Secured Communication Channels in Software-Defined Networks

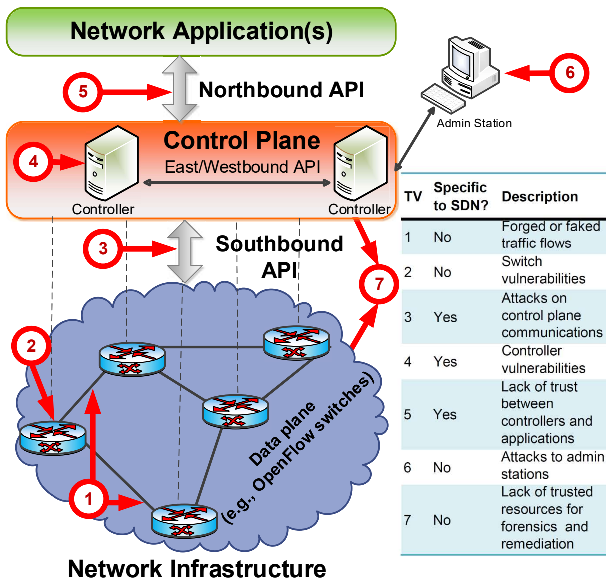
Beytüllah Yiğit, Gürkan Gür[†], Bernhard Tellenbach[†] and Fatih Alagöz
Department of Computer Engineering, Bogazici University
34342 Istanbul, Turkey
[†]Zurich University of Applied Sciences (ZHAW)
8401 Winterthur, Switzerland
Email: {beytullah.yigit, fatih.alagoz}@boun.edu.tr, {gueu, tebe}@zhaw.ch

***Abstract*—Software-Defined Networking (SDN) brings new opportunities to alleviate the existing security deficiencies of traditional networks. However, it also introduces new issues, a primary one being the vulnerabilities related to data and control plane communications. This work presents a security architecture to address security problems regarding data exchange in software-defined networks. To this end, a cryptographic key generation application is proposed to generate certificates which are used for securing communication of SDN entities (controller, switch and application). We also provide an overview of related literature focusing on key elements in such architecture. In our model, Transport Layer Security (TLS) can be activated between SDN nodes to provide confidentiality, integrity, authentication and authorization with special certificate fields. Besides, an integrated security module further strengthens the communication security by applying Access Control List (ACL), hardening TLS configuration and reducing the impact of private key hijacking. It also facilitates security administration tasks via per-channel activation/deactivation of TLS protocol and monitoring of real-time security alarms.**

I. Introduction

SOFTWARE-DEFINED networking (SDN) is a key technology for achieving the flexibility and scalability levels required by future network infrastructure. It relies on the main idea of decoupling the control and the data planes: network intelligence and network states are logically centralized while the underlying network infrastructure is abstracted by the controller for different applications. The software-based controller determines the network decisions (*control plane*), while switches handle the actual forwarding of data packets (*data plane*). A simplified view of a generic SDN architecture and threats against SDN are shown in Fig. 1 [1]. In this setting, the controller can be used to enhance network security by exploiting the global network view. Security applications at the controller can analyze and correlate network data to infer the network status spatially and temporally. Furthermore, they can run attack detection algorithms and help to understand the nature of various network threats and security incidents. The information collected and analyzed by the control plane can then be used to enforce appropriate security policies in the data plane. Overall, these characteristics provide potent capabilities to improve the security level of the communication infrastructure.

This work was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under Grant 117E165.



TV	Specific to SDN?	Description
1	No	Forged or faked traffic flows
2	No	Switch vulnerabilities
3	Yes	Attacks on control plane communications
4	Yes	Controller vulnerabilities
5	Yes	Lack of trust between controllers and applications
6	No	Attacks to admin stations
7	No	Lack of trusted resources for forensics and remediation

Fig. 1. SDN architecture and threat vector (TV) map [1].

Although SDN offers opportunities to address security related network issues, it also involves various comparable challenges which cast worthwhile technical questions such as control plane saturation attacks and detection of malicious application. Threat vectors against SDN are also shown in Fig. 1 [1]. As seen in the figure, threat vectors ①, ②, ⑥ and ⑦ already exist in traditional networks. On the contrary, threat vectors ③, ④ and ⑤ are specific to SDN and stem from the separation of the control and data planes and the addition of a new component to the network, namely the controller. Threat ① considers generating malicious flows for Denial of Service (DoS) attacks. System and software vulnerabilities in the switch, controller and admin stations result in Threat ②, ④ and ⑥, respectively. Threat ③ includes attacks on the confidentiality, integrity and availability of the communication channel between the control and data planes. For example, they can be carried out by exploiting a (configuration) flaw in the TLS protocol or by carrying out a DoS attack on a network

TABLE I
RELATED LITERATURE

Work	Contributions and employed key method(s)	Key Generation
Samociuk [2]	Advantages and drawbacks of TLS, SSH and IPSec protocols are compared for SB interface.	Not Applicable (N/A)
Lam et al. [3]	Implementation of TLS support for cluster communications (EWB) in ONOS controller	N/A
Lam et al. [4]	Identity-based cryptography to secure SB interfaces	Partially
Kang et al. [5]	Pre-shared secrets for authentication of switches by extending OpenFlow to solve Data Path ID duplication problem.	No
Agborubere et al. [6]	Review of TLS security flaws and proposal of a TLS extension for OpenFlow	No
Liyanage et al. [7]	Host Identity Protocol to authenticate switches	No
Othman et al. [8]	Not only controller but also switches can install flow rules to other switches by using digital signatures.	No
Cui et al. [9]	A mutual authentication mechanism provided by certificates for NB interface	No
Paladi et al. [10]	A framework to establish a secure communication channel for OpenFlow via certificates is developed. It uses Intel SGX which offers strong isolation of cryptographic data by reserving a dedicated private area in the memory.	Partially
Qin et al. [11]	Authentication for controllers and their messages by using hash chains	Partially
Peng et al. [12]	Usage of Quantum Key Distribution technology to secure SB interface	Partially
Toseef et al. [13]	Certificate-based AAA services for SDN experimental testbeds (e.g. OFELIA)	Partially
Banse et al. [14]	Authentication and authorization of SDN applications by controlling certificate fields	No

link or communication endpoint. Threat ⑤ arises from lack of trust between the controllers and applications while Threat ⑦ points lack of reliable and trustworthy information to understand the cause of a detected problem. Threats ⑦ also includes absence of secure and stable system snapshots for remediation. Therefore, security in SDN is a vibrant research field with a wide range of topics. Most of them can be assigned to the body of work either on using SDN as a security enabler or on how to implement SDN architectures so that they are themselves secure. In this work, we focus on the latter and consider the security of SDN communication channels. Specifically, there exist three key communication interfaces in SDN: SB (southbound), EWB (east-westbound) and NB (northbound) as shown in Fig. 1. SB interface is located between the controller and switches. OpenFlow has emerged as the *de facto* SB protocol. OpenFlow specification recommends the use of TLS with mutual authentication between the controllers and their switches[1]. However, this feature is optional starting with OpenFlow version 1.1.0. In the control plane, the EWB interface covers controller communications in a cluster while the NB interface provides communication between network applications and the controller. Although EWB and NB interfaces do not have agreed-upon protocols, a REST API, which is TLS compliant, is typically used for them.

In essence, TLS is imperative for protecting communication in SDN interfaces. Nevertheless, relying on pure TLS usage alone is not sufficient, albeit being instrumental. First, TLS must be hardened in the whole network since it contains some security flaws due to obsolete or vulnerable versions. Furthermore, TLS must be used with mutual authentication to limit the possibility of adding a bogus device. Hence, the communicating entities should have proper certificates. In that situation, one needs to consider the security of these certificates and handle possible outcomes of private key hijacking (stealing). Additionally, an Access Control List (ACL) for network elements should regulate access since a switch or a controller is authorized to connect to only a subset of controllers. From the network management perspective, a security architecture must handle network devices which lack TLS support, generate alarms about security incidents and allow per-channel activation/deactivation of TLS protocol.

[1]The specification also mentions auxiliary OpenFlow connection which can use Datagram Transport Layer Security (DTLS).

In this work, we propose *Secure Communication Architecture for Software-Defined Networks (SECAS)* to overcome the aforementioned security problems. We design a cryptographic key generation model and implement it with an SDN application that can generate and manage key material for communicating entities and then issue and manage corresponding X.509 certificates. Special certificate fields are designed to limit the impact of private key hijacking. Moreover, TLS configuration is hardened for all interfaces through features such as removal of TLS version and removal of support for weak cipher suites. From the operational perspective, real-time security alarms are dispatched for security breaches like unauthorized access requests. TLS usage in each interface can be managed or monitored separately via a user interface.

II. RELATED WORK

In this section, we present the literature on communication channel security and authentication of network elements in SDN. Table I shows some related technical works summarizing their main methods and key generation capability. The papers [2] and [3] do not entail a proposal for SDN security. Samociuk compares security protocols, TLS, SSH and IPSec in [2]. He also mentions about certificate generation via Public Key Infrastructure (PKI). Lam et al. focuses on EWB interface and implements TLS support in that channel for ONOS controller in [3]. However, only intra-cluster communication is secured and inter-cluster communication remains unprotected. Their experiments show that TLS overhead in terms of number of messages is about 5%.

Lam et al. propose to apply identity-based cryptography (IBC) to secure SB communications [4]. They modify TLS protocols to use them with IBC. In IBC, public keys are generated according to the identity of the user corresponding to, for instance, MAC address. The private keys are generated in Private Key Generator (PKG) which is a separate security module. Hence, an entity first creates its public key according to its identity and then communicates with PKG to obtain its private key. Kang et al. propose a lightweight data plane authentication scheme to solve data plane identifier (DPID) duplication problem where switches have the same DPID causing the controller to malfunction [5]. Liyanage et al. use the Host Identity Protocol (HIP) to authenticate switches in [7]. In HIP, similar to IBC, public keys are used as Host Identity which are then used to mutually authenticate end nodes. Their proposal changes the SDN architecture and switch design by adding security gateways between the controller and switches to use IPSec tunneling.

In [4] and [5], communicating entities need to possess a secret key beforehand which means that they must employ symmetric key distribution. Furthermore, TLS and OpenFlow protocols need to be modified for [4], [6] and [5], respectively. Likewise, SDN architecture must include IPSec capable security gateway in [7]. Modification of a widely deployed protocol and extra component integration to SDN limit the usability of these proposals.

The works in the technical literature reveals that incumbent research generally does not address key generation or distribution. Hence, a key generation application for SDN can be beneficial and necessary since it can provide public/private keys and certificates regarding all mentioned works. Moreover, these proposals only focus on a part of the overall problem such as switch authentication or SB interface security. Thus, a holistic approach is crucial for securing SDN communication channels in an efficient and robust way. To the best of our knowledge, our architecture *SECAS* is the first work which provides a complete set of features like certificate generation and protection, ACL, security alarms and TLS hardening for communication security in SDN.

III. SECAS: A Secure Communication Architecture for Software-Defined Networks

Software-defined networks entail control plane channels (i.e. SB, NB, EWB links) which must be secured to implement a robust SDN architecture. If the data exchange is not protected, an attacker can impersonate a controller or switch and carry out malicious activities which can have serious consequences, depending on the traffic flowing over this infrastructure.

In this overall setting, a powerful and motivated attacker/adversary can control the network infrastructure; hence block, save, change, drop or replay all communication in the SDN. Moreover, it can deploy fake network entities to initiate a Sybil attack and subsequently disrupt the network communication. Therefore, authentication and authorization are crucial functions for this system.

In this work, we propose *SECAS* for secure communication of controller, switch, and application entities in the SDN and aim to overcome the SDN-specific threats ③ and ⑤. To this end, the attacker/adversary is restricted primarily by the deployed cryptographic methods[2]. *SECAS* consists of two main components: a key generation application to generate required cryptographic material explained in Section III-A and a security module (*SecMod*) to control TLS usage described in Section III-B.

A. SECAS Key Generation Model for SDN

The employment of TLS in SDN involves some extra operational steps for facilitating and maintaining secure communications. In particular, certificate generation for network entities, signing of certificates with private keys and distribution of cryptographic material to the network devices are required. To generate certificates and public-private key pairs, a key generation application is instrumental. In the application, a Certificate Authority (CA) is generated as the first step. All other certificates are trusted to CA. Hence, each entity only needs the CA certificate and its own certificate. Typically, self-signed certificates generated outside of the software-defined network are used to enable TLS in SDN. However, self-signed certificate generation is cumbersome and has scalability issues since these certificates must be transferred to each communicating entity which leads to the common key distribution challenge.

SECAS key generation application contains only one CA and no intermediate subordinate CA. The system administrator can choose cryptographic algorithms and key length according to the security requirements. *SECAS* model supports two widely used asymmetric algorithms: RSA and Elliptic Curve Cryptography (ECC). Although all key lengths are supported, the security administrator needs to be aware that larger key length substantially increases the time required for key generation and exchange. ECC support is sensible since it offers the same level of security with relatively small key sizes. For instance, security level of 224-bit ECC is equivalent to 2048-bit RSA. Besides, smaller key size enables faster authentication.

In *SECAS*, an SDN application takes over the job of creating and managing the certificates and key material needed to protect the communication. The sequence diagram of certificate generation is depicted in Fig. 2. First, that application generates a key pair and issues a CA certificate by using the generated key pair according to designated security parameters, e.g. the chosen asymmetric algorithm and key length. The CA certificate and key pair can be stored in password-protected Java Key Store (JKS) or Public-Key Cryptography Standards (PKCS) data format. The password for the key stores are dynamically generated according to password generation parameters (password length and character complexity such as lower letters, upper letters, numbers and special characters). After the CA generation, the network administrator can make certificate request via the user interface or a REST call. There are three certificate types: switch (C_s), controller (C_c) and

[2]Please note that, we intentionally exclude *DoS* attacks on SDN communication channels in this work. That issue deserves an availability-oriented solution, which is out of scope for this paper.

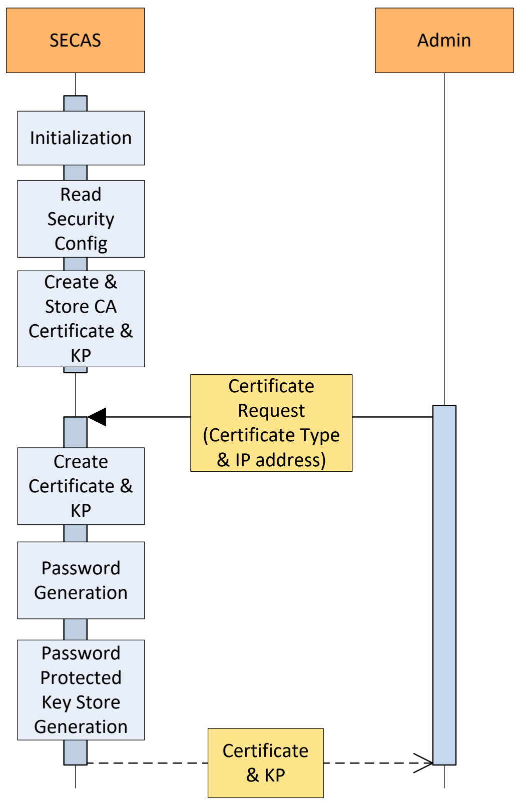

SECAS
Admin
Initialization
Read Security Config
Create & Store CA Certificate & KP
Certificate Request (Certificate Type & IP address)
Create Certificate & KP
Password Generation
Password Protected Key Store Generation
Certificate & KP

Fig. 2. Certificate and key pair (KP) generation in SECAS key generation model.

application (C_a). C_s and C_a certificates are used for SB and NB interface, respectively. C_c certificates are used for SB, EWB and NB interfaces.

SECAS utilizes only one CA in its key generation model. However, one of the biggest concerns in using a single trusted CA is the private key compromise (key hijacking). Plain password protection of keys is usually ineffective since passwords are written somewhere in the device where keys are also located. Thus, if an attacker can somehow obtain a certificate or key with its corresponding password, he/she can connect to any element in the network. To alleviate this threat, our certificates contain two internal information fields about communicating entity: type and IP address of the entity. In that way, the impact of stealing the private key from a device other than those operating the CA is limited since an attacker can only impersonate one entity from the specific type written in the certificate and can only communicate from a static IP address recorded in the certificate. Accordingly, a certificate request shown during the certificate generation in Fig. 2 contains two parameters: `certificate_type` and `IP_address`. When a certificate request is received from the network administrator, the certificate and key pair are generated according to the security requirements and input parameters. Besides, the administrators can make the request to revoke a certain certificate of a device. Accordingly, *SECAS* generates and publishes the Certificate Revocation List (CRL) to announce the devices which should no longer be trusted. In our model, certificates and key pairs which are requested by the administrator must be transferred to the network devices manually. Developing secure key distribution mechanism is out of scope of this work.

B. *SECAS Security Module (SecMod)*

Certificate generation is not solely adequate for secure communication in a TLS environment. We need to handle mutual authentication with special certificate fields, notify the network administrator about security incidents, harden TLS configuration and comply with the requirements in realistic settings like ACL. Hence, a security module in the controller is beneficial to secure the communication infrastructure of SDN in an integrated manner. The best location for the security module is the controller since controllers are involved in all communication interfaces. Besides, it is difficult to interfere with switch software or network applications as they are produced by different vendors and a large developer base.

1) TLS Hardening: TLS protocol has some vulnerabilities leading to some significant attacks such as BEAST, CRIME and BREACH [2]. Despite such security incidents, TLS is the de facto standard on the Internet to secure communications between clients and servers, and more than half of the network traffic is protected by TLS. Thus, a proper implementation of TLS is prevailingly accepted as secure [2].

TLS protocol has evolved over years and has different versions deployed across connected devices. It employs different cipher suites which are set of of cryptographic algorithms and used for key exchange, signature generation, encryption and integrity. For secure SDN communication channels, we need to harden TLS configuration by disabling older and vulnerable TLS versions, and removing weak and vulnerable cipher suites. In SECAS, the network administrator can determine the TLS configuration to deploy in the whole software-defined network. That configuration contains allowed TLS versions and cipher suites in preferred order. Thus, only secure TLS versions and cipher suites are maintained in the controller and TLS attacks like CRIME and BEAST can be prevented.

2) Security Management Functions: In most of the controller software implementations, TLS communication must be activated before the controller is started and it cannot be turned on or off without taking the controller offline. However, this situation is impractical since the controller must be restarted and then preserve all its operational context to enable or disable merely a security feature. Additionally, the security administrator is required to start or stop TLS for different reasons such as changing security policies. As an example, let us consider that all network applications are running in a compute node near to the controller and TLS is deactivated in NB interface since it is assumed to be physically secure in a data center environment. The administrator needs to activate TLS in NB interface when a remote application is deployed. Nevertheless, intermittent restart of the network controller is not a viable requirement due to changing security policies. To address this management deficiency, *SecMod* allows activation/deactivation of TLS communication on-the-fly in every interface separately via user controls. Thus, the administrator can manage the network in an agile manner as its requirements are constantly changing.

TABLE II
TLS SUPPORT IN VARIOUS COMMERCIAL SWITCHES.

Switch Model	TLS Support
HP 3500	No
Brocade MLXe	No
Alcatel-Lucent OmniSwitch	No
Open vSwitch	Yes

3) TLS-less Operation: A network consists of a diverse set of elements (switches and network applications). Particularly, switches can have different vendors and capabilities. Table II shows TLS support of various switch vendors [15]. Although more capable switches are being released alleviating this deficiency, SDN network elements in the field may still lack TLS support. Thus, *SECAS* also supports network elements whose TLS support is absent via a fall-back scenario. The administrators can mark these elements in the configuration to permit plain-TCP connections to the controller while other elements still connect via TLS[3]. Despite the fact that this is not a desired and recommended situation, it may occur due to practical constraints in operational networks.

4) Access Control: A distributed SDN control plane can have more than one controller cluster. Controllers have two types of communication through EWB interface: intra-cluster and inter-cluster. In this setting, each controller manages a subset of switches and communicates with only a set of controllers. Hence, switches must connect to the controllers assigned to them and a controller can only connect to allowed peers. To this end, *SecMod* supports ACL for each controller and can further restrict access. An ACL includes IP addresses of valid switches and controllers which can connect to a controller. When a device sends a connection request to a controller, *SecMod* controls whether the ACL contains the IP address of the device. If it does not, the connection request is rejected by the security module. Hence, no device which is not in the list can access the controller.

5) Protection against Private Key Hijacking: Private keys can be hijacked by hacking an SDN communicating entity or physically compromising it. As mentioned earlier, two special fields, `certificate_type` and (device) `IP_address`, are added to the certificates to cope with the hijacking issue. *SecMod* overrides the certificate verification to control these two fields. Besides, client authentication is enforced since the default mode of TLS considers only server authentication. After certificate generation, the client and server already have necessary certificates and key pairs which are used as key-stores to authenticate themselves. They also have the CA certificate to use as trust-store. *SecMod* takes over the certificate verification when there is a connection attempt to the controller. It performs the usual certificate verification using the device's trust-store. *SecMod* also checks the CRL to control whether the certificate is revoked or not. Additionally,

[3]The controller can not be marked since the selected controller software must support TLS anyway.

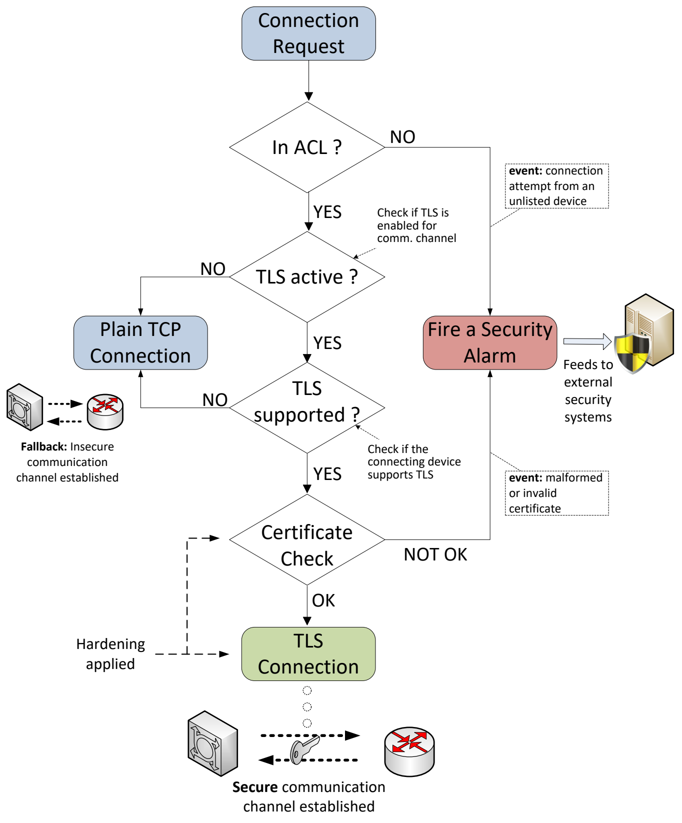

Connection Request
In ACL ?
NO
YES
event: connection attempt from an unlisted device
Check if TLS is enabled for comm. channel
NO
TLS active ?
YES
Plain TCP Connection
Fire a Security Alarm
Feeds to external security systems
NO
TLS supported ?
Fallback: Insecure communication channel established
Check if the connecting device supports TLS
YES
event: malformed or invalid certificate
Certificate Check
NOT OK
OK
Hardening applied
TLS Connection
Secure communication channel established

Fig. 3. Operation flow of *SecMod*.

the module checks the certificate type and controls whether IP address of the connecting party is identical with the respective field in the certificate.

6) Security Monitoring and Situation Awareness: Network administrators operate large networks and they can not control every network element and their states in real time. SDN provides powerful mechanisms to monitor network status such as global network view and flow statistics. *SecMod* leverages the SDN infrastructure to generate real-time alarms for the following security incidents: 1) connection requests from IP addresses not in the ACL, 2) the use of invalid or untrusted certificates in the TLS handshake and 3) a mismatch between the IP address or the type of the connecting device and the corresponding information in the certificate shown during the TLS connection setup. These alarms specifically indicate which devices are affected from which security incidents. For further analysis by the administrator(s), they can be seen from a graphical user interface. These alarms can also be used to determine the certificates which need to be revoked.

7) How They All Fit Together – SecMod Operation Flow: Fig. 3 depicts the operation flow of the *SECAS SecMod* when a switch initiates a connection to the controller. The other cases such as controller-to-controller communication are handled in a similar way. When the connection request is received by the controller, *SecMod* checks whether the IP address of the connecting switch is in the ACL. If it is not, the connection is rejected and a security alarm is generated. Otherwise, the module checks if TLS is active and the switch supports TLS. If that is not the case, the connection request is accepted but the controller starts managing the switch via plain OpenFlow. Otherwise, a TLS handshake is started. When the certificate of the switch is received, its verification is started. The certificate must be issued by the CA trusted by the controller. Additionally, its type must be "switch" and the IP address of the connecting switch must be the same IP address embedded in the certificate. If any of these checks fails, a security alarm is triggered. According to the alarm, the administrator(s) may decide to revoke the certificate of the connecting switch. If all controls are passed, the TLS handshake is completed and communication between the switch and the controller over the TLS channel can start.

IV. Discussion

Persistently, there is a performance concern when applying any security mechanism to a communication channel in data networks. Such concerns are common excuses to disable/ignore security functions based on a cost-benefit perspective. Besides, data centers, touted as leading execution points of SDN implementations, have relatively controlled and restricted environments. Therefore, security hardening and cyberattacks on the communication channels in these domains are regarded as relatively far-fetched issues. These factors have led to the current situation where securing communication channels via TLS is generally overlooked or comes later in the list of security priorities [2].

There are different elements of overhead due to adoption of TLS solutions for SDN security. First, encryption and certificate usage naturally leads to computational costs. However, modern hardware minimizes these costs with perpetual improvements. Processors utilize built-in instruction sets to efficiently handle encryption and mitigate performance issues. Hence, the performance overhead of TLS practically becomes manageable nowadays [3]. To render the related trade-offs of the *SECAS* key generation application in a tangible way, a performance simulation was conducted by generating certificates with different key sizes of RSA and ECC algorithms. We used BouncyCastle cryptography library and Java 1.8 on a 64-bit Ubuntu machine with 2.5GHz CPU, 8GB RAM. We ran each simulation for 100 times and investigated the average generation time. The results indicated that ECC has faster certificate generation speed and that generation time does not change significantly as the key size increases. In contrary, as the key size of RSA increases, the required time for certificate generation increases drastically. For instance, NIST allows 2048-bit or longer RSA keys to be used in TLS and *SECAS* key generation application can generate and pack 2048-bit certificate in less than 0.5 second on average while 224 bit ECC which is equivalent to 2048 bit RSA, took only 8 milliseconds. In addition to this investigation of key generation performance, as future work, we plan to perform system-level overhead analysis of *SECAS*.

Another considerable drawback is the certificate and key distribution since it may require manual steps like transferring generated certificates to network elements. However, key distribution is an inherent problem which can be addressed according to different requirements. Efficient mechanisms can be devised based on existing solutions to address different security policies, system attributes and use-cases. Besides, network administrators already deal with such operational challenges in the field and are used to address this kind of deployment tasks thanks to practices such as scripting. Essentially, SDN usage cannot be expanded without securing it by accepting some bearable overhead.

Since key generation must be done before TLS handshake, the key generation application does not work in real time. Thus, failure or saturation of the application do not impose a major risk and can be mitigated using various techniques (e.g. additional hardware resources). Besides, according to our simulation, 2048-bit RSA certificate generation of a large network with 1000 switches, 10 controllers and 100 applications would take less than 10 minutes. This run-time can be shortened by using a dedicated hardware module, deploying multiple instances of the key generation application by cloning the CA key pair on different machines. From an architectural viewpoint, *SECAS* can be used with intermediate CAs to achieve better scalability. Additionally, if one uses at least two sub-CAs with a root CA where the private key is only offline available, he significantly reduces the risk of single-point-of-failure and the system might have a recovery path if only one sub-CA certificate is revoked and no secure connection is possible anymore because of this situation. However, the security administrators should keep in mind that each additional layer significantly increases the time required for certificate verification.

V. Conclusions

In this work, we first present a concise overview of SDN strengths and weaknesses from the security perspective. There are many works which use SDN as a security enabler. However, the security of SDN is also a crucial topic. Therefore, we propose *SECAS* — a secure communication architecture including a key generation application to eliminate security threats to the communication channels of SDN. *SECAS* provides mutual authentication between SDN entities, end-to-end TLS encryption for all communication interfaces, certificate verification and real-time notifications/alarms about security violations. It also hardens the TLS configurations while applying ACL to further restrict access. *SECAS* considers real-world requirements of SDN such as on-the-fly TLS activation/deactivation and support for TLS-incapable devices. We also present a discussion on how *SECAS* provides secured communication channels in SDN.

As a future research direction, the security module can be further extended to detect malicious network devices and then automatically revoke their certificates. Besides, overhead of TLS to the SDN system in terms of bandwidth, latency and throughput with respect to different security parameters (e.g., key length and deployed algorithms) can be investigated. Accordingly, particular impact of TLS to the network operation according to pertinent security requirements can be explicitly conveyed.

References

[1] D. Kreutz, F. M. 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*, ser. HotSDN '13. ACM, 2013, pp. 55–60.

[2] D. Samociuk, "Secure communication between OpenFlow switches and controllers," *Proceedings of the 7th International conference on Advances in Future Internet (AFIN).*, vol. 39, 2015.

[3] J. H. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian, "TLS channel implementation for ONOS's east/west-bound communication," in *Electronics, Communications and Networks V*. Springer, 2016, pp. 397–403.

[4] J. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian, "Securing SDN southbound and data plane communication with IBC," *Mobile Information Systems*, vol. 2016, 2016.

[5] J. W. Kang, S. H. Park, and J. You, "Mynah: Enabling lightweight data plane authentication for SDN controllers," in *24th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2015, pp. 1–6.

[6] B. Agborubere and E. Sanchez-Velazquez, "OpenFlow communications and TLS security in software-defined networks," in *IEEE Green Computing and Communications (GreenCom)*, 2017, pp. 560–566.

[7] M. Liyanage, M. Ylianttila, and A. Gurtov, "Securing the control channel of software-defined mobile networks," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, June 2014, pp. 1–6.

[8] O. M. Othman and K. Okamura, "Securing distributed control of software defined networks," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 13, no. 9, p. 5, 2013.

[9] H. Cui, Z. Chen, L. Yu, K. Xie, and Z. Xia, "Authentication mechanism for network applications in SDN environments," in *20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2017, pp. 1–5.

[10] N. Paladi and C. Gehrmann, "TruSDN: Bootstrapping trust in cloud network infrastructure," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2016, pp. 104–124.

[11] H. Qin and N. Wang, "A data-origin authentication protocol based on ONOS cluster," in *MATEC Web of Conferences*, vol. 56. EDP Sciences, 2016.

[12] Y. Peng, C. Wu, B. Zhao, W. Yu, B. Liu, and S. Qiao, "QKDFlow: QKD based secure communication towards the OpenFlow interface in SDN," in *4th International Conference on Geo-Informatics in Resource Management and Sustainable Ecosystems GRMSE*. Springer, 2016, pp. 410–415.

[13] U. Toseef, A. Zaalouk, T. Rothe, M. Broadbent, and K. Pentikousis, "C-BAS: Certificate-based AAA for SDN experimental facilities," in *2014 Third European Workshop on Software Defined Networks*, 2014, pp. 91–96.

[14] C. Banse and S. Rangarajan, "A secure northbound interface for SDN applications," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 834–839.

[15] N. Dayal, P. Maity, S. Srivastava, and R. Khondoker, "Research trends in security and DDoS in SDN," *Security and Communication Networks*, vol. 9, no. 18, pp. 6386–6411, 2016.

Beytüllah Yiğit (beytullah.yigit@boun.edu.tr) is a Ph.D. candidate in the Department of Computer Engineering, Bogazici University. His current research focuses on SDN and network security.

Gürkan Gür (gueu@zhaw.ch) is a senior researcher at Zurich University of Applied Sciences (ZHAW) in Switzerland. His research interests include information security, Future Internet and information-centric networking.

Bernhard Tellenbach (tebe@zhaw.ch) is a professor and head of the information security research group at ZHAW. He received his Ph.D. degree in 2013 from ETH Zurich. His research interests include information security, network security, system security, and security training and testing.

Fatih Alagöz (fatih.alagoz@boun.edu.tr) is a professor in the Computer Engineering Department of Bogazici University, Turkey. He obtained his DSc degree in Electrical Engineering in 2000 from George Washington University, USA. His current research areas include wireless/mobile/satellite networks and network security.