This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

HiAuth: Hidden Authentication for Protecting Software Defined Networks

Osamah Ibrahiem Abdullaziz, *Student Member, IEEE,* Li-Chun Wang, *Fellow, IEEE,* and Yu-Jia Chen

***Abstract*—Software defined networking (SDN) enables network function programmability for ease of configuration and maintenance, and also allows network administrators to change traffic rules on the fly. However, denial of service (DoS) attacks pose security challenges on the centralized control plane of SDN. Although the transport layer security (TLS) can help secure the control plane, it is computationally intensive, complex to configure, and not mandatory in OpenFlow protocol. In this paper, we present a lightweight authentication solution, called Hidden Authentication (HiAuth), to protect the SDN controller by hiding the identities of the forwarding devices into the control packets via efficient bitwise operations. HiAuth is the first to incorporate information hiding techniques into OpenFlow to provide security against DoS attacks. HiAuth exploits the IP identification field of IPv4 and the transaction identification field of OpenFlow in two authentication schemes. The experimental results show that HiAuth can effectively mitigate intruder DoS attacks and provide high undetectability to attackers.**

***Index Terms*—Software defined networking (SDN), OpenFlow, Denial of service (DoS) attacks, Information hiding.**

I. Introduction

SOFTWARE defined networking (SDN) decouples the network control from the forwarding devices hence simplifies and enhances network management [1]. In general, SDN architecture consists of three layers: application plane, control plane, and data plane. Fig. 1 illustrates the architecture and the interfaces between the three layers of SDN. The northbound application programming interfaces (APIs) allow SDN applications to convey routing and security policies into the control plane. Subsequently, SDN control plane enforces those policies on the forwarding devices through the southbound APIs. OpenFlow is the first SDN standard which defines open southbound interfaces for controlling network flows.

The SDN centralization of network intelligence with the availability of a global view of the entire network improves programability and scalability for future network and service management. As a result, many new SDN applications are proposed to optimize the network performance from different aspects, including throughput maximization [2] [3], deterministic delay guarantee [4], and security enhancement [5] [6]. Clearly, the capability to influence the network behavior through software from a logically centralized control brings several advantages. However, software vulnerabilities become a concern [7]. More severely, the centralization of the control plane can cause a single point of failure. Unfortunately, transport layer security (TLS) is optional in OpenFlow protocol. Because of its configuration complexity, TLS is not adopted by some OpenFlow-enabled switches and controllers [8]. As a result, the legitimacy of the forwarding devices can not be verified. Hence, malicious attackers can easily launch denial of service (DoS) attacks against the SDN controller.

To resolve the DoS issue, we first generalize the threat concerns of DoS attacks against the SDN controller. In particular, we study the possible DoS attack scenarios, including the outsider and the insider attack classes. Then, we evaluate the DoS impacts on the SDN controller. Finally, we present a lightweight authentication mechanism, Hidden Authentication (HiAuth), to mitigate intruder DoS attacks.

HiAuth has two important features: 1) obscure and 2) lightweight. Firstly, for obscurity, HiAuth mimics the original statistical distributions of the values generated by the operating systems to stay undetected by attackers. Secondly, for being lightweight, HiAuth only relies on simple bitwise operations for computations. Thus, it does not require specialized hardware. To the best of our knowledge, HiAuth is the first to incorporate information hiding techniques into OpenFlow protocol to provide security against DoS attacks at the protocol level. Here, we propose two HiAuth schemes:

- IP identification (IPID) based HiAuth which conceals the identities of the forwarding devices within the IPID field of the IPv4 header.
- Transaction identification (XID) based HiAuth which conceals the identities of the forwarding devices within the XID field of the OpenFlow header.

IPID-based HiAuth provides authentication at the network layer. However, in the case if IPv4 becomes obsolete in the future or a complete transition to IPv6 occurs, XID-based HiAuth can be utilized to provide the same level of authentication.

The rest of the paper is organized as follows. Section II provides a background on SDN security challenges and network information hiding. Section III investigates the security of the SDN control channel including the TLS usability and the threat of DoS attacks on SDN controller. Section IV reviews the existing work on IPID-based information hiding and Section V details the proposed authentication schemes. Section VI presents the experimental results. Section VII provides recommendations and discusses the strengths and the limitations of the HiAuth schemes. Finally, we state our concluding remarks and future work in Section VIII.

II. Background

In this section, we discuss the impact of SDN features on its security and introduce the concept of network information

Osamah Ibrahiem Abdullaziz, Li-Chun Wang, and Yu-Jia Chen are with National Chiao Tung University. Corresponding author: Li-Chun Wang (Email: lichun@g2.nctu.edu.tw)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

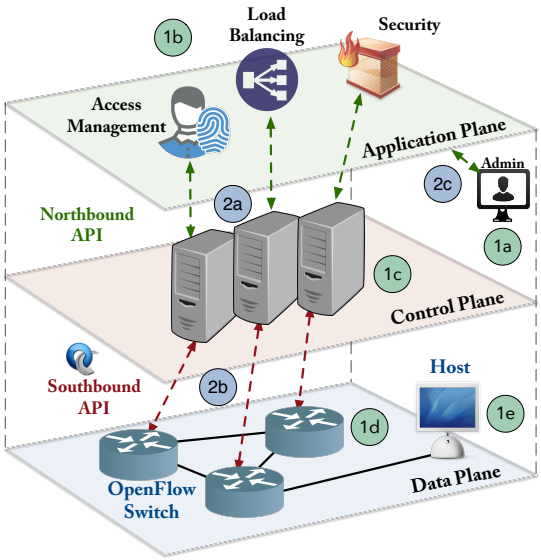


Fig. 1: SDN architecture and its security challenges.

hiding.

A. Security in SDN

SDN has two main advantages: network programmability and centralized network management. First, with separation of control plane and data plane, programmable SDN allows network policies to be modified by software instead of manual configurations as compared to traditional networks. To update a network policy in traditional networks, each device has to be manually configured, which may cause misconfiguration and thus leads to security vulnerabilities [9]. Secondly, the centralized control logic can facilitate network management because of the availability of the network global view [10].

Security in SDN has two aspects, namely security through SDN and security of SDN. On the one hand, security through SDN focuses on utilizing SDN features to resolve traditional network security issues. SDN security applications can inspect packets through the control plane. After security analysis, these applications can drop or redirect the traffic to security middleboxes [11]. On the other hand, security of SDN deals with the security challenges caused by SDN, e.g., single point of failure.

To further elaborate the differences between these two aspects, we take DoS prevention as an example. In security through SDN, DoS attack on traditional networks can be prevented by SDN applications [6] [12]. In [12], Defense4All is designed to detect and mitigate DoS attacks. Defense4All performs two main tasks, namely (a) behavior monitoring by analyzing traffic statistics, and (b) traffic redirecting by setting flow entries to reroute the DoS traffic. A web server under a DoS attack within a network can be protected by such SDN applications. In the case of security of SDN, DoS attack against control plane can be prevented at the protocol level. TLS and rate limiting techniques are solutions recommended by the current SDN standards. The proposed HiAuth mechanism belongs to the latter aspect in which it aims to provide a lightweight packet level authentication to prevent DoS attacks against the control plane.

Security of SDN architecture faces system and network-traffic related challenges. Fig. 1 illustrates these two kinds of security challenges, where label (1) and label (2) refer to the system and network-traffic related challenges, respectively.

1) *System Related Security Challenges*

SDN inherits most of the system-related security threats from traditional networks. More dangerously, the benefits that SDN brings to the network management also intensify the impacts of these threats, which are described as follows.

a) *Administrative Device Security* - The administrative device is the management machine which is used by network administrators to access the application plane. Vulnerable management machines put both data and the entire network at risk. One countermeasure is to keep the management machine updated and adopt strong security scheme, such as multi-level authentication.

b) *Application Security* - Practically, no software comes without bugs or flaws. Network applications and services will always be targeted by malicious attackers. SDN heavily relies on software to manage the network. If the network behaviors can be changed by software, it is susceptible to software related vulnerabilities as well. Therefore, a malicious application could potentially compromise the network. In this case, timely reliable software update and patching are essential.

c) *Controller Device Security* - SDN controller is the most important element in the SDN architecture. Similar to management device security, the controller operating system and its applications are attractive targets and share the same threats. A faulty or malicious controller could violate the desired behavior and thus harm the entire network. Existing security solutions in traditional networks and continuous update can be applied to protect the controller from these threats.

d) *Forwarding Device Security* - It may seem that a faulty or a malicious switch is harmless to the network since SDN moves the intelligence from the forwarding devices to its controller. This is not true because a single switch can be exploited to discard or reroute network traffic. In the worst case, a switch can be used to overload the controller with forged requests in a DoS attack.

e) *Host Device Security* - The security of a host in SDN is equally important. A malicious host can be exploited to indirectly saturate controller resources. When a host communicates with another host, the SDN switch checks its flow tables for a matching rule. If not matched, the switch requests routing information from the controller. By exploiting this procedure, an attacker can launch a DoS attack against the controller.

2) *Network Traffic Related Security Challenges*

Unlike system-related security, network-traffic related security focuses on the vulnerabilities of communication protocols. SDN transforms the network from vertically centralized to horizontally distributed architecture. This

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

design extends the mutual-communications [13] among components and therefore brings new network security challenges as follows.

a) *Administrative Channel Security* - Refers to the security of the channel between the administrator device (can be a mobile device) and the SDN applications. Lack of security protocols in this channel may lead to network policy tampering through man-in-the-middle (MITM) attacks.

b) *Management Channel Security* - Refers to the security of the channel between the SDN applications and the network controller. This is especially critical when SDN applications and controller run on different physical machines. The lack of trust and authentication mechanism between SDN application and controller may lead to security threats such as MITN and DoS attacks.

c) *Control Channel Security* - Refers to the security of the channel between the SDN controller and the forwarding devices. In the absence of an authentication protocol in this channel, DoS and MITM attacks can target control channel devices.

B. Network Information Hiding

Intuitively, information hiding aims to intelligently insert data into a suitable carrier, while keeping the tampering of that carrier imperceptible [14]. Information hiding applications include steganography [15], fingerprinting, watermarking [16], and authentication. Commonly, a digital carrier such as text, image or video is exploited for hiding information. More recently, network protocols are utilized for information hiding [17]–[20]. Network information hiding deals with hiding information into network traffic. In general, network information hiding schemes are evaluated by their capacity, robustness and undetectability [21]. Capacity is the amount of hidden information per second. Robustness refers to the resistance against altering hidden information in the noisy channels. Lastly, undetectability is the ability to avoid detection. In general, there are three kinds of network information hiding schemes:

- *Packet data unit (PDU) scheme* - Inserts data into packet headers. PDU schemes can be easily implemented and provide high capacity. In [22] and [23] packet length is used to hide information, whereas the time to live (TTL) field of IP protocol is utilized in [24]. On the other hand, [17] exploits Etherleak vulnerability [25] to hide information by padding small Ethernet frames.
- *Protocol behavior (PB) scheme* - Encodes data by exploiting network protocols' behaviors. In [26], the retransmission of reliable protocols is exploited. In another example, the reception and the absence of packets within a predefined period of time is utilized to encode information in [27].
- *Network applications and services (NAS) scheme* - Exploits the characteristics of network applications to hide information. In [18] the encrypted packets of silence signal in Skype VoIP application is exploited.

TABLE I: IPID generation in various operating systems.

OS	Version	Global counter	Per-dst counter	Randomly generated
Windows	XP/7/8/10	✓		
Ubuntu	16.04		✓	
Android	4.3		✓	
MacOS	10.12.5			✓
iOS	6.1.3			✓
OpenBSD	5.2			✓

Among the investigated PDU-based schemes, the IPID field [28] is considered for information hiding [29]–[33]. In our recent work [19], called AIPISteg, both the IPID field and the packet payload are utilized to hide information. The IPID can be used to recover a fragmented packet at the destination. IP fragmentation can divide a packet into smaller ones with smaller maximum transfer unit (MTU). For this reason, IPID uniquely associates fragments to packets for reassembly at the destination. Despite the fact that IPID is defined in [28], the details of its generation algorithm is not given. As a result, IPID is openly implemented by vendors but still complies to the requirements in the specification. To the best of our knowledge, there are three IPID generating mechanisms: global incremental counter, per-destination incremental counter, and pseudo random generation. The IPID generation mechanisms for various operating systems are summarized in Table I.

Generally, if a unique IPID is desired, the maximum possible period before repeating an IPID is $2^{16} - 1$ since the IPID field is 16-bit in size. Therefore, the easiest implementation of IPID generation mechanism is an incremental counter. In Windows-based operating system, a global incremental counter is used for IPID generation. That is, network applications share the same counter. On the other hand, Linux-based operating systems, such as Ubuntu, utilize per-destination incremental counter, where counters are created for every destination [34]. Alternatively, Unix-based operating systems, such as Mac OS X, iOS and OpenBSD, follow the pseudo random IPID generation process. Although the IPID values are randomly generated, the aforementioned platforms ensure that the IPIDs are not repeated too soon.

III. Control Channel Security

A. TLS Usability and Adoption

Security is essential, but it is not always considered when designing new network architectures or protocols [35]. Likewise, the security of SDN architecture is afterthought. One evidence to this fact is the widespread failure in the adoption of TLS for the control channel. Since after the earliest OpenFlow specification v1.0.0, the communication between the controller and the switch optionally recommends TLS connection. That is, the subsequent OpenFlow specification versions, including the latest v1.5, make TLS an optional communication mode. Consequently, many SDN vendors do not support TLS in their products. A survey on the TLS adoption by various SDN vendors was presented in [8], showing that many controllers and switches do not support TLS connection. In addition to being optional in the specification, the failure in TLS adoption may be regarded to the following reasons:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

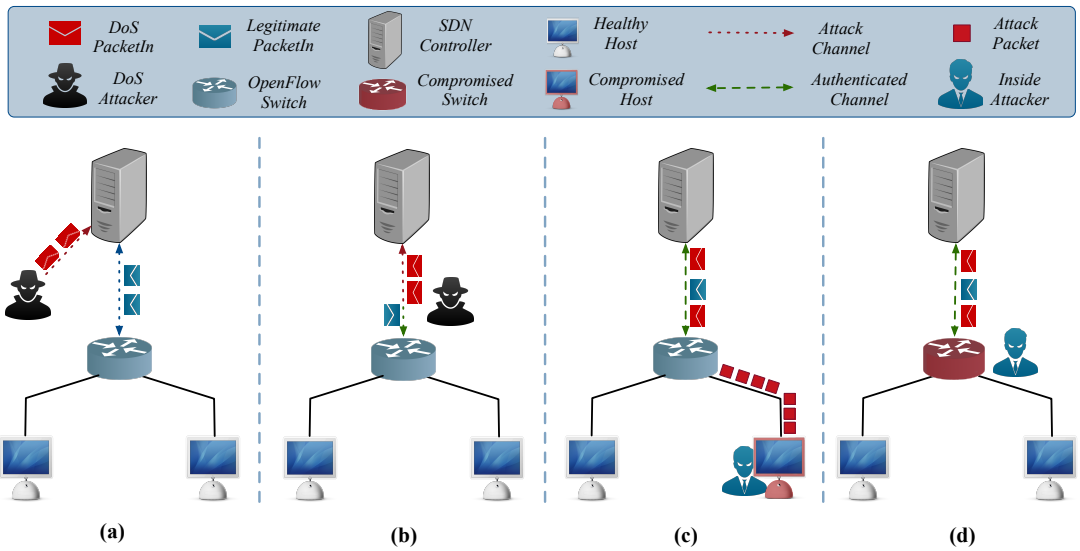


Fig. 2: DoS attacks scenarios against SDN controller. (a) Unauthenticated channel DoS, (b) Man-in-the-middle DoS, (c) Compromised host DoS, and (d) Compromised switch DoS.

- *Configuration complexity*: the tasks involved in the proper configuration of a secured TLS connection are challenging [36] [37].
- *Lack of mutual support*: the lack of TLS support in both the controller and the switch simultaneously. TLS can be CPU-intensive for some switches [38];
- *Rapid standardization*: vendors put more attention on the rapid development of OpenFlow.

Recent studies show that configuring TLS is challenging even for an experienced user. In [36], the usability issues in deploying TLS were investigated. Participants in this study are the knowledgeable individuals in the field of security and privacy-enhancing technologies. The results in [36] was evaluated based on TLS server rating criteria [39]. Only 21% of the participants were graded A, while 57% and 7% and 14% of participants were graded B, C, and failed, respectively. Other studies focused on the effect of certificate warnings and the linguistic difficulty of browser warning messages on non-expert users [40], [41].

To verify this finding, we performed an empirical analysis on the TLS configuration of the top 450 web servers reported by alexa.com. In this analysis, we also utilize Qualy's SSL test and evaluation criteria [39]. Our result shows that 59% of the tested servers scored grade A, whereas 26% of them scored below. Fig. 3 shows the result of TLS configuration analysis.

B. DoS Attacks Against SDN Controller

Due to the centralized nature of SDN, the lack of security enforcement, and TLS usability issues, DoS becomes a major concern. Here, we systematize the possible scenarios where DoS attacks can be launched against SDN controller. Note that there are two classes of DoS attackers, namely outsider and insider. An outsider attacker is an intruder by the network legitimate devices. On the other hand, an insider attacker is a compromised legitimate device that can communicate with other members in the network [42]. Accordingly, we dictate that the DoS attacks against the SDN controller can be in one of the following scenarios:

a) *Unauthenticated channel outsider DoS (Fig. 2(a))*: In the absence of TLS authentication mechanism, switches communicate with the controller through a plain TCP connection. Specifically, a switch only requires the IP address and the port number of the controller to establish the communication. In that way, an intruder can pretend to be a legitimate switch and flood the controller with a large number of packets. For example, a tool for identifying SDN networks and attacking its controller was introduced in [43].

b) *Man-in-the-middle (MITM) outsider DoS (Fig. 2(b))*: Similarly, in the absence of an authentication mechanism, a MITM attack is possible [44]. In this scenario, the MITM attacker impersonates a legitimate switch and attempts to connect to the controller for the purpose of disrupting the controller connection with the legitimate switch. For instance, in [45] a vulnerability in Open Floodlight controller is exploited which allows an attacker to impersonate a legitimate switch and disable its control link.

c) *Compromised host insider DoS (Fig. 2(c))*: Hosts software vulnerabilities may present opportunities for an attacker to launch an insider DoS attack. In this situation, an attacker exploits the vulnerability of a legitimate host to gain access to the network and carefully crafts packets to its connected switch to exhaust the controller resources [46] [47].

d) *Compromised switch insider DoS (Fig. 2(d))*: As explained in Section II-A, malicious switches may harm the SDN network. Similar to type (c) attack, a malicious switch may exhaust the controller resources and ultimately affect the network availability [47].

The outsider class of DoS attacks can be prevented by an authentication mechanism that can verify the identity of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

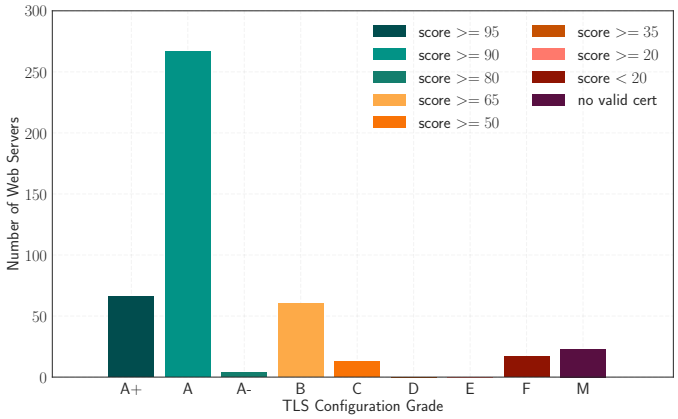


Fig. 3: Analysis of the TLS configuration quality of the top 450 web servers.

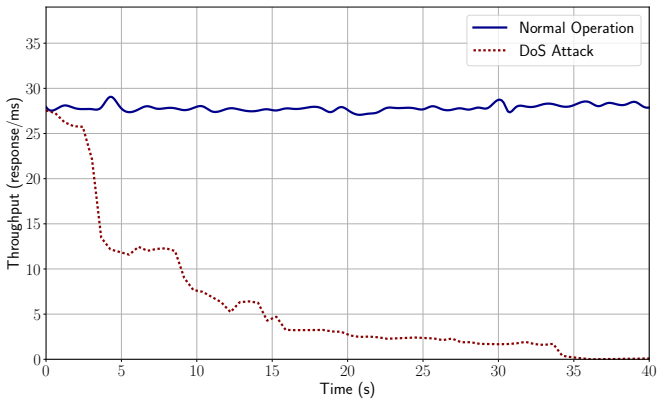


Fig. 4: Controller throughput during normal operation and during DoS attack (dotted line) [46].

forwarding devices. In this case, when TLS is not supported, the control plane resources can be easily violated. HiAuth aims to prevent outsider DoS attacks through a lightweight authentication mechanism.

The insider class of DoS attacks can be prevented by various techniques including request rate limiting [48]–[51], controller scheduling [52] and request classification and prioritization [53]–[55]. For example, a rate limiting approach is introduced in OpenFlow 1.3.0, which enables the controller to lower the request rates of the switches when the rates are more than it can handle. In addition, AvantGuard [48] and LineSwitch [49] utilize rate limiting to mitigate a traditional TCP-based DoS attack, known as SYN flooding, in SDN architecture. FloodGuard [50] and FlowSec [51] introduce protocol-independent rate limiting techniques.

SDN network implements two modes for flow routing rule installation into switches, namely proactive and reactive. In the proactive mode, flow routing rules are pre-installed in advance into switches for anticipated routing destinations. On the contrary, in the reactive mode, flow routing rules are installed by the controller in run time as response to packet-In messages. Switches in the reactive mode networks send packet-In message to the controller when no rule is matched in the flow tables. The packet-In messages are the only switch messages to which the controller is required to respond.

In our previous work [46], we simulated a DoS attack to demonstrate the impacts of DoS on SDN network. A reactive network was simulated, where the controller spends some resources to answer packet-In messages. This DoS attack generates a large number of carefully crafted packets with unique flow attributes to cause the no-match events and the packet-In messages. As a consequence, these huge number of attack packets can impair the network usage in two ways: saturate the switch flow tables and then deny legitimate rule installation and exhaust controller resources and keep it busy from responding to the legitimate switches' packet-In messages.

Fig. 4 shows the experimental result of the controller throughput during normal operation and during DoS attack. The x-axis represents the time in seconds. while the y-axis represents the throughput, which is the number of controller responses per millisecond. The average throughput of the controller is first evaluated during normal operation. Subsequently, with the same configuration, the throughput of the controller is evaluated during the DoS attack.

To mitigate the threat of DoS attacks in SDN, it seems applicable to replicate the control plane [7], i.e., changing the control plane from centralized to distributed architecture. However, this solution may not be viable. In [56], it is shown that by merely utilizing multiple controllers (replicas) in SDN networks may not be sufficient to avoid a single point of failure. It is observed that the load of the backup controllers may exceed their capability and results in cascaded failures.

IV. IPID-based Information Hiding

Most of the IPID-based information hiding methods assume that the IPID value is random [29]–[33]. It is not true in most operating systems as shown in Table I. In particular, at least three IPID-based random data embedding schemes are proposed in the literature: (1) 2-byte plain data embedding (PDE) [29], (2) 1-byte encrypted data and 1-byte random data embedding (EDE-1) [30], and (3) 2-byte encrypted data embedding (EDE-2) [31]–[33]. In PDE method, the 16 bits of the IPID field are replaced by the ASCII representation of the hidden information. Specifically, Rowland's proposal [29] is an example of PDE and the pioneering work in the area of IPID-based information hiding. Although it is straightforward and easy to implement, this method has the following weaknesses:

- *Easily detectable* - It results in abnormal IPID distribution, which can be easily distinguished from the original IPID distribution regardless of the utilized IPID generation mechanism.
- *Vulnerable to unauthorized access* - The embedded data are not protected and can be easily extracted from the IPID if anomaly is detected.
- *Vulnerable to IP fragmentation* - The IPID is duplicated when fragmentation occurs. Hence, the recipient receives multiple copies of the same data.

Later, the authors of [30] developed another method to encrypt the data by using Toral Automorphism [57] before the embedding process. Next, a byte of the encrypted data is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

embedded into the first byte of the IPID and the second byte is randomly generated for the purpose of packet identification. Also, they suggested to probe the network for maximum transmission unit (MTU) prior to initiating the communication session. This method improves the secrecy of the embedded data but the channel capacity is greatly reduced. In addition, conventional path MTU discovery [58] relies on Internet control message protocol (ICMP) to probe the transmission path for the smallest packet size. Unfortunately, more and more networks drop ICMP packets to avoid DoS attacks [59]. Alternatively, [31]–[33] suggested to utilize packetization layer path MTU discovery (PLPMTUD) [60], which depends on TCP to probe for a network's MTU setting. In their work, the full two bytes of the IPID field is replaced with two bytes of the encrypted data.

Indeed, the knowledge of MTU is critical for the aforementioned methods. In practice, however, the MTU value may vary even for the same communication session. Moreover, some networks still connect to non-compliant devices, which may fragment packets even if fragmentation is not permitted by the network applications [61]. In addition to the drawback of relying on path probing, random IPID-based methods also neglect the mechanism in which the IPID values are generated by different operating systems. The existing works assume that encrypting the data results in random values, which are suitable for IPID. In fact, exploiting the randomness of the encrypted data as IPID values for secret communication applications is not viable. This is because the random IPID values generated by operating systems exhibit statistical distribution which is different form those of the existing IPID-based methods. For this reason, the existing IPID-based methods are vulnerable to anomaly detection.

To overcome the shortcomings of the existing works, we propose an authentication application of information hiding – HiAuth. In this scheme, the statistical distributions of the IPID values generated by various operating systems are analyzed first. Next we introduce a corrective mechanism to rectify the distribution of the HiAuth IPID values. Furthermore, a fast stream cipher is utilized to protect the hidden data and provide randomness to the generated IPID values. Finally, we develop the solutions to handle data duplication when packet fragmentation occurs.

V. Hidden Authentication

HiAuth is a packet-level authentication mechanism aiming to mitigate the class of outside DoS attacks against the control plane. On one hand, HiAuth is lightweight in terms of computation since it only needs a bitwise operation and a simple mapping function to hide the identity information of a device into control packet header. On the other hand, HiAuth configuration is rather straightforward and easy when compared to TLS configuration. In this section, we introduce two HiAuth schemes namely, IPID-based HiAuth and XID-based HiAuth.

A. IPID-based HiAuth

IPID-based HiAuth is designed to be utilized in the platforms that implement randomness in their IPID generation. Despite the work published in the literature on hiding information into the IPID field, it is clear that most of those methods neglect the obvious alteration which is introduced to the distribution of IPID. Furthermore, all the methods are functional only when the network MTU is predetermined. Therefore, we propose IPID-based HiAuth to provide packet authentication and overcome these shortcomings.

The IPID-based HiAuth encoding and decoding processes are symmetric and consist of three functions: (1) one-time pad generation, (2) data mixing, and (3) distribution corrective mechanism. HiAuth encoding process requires the following inputs: a) a 256-bit key, a 64-bit block counter, and a 64-bit nonce to be used in the one-time pad generator, and b) a 12-bit device identification to be used in the data mixing function. The output of these two functions is then mapped into a 12-bit IPID base value. Finally, a distribution corrective mechanism converts the IPID base value into a distribution compliant 16-bit IPID which will be used for the control packet transmission.

1) One-time Pad Generation (OPG): is an important function of HiAuth to maintain the security of the authentication process. OPG relies on high-speed stream ciphers to produce key stream. For instance, RC4 and ChaCha stream ciphers are known for their simplicity and speed in software. Here, we exploit ChaCha [62] for its proven security. ChaCha is a secure, fast and remarkably simple stream cipher algorithm. Recently, ChaCha has gained more popularity especially after Google included it in its openSSL cipher suite [63]. Although ChaCha is mainly utilized for encryption, its core design can be treated as one-time pad generator. When compared to the well-know Advanced Encryption Standard (AES), ChaCha is three times faster in software implementation running on platforms that lack specialized AES hardware [64]. In addition, ChaCha is not vulnerable to timing attacks as in the case of AES [65].

ChaCha state is a matrix of 4×4 32-bit unsigned integers (16 words) which get scrambled by a function called quarter round (QR) for r number of rounds. The initial state X consists of 4 words of constant c, 8 words of key k, 2 words of block counter b (incremented after r rounds), and 2 words of nonce n.

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \Leftarrow \begin{pmatrix} c & c & c & c \\ k & k & k & k \\ k & k & k & k \\ b & b & n & n \end{pmatrix}$$

The number of rounds can be 8, 12 or 20. That is, if r = 8, the QR function will run 32 times before producing an output stream. A QR performs arithmetic operations including $\boxplus$ addition modulo 2^{32}, $\lll$ rotation, and $\oplus$ exclusive-or on 4 words each time and alternate between column or diagonal rounds (see Algorithm 1).

Once the QR function has ran for the specified number of rounds, the updated state is added to the current state matrix resulting in a 64-byte of key stream. IPID-based HiAuth extracts 12-bit pad S_r from the key stream to be used in the data mixing function. The parameters of ChaCha algorithm and the device ID can be configured either offline by the network administrator or over the network through existing key

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

Algorithm 1: ChaCha Quarter-round Function

Input: four 32-bit words a, b, c, and d
Output: scrambled words a, b, c, and d

begin
  $a \leftarrow a \boxplus b$;
  $d \leftarrow (a \oplus b) \lll 16$;
  $c \leftarrow c \boxplus d$;
  $b \leftarrow (c \oplus b) \lll 12$;
  $a \leftarrow a \boxplus b$;
  $d \leftarrow (a \oplus d) \lll 8$;
  $c \leftarrow c \boxplus d$;
  $b \leftarrow (c \oplus b) \lll 7$;
return a, b, c, d

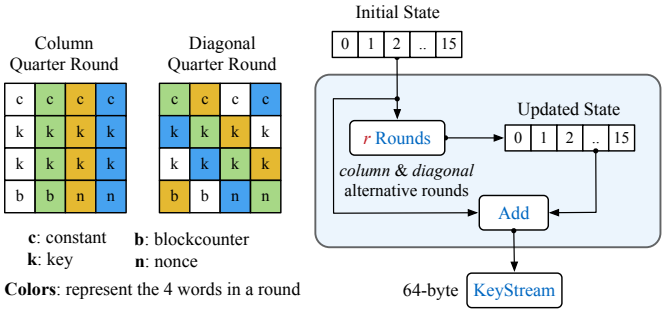


Fig. 5: Illustration of ChaCha stream cipher.

exchange protocols such as Diffie Hellman [66]. Fig. 5 shows ChaCha block function and the arrangement of the column and the diagonal rounds.

2) Data Mixing (DM): is a simple mixing function that provides security and randomness to its input. It consists of bitwise exclusive-or (XOR) operation and a lookup table mapping. The input to DM function is a 12-bit one-time pad, which is taken from OPG, and a 12-bit device ID D_i. First, DM performs an XOR operation between S_r and D_i to produce a 12-bit table entry sequence T_s. The XOR operation is computationally cheap and fast approach to scramble the input data. Given an XORed bit, the probability that the input bit is equal to 0 or 1 is 0.5. As such, the security of bitwise operation output strongly depends on the randomness of key stream X. That is, the table entry, which is the result of $S_r \oplus D_i$, should not reveal any information to an attacker about the device ID.

Subsequently, T_s is used in a lookup table T to output a 12-bit value, which represents the 12 least significant bits (LSB) of the IPID, denoted as IPID base value $IPID_b$. Here, the table consists of 4,096 entries, each of which has 12 bits. In particular, each table entry is associated with a unique 12-bit decimal value as its output $IPID_b$. To speed up the process of mapping a table entry into $IPID_b$, indexing is used instead of conventional matching. Indexing requires a single memory access and has a constant hit time (i.e., time required to find a bit sequence). To implement index mapping, the lookup table entries are sorted and used as the index for the respective $IPID_b$. In other words, the result of the XOR operation is an index pointing to an $IPID_b$.

3) Distribution Corrective Mechanism (DCM): is a platform specific function combined with OPG and DM to generate IPID values. The generated IPID values will comply with the IPID distribution of the platform's operating system. This is to ensure high undetectability of the hidden information. DCM is based on UNIX IPID generation, which is used in the Darwin v.16.3.0. As stated, IPID generation algorithms exploit one of the IPID generation mechanisms presented in Table I. In UNIX-based systems, such as Darwin and OpenBSD, IPID values are randomly generated. In fact, these systems display distributions that are different from uniform random distribution. This is due to the searchable queue implementation which compromises between IPID predictability and IPID collision avoidance [67]. DCM is designed as a selector for compliant IPID values. The remaining 4 MSB of the full IPID field are utilized to generate 16 IPID candidates to prevent reusing the same IPID within a searchable queue. The searchable queue is utilized to store recently used IPIDs. The size of the queue is 2^{12}, which provides a good tradeoff between randomness and non-repetition while taking performance into account [67]. The following steps and Algorithm 2 details the encoding process of IPID-based HiAuth.

Algorithm 2: IPID-based HiAuth Encoding Algorithm

Input: key k, counter b, nonce n, 12-bit D_i and lookup table T, where length of $T = 2^{12}$
Output: The $IPID$ to be used

ChaCha initialization
  $X \leftarrow$ init(k, b, n) `//only once`
Generate 12-bit sequence using ChaCha
  $X' \leftarrow X$ `//state working copy`
  $X' \leftarrow$ quarter-rounds
  $X \leftarrow X \boxplus X'$
  $S_r \leftarrow$ 12-bit MSB X `//one-time pad`
eXclusive OR
  $T_s \leftarrow S_r \oplus D_i$ `//table entry`
Perform table lookup
  $IPID_b \leftarrow$ **return** *result* `//IPID base value`
Generate IPID options
for *each combination in a 4-bit sequence* **do**
  IPIDChoice $\leftarrow$ 4-bit sequence $\|$ $IPID_b$
IPID Selection
while *IPIDChoice in queue* **do**
  choose another IPIDChoice
append IPIDChoice to queue
Correct the IPID generation
if *queue size* $>$ *4096* **then**
  pop the first queue element
return $IPID$

Step 1: Configure the one-time pad generator by initializing ChaCha.
Step 2: Assign a switch ID to identify the switch.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

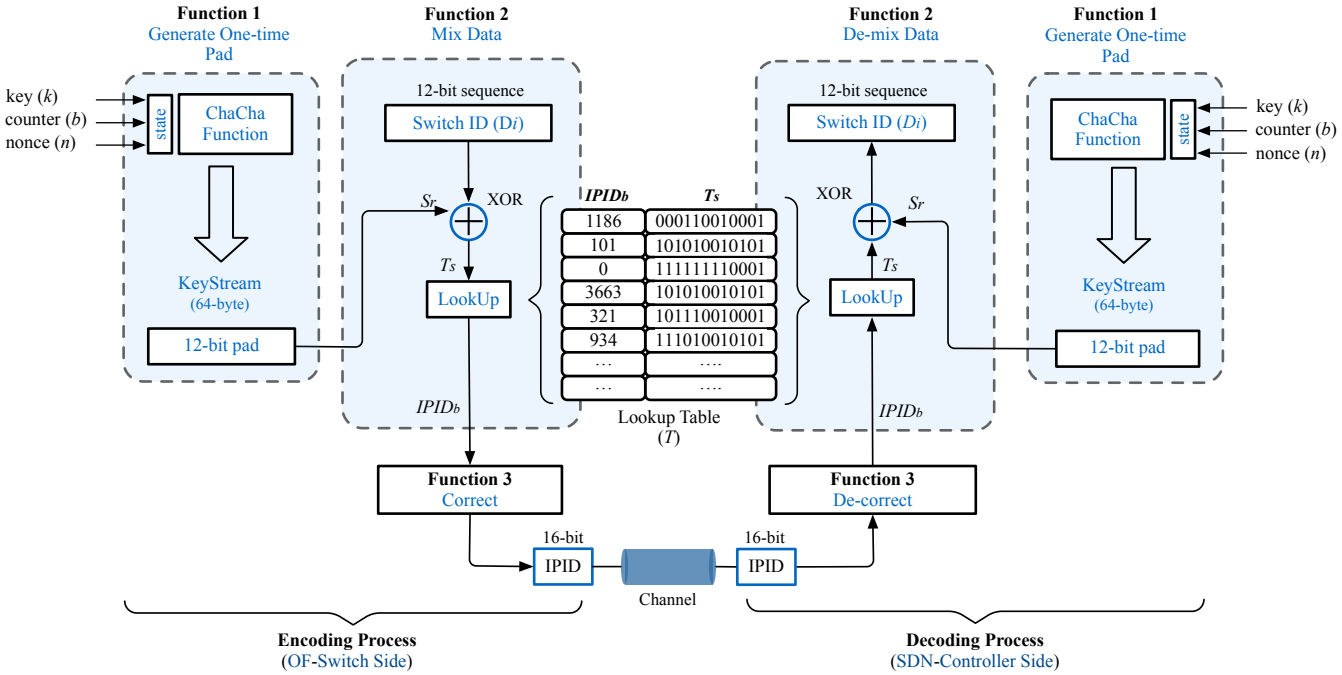


Fig. 6: IPID-based HiAuth

Step 3: Obtain one-time pad from ChaCha key stream.
Step 4: XOR the D_i and the S_r obtained from *Step* 2 and *Step* 3.
Step 5: Map the obtained 12-bit sequence T_s to an $IPID_b$ value by using the lookup table.
Step 6: Append the remaining 4 bits to $IPID_b$ to obtain 16 IPID candidates.
Step 7: Utilize the frist-in-first-out (FIFO) queue to store the used IPID value.
Step 8: Select an IPID value randomly from the 16 IPID candidates
Step 9: Check if the selected IPID is recently used. That is, the selected IPID is used if it does not exist in the queue; otherwise, repeat *Step* 8.
Step 10: Pop out the first IPID value if the queue size exceeds 2^{12}.
Step 11: Repeat *Step* 3 to *Step* 10 for every packet-In message transmission.

The decoding process verifies the identity of a device by reversing the encoding process. Fig. 6 illustrates the IPID-based HiAuth encoding and decoding processes.

4) IP Fragmentation Handling: All the existing works on IPID information hiding are only operational when the MTU is known. They also assume that the MTU will not change along the communication path. Nevertheless, this assumption may not be valid since the communication path may change for the same session. Here, we suggest two solutions to mitigate the effect of IP fragmentation on IPID-based HiAuth.

- *Fragmentation header monitoring* - IP header contains fragmentation related fields including 3-bit flag and 13-bit fragment offset. These fields can be collaboratively utilized to detect IP fragmentation and perform packet reassembly. In particular, the more fragment flag can indicate the existence of fragments whereas fragment offset indicates the part of the original packet which is carried in the fragment.
- *Application-level solution* - IPID-based HiAuth encoding functions yield randomness in the generated IPID values. In particular, OPG produces a stream of pseudo random pads to scramble the device ID in DM. Eventually, DCM ensures a sufficiently long period before repeating IPID values in consecutive packets. IPID-based HiAuth can include a mechanism to handle duplication as part of its implementation.

B. XID-based HiAuth

While IPID-based can provide packet authentication at the network layer, the following scenarios might render the authentication obsolete:

- *IPv4 specification update*: in the case if the IPv4 specification gets updated to more closely match IPv6 [61]. In IPv6, IPID field will only be available in fragmented packets [68].
- *Transition to IPv6*: in the case if IPv4 becomes obsolete when a full transition to IPv6 occurs. In fact, the transition from IPv4 to IPv6 is very challenging and will take very long especially with techniques such as network address translation and classless inter-domain routing.

In such scenarios, XID-based HiAuth can provide the same level of authentication. To the best of our knowledge, XID-based HiAuth is the first work that exploits XID for two purposes, facilitating message pairing and providing side channel communication.

XID is a 32-bit OpenFlow protocol header field which stands for transaction identifier. The main purpose of XID is to match synchronous control messages. That is, a synchronous

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management

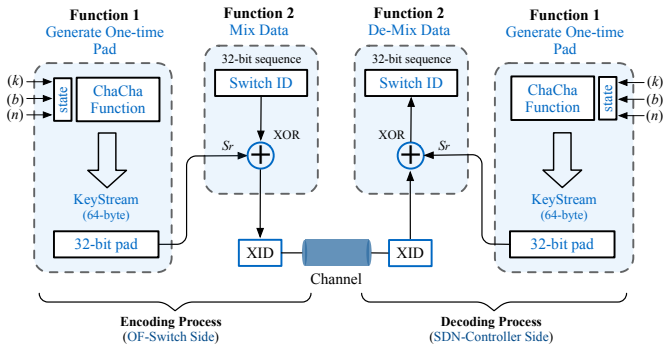


Fig. 7: XID-based HiAuth

reply message, such as packet-Out, will contain identical XID value as the one used in packet-In message. Hence, a switch requesting for routing information from the controller can differentiate multiple replies and match each reply to its corresponding request. In OpenFlow, the generation mechanism for XID is not described. Thus, the controller vendors can freely implement its generation algorithm as long as it serves its designed purpose. For instance, Ryu SDN controller utilizes a Python module, which implements pseudo-random number generator (i.e., random.randint), to generate XID values. Because XID is generated with a uniform distribution, XID-based HiAuth does not include any corrective mechanism to alter the generated distribution. Therefore, the encoding and decoding processes consist of the two functions namely OPG and DM.

Similarly, OPG exploits ChaCha to provide key stream for XID generation. In contrast to IPID-based HiAuth, XID-based HiAuth extracts 32-bit pad S_r from the key stream to be used in the data mixing function. A 32-bit device ID D_i is XORed with S_r in the DM function. The output of DM is then used as an XID value for the control channel packet. Here, the lookup table is removed to keep the computation and storage cost as low as possible. In fact, the absence of the lookup table in data mixing function does not weaken the security of this HiAuth variant since the length of pad is long enough. Fig. 7 shows the encoding and the decoding processes of the XID-based HiAuth.

VI. Experimental Results

In this section, we evaluate the undetectability and the performance of proposed schemes. In particular, the undetectability analysis shows the ability of HiAuth to withstand anomaly detection. Furthermore, the performance analysis includes overhead evaluation and the ability of HiAuth to drop attack packets. The IPID and XID values will be evaluated against those generated by a UNIX-based operating system and Ryu SDN controller, respectively. Table II shows the settings used in our experiments.

A. IPID-based HiAuth Undetectability

The IPv4 IPID field is 16-bit in length and can be assigned any value between 0 and $2^{16} - 1$ in decimal representation. This range of values is sufficient to uniquely identify packets in a communication session for a reasonable period of time. In

TABLE II: Experiments settings

Setting type	Parameter	Value
Common settings	key size	256-bit
	nonce size	64-bit
	counter size	64-bit
	queue & table size	4096[1]
	correction bits	4-bit[1]
	No. of rounds	20
Undetectability analysis	sample size	65536
	IPID value	$[0, (2^{16} - 1)]$
	XID value	$[0, (2^{32} - 1)]$
	Histogram mode	count[1]/ probability[2]
Overhead analysis	TLS version	1.0
	TLS record layer	TLSv1.2
	encryption	3DES
	authentication	SHA-1
	key exchange	RSA
	CPU	Intel Core i7
DoS mitigation	Ryu	v4.27
	Mininet version	2.2.1
	attack rate	0 - 300 (packet_in/s)
	http request rate	500 (request/s)
	test duration	60s

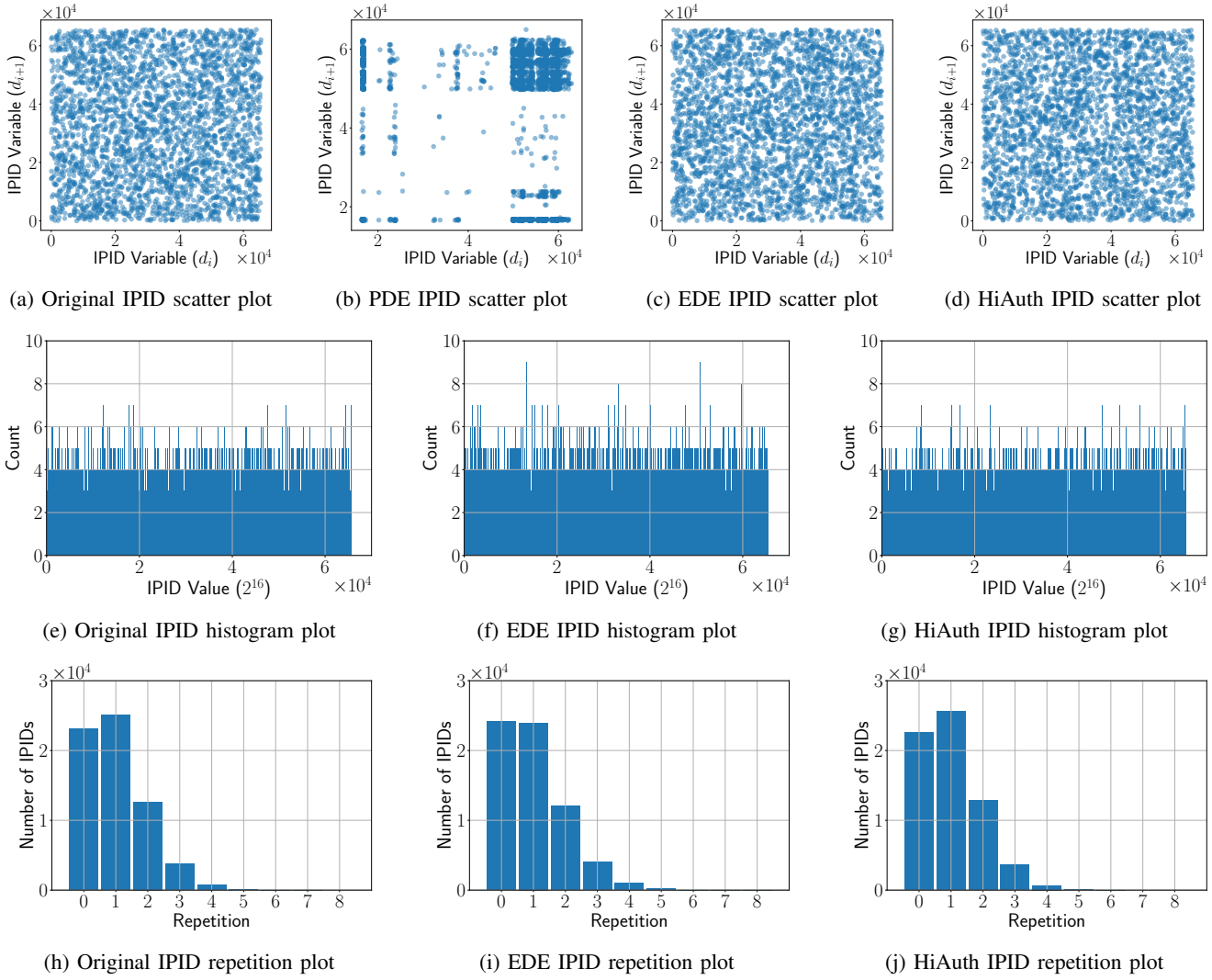
[1] IPID-based HiAuth specific [2] XID-based HiAuth specific

general, information hiding schemes are evaluated in terms of capacity, robustness and undetectability. For channel capacity, the IPID-based HiAuth can offer a maximum of 12 bits per packet. This translates into 4, 096, and is the number of devices that a network administrator can identify within the network when using IPID-based HiAuth. For being robust to alteration, HiAuth relies on the services provided by the upper-layer protocols for error checking and reliable transmissions, such as TCP in the case of OpenFlow protocol.

For undetectability, we compare four implementations, namely a) original UNIX-based platform IPID, Darwin 16.3.0, b) PDE IPID, c) EDE IPID, and d) HiAuth IPID. Visual benchmarking methods, including scatter, histogram, and repetition plots, are utilized in our evaluation. Here, each resulting IPID sample is compared to the original sample. To reproduce similar PDE results as in [29], English sentences in their ASCII binary representation are directly embedded into the IPID field. Since it is assumed that the IPID value is always random in [30]–[33], we simulate their results by using a pseudo random generator to produce uniform distribution.

Figs. 8(a)-(d) illustrate the original, PDE, EDE and HiAuth IPID scatter plots, respectively. Here, the successive pairs of the IPID values are plotted against each other. Let d and i denote the IPID and its sequence number, respectively. The x-axis represents an IPID value, while the y-axis represents the successive IPID value in the captured sequence. That is, the points $(x, y) = (d_i, d_{i+1})$ are plotted without connecting such as $(d_1, d_2), (d_3, d_4), (d_5, d_6)$ and so on. As demonstrated in Fig. 8(b), it is almost effortless to identify patterns in the case of PDE IPID which makes such methods easily detectable. Note that the randomness of PDE follows the frequencies of alphabets occurrence in English sentences (i.e., the 'e' highest and 'z' the lowest). On the contrary, the remaining samples are evenly distributed and display near zero correlation with no obvious patterns. This makes it difficult to distinguish the ordinary IPID traffic from the IPID values generated by the other information hiding methods.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management



(a) Original IPID scatter plot (b) PDE IPID scatter plot (c) EDE IPID scatter plot (d) HiAuth IPID scatter plot

(e) Original IPID histogram plot (f) EDE IPID histogram plot (g) HiAuth IPID histogram plot

(h) Original IPID repetition plot (i) EDE IPID repetition plot (j) HiAuth IPID repetition plot

Fig. 8: Undetectability analysis

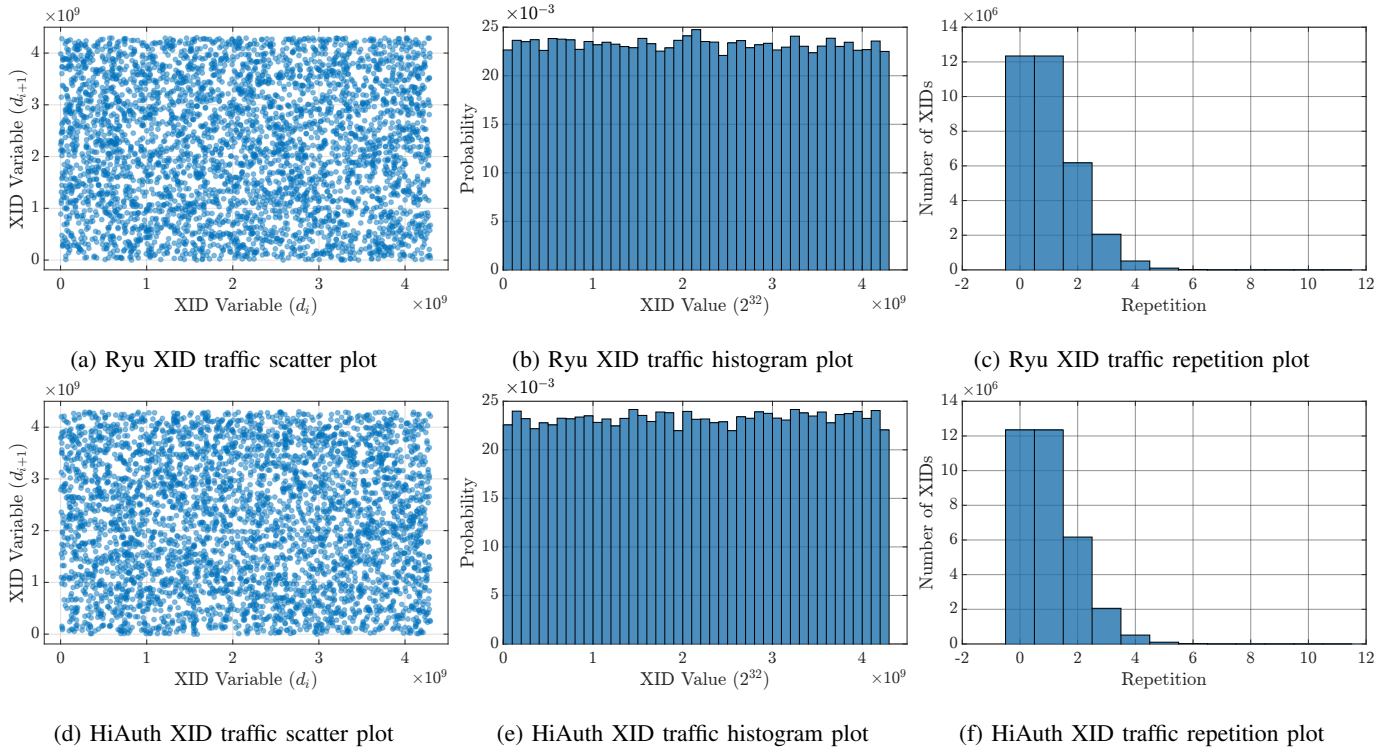
Next, we utilize histogram plots to analyze the uniformity of the IPID distributions. Figs. 8(e)-(g) show the histograms of the original, EDE and HiAuth IPID values. The x-axis represents the IPID values, while the y-axis represents the frequency of the respective IPID value. Here, the sample size is equal to 2^{16}, for which each IPID value occurs fairly at least once. Also, the histogram bin size is chosen to be 2^{16} to represent the count of each value in the sample.

When the original IPID is compared to the EDE IPID (see Figs. 8(e)-(f)), the frequencies of the EDE IPID traffic differ from those of the original IPID traffic. For instance, the highest count of an IPID in the original and HiAuth IPID samples is seven, while the highest count of the EDE IPID values is nine. Also, the number of the IPID values with count six is much higher than what can be seen in Fig. 8(e). In contrast, HiAuth IPID histogram closely resembles the original IPID histogram as suggested by the comparison in Figs. 8(e) and 8(g).

To distinctly observe the differences between original IPID and EDE IPID, we evaluate the number of the incidences that an IPID has occurred in a sample, which is called repetition in this paper. Figs. 8(h)-(j) depict the repetition plots for the implementations in comparison. The x-axis represents the number of repetitions, while the y-axis represents the number of IPID values associated with the respective repetition. Note that in the case of Fig. 8(i), the number of IPID values that has not occurred in the sample ($x = 0$) is much larger than that in Figs. 8(h) and 8(j). Thus, histogram and repetition plots expose the distribution differences between the EDE and original IPID traffic. In contrast, HiAuth IPID values demonstrates similar uniformity and repetition trends to the original IPID values. Therefore, visual benchmarking suggests that HiAuth outperforms the existing methods in terms of undetectability.

Clearly, the conventional IPID-based methods neglect the purpose of the IPID. They fail to ensure a sufficiently long period before repeating an IPID, and merely assume that the IPID is always random. Because of this, it is fairly easy to detect these methods if they are utilized in counter-based IPID platform (e.g., Windows and Linux). Nonetheless, even if they are utilized on platforms which manifest randomness, repeating IPID values too soon can easily render the hidden information compromised.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management



(a) Ryu XID traffic scatter plot (b) Ryu XID traffic histogram plot (c) Ryu XID traffic repetition plot

(d) HiAuth XID traffic scatter plot (e) HiAuth XID traffic histogram plot (f) HiAuth XID traffic repetition plot

Fig. 9: Undetectability Analysis of XID-based HiAuth.

B. XID-based HiAuth Undetectability

The XID field of OpenFlow protocol is 32-bit long and can be any value between 0 and $2^{32} - 1$. This range of values is utilized as unique transaction identifier sto match request messages to response messages. Similar to IPID, OpenFlow does not describe how the XID value should be generated. Because of that, XID value is randomly generated by SDN vendors. Accordingly, HiAuth can utilize XID to provide 32 bits of authentication channel capacity. This capacity enables network administrators to identify 4.3×10^9 devices when using XID-based HiAuth. The disadvantage of using XID for authentication is one-sided entity verification. In other words, a single XID-based HiAuth mechanism can be used to authenticate a switch to a controller, but not in the opposite direction.

In our experiments, we evaluate HiAuth XID traffic against the XID traffic generated by Ryu controller. Similar to the IPID-based HiAuth, XID-based HiAuth is benchmarked by using scatter plot for randomness and histogram as well as repetition plots for distribution uniformity. Fig. 9 shows the undetectability analysis of Ryu and HiAuth XID traffic. The scatter plots in Figs. 9(a) and 9(d) demonstrate the randomness of Ryu and HiAuth XID traffic, respectively. Successive pairs of the XID values are plotted against each other. It is observed that both results show randomness with near zero correlation. Furthermore, Figs. 9(b) and (e) illustrate the histograms of Ryu and HiAuth XID traffic, respectively. The *x*-axes represent the value of the XID and the *y*-axes represent the probability of the respective range of XID values. Both histograms show fairly uniform distributions although there are some statistical fluctuations due to the randomness. Finally, Figs. 9(c) and (f) depict the repetition plots. From the figure, one can see that HiAuth XID traffic resembles the randomness and the statistical distribution of Ryu XID traffic.

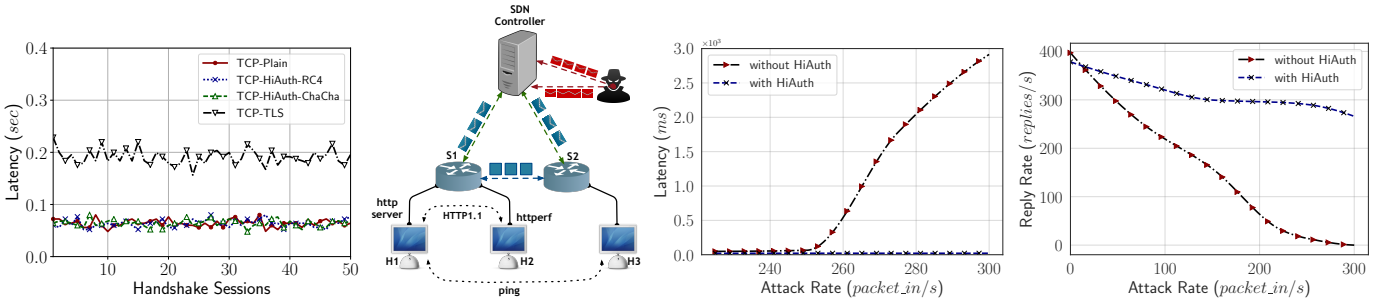
C. Performance Analysis

The latency of TCP handshake process with different implementation is analyzed to evaluate the imposed overhead. For this experiment, we implement a client-server software for plain TCP, TCP with HiAuth (both ChaCha and RC4) and TCP with TLS (see Fig. 10(a)). The evaluation is carried out on physical hardware instead of virtual machines to avoid emulation overhead. Table II shows the details of the software and hardware used for this experiment. Here, the *x*-axis represents the number of handshake samples while the *y*-axis shows the respective latency in seconds. The results suggest that HiAuth is lightweight and does not introduce obvious overhead.

The ultimate goal of HiAuth is to drop unauthorized control packets. To establish this, we implement IPID-based HiAuth proof-of-concept which runs on forwarding devices in Mininet and Ryu SDN controller. The experimental setup consists of an SDN controller, DoS attacker, two OpenFlow switches and three hosts connected as showing in Fig. 10(b). To avoid causing DoS due to bandwidth saturation, each one of the switches as well as the attacker has a separate control channel that connects to the controller. This way, the evaluation will only reflect the impact on the controller resources.

Provided that the controller and the switches are synchronized (in terms of key stream generation), legitimate messages will be allowed and attack messages will be dropped. We

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2909116, IEEE Transactions on Network and Service Management



(a) Overhead (b) Expt. setup for (c)-(d) (c) Host reachability (d) Web server performance

Fig. 10: Performance analysis

demonstrate HiAuth ability to mitigate outsider DoS attack by two sets of experiments, namely hosts reachability using ping test and web server reply rate using httperf tool. It is important to note that, adding flow rules to the switches is disabled to stimulate legitimate mismatch events for performance measurement. On one hand, a ping test between H1 and H3 is performed to measure the latency during a DoS attack with and without HiAuth, as shown in Fig. 10(c). The results shows that with HiAuth, host reachability is maintained, while the latency significantly increases as the controller gets pre-occupied processing attack packets in the absence of HiAuth. On the other hand, the reply rate of a web server running on H1 is benchmarked by running httperf tool on H2. The client of httperf sends 500 HTTP/1.1 request/s for 60 second to the web server and reports the reply rate as a performance metric. In Fig. 10(d), the x-axis represents the attack rate in packet_in per second while the y-axis indicates the web server respective replies per second. The results demonstrate that without HiAuth, the controller fails to process legitimate packet_in messages which renders the web server reply rate to zero.

VII. Discussions

A. Security

Unlike cryptography which protects data by scrambling, information hiding protects data by concealing its very existence. In general, the use of information hiding techniques is not publicly shared. The security of the hidden information is based on the ability to stay undetected. HiAuth exploits this opportunity to conceal control packet authentication. Although the k and n in the OPG and the D_i in the data mixing function are the same throughout the authentication process, the combination of HiAuth three components will always ensure to produce values, which comply with the original IPID and XID values.

B. Usability and computational complexity

One of HiAuth primary advantages is its adoption simplicity. As described in Section III, SDN vendors may not support TLS because of its complexity and the usability challenges. On the contrary, HiAuth configuration is straightforward. To configure HiAuth, network administrators will only need to set few parameters, namely k, b, n and D_i. Additionally, HiAuth is completely transparent to application layer. It requires replacing the generator for the IPID or XID (IP layer or TCP layer) values depending on the scheme used.

In terms of computational complexity, HiAuth only perform arithmetic operations on the fix-sized data. The XOR operation scrambles the unique IDs and creates a randomized table entry. The lookup table, in the case of IPID-based HiAuth, involves performing an indexed search through an array of size 2^{12}. Also, the one-time pads can be pre-computed before receiving packets, resulting in a speedy identity verification. As compared to TLS, HiAuth is lightweight because it does not require dedicated hardware for its computations.

C. Limitations

In Section III, we presented four DoS attack scenarios in which outsider class, in particular scenarios (a) and (b), can be easily mitigated by HiAuth. However, neither HiAuth nor TLS can prevent scenarios (c) and (d) since an attacker can utilize an authenticated switch to launch the attack. This is a limitation of HiAuth which we leave for future extension.

D. Suggested Security Enhancement

To improve the security of HiAuth, we recommend the following actions:

- *Frequent table permutation*: Table permutation can be performed after a certain number of packet transmissions. This will increase the resistivity against the brute force attacks.
- *Limited number of authentication attempts*: It is also possible to limit the number of authentication attempts within a specified period of time to ensure that guess attempts will take long time.
- *Blacklisting*: Blacklisting approach can be added to prevent brute-force attack to obtain the secret key.
- *Platform utilization*: Utilize the proposed mechanism in operating systems and controllers that randomly generate IPID and XID values to avoid being detected by attackers.

VIII. Conclusions and Future Work

Because of SDN centralization nature, DoS attacks become a crucial security issue. Unfortunately, there has been a widespread failure to adopt TLS in SDN control channel due to its configuration complexity and lack of standard enforcement. In this paper, we present a lightweight authentication solution,

called Hidden Authentication (HiAuth), to protect the SDN controller against DoS by hiding the identities of the forwarding devices into the control packets via efficient bitwise operations. HiAuth is the first to incorporate information hiding techniques into OpenFlow to provide security against DoS attacks. The experimental results show that HiAuth can effectively mitigate outsider DoS attacks and provide high undetectability to attackers. Some interesting research topics can be extended from this work, such as using information hiding to mitigate distributed DoS and insider DoS attacks.

REFERENCES

[1] Z. Cao, S. S. Panwar, M. Kodialam, and T. Lakshman, "Enhancing mobile networks with software defined networking and cloud computing," *IEEE/ACM Transactions on Networking*, 2017.

[2] M. Huang, W. Liang, Z. Xu, and S. Guo, "Efficient algorithms for throughput maximization in software-defined networks with consolidated middleboxes," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 631–645, 2017.

[3] J. Chase, R. Kaewpuang, W. Yonggang, and D. Niyato, "Joint virtual machine and bandwidth allocation in software defined network (SDN) and cloud computing environments," in *IEEE International Conference Communications (ICC)*, pp. 2969–2974, 2014.

[4] Y.-J. Chen, L.-C. Wang, F.-Y. Lin, and B.-S. P. Lin, "Deterministic quality of service guarantee for dynamic service chaining in software defined networking," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 991–1002, 2017.

[5] Y.-J. Chen, F.-Y. Lin, L.-C. Wang, and B.-S. Lin, "A dynamic security traversal mechanism for providing deterministic delay guarantee in SDN," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pp. 1–6, IEEE, 2014.

[6] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *IEEE 35th Conference on Local Computer Networks (LCN)*, 2010.

[7] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[8] K. Benton, L. J. Camp, and C. Small, "OpenFlow vulnerability assessment," in *Proceedings of the second ACM workshop on Hot topics in software defined networking*, 2013.

[9] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *IEEE Communications Magazine*, vol. 44, no. 3, pp. 134–141, 2006.

[10] S. J. Vaughan-Nichols, "OpenFlow: The next generation of the network?," *IEEE Computer*, vol. 44, no. 8, pp. 13–15, 2011.

[11] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in software defined networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2317–2346, 2015.

[12] Opendaylight, "Defense4all: SDN application for detecting and driving mitigation of DoS and DDoS attacks." OpenDaylight technical workstream, 2013.

[13] Y.-J. Chen, L.-C. Wang, and C.-H. Liao, "Eavesdropping prevention for network coding encrypted cloud storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2261–2273, 2016.

[14] W. Mazurczyk and L. Caviglione, "Information hiding as a challenge for malware detection," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89 – 93, 2015.

[15] W. Mazurczyk and L. Caviglione, "Steganography in modern smartphones and mitigation techniques," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 334–357, 2015.

[16] J.-M. Guo, G.-H. Lai, K. Wong, and L.-C. Chang, "Progressive halftone watermarking using multilayer table lookup strategy," *IEEE Transactions on Image Processing*, vol. 24, no. 7, pp. 2009–2024, 2015.

[17] B. Jankowski, W. Mazurczyk, and K. Szczypiorski, "Padsteg: Introducing inter-protocol steganography," *Telecommunication Systems*, vol. 52, no. 2, pp. 1101–1111, 2013.

[18] W. Mazurczyk, M. Karaś, and K. Szczypiorski, "Skyde: A skype-based steganographic method.," *International Journal of Computers, Communications & Control*, vol. 8, no. 3, 2013.

[19] O. I. Abdullaziz, V. T. Goh, H.-C. Ling, and K. Wong, "AIPISteg: An active IP identification based steganographic method," *Journal of Network and Computer Applications*, vol. 63, pp. 150–158, 2016.

[20] L. Caviglione, M. Podolski, W. Mazurczyk, and M. Ianigro, "Covert channels in personal cloud storage services: The case of Dropbox," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1921–1931, 2017.

[21] N. F. Johnson, Z. Duric, S. Jajodia, and N. Memon, "Information hiding: Steganography and watermarking - attacks and countermeasures," *Journal of Electronic Imaging*, vol. 10, no. 3, pp. 825–826, 2001.

[22] O. I. Abdullaziz, V. T. Goh, H.-C. Ling, and K. Wong, "Network packet payload parity based steganography," in *IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET)*, 2013.

[23] L. Ji, W. Jiang, B. Dai, and X. Niu, "A novel covert channel based on length of messages," in *IEEE International Symposium on Information Engineering and Electronic Commerce*, 2009.

[24] S. Zander, G. J. Armitage, and P. Branch, "An empirical evaluation of IP time to live covert channels," in *15th IEEE International Conference on Networks*, 2007.

[25] O. Arkin and J. Anderson, "Etherleak: Ethernet frame padding information leakage," 2003.

[26] W. Mazurczyk, M. Smolarczyk, and K. Szczypiorski, "Retransmission steganography and its detection," *Soft Computing*, vol. 15, no. 3, pp. 505–515, 2011.

[27] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *ACM Proceedings of the 11th Conference on Computer and Communications Security*, 2004.

[28] J. Postel, "Internet protocol." RFC791, 1981.

[29] C. H. Rowland, "Covert channel in the TCP/IP protocol suite," *First Monday: Peer-reviewed Journal on the Internet*, vol. 2, no. 5, 1997.

[30] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Workshop on Multimedia Security at ACM Multimedia*, 2002.

[31] B. Xu, J.-Z. Wang, and D.-Y. Peng, "Practical protocol steganography: Hiding data in IP header," in *IEEE Asia International Conference on Modelling & Simulation*, 2007.

[32] D. Dhobale, V. Ghorpade, B. Patil, and S. Patil, "Steganography by hiding data in TCP/IP headers," in *IEEE 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 2010.

[33] R. M. Goudar, S. J. Wagh, and M. D. Goudar, "Secure data transmission using steganography based data hiding in TCP/IP," in *ACM Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, 2011.

[34] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Springer Proceedings of the 7th International Conference on Information Hiding*, 2005.

[35] R. Hu, Y. Qian, H.-H. Chen, and H. Mouftah, "Cyber security for smart grid communications: Part I," *IEEE Communications Magazine*, vol. 50, no. 8, 2012.

[36] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl, "I have no idea what i'm doing - on the usability of deploying HTTPS," in *Proc. of the 26th USENIX Security Symposium*, vol. 17, pp. 1339–1356, 2017.

[37] S. Fahl, Y. Acar, H. Perl, and M. Smith, "Why eve and mallory (also) love webmasters: A study on the root causes of SSL misconfigurations," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 507–512, ACM, 2014.

[38] Floodlight-developers, "TLS support," 2018.

[39] S. Qualys, "SSL server test," 2014.

[40] D. Akhawe and A. P. Felt, "Alice in warningland: A large-scale field study of browser security warning effectiveness.," in *USENIX security symposium*, vol. 13, 2013.

[41] M. Harbach, S. Fahl, P. Yakovleva, and M. Smith, "Sorry, i don't get it: An analysis of warning message texts," in *International Conference on Financial Cryptography and Data Security*, pp. 94–111, Springer, 2013.

[42] M. Raya and J.-P. Hubaux, "Securing vehicular ad hoc networks," *Journal of Computer Security*, vol. 15, no. 1, pp. 39–68, 2007.

[43] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *ACM Proceedings of the second workshop on Hot topics in software defined networking*, 2013.

[44] W. Wang, J. McNair, and J. Xie, *Authentication and Security Protocols for Ubiquitous Wireless Networks*. Book Chapter of Ambient Intelligence, Wireless Networking, and Ubiquitous Computing, Athanasios Vasilakos and Witold Pedrycz, Eds. Artech House, 2006.

[45] J. M. Dover, "A denial of service attack against the Open Floodlight SDN controller." Research report, 2013.

[46] O. I. Abdullaziz, Y.-J. Chen, and L.-C. Wang, "Lightweight authentication mechanism for software defined network using information hiding," in *IEEE Global Communications Conference (GLOBECOM)*, 2016.

[47] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, 2016.

[48] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *ACM Conference on Computer & communications security*, 2013.

[49] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Tackling control plane saturation attacks in software-defined networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1206–1219, 2017.

[50] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *IEEE/IFIP 45th International Conference on Dependable Systems and Networks (DSN)*, 2015.

[51] M. Kuerban, Y. Tian, Q. Yang, Y. Jia, B. Huebert, and D. Poss, "FlowSec: Dos attack mitigation strategy on sdn controller," in *IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2016.

[52] S. Lim, S. Yang, Y. Kim, S. Yang, and H. Kim, "Controller scheduling for continued SDN operation under DDoS attacks," *Electronics Letters*, vol. 51, no. 16, pp. 1259–1261, 2015.

[53] R. Mohammadi, R. Javidan, and M. Conti, "SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 487–497, 2017.

[54] T. Wang and H. Chen, "SGuard: A lightweight SDN safe-guard architecture for DoS attacks," *China Communications*, vol. 14, no. 6, pp. 113–125, 2017.

[55] L. Wei and C. Fung, "FlowRanger: A request prioritizing algorithm for controller DoS attacks in software defined networks," in *IEEE International Conference on Communications (ICC)*, pp. 5254–5259, 2015.

[56] G. Yao, J. Bi, and L. Guo, "On the cascading failures of multi-controllers in software defined networks," in *IEEE 21st International Conference on Network Protocols (ICNP)*, 2013.

[57] D. Arrowsmith and C. M. Place, *An Introduction to Dynamical Systems*. Cambridge University Press, 1990.

[58] J. Mogul and S. Deering, "Path MTU discovery." RFC 1191, 1990.

[59] A. Householder, A. Manion, L. Pesante, and G. M. Weaver, "Managing the threat of denial-of-service attacks," *CMU Software Engineering Institute CERT Coordination Center*, 2001.

[60] M. Mathis and J. Heffner, "Packetization layer path MTU discovery." RFC 4821, 2007.

[61] J. Touch, "Updated specification of the IPv4 ID field." RFC6864, 2013.

[62] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC*, vol. 8, pp. 3–5, 2008.

[63] E. Bursztein, "Speeding up and strengthening HTTPS connections for chrome on android," 2018.

[64] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," tech. rep., 2018.

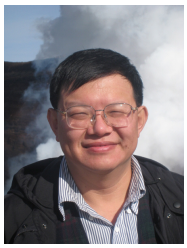
[65] D. J. Bernstein, "Cache-timing attacks on AES," 2005.

[66] W. Diffie, P. C. Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[67] Apple Inc., "IP ID generation is a fascinating topic," 2017.

[68] R. Hinden, "Internet protocol version 6 (IPv6) specification," 2017.

Osamah Ibrahiem Abdullaziz received the B.S. and M.Eng.Sc. degrees from Multimedia University, Malaysia in 2011 and 2015, respectively. He is currently a Ph.D. candidate at the department of Electrical Engineering and Computer Science, National Chiao Tung University, Taiwan. His current research interests include software defined networks, multi-access edge computing, virtualization and network information hiding.



Li-Chun Wang (M'96 – SM'06 – F'11 received the B.S. degree from National Chiao Tung University, Taiwan, R.O.C. in 1986, the M.S. degree from National Taiwan University in 1988, and the Ms. Sci. and Ph. D. degrees from the Georgia Institute of Technology , Atlanta, in 1995, and 1996, respectively, all in electrical engineering.

From 1990 to 1992, he was with the Telecommunications Laboratories of Chunghwa Telecom Co. In 1995, he was affiliated with Bell Northern Research of Northern Telecom, Inc., Richardson, TX. From 1996 to 2000, he was with AT&T Laboratories, where he was a Senior Technical Staff Member in the Wireless Communications Research Department. Since August 2000, he has joined the Department of Electrical and Computer Engineering of National Chiao Tung University in Taiwan and is the current Chairman of the same department. His current research interests are in the areas of radio resource management and cross-layer optimization techniques for wireless systems, heterogeneous wireless network design, and cloud computing for mobile applications.

Dr. Wang won the Distinguished Research Award of National Science Council, Taiwan in 2012, and was elected to the IEEE Fellow grade in 2011 for his contributions to cellular architectures and radio resource management in wireless networks. He was a co-recipient(with Gordon L. Stuber and Chin-Tau Lea) of the 1997 IEEE Jack Neubauer Best Paper Award for his paper "Architecture Design, Frequency Planning, and Performance Analysis for a Microcell/Macrocell Overlaying System," IEEE Transactions on Vehicular Technology, vol. 46, no. 4, pp. 836-848, 1997. He has published over 200 journal and international conference papers. He served as an Associate Editor for the IEEE Trans. on Wireless Communications from 2001 to 2005, the Guest Editor of Special Issue on "Mobile Computing and Networking" for IEEE Journal on Selected Areas in Communications in 2005, "Radio Resource Management and Protocol Engineering in Future Broadband Networks" for IEEE Wireless Communications Magazine in 2006, and "Networking Challenges in Cloud Computing Systems and Applications," for IEEE Journal on Selected Areas in Communications in 2013, respectively. He is holding 10 US patents.

Yu-Jia Chen received the B.S. degree and Ph.D. degree in electrical engineering from National Chiao Tung University, Taiwan, in 2010 and 2015, respectively. He is currently an assistant professor at the department of communication engineering in National Central University. His research interests include Internet of Things (IoT), wireless sensing, and network security. Dr. Chen has published more than 30 articles in peer-reviewed journal and conference papers. He is holding three US patent and three ROC patent.