enables the Massif heap profiler. The results are presented on Figure 6.

6. Conclusion and Future Work

In this paper, we propose a new communication protocol based on group key agreement approach able to address the inside communication of a CoC system. Following the design of the proposed SSPSoC protocol, we validate and simulated it within an SDN environment. Our results focused on the evaluation of two GKA schemes according to their scalability and their power consumption. As far as the results are concerned, we noticed that the Teng protocol has far better performance and significantly lower power consumption based on the number of participants and that makes it more appropriate option for the third phase of our SSPSoC protocol. From the other side the Sharma protocol, even without using pairing as Teng protocol, has higher cost and memory consumption. We believe that these results are obtained due to the authenticity of every participant that



(a) Sharma protocol



(b) TENG protocol

FIGURE 6: Memory consumption of two GKA protocols.

the Sharma protocol is considering in contrast to the Teng protocol which does not consider the authenticity of the group participants.

Possibly, future work could be the implementation of the SSPSoC protocol within the context of MPSoC. In order to achieve this, a cyclic accurate simulator should be chosen and some appropriate modifications should be made in the SSPSoC protocol. Another possible extension could be to add more controllers in our system and create groups of controllers; with this way secure communication is provided not only between ICs but also between the different abstraction layers of a MPSoC architecture.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] P. Sethi and S. R. Sarangi, "Internet of things: architectures, protocols, and applications," *Journal of Electrical and Computer Engineering*, vol. 2017, Article ID 9324035, 25 pages, 2017.

[2] G. Bousdras, F. Quitin, and D. Milojevic, "Template architectures for highly scalable, many-core Heterogeneous SoC: could-of-chips," in *Proceedings of the 13th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip, ReCoSoC 2018*, pp. 1–7, IEEE, July 2018.

[3] S. Kumar, A. Jantsch, J.-P. Soininen et al., "A network on chip architecture and design methodology," in *Proceedings of the IEEE Computer Society Annual Symposium, VLSI, 2002*, pp. 117–124, IEEE, 2002.

[4] P. Ezhumalai, A. Chilambuchelvan, and C. Arun, "Novel NoC topology construction for high-performance communications," *Journal of Computer Networks and Communications*, vol. 2011, Article ID 405697, 6 pages, 2011.

[5] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] L. Cong, W. Wen, and Z. Wang, "A configurable, programmable and software-defined network on chip," in *Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications, WARTIA 2014*, pp. 813–816, IEEE, September 2014.

[7] R. Sandoval-Arechiga, J. L. Vazquez-Avila, R. Parra-Michel, J. Flores-Troncoso, and S. Ibarra-Delgado, "Shifting the network-on-chip paradigm towards a software defined network architecture," in *Proceedings of the International Conference on Computational Science and Computational Intelligence, CSCI 2015*, pp. 869-870, IEEE, USA, December 2015.

[8] R. Sandoval-Arechiga, R. Parra-Michel, J. L. Vazquez-Avila, J. Flores-Troncoso, and S. Ibarra-Delgado, "Software defined networks-on-chip for multi/many-core systems: a performance evaluation," in *Proceedings of the 12th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2016*, pp. 129-130, USA, March 2016.

[9] A. Scionti, S. Mazumdar, and A. Portero, "Software defined Network-on-Chip for scalable CMPs," in *Proceedings of the 14th International Conference on High Performance Computing and Simulation, HPCS 2016*, pp. 112–115, Austria, July 2016.

[10] K. Berestizshevsky, G. Even, Y. Fais, and J. Ostrometzky, "SDNoC: software defined network on a chip," *Microprocessors and Microsystems*, vol. 50, pp. 138–153, 2017.

[11] S. Ellinidou, G. Sharma, J.-M. Dricot, and O. Markowitch, "A SDN solution for system-on-chip world," in *Proceedings of the 5th International Conference on Software Defined Systems, SDS 2018*, pp. 14–19, Spain, April 2018.

[12] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] D. Samociuk, "Secure communication between OpenFlow switches and controllers," in *Proceedings of the AFIN 2015: The Seventh International Conference on Advances in Future Internet*, vol. 39, 2015.

[14] H. Zhang, Z. Cai, Q. Liu, Q. Xiao, Y. Li, and C. F. Cheang, "A survey on security-aware measurement in SDN," *Security and Communication Networks*, vol. 2018, Article ID 2459154, 2018.

[15] G. Sharma, V. Kuchta, R. A. Sahu, S. Ellinidou, O. Markowitch, and J. Dricot, "A Twofold Group Key Agreement Protocol for NoC based MPSoCs," in *Proceedings of the 2018 16th Annual Conference on Privacy, Security and Trust (PST)*, pp. 1-2, August 2018.

[16] G. Sharma, S. Ellinidou, V. Kuchta, R. A. Sahu, O. Markowitch, and J.-M. Dricot, "Secure communication on noc based mpsoc," in *Proceedings of the International Conference on Security and Privacy in Communication Systems*, pp. 417–428, Springer, 2018.

[17] O. N. Foundation. OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06), 2015.

[18] R. Klöti, V. Kotronis, and P. Smith, "OpenFlow: a security analysis," in *Proceedings of the 2013 21st IEEE International Conference on Network Protocols, ICNP 2013*, pp. 1–6, Goettingen, Germany, October 2013.

[19] A. Miyaji, M. Nakabayashi, and S. Takano, "New explicit conditions of elliptic curve traces for FR-reduction," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E84-A, no. 5, pp. 1234–1243, 2001.

[20] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *ACM SIGPLAN Notices*, vol. 42, no. 6, pp. 89–100, 2007.

[21] R. Rong and J. Liu, "Distributed mininet with symbiosis," in *Proceedings of the 2017 IEEE International Conference on Communications, ICC 2017*, pp. 1–6, IEEE, May 2017.

[22] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *Proceedings of the 2013 Workshop on Software Defined Networks for Future Networks and Services, SDN4FNS 2013*, pp. 1–7, IEEE, Trento, Italy, November 2013.

[23] T. Dierks, "The transport layer security (TLS) protocol version 1.2," Tech. Rep. RFC5246, 2008.

[24] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*, pp. 151-152, ACM, August 2013.

[25] G. Sharma, R. A. Sahu, V. Kuchta, O. Markowitch, and S. Bala, "Authenticated group key agreement protocol without pairing," in *Proceedings of the International Conference on Information and Communications Security*, pp. 606–618, Springer, 2017.

[26] T. Jikai and W. Chuankun, "An identity-based group key agreement protocol for low-power mobile devices," *Chinese Journal of Electronics*, vol. 25, no. 4, pp. 726–733, 2016.

[27] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proceedings of the The Computer Security Foundations Workshop VII*, pp. 187–191, IEEE, Franconia, NH, USA, 1994.

[28] M. J. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," Tech. Rep. SP 800-38D, 2007.

[29] Mininet Project. 2017. http://mininet.org/.

[30] Linux Foundation, "OvS Open vSwitch," 2016. http://www.openvswitch.org.

[31] Ryu SDN Framework Community, "Component-based software defined networking framework build SDN agilely," 2017. https://osrg.github.io/ryu/.

[32] Ben Lynn, PBC Library. 2018. https://crypto.stanford.edu/pbc/.