Improved Handshaking Procedures for Transport Layer Security in Software Defined Networks

Xue Jun Li
Department of EEE
Auckland University of Technology
Auckland, New Zealand
xuejun.li@aut.ac.nz

Maode Ma
College of Engineering
Qatar University
Qatar
mamaode@qu.edu.qa

Cho Wai Hlaing
School of EEE
Nanyang Technological University
Singapore
ch0097ng@e.ntu.edu.sg

***Abstract*— Software defined networking (SDN) has emerged as a new technology to enhance the flexibility, resilience, and automated centralized management of a network. Recently several reports have identified possible vulnerabilities, which may affect its authenticity, availability, confidentiality and integrity. This paper analyzes several types of security issues in SDNs, especially on how to secure the communication between the control plane and the data plane. The state-of-the-art security protocol TLS in SDNs has been verified using the Scyther Tool. Two security schemes, namely TLSHPS and TLSIHP are proposed to improve the handshaking procedures of the TLS. Security analysis with the Scyther tool shows that both proposed schemes work well to prevent various cyber attacks.**

Keywords—Internet of Things, software-defined networks, security, handshaking procedure, transport layer security.

I. Introduction

The Internet of Things (IoT) augments the application of the Internet to connect physical objects, which are embedded with microcontrollers, sensors, and actuators to realise smart applications such as intelligent transportation systems (ITS), smart cities. The growing complexity of IoT and its ever-increasing data traffic pose a significant challenge to traditional network management on how to respond to malicious events securely and timely.

As we know, traditional network management is complex and dedicated to specific services. Every major network equipment vendor has a unique network infrastructure, firmware and other related software, which are only compatible with its hardware. This unfortunately hinders the innovation progress of network technologies, which leads to higher operation and management costs than equipment costs, especially when additional hardware is added to a current network due to service expansion [1].

To address the bottleneck of traditional network management, software defined networks (SDNs) have been proposed. SDNs offer promising opportunities for network management in terms of flexibility, simplicity and programmability. SDNs can reduce the operating cost including maintenance, error handling. SDN integrates the control plane of multiple devices into one instance to simplify the network management, offering robustness against outage. Consequently, SDNs oversee the entire network and can speed up the service delivery by providing both virtual and physical network devices from a central location. Furthermore, SDNs allow network engineers and administrators to configure the network without affecting the existing users. Finally, physical and virtual switches and network devices can be managed from a central controller.

In this paper, we focus on the security enhancement for software defined mobile networks (SDWNs), which have three layers: data plane, controller layer and application layer. As the brain of an SDN, the controller oversees the whole network. However, the controller deployments, protocols and software are new, thus the history of attacks to SDNs is largely unknown. Therefore, it is critical to consider security threats carefully before adopting an SDN. For example, an SDN physically separates the control and data planes of network devices. As such, various attacks can affect the security of SDNs [2], which includes their confidentiality, integrity, authenticity and availability.

Currently transport layer security (TLS) is used with data encryption protocol within the Openflow to secure the communication link between the control plane and the data plane in SDNs. However, the improper configurations of TLS could make an SDN vulnerable. Authentication and key exchange are important to TLS protocol security. The vulnerability of TLS can be verified using the Scyther tool. This motivates us to study those vulnerable points and identify the potential attack scenarios that could compromise the security of an SDN with TLS protocol. All attainable potential attacks are identified for each layer, then methods to prevent these attacks are described. We propose two security schemes to improve handshaking procedures and enhance the TLS protocol.

The rest of this paper is organised as follows. Section II briefly revisits SDNs and their security issues, followed by discussions on different types of cyberattacks in SDNs. Section III presents the proposed security schemes with improved handshaking procedures for TLS, followed by the security verification results in Section IV. Section V concludes the paper.

II. Related Work

A. Software Defined Networking

The architecture of SDNs comprises three layers: the application plane, the control plane (known as the brain of an SDN) and the data plane (infrastructure layer) where all the network devices reside in [3]. As shown in Fig. 1, network forwarding devices are residing in the data plane, where network virtualization is implemented through the control layer. The control layer supervises the network forwarding behavior by providing the consolidated control functionality through open application programming interfaces (APIs) [4]. It resides on a server. Network policies and flow of traffic throughout the network are managed by the controller plane [3]. Finally, the application layer comprises the end user's application that uses the SDN communications and services [5], which may include network configuration, monitoring and management by leveraging the network information [2].

The northbound interface connects the control layer and the application layer. It would be through RESTful APIs of SDN controllers [6]. The southbound interface offers communication between the infrastructure layer and the controller layer. It would use southbound protocols like Openflow, etc.

B. *Network Security*

Four major security objectives are discussed as follows.

Confidentiality is protecting the information from disclosure to unauthorized parties, which is ensured by encryption and access control. Currently, TLS protocol is used to encrypt the communication between the controller and the data plane devices. Access control involves granting access from management interfaces of the network devices after authentication is successful.

Authenticity is the assurance that communicating entity is the one claimed. A digital signature is the usual cryptographic method to provide authenticity. For the network devices, they exchange keys to provide authenticity [2]. For SDNs, TLS exchanges digital certificates during the TLS handshake process to check the authentication. Digital certificates provide protection against impersonation attacks and ensure authenticity by verifying the digital signature, certificate chain, activation and expiry date and the revocation date [7].

Integrity is to protect the information from being modified by unauthorised parties. In SDNs, the southbound API could be a potential attack vector for malicious intruders to compromise the integrity [8]. For instance, if a malicious intruder places a device on transmission tunnel between the switch and a controller, or simply duplicates the traffic flow to that device, then an attacker could get the configuration setting, delete the original rule and create a new rule to modify the original data flow [9]. This type of attack is known as Man-in-the-Middle (MITM) attack. Therefore, the rule of network routing flow and limitation of messages transmitted between layers are needed to ensure data integrity. For example, Message Authentication Code (MAC) can be adopted for this purpose.

Availability is to guarantee reliable access that authorized people can access the information when it is required. The controller is the most important part of SDN availability. If one of the data plane devices is down due to denial of service (DOS) attack or configuration error or hardware breakdowns, the controller can rapidly redirect the flow of network paths. However, if the controller is being attacked or unavailable due to configure issues, the network devices can only accomplish the predefined rules. To ensure the availability, configuration and technical error should be avoided.

C. *Attack Types and Their Solutions*

As illustrated in Fig. 2, at the data plane, attackers could launch DOS attacks to reduce SDN availability by gaining illegitimate access to the network devices. The DOS is an attack to suspend the availability of service and can affect the system, which can affect the network by crashing or flooding traffic. Traffic diversion attack can happen at the data plane, which can compromise network devices to alter the network route and allow attackers to eavesdrop on traffic flows. Additionally, an attacker may attempt a replay attack to spythe flows on the southbound communication to control the flows that are in use and the type of network traffic is

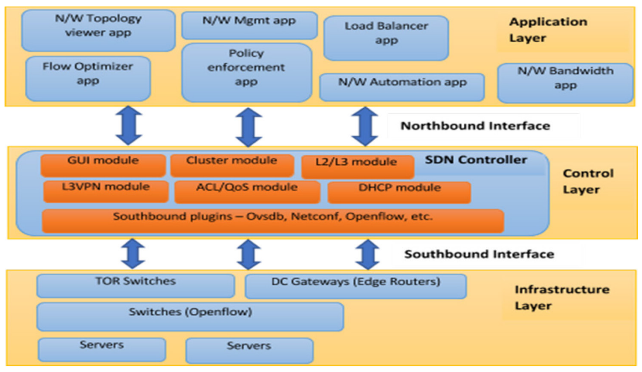


Fig. 1. Illustration of the SDN Architecture

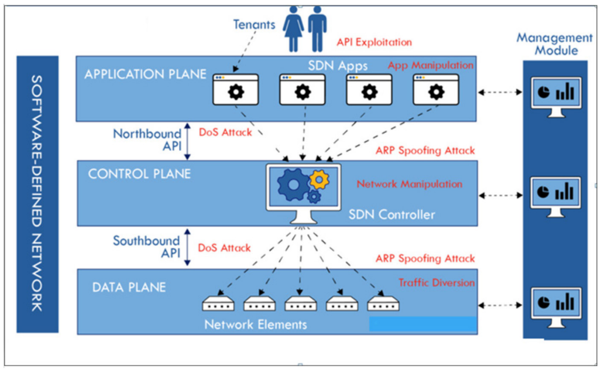


Fig. 2. Illustration of Security Threats on the SDNs

being allowed over the network. An MITM attack can also be launched by spoofing the northbound API messages or southbound messages. By a successful attack, the attacker can sniff, modify and even stop the network traffic.

Noteworthily, the controller layer is a primary target for attackers as it is the central point of failure that could affect the entire SDN. A resource exhaustion attack can happen on the controller, which requires receiving event updates and it can cause a delayed response in Packet_In and Packet_Out messages [10]. A network manipulation attack is another attack that occurs on the controller layer. A compromised controller can produce false data and open other vulnerabilities in the entire network.

At the application layer, an attacker can launch an app manipulation attack using SQL injection or cross-site scripting techniques. Application vulnerability could lead to disruption of service or a malfunction, or allowing the attackers to spy on data. An adversary can cause the overall disruption of SDN services [11].

In summary, at data plane, attackers can launch the traffic diversion and DOS attacks due to short of (1) authentication mechanisms to verify the authenticity of communication between data plane devices or network elements and controller and (2) unbounded flow state memory allocation where a malicious intruder takes advantage of the huge memory space required for the forwarding elements [12]. Therefore, to prevent spoofing of southbound transmission, most organizations use the TLS protocol to authenticate and encrypt the traffic [10]. The solution may differ according to the southbound protocol being used in SDNs. For instance, OpenFlow protocol uses the TLS session and some protocols use shared-secret key and nonce to prevent replay attacks.

To secure the control plane, there should be different access privileges for network engineers or administrators to prevent unauthorized access to the SDN controller. Regular audit and logging trails can be used to identify the illegitimate changes or access from an attacker. Using rate-limiting and packet dropping techniques can defend the DOS attack. For securing the application layer, keeping servers updated with the latest patches can prevent the kind of API exploitation attack. Using strong encryption techniques can prevent ARP spoofing attacks on the application layer or the control plane [13].

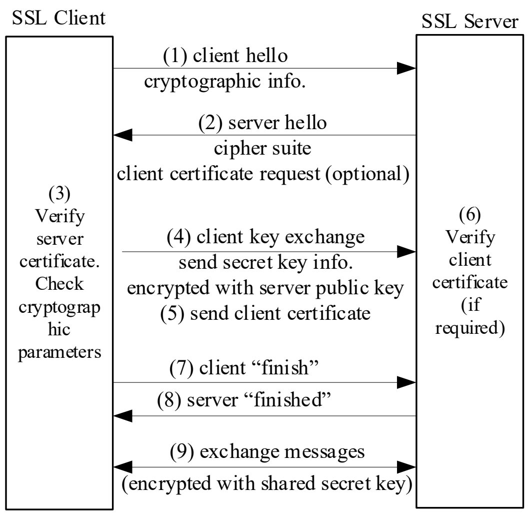
D. Drawbacks of TLS

The Open Flow protocol used in SDNs for southbound API between the controller and network devices is under the study [14]. It is important to tighten the security of the Open Flow protocol to keep confidentiality, integrity and prevent data spying [15].

TLS is an encryption protocol designed to provide reliable transmission security throughout the Internet and allow clients and servers to communicate confidentially. The TLS has two layers, namely record protocol and handshake protocol. Data link encryption and device authentication are completed at the record layer. The handshaking process is complicated to negotiate the secure attributes, such as session identifier, certificate and cipher specification. Fig. 3 shows the client-server handshake process. To secure TLS, a strong private key and valid certificate using the strong cipher suite are used for the authentication of the network devices.

Unfortunately, the TLS heavily relies on pre-master key (or shared secrete key). Thus improper configuration can lead to security risk [16] [17]. For instance, if an attacker could obtain the pre-master key and place the device between the switch and the controller, it will lead to an MITM attack as shown in Fig. 4 [9]. An unsecured key exchange can also lead to an MITM attack [3]. To overcome it, the controller needs to check all TLS handshake messages to verify the protocol version, cipher suites, certificate. Since the controller inspects all network traffic to enforces policies, it can detect and reroute the flow of the MITM attacks.

If a malicious hacker can obtain the pre-master key (which is shared between client and server and generated by either side) and interpret the TLS flow by taking the advantage of shared secrete key, then the integrity of transmission data becomes compromised.



SSL Client
SSL Server
(1) client hello
cryptographic info.
(2) server hello
cipher suite
client certificate request (optional)
(3) Verify server certificate. Check cryptographic parameters
(4) client key exchange
send secret key info.
encrypted with server public key
(5) send client certificate
(6) Verify client certificate (if required)
(7) client "finish"
(8) server "finished"
(9) exchange messages
(encrypted with shared secret key)

Fig. 3. Client-Server TLS Handshaking Process

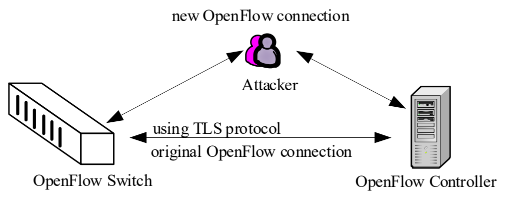


Fig. 4. The Man-in-the-Middle Attack

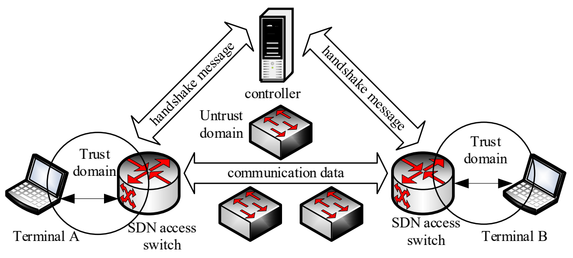


Fig. 5. SDWN Network Architecture

III. Proposed Schemes

A. Proposed TLSHPS

The first proposed scheme is called TLS Handshake Procedure Simplification (TLSHPS), which aims to simplify the TLS handshaking procedures and enhance confidentiality of the connection between the controller and switches. Consequently, it can reduce the transmission delay. By this scheme, the controller generates and issues the symmetry key for the two communication parties instead of using key exchange to prevent from key lost. In addition, the controller will verify the certification instead of the terminal.

Consider the network architecture in Fig. 5, we assume: (1) TLS protocol is used to protect the communication between the controller and the SDN switches. (2) The trust domain covers the access network from the terminal to the SDN switches. The controller will validate the attributes of all TLS handshake messages attributes, including protocol versions, cipher suites, certificates and compression methods. Then, it will decide to forward or block the flow.

As shown in Fig. 6, the proposed TLSHPS protocol works in seven steps. (1) Clients initiates the communication and sends the Certificate message, Certificate Verify message and Client Hello message, including the cipher suites, the protocol version, a random number, and session ID. The Certificate message contains the Client Certificate Chain. The Certificate Verify message includes the digitally signed hash value of the combination of Client Hello and Certificate message. (2) The controller checks the timestamps and nonce in the Client Hello message. If the verification is successful, the controller forwards the initial handshake message to the server directly. Otherwise, the controller drops the connection. (3) The server returns the Server Hello message, Certificate message and Certificate Verify message to the controller after receiving the Client Hello message. (4) The controller checks the timestamp and nonce in the Server Hello message, which is forwarded to the client upon successful verification (similar to Step 2). (5) The controller generates a symmetric key for this TLS session after the Certificate Verification of the client and server is successful. (6) The client and server will receive the Key message and identity message from the controller. The

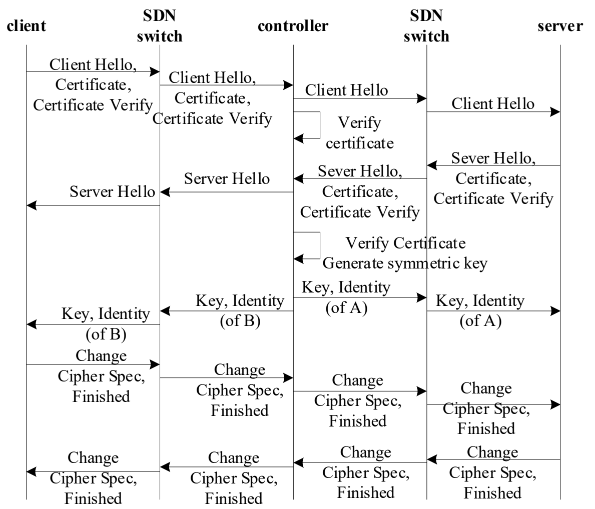

client
SDN switch
controller
SDN switch
server
Client Hello, Certificate, Certificate Verify
Client Hello, Certificate, Certificate Verify
Client Hello
Client Hello
Verify certificate
Sever Hello, Certificate, Certificate Verify
Sever Hello, Certificate, Certificate Verify
Server Hello
Server Hello
Verify Certificate Generate symmetric key
Key, Identity (of A)
Key, Identity (of A)
Key, Identity (of B)
Key, Identity (of B)
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished

Fig. 6. Flow Diagram of the TLSHPS

Key message is constructed by the symmetric key, the nonce and timestamp. The nonce and timestamp are received in the Client Hello and Server Hello messages and will be returned to the sender by the controller. (7) Finally, the client and server exchange ChangeCipherSpec and finished messages.

By the proposed TLSHPS protocol, the controller intercepts the certificate messages and also sends the identity information of the correspondent terminal. The verification of the handshake message integrity is not between the two communication terminals, but between the terminal and the controller. At Step 5, the controller generates and issues the symmetric key for the two terminals through the encrypted link between the controller and the SDN switch. The symmetric key is used as the *pre_master _secret*.

B. *Proposed TLSIHP*

If the controller is utilized for verifying and generating the symmetric key for TLS communications, heavy traffic flows might overload the controller because the controller is a single entity, which is responsible for the centralized network management in SDNs. Therefore, another scheme of TLS with Improved Handshaking Procedure (TLSIHP) has been proposed, under which edge controller and fabric controller (FC) in the control plane are used to support different functions and services. The former verifies the handshake message, and the latter manages routing and other tasks. The controller will only verify the certificate and restrict the cipher suites to use the strong authentication approach. The policy rule will check the TLS handshaking attribute; cipher suites and certificate issuer. The controller performs the verification and decides which TLS connection will be established or blocked.

In the SDN architecture shown in Fig. 7, edge switches forward the TLS handshake message to the controller for verification purpose. The controller checks the TLS flow

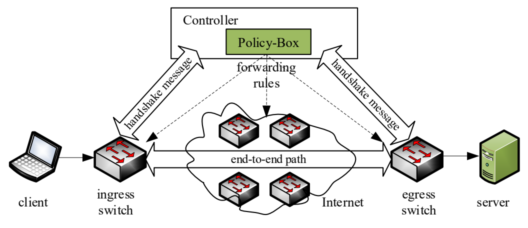


Fig. 7. SDWN Network Architecture with Edge Switches

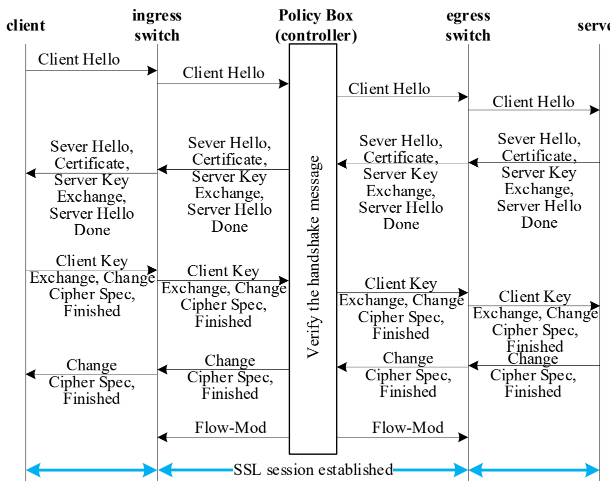

client
ingress switch
Policy Box (controller)
egress switch
server
Client Hello
Client Hello
Client Hello
Client Hello
Verify the handshake message
Sever Hello, Certificate, Server Key Exchange, Server Hello Done
Sever Hello, Certificate, Server Key Exchange, Server Hello Done
Sever Hello, Certificate, Server Key Exchange, Server Hello Done
Sever Hello, Certificate, Server Key Exchange, Server Hello Done
Client Key Exchange, Change Cipher Spec, Finished
Client Key Exchange, Change Cipher Spec, Finished
Client Key Exchange, Change Cipher Spec, Finished
Client Key Exchange, Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Change Cipher Spec, Finished
Flow-Mod
Flow-Mod
SSL session established

Fig. 8. Flow Diagram of the TLSIHP

attributes according to the policy rule defined in the controller [18]. Once the TLS handshake message is classified, the policy box installs the forwarding rule to the switches to build the connection between client and server. The TLSIHP only verifies the TLS flow attributes and check the particular values inside the TLS Hello messages. Then, it searches for particular TLS parameters, aiming to match their values with the policy rule defined inside the controller policy box. Fig. 8 shows the detailed flow of the TLSIHP handshake process.

(1) A client sends a Client Hello message, Certificate message and Certificate Verify message to the server. (2) The controller will verify TLS flow attributes, including client TLS protocol version, cipher suites, certificate issuers, and nonce. Upon successful verification, the controller forwards it to the server. Otherwise, it will be blocked. (3) After receiving the Client Hello message, the server sends the Server Hello message, Certificate message and Server Key (key identity of server) to the client. (4) The controller checks the server TLS protocol version, cipher suites, certificate issuers, timestamp and nonce (Similar to Step 2). Upon successful verification, the controller will forward it to the client. Otherwise, the controller will stop it. (5) The client sends the Client Key exchange, ChangeCipherSpec and Finished message to the server after receiving the Server Hello message, Server Key Exchange and so on. (6) Finally, the server sends the ChangeCipherSpec and Finished message to the client after receiving the client message.

The difference between the TLSHPS and the TLSIHP protocol is that the TLSHPS protocol verifies the certificate and generates the symmetric key for client and server and issues it to authenticate the confidentiality of transmission between two terminals. By the TLSIHP, the controller verifies the certificate only, which can significantly reduce the load of the controller.

IV. Security Verification Results and Discussions

Scyther is an automatic security protocol verification tool, which is widely used for the classification of cryptography protocol secrecy, including robustness and integrity. To use Scyther, the protocol description is written in the Security Policy Definition Language (SPDL). It takes a role-based description of a protocol as input and a claim is declared to verify the intended security properties. Scyther provides three functions to check the protocol verification:

(1) *verify protocol*—only the claim events are verified or falsified by Scyther. (2) *verify automatic claims*—Scyther can automatically generate claims and check them even security properties are not specified as claim events. (3) *characterise roles*—Scyther analyses each protocol role and provides all finite representation traces, which includes an execution of the protocol role.

A. Verification of the TLS protocol

The verification process includes the TLS v1.2 for mutual TLS_RSA protocol. The RSA authentication is the most commonly deployed mode of the TLS [19]. By the TLS-RSA, four single-use symmetric keys are generated to as session keys to authenticate and encrypt the application data in consequent transmission. Therefore, the TLS-RSA protocol will be tested by the Scythe tool.

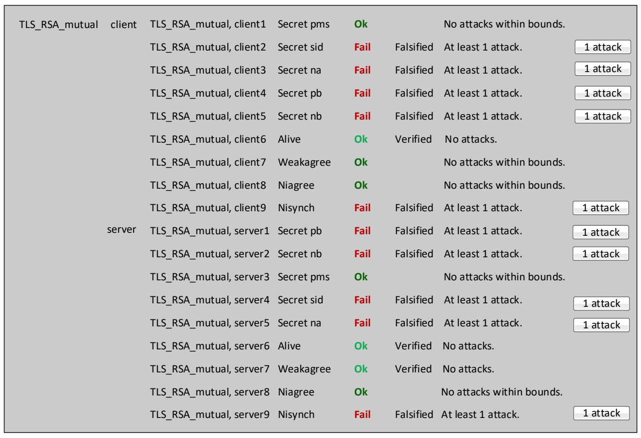
The SPDL language is used to write the process of the TLS handshaking. Two roles have been identified as a client and a server. It makes the global declaration for the required variable names, hash function, user types and miscellaneous and asymmetric keys. Then, the role definitions as sequences of events are identified. Once the protocol description in SDPL is ready, one can click the verify tab in the first panel of Scythe GUI, it provides three types of verification. For the TLS-RSA protocol, automatic claims function is run to generate claims and check them. Fig. 9 shows the outcome of the Scyther verification protocol function for the TLS protocol.

From Fig. 9, the TLS protocol is secure and with some vulnerability points. A status is either *OK* or *Fail*. The status *OK* means that the claim has been verified no attacks or no attacks within bounds. The status *Fail* means that the claim has been falsified with at least one attack. The *1 attack* button under the class can be clicked and it will open up a new window, which shows the point where the attack appears. In particular, a random fresh pre-master secret (PMS) key is generated by the client side, which is the only secret information used in the computation of all session keys [19]. Hence, the secrecy of PMS fundamentally determines the secrecy of the session keys. The Scyther tool observes that the session key secrecy has been attacked using {LKRactor} by service side AKC attack. The hacker has essentially eavesdropped on a regular handshake. Then, the hacker has decrypted {PMS}pk(s) by using the sk(S) to obtain the PMS and computed the session keys. It allows the hacker to hijack any subsequent transmitted messages or implant rendition to establish an unsecured transmission channel. Therefore, it proves that the TLS is unsecure.

B. Verification of the Proposed TLSHPS and TLSIHP

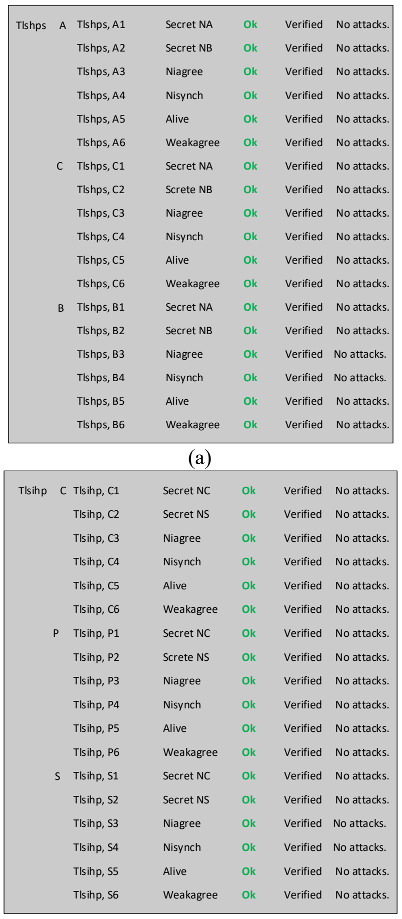
According to the TLSHPS protocol description, the controller records the timestamp and nonce in the Client and Server Hello message and holds on for a time window. The TLSHPS has been written in SPDL to test with the Scythe. After running the *verify protocol*, it is approved that the TLSHPS protocol does not carry any loose hole and it ensures secure transmission. Fig. 10(a) shows the result window for the TLSHPS testing by the Scyther. Most of the claims have been used to verify the security properties to show status OK, without attacks discovered within bounds. Therefore, it can conclude that the TLSHPS is secure.

Next, the description of the proposed TLSIHP is presented in the SPDL language for the evaluation. As shown in Fig. 10(b), all claims have been verified with OK



TLS_RSA_mutual	client	TLS_RSA_mutual, client1	Secret pms	Ok		No attacks within bounds.	
		TLS_RSA_mutual, client2	Secret sid	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, client3	Secret na	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, client4	Secret pb	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, client5	Secret nb	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, client6	Alive	Ok	Verified	No attacks.	
		TLS_RSA_mutual, client7	Weakagree	Ok		No attacks within bounds.	
		TLS_RSA_mutual, client8	Niagree	Ok		No attacks within bounds.	
		TLS_RSA_mutual, client9	Nisynch	Fail	Falsified	At least 1 attack.	1 attack
	server	TLS_RSA_mutual, server1	Secret pb	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, server2	Secret nb	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, server3	Secret pms	Ok		No attacks within bounds.	
		TLS_RSA_mutual, server4	Secret sid	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, server5	Secret na	Fail	Falsified	At least 1 attack.	1 attack
		TLS_RSA_mutual, server6	Alive	Ok	Verified	No attacks.	
		TLS_RSA_mutual, server7	Weakagree	Ok	Verified	No attacks.	
		TLS_RSA_mutual, server8	Niagree	Ok		No attacks within bounds.	
		TLS_RSA_mutual, server9	Nisynch	Fail	Falsified	At least 1 attack.	1 attack

Fig. 9. The Result of TLS Verification Claim using Scyther Tool



Tlshps	A	Tlshps, A1	Secret NA	Ok	Verified	No attacks.
		Tlshps, A2	Secret NB	Ok	Verified	No attacks.
		Tlshps, A3	Niagree	Ok	Verified	No attacks.
		Tlshps, A4	Nisynch	Ok	Verified	No attacks.
		Tlshps, A5	Alive	Ok	Verified	No attacks.
		Tlshps, A6	Weakagree	Ok	Verified	No attacks.
	C	Tlshps, C1	Secret NA	Ok	Verified	No attacks.
		Tlshps, C2	Screte NB	Ok	Verified	No attacks.
		Tlshps, C3	Niagree	Ok	Verified	No attacks.
		Tlshps, C4	Nisynch	Ok	Verified	No attacks.
		Tlshps, C5	Alive	Ok	Verified	No attacks.
		Tlshps, C6	Weakagree	Ok	Verified	No attacks.
	B	Tlshps, B1	Secret NA	Ok	Verified	No attacks.
		Tlshps, B2	Secret NB	Ok	Verified	No attacks.
		Tlshps, B3	Niagree	Ok	Verified	No attacks.
		Tlshps, B4	Nisynch	Ok	Verified	No attacks.
		Tlshps, B5	Alive	Ok	Verified	No attacks.
		Tlshps, B6	Weakagree	Ok	Verified	No attacks.

(a)

Tlsihp	C	Tlsihp, C1	Secret NC	Ok	Verified	No attacks.
		Tlsihp, C2	Secret NS	Ok	Verified	No attacks.
		Tlsihp, C3	Niagree	Ok	Verified	No attacks.
		Tlsihp, C4	Nisynch	Ok	Verified	No attacks.
		Tlsihp, C5	Alive	Ok	Verified	No attacks.
		Tlsihp, C6	Weakagree	Ok	Verified	No attacks.
	P	Tlsihp, P1	Secret NC	Ok	Verified	No attacks.
		Tlsihp, P2	Screte NS	Ok	Verified	No attacks.
		Tlsihp, P3	Niagree	Ok	Verified	No attacks.
		Tlsihp, P4	Nisynch	Ok	Verified	No attacks.
		Tlsihp, P5	Alive	Ok	Verified	No attacks.
		Tlsihp, P6	Weakagree	Ok	Verified	No attacks.
	S	Tlsihp, S1	Secret NC	Ok	Verified	No attacks.
		Tlsihp, S2	Secret NS	Ok	Verified	No attacks.
		Tlsihp, S3	Niagree	Ok	Verified	No attacks.
		Tlsihp, S4	Nisynch	Ok	Verified	No attacks.
		Tlsihp, S5	Alive	Ok	Verified	No attacks.
		Tlsihp, S6	Weakagree	Ok	Verified	No attacks.

(b)

Fig. 10. Verification Results for (a) TLSHPS (b) TLSIHP by Scyther Tool

status. Thus, the security property has been successfully verified and the proposed TLSIHP is also secure. As compared to the standard TLS, the proposed TLSHPS and TLSIHP protocol can reduce about 13% and 21% of the delay in the handshake, respectively.

C. Security Analysis

Ability against replay attacks—A replay attack is a kind of network attack, which maliciously repeated valid data transmission [18] or is delayed as the attacker intercepts the message and resends an old message. For both proposed schemes, a fresh timestamp is used only once. A nonce is a random number generated by the client or server. The controller records the nonce and timestamp in the Client

Hello and Server Hello messages. The records are held for a time window. The controller drops a message if its timestamp falls outside the window. If an attacker resends a message to a terminal inside the time window, the SDN switch will forward the handshake message to the controller. Then the controller will check the record of the nonce and timestamp, and the message will be rejected. Therefore, a replay attack can be prevented.

Message Integrity—The message integrity is the most important security property so that the correctness of the information is guaranteed. It is to defend against information alteration or destruction in the transmission. Therefore, by both proposed schemes, a cipher suite allows using a combination of pseudo-random function (PRF) algorithm, the message authentication code algorithm, the key exchange algorithm and the encryption algorithm.

Ability against impersonation attacks—By the impersonation attack, the identity of a legitimate party in the SDN is successfully assumed by an adversary [11]. Using asymmetrical cryptography can prevent the impersonation attacks. The client / server puts its public key into the certificate message and a signature signed by its private key into the Certificate Verify message. The controller decrypts the message using the public key and verifies the decryption results. The client / server can prove that it has a private key compatible with the public key to pass the verification successfully. It can prevent impersonation attacks.

Ability against MITM attacks—By both proposed schemes, each handshake message is only transmitted between the controller and the SDN switch, encrypted by the TLS protocol provided in the SDWN. No information could be obtained even if an attacker intercepts the encrypted handshake messages. Furthermore, the verification of the hash value in the Finished message will fail if the attacker tampers with the handshake messages [18].

V. Conclusion

In this paper, the security of the TLS for the SDNs has been evaluated by Scyther tool to discover its vulnerability. To improve the TLS protocol, two security enhancement schemes, the TLSHPS and the TLSIHP, have been proposed and evaluated. The security verification results show that both proposed schemes can prevent some malicious attacks successfully. Moreover, they can satisfy the security property of confidentiality, integrity and authentication. Finally, they both reduce the handshake delay as compared to the TLS.

References

[1] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys & Tutorials,* vol. 16, no. 4, pp. 1955-1980, 2014.

[2] L. Schehlmann, S. Abt, and H. Baier, "Blessing or curse? Revisiting security aspects of Software-Defined Networking," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, Rio de Janeiro, Brazil, 2014, pp. 382-387.

[3] H. Arora. (2017). *Software Defined Networking (SDN) - Architecture and role of OpenFlow*. Available: https://www.howtoforge.com/tutorial/software-defined-networking-sdn-architecture-and-role-of-openflow/

[4] Open Networking Foundation. (2013, 1 September). *Software-Defined Networking: The New Norm for Networks*. Available: https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/

[5] Open Networking Foundation. (2013). *SDN Security Considerations in the Data Center*. Available: https://opennetworking.org/sdn-resources/solution-briefs/sdn-security-considerations-in-the-data-center/

[6] W. Zhou, L. Li, M. Luo, and W. Chou, "REST API Design Patterns for SDN Northbound API," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, 2014, pp. 358-365.

[7] IBM Knowledge Center. (2019). *How TLS provides identification, authentication, confidentiality, and integrity*. Available: https://www.ibm.com/docs/en/ibm-mq/9.1?topic=tls-how-provides-identification-authentication-confidentiality-integrity

[8] N. Lawrence. (2018). *SDN and its Role in Automating & Scaling in the Data Center*. Available: https://www.cisco.com/c/en/us/solutions/collateral/enterprise/cisco-on-cisco/cs-en-08022017-sdn-thesis.html

[9] D. Samociuk, "Secure Communication Between OpenFlow Switches and Controller," in *AFIN2015-The Seventh International Conference on Advances in Future Internet*, Venice, Italy, 2015.

[10] S. Hogg. (2014). *SDN Security Attack Vectors and SDN Hardening*. Available: https://www.networkworld.com/article/2840273/sdn-security-attack-vectors-and-sdn-hardening.html

[11] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," presented at the Proceedings of the first workshop on Hot topics in software defined networks, Helsinki, Finland, 2012. Available: https://doi.org/10.1145/2342441.2342466

[12] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions," in *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, B. B. Gupta, G. M. Perez, D. P. Agrawal, and D. Gupta, Eds. Cham: Springer International Publishing, 2020, pp. 341-387.

[13] D. Asturias. (2017). *9 Types of Software defined network attacks and how to protect from them*

[14] B. Agborubere and E. Sanchez-Velazquez, "OpenFlow Communications and TLS Security in Software-Defined Networks," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017, pp. 560-566.

[15] SDxCentral Studios. (2016). *What Is an OpenFlow Controller?* Available: https://www.sdxcentral.com/networking/sdn/definitions/openflow-controller/

[16] H. Böck. (2016). *TLS – The most important crypto protocol on the Internet*. Available: https://int21.de/slides/hackpra-tls/#/

[17] C. Schum, "Correctly implementing forward secrecy," 2015.

[18] A. Ranjbar, M. Komu, P. Salmela, and T. Aura, "An SDN-based approach to enhance the end-to-end security: SSL/TLS case study," presented at the NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, Istanbul, Turkey, 2016. Available: https://doi.org/10.1109/NOMS.2016.7502823

[19] D. Basin, C. Cremers, and M. Horvat, "Actor Key Compromise: Consequences and Countermeasures," in *2014 IEEE 27th Computer Security Foundations Symposium*, 2014, pp. 244-258.