Suhail Ahmad
Ajaz Hussain Mir

SECURING CENTRALIZED SDN CONTROL WITH DISTRIBUTED BLOCKCHAIN TECHNOLOGY

Abstract *Software-Defined Networks (SDN) advocate the segregation of network control logic, forwarding functions and management applications into different planes to achieve network programmability and automated and dynamic flow control in next-generation networks. It promotes the deployment of novel and augmented network-management functions in order to have flexible, robust, scalable, and cost-effective network deployments. All of these features introduce new research challenges and require secure communication protocols among segregated network planes. This manuscript focuses on the security issue of the southbound interface that operates between the SDN control and the data plane. We have highlighted the security threats that are associated with an unprotected southbound interface and those issues that are related to the existing TLS-based security solution. A lightweight blockchain-based decentralized security solution is proposed for the southbound interface to secure the resources of logically centralized SDN controllers and distributed forwarding devices from opponents. The proposed mechanism can operate in multi-domain SDN deployment and can be used with a wide range of network controllers and data plane devices. In addition to this, the proposed security solution has been analyzed in terms of its security features, communication, and re-authentication overhead.*

Keywords SDN, SDN security, blockchain, southbound interface, TLS, threats in SDNs

Citation Computer Science 24(1) 2023: 5–30

1. Introduction

The extensive efforts of researchers over the years to transform the Internet to a more reliable, programmable, open, manageable, and secure infrastructure have resulted in SDN. SDN introduces a novel networking architecture that segregates network intelligence from packet forwarding by placing network control logic into an external software-based controller. Software-based SDN controllers form the SDN control plane and provide a birds-eye view of an entire network at a single central point. This centralized network control simplifies network management and enables dynamic and automated flow control. Besides the centralized network control, SDN fosters the concept of network programmability, wherein diverse network functions are implemented as software applications – either on top of the SDN controller or as independent data-analysis functions.

The triple-layered SDN architecture that was proposed by the Open Network Foundation (ONF) is shown in Figure 1.

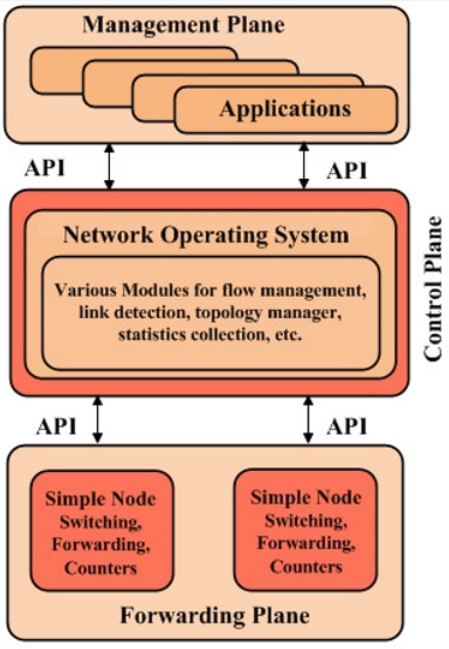

Figure 1. SDN Architecture

The bottom layer (or the data plane) includes simple forwarding devices that are called switches; these match any received packet headers against the flow rules or instructions that are issued by the SDN controller. The middle layer (or the control plane) comprises SDN controllers, which may implement a specific network operating system (NOS) to abstract low-level control logic details from management applications

and convert high-level network policies into flow rules to properly configure and manage the entire data plane infrastructure [33]. The top layer (or the management plane) involves various SDN applications that implement specific network-control functions like load balancing, routing, firewalls, etc. The applications in the management plane can be implemented by using either controller-specific programming languages or high-level SDN programming languages (like Frenetic [23], Pyretic [42], Procera [55], etc).

The information exchange among the three layers/planes in SDN takes place over the southbound, northbound, and eastbound/westbound interfaces. The management applications use the northbound interface to exchange information with the SDN control plane. It can be compared with the POSIX or win32 standard of operating systems, which provide the necessary abstractions to ensure controller and programming language independence. The primary goal here is to conceal complex network-wide configurations and state management to simplify the roles of network administrators by providing suitable high-level abstractions.

The eastbound interface is used to exchange information between SDN controllers, while the westbound interface enables information exchanges between centralized SDN controllers and traditional distributed control plane. Neither the eastbound nor the westbound interface has been standardized, and various controllers [13, 22, 32, 44, 49, 51] use diverse approaches for inter-controller communication and to communicate with legacy distributed control [28–30]. On the other hand, the open vendor agnostic interface between the data and the control plane is called the southbound interface (SBI); in this manuscript, our main goal is to secure the information exchange at this interface.

In brief, the SDN paradigm is a promising architecture that is still in its early stage of development and requires further advancements in all of the three planes, and more importantly, in the inter-plane communication protocols in order to realize its full benefits in production and realistic deployments. The two key aspects that act as impediments in widespread SDN acceptance are interfacing with legacy distributed controls and SDN security. The former has received much attention from the research community, and numerous hybrid SDN controllers and data plane devices (DPDs) have been proposed (which we analyzed in our previous work [3]). The security aspect of SDN requires more consideration in terms of the development and testing of secure and protected SDN layers and inter-communication mechanisms in order to withstand diverse real-world security threats.

1.1. Contributions

The primary focus of this manuscript is to protect SDN control traffic from adversaries and malicious devices. To the best of our knowledge, this is first time a blockchain-based multi-domain authentication and key-exchange mechanism has been proposed to secure a centralized SDN control. The proposed lightweight blockchain-based decentralized security mechanism for the southbound interface provides additional security

features as compared to the existing approaches. In this manuscript, the major contributions that are made are as follows:

- We provide an overview of key resources in different DPDs and SDN controllers.
- We highlight how an insecure southbound interface jeopardizes the resources of both the control and data planes.
- We summarize various security threats or attacks that are plausible over an unprotected SBI.
- We emphasize the various vulnerabilities and limitations of the most prominent TLS-based security solution for the SBI.
- We describe our proposed robust, secure, and distributed mechanism for SBI security.
- We analyze the proposed security mechanism in terms of its security features, communication overhead, and re-authentication cost.

1.2. Paper outline

The manuscript is organized as follows: Section 2 provides background information regarding the southbound interface and blockchain technology. Section 3 highlights southbound interface security vulnerabilities/threats/attacks, and Section 4 presents related works and the challenges in the existing approaches. Section 5 presents the proposed approach, and Section 6 elaborates the security features and analyzes the proposed mechanism. Finally, the paper is concluded in Section 7.

2. Background: Southbound Interface (SBI) and blockchain technology

The three distinguishing features of the SDN paradigm are dynamic traffic-flow control, centralized network visibility, and network programmability. Such features have revolutionized the overall network control and basic packet forwarding. However, these features have also introduced new security challenges and threat vectors into the network architecture. The separation of the forwarding infrastructure, control functions, and management applications into three planes have introduced new threat interfaces and vulnerable targets. The information exchange among the SDN planes require secure communication channels that necessitate the use of cryptographic mechanisms in order to ensure message integrity, confidentiality, and source authentication. In this paper, our goal is to authenticate devices in the SDN domain and secure the traffic between the control and data planes. Therefore, we first elaborate the services that are offered and resources that are involved in these two planes; in the next section, we highlight how an insecure SBI can be exploited by opponents to disrupt such services and resources.

2.1. SDN Control Plane

The SDN control plane usually involves general-purpose hardware that executes NOS [3]. NOS is comprised of basic control programs that are necessary to manage DPDs. It hides the intricate network control logic from applications and presents a simplified and comprehensive view of the network devices to the applications (which is analogous to the operating system of a PC). More specifically, NOS encompasses the necessary control programs for topology detection and network traffic management. The core control modules that are commonly observed in the majority of NOSs are shown in Figure 2.

Along with the topology-manager module, the link-detection module provides conversant topology information. The controller discovers switches, hosts, and links in the network with the help of an initial handshake process, packet_in messages, and LLDP protocol, respectively [31]. Using this topology information, traffic flow paths are defined across the network by the decision-making module. Two other important core modules are the flow and storage managers. The former uses the SBI to express and alter traffic-flow rules in DPDs, whereas the latter manages the network state information. In addition to these, the SDN controller usually involves other supplementary modules like a statistic-collector module, a dedicated queue-manager module, and a module manager for flow-statistic collection, the management of various queues, and the orchestration of information exchanges between different controller modules, respectively.

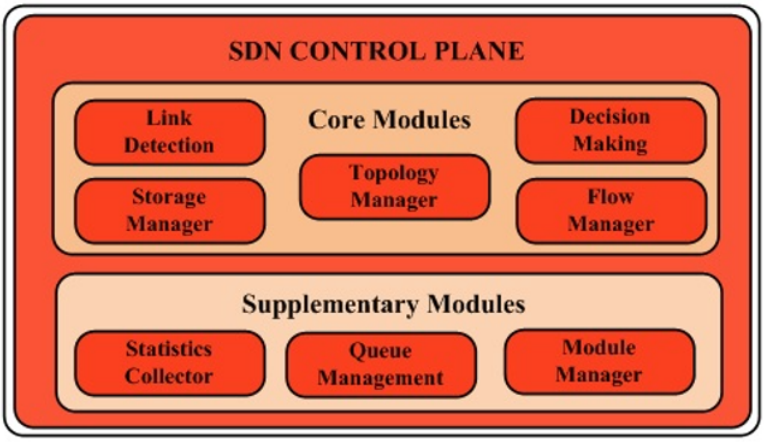

Figure 2. SDN Controller Modules

In short, the intelligence of the entire network resides in the SDN control plane. Logically or physically centralized SDN controllers enable dynamic network control and vibrant network monitoring. In dynamic network control, the forwarding devices are programmed as per the policy directives that are expressed by the applications in the management plane and can be reconfigured as per the changing network state. In

network monitoring, on the other hand, SDN controllers retrieve flow-statistic information from the forwarding devices that can be analyzed by numerous applications in the management plane (like load balancing, security, etc.). If traffic congestion or other anomalies like security attacks are detected, the SDN control plane dynamically reprograms the flow rules so that the network traffic is diverted to under-utilized paths or an intrusion-detection system (IDS). The network traffic-statistic information is also beneficial for network provisioning in order to meet future traffic demands. Both of these control functions involve bi-directional information exchanges between the control and data planes via the SBI. Therefore, it necessitates a reliable information exchange at the SBI in order to determine the correct network state and detect anomalies like congestion, attacks, or faults in the network. To prevent opponents or malicious devices from accessing the network state information and disturbing the smooth functioning of the network necessitates protecting the communication channel between the control and data plane. However, most SDN controllers do not use secure SBIs except for a few (like OpenDayLight [44], ONOS [13], and Rosemary [51]).

2.2. SDN data plane

The SDN paradigm emerged with a notion of simple and dumb forwarding devices that are managed by centralized SDN controllers. Immediately after the introduction of OpenFlow [37], numerous network-programming proposals [23, 42, 55] promoted this two-tier programming model wherein the entire network intelligence and stateful processing are assigned to the SDN controller, and stateless DPDs forward packets as per the forwarding rules that are communicated by the remote controller. This centralized network intelligence is suitable for those network deployments where the forwarding state changes does not have stern real-time requirements and mostly depend on the global network state. However, it can become a bottleneck for various applications wherein wire-speed reactions are required for various events like simple port/flow state changes. Consequently, researchers have proposed numerous approaches over the last few years for offloading some control tasks and stateful packet processing inside DPDs/switches in order to address the control latency issue of the two-tier programming model. Here, we first present the traditional SDN data plane using OpenFlow; then, an overview of a stateful SDN data plane is provided.

2.2.1. Conventional SDN data plane using OpenFlow

The main aim of OpenFlow designers was to promote low-cost and high-performance network implementations that can provide flexible and adaptable innovations [37]. The authors of OpenFlow proposed an abstract model of a programmable flow table that is commonly termed as "match/action" abstraction. This match/action abstraction is comprised of the following three main parts (as shown in Figure 3a): (i) rule – a combination of header fields that range from Layer 2–Layer 4; (ii) action – one or more actions that specify the further processing of received packets; and (iii) stats – counters for statistic collection that is relevant to the matching rule.

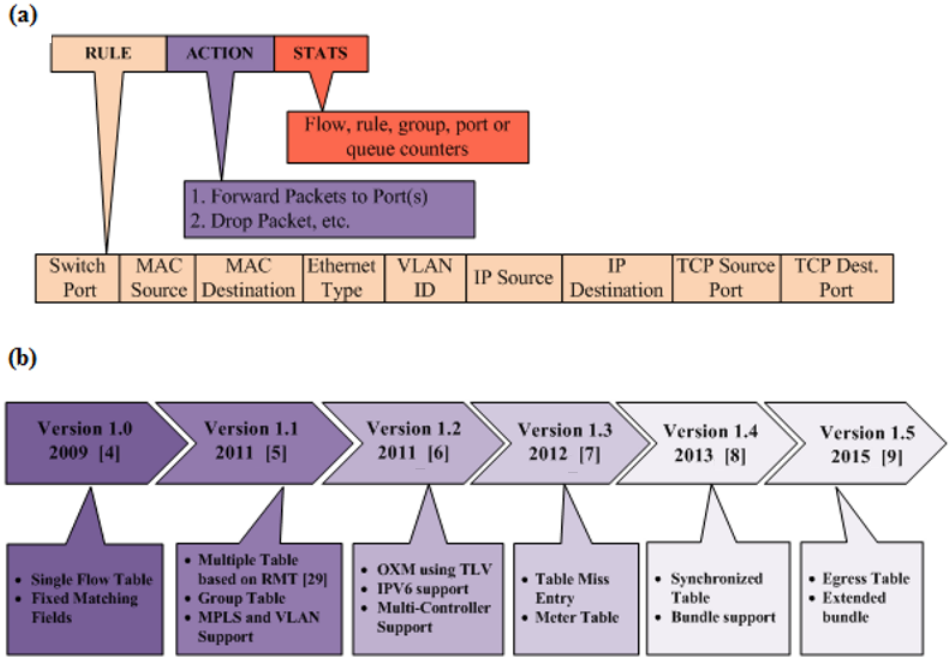

Figure 3. (a) Flow rule – Match, Action, Stats in OpenFlow; (b) OpenFlow evolution

Since its inception, OpenFlow has gone through a brisk evolution. The basic version 1.0 [45] provided only a single flow table with 12 fixed header fields, whereas the latest version 1.5 [46] introduced a number of advanced functions along with 41 matching fields. The evolution of OpenFlow is shown in Figure 3b, along with the most prominent features that were added with each version. After the standardization of version 1.0, it immediately became evident that it was very restraining; subsequently, three important advances (flexible match/action, multiple flow tables, and tailored stateful extensions) were made to OpenFlow in the advanced versions.

The most important extension in OpenFlow is group tables, which provide elementary stateful operations in the DPDs. For instance, the original OpenFlow specification required remote controller intervention to instantiate a new flow rule in case of a link/port failure. Such controller dependence will result in delayed responses; in the intervening period, those packets that were destined to a failed port would be lost. The fast failure group table contains multiple action buckets to address this issue. Another group table is select that enables load balancing in the data plane. Group tables are quite suitable for handling such situations, but they are still optional and loosely specified in the latest OpenFlow specification. Also, these extensions in the OpenFlow evolution approve the essence of stateful operations inside the DPDs.

2.2.2. Stateful SDN data plane

Initially, OpenFlow proved to be quite pragmatic, as it opened doors for programmability that was well beyond what was realizable at that time with closed commercial switches. It provided more flexibility with immediate deployability by a simple firmware upgrade. However, the increased control latency due to the continuous data and control plane signaling that is required in OpenFlow made the research community consider other options like offloading limited control tasks and stateful flow processing inside the DPDs. The initial breakthrough in this direction was provided by P4 [18], OpenState [16], and FAST [43], which defined switch-level primitives inside the DPDs to handle the flow states. Their basic idea was to allow the programmer to define stateless flow tables in the switches (like OpenFlow) and also those flow rules that were termed as stateful rules that changed as per the state of a particular flow. Since then, new switch architectures [17, 53], stateful data plane platforms [16, 48, 59], compilers, programming languages, and frameworks [24, 25, 27, 52] have been proposed.

The stateful SDN data plane overcomes the limitations of OpenFlow by providing more-significant programming abstractions in the data plane that simplify the configuration and stateful operations inside the DPDs. The overall processing of the stateful data plane can be summarized as follows:

i) *state storage:* per-flow state information is stored inside DPDs (including distributed state storage);
ii) *state transition:* allows one to perform programmatically formalized in-switch state transitions upon receiving packet or occurrence of event in data plane;
iii) *autonomous decisions:* capability to perform autonomous decisions inside DPDs based on locally stored state information without involving SDN controller.

Therefore, the stateful DPDs store historical information (reflecting the current state of a flow) of incoming and outgoing packets as per the instructions that are received from the remote controller. The state information of a particular flow changes either due to sending/receiving packets or the occurrence of an event like a link/port change. This stateful data-plane processing does not violate the basic SDN principle of separate control and data planes, as the centralized SDN controller still controls (i) which states to be offloaded to the switch (keeping in view the performance) and (ii) how the states will be managed by the switch.

2.3. Blockchain technology

A blockchain is basically a distributed collection of data-elements wherein each element is termed a block. All such blocks are interlinked chronologically to form a chain that is secured by cryptographic measures [58]. The existing blockchain systems are broadly classified into three types: public, private, and consortium blockchains. In this manuscript, we have used the consortium blockchain, which is a semi-decentralized system wherein only a limited set of nodes are used to validate a block through consensus. The read permissions of the blockchain records are either public or restricted.

In the last decade, numerous domains have employed blockchain-based solutions ranging from production to IoT. Various studies [6, 35, 38] have also advocated the use of blockchain technology in SDNs. The primary motivation to develop a blockchain-based authentication mechanism for the SDN environment is its distributed nature and reduced communication overhead among domains/datacenters/regions. These features are very useful in SD-WANs and multi-tenant data-center networking.

3. Motivation: SBI security vulnerabilities

All of the SDN layers and interfaces are susceptible to different types of security attacks, which can either compromise the target elements of a specific layer or launch an assault from one layer to subvert another layer [19]. Some vulnerabilities and security threats can be leveraged by using conventional techniques/mechanisms that are commonly employed in legacy networks, whereas some challenges require new mitigation techniques/security protocols due to the different architecture.

Attacks in SDN can be classified according to the primary objective of the attacker. For instance, if an attacker eavesdrops at the SBI, the main goal of the attacker can be to access and tamper with critical data plane information; therefore, it can be classified as an unauthorized disclosure. An insecure control-data plane channel can be easily sniffed by an adversary to retrieve topology, monitoring, or other management information. A man-in-the-middle attack is also very likely over an insecure channel wherein an intruder can subvert the information exchange between the SDN controllers and the DPDs. In this section, we have highlighted various security threats and challenges that can be confronted by the SDN control and data planes due to an insecure SBI.

3.1. Data plane threats

Stateful or stateless SDN DPDs involve numerous resources like multiple-flow tables, state memory, and packet processors that can be overwhelmed/exhausted by attackers if an insecure communication exchange takes place at the SBI. The various security threats against DPDs can be categorized as follows:

i) ***Flow table mutations/flooding:*** SDN controllers communicate flow-rules to the DPDs over the SBI. If the control-data plane channel is not protected with cryptographic mechanisms, an intruder/adversary can easily modify the rules or overwrite/flush the existing traffic-management rules in the DPDs. The attacker can disturb the flow tables and degrade the overall performance by (i) flooding the switch tables with spurious flow rules and (ii) setting the parameters to generate a continuous table miss, which would force devices to raise false alarms for the installations of flows. The first case may result in a flow table flooding attack wherein an attacker stores bogus rules in the flow tables and the legitimate requests will be quite frequently forwarded to the SDN controller [57]. On the other hand, the second case may result in a denial of service for legitimate requests.

ii) ***Fuzzing attacks:*** In an open control channel, an attacker can inject cleverly crafted control packets into the data plane that contain malicious or malformed headers that can expose prevailing vulnerabilities or can disturb the normal packet flow processing. Such malformed control packets can lead forwarding switches to an undesired state that can have a detrimental effect on the overall switch stability and performance [57].

iii) ***Reconnaissance attacks:*** Such attacks are also termed as side-channel attacks wherein an attacker can commonly clout the resultant response of a target device against a particular network situation to deduce certain implied information that an attacker can subsequently use to launch other kinds of attacks. For instance, an adversary can determine the control latency by sending specific packets to a particular switch and later on use the same information to launch a flow table flooding attack [57].

iv) ***State memory exhaustion attacks:*** In a stateful SDN data plane, each data plane device allocates memory for state transitions that are generated by incoming packets or switch-level events. To save state information, data structures like array-based variables or state tables are used in different stateful DPDs [16, 59]. With an insecure southbound channel, however, an attacker can exploit the open channel to store unnecessary state information into the DPDs to exhaust the state memory.

v) ***Switch malfunction attacks:*** Another threat vector that is derived from an unprotected switch-controller channel is the possibility that an attacker may force the execution of CPU-intensive tasks on a switch. For instance, an attacker may impersonate the SDN controller and can send flow-status queries to the switch, forcing a switch to incessantly compute the required information. The other switch-related attacks are identity hijacking and spanning tree poisoning. In the former case, an attacker may impersonate a legitimate switch to disconnect a genuine one, whereas in the later case, an attacker can fabricate fake links by using crafted LLDP packets to poison the spanning tree protocol.

3.2. Control plane threats

The control plane, which holds all of the network state information and regulates the network resources in an effective manner, faces the following threats due to an insecure SBI:

i) ***Packet-in flooding:*** Without cryptographic protection at the SBI, the attacker can compromise numerous switches and can flood them with malicious flow packets. In a conventional SDN, the DPDs transform such flow requests into packet-in messages and send them to the controller for flow-rule information. The controller processes such requests, resulting in a waste of its computational power. Such an attack has a two-fold negative effect, as the resources of both the switch and controller are wasted; in the worst case, this may result in a denial of service for legitimate flows.

ii) ***Topology poisoning:*** As discussed in the previous section, the SDN controller discovers switches, hosts, and links in the network with the help of an initial handshake process, packet_in messages, and LLDP protocol, respectively [31]. In order to have up-to-date topology information at the centralized SDN controller, the said message exchange takes place continuously over the data-control channel. An insecure message exchange can be exploited by an opponent to poison such messages with fake information; in the worst case, this may result in having incorrect topology information with fake links and nodes at the centralized controller. Topology poisoning can also be used to perform a host-location hijacking attack in which an attacker crafts LLDP packets to poison the topology information in order to divert the traffic of a legitimate host toward an attacker's device [26]. In addition to this, an adversary may also exploit the device-learning messages to create message-forwarding loops [1].

iii) ***NOS service disruption:*** Like a traditional PC operating system, the NOS in the SDN control plane controls network resources to simplify the network management and, at the same time, attempts to provide cost-effective resource utilization. However, most NOSs support limited security features except for a few (like ONOS [13], ODL [44], and Rosemary [51]). With an open data-control plane channel and lack of authentication mechanisms, any reprobate data plane device can additionally exploit the SDN controller's mis-configuration and vulnerabilities in order to achieve diverse objectives like executing control commands that kill core-controller processes, redirecting information that is anticipated at a legitimate device, tampering with and accessing controller databases, and modifying internal data that may lead the SDN controller to an erratic state; in the wort case, this can kill vital controller processes or tamper with internal data structures to achieve a denial-of-service attack or controller non-availability.

All of these security threats are summarized in Table 1. Apart from these security threats, there are numerous other vulnerabilities in different planes and interfaces (as reported by the researchers in [1, 5, 7, 8, 19]); however, our aim here is to highlight only those issues that are associated with an unprotected SBI.

Table 1
Security threats in SDN Data and Control Plane due to unprotected SBI

Network Plane	Attacks	Main Highlights
Data Plane Threats	Flow Table Mutations/Flooding	Attacker modifies DPD flow table entries or can cause overflow of rules and can also flood controller with packet_in messages;
	Fuzzing Attacks	Malformed control packet injection to expose vulnerabilities;

Table 1 con't

Network Plane	Attacks	Main Highlights
	Reconnaissance attacks	Attacker determines certain information that can be subsequently used to launch other kinds of attacks;
	State memory	Opponents flood stateful SDN devices with bogus state exhaustion information to consume state storage memory;
	Switch malfunction	Attacker forces execution of CPU-intensive tasks on attacks switch to disturb normal processing at DPD.
Control Plane Threats	Packet_in Flooding	Fake packet_in messages disrupt normal processing of packets and, in worst case, results in denial of service for legitimate flows;
	Topology Poisoning	To divert traffic of legitimate host toward an attacker's device or create forwarding loops in data-link layer;
	Service disruption	Without protection in place, core controller processes and internal data structures can be tampered with, which can have detrimental impact on overall controller operations.

4. Related work

As observed in the previous section, the SBI is highly prone to various attacks without security protocols, and attackers can breach SDNs while remaining unobserved. The ONF recommended TLS for securing the information exchange at the SBI. One of the impediments for network operators when using TLS is the tedious configuration, as both the controllers and the DPDs require certificate authority's (CA) keys, certificates, and the signing of these certificates with the CA's key [12]. The keys and certificates need to be deployed prior to the actual network implementation. This complicated configuration that is required at both ends is a major hindrance for TLS adoption for the SBI.

On the other hand, network administrators in SDNs have the flexibility to choose DPDs [20, 50, 54, 56] as per the requirements; accordingly, they must use security protocols as per the availability of resources in these DPDs. TLS provides such flexibility and backward compatibility by enabling distant parties to choose from different security algorithms and protocol versions during the initial handshake process. However, this flexibility is a primary security concern in TLS, as it may jeopardize the network's resources and control traffic in terms of integrity, availability, and confidentiality. Since its inception, the TLS protocol suite has confronted numerous practical and theoretical attacks such as collision attacks [14, 15], RACOON [39], CRIME [9, 21], POODLE [41], Triple Handshake [40], DROWN [11], etc. Consequent to such attacks, TLS evolved from version 1.1 to version 1.3. The latest version 1.3

introduced a faster handshake process, new security features, and restricted use of vulnerable cipher suites. However, the authors of [36] claimed that a downgrade attack is still feasible in TLS 1.3.

The authors of [34] proposed identity-based cryptography (IBC) for SBI security. The proposed approach relieves the end parties from obtaining public keys from a central authority to derive session keys. However, the authors created a simplified set-up where the roles of the various entities were not clearly mentioned. On the other hand, the authors of [47] deployed an intrusion-detection center (IDC) and KDC between the control and data planes to monitor SBI traffic and secure in-transit control traffic. They did not evaluated the proposed system in terms of security features or other performance parameters.

The authors of [2] pointed out various deficiencies in TLS-based implementations with OpenFlow and recommended certain changes in order to improve overall network security. The highlighted issues are support for vulnerable versions, support for vulnerable protocols, optional client verification, etc. They have necessitated mutual authentication for seamless and secure network connectivity. Keeping all of these issues in view, we present an authentication and key-exchange mechanism in the next section that addresses most of these vulnerabilities.

5. Proposed lightweight authentication and key-exchange mechanism

The top-level diagram of the proposed approach in a multi-domain SDN environment is shown in Figure 4. It involves the following entities that provide reliable device authentication and key-exchange mechanisms in the SDN environment:

- **Domains:** The SDN environment can be segregated into various domains, including SDN controllers that provide necessary services to the DPDs. Furthermore, each domain involves many DPDs, a service manager (SM), and a witness peer (WP).
- **SDN Controller:** As discussed in Section 2.1, the controller implements the network control logic and provides all of the forwarding and traffic-regulating services to the DPDs. When a data plane device initiates a connection with the SDN controller, the controller initially diverts it to the service manager for the necessary authentication; once authenticated, the required services are provided to the DPDs.
- **Switches or DPDs:** Irrespective of stateful or stateless DPDs, the primary function of these devices is to forward packets. The forwarding decision is based on the flow rules that are communicated by the SDN controller. In the proposed scenario controller, however, the services are only available if the DPDs are authentic.
- **Authentication Server (AS):** AS is a trusted central authority that publishes the public parameters and cryptographic functions that are used. Furthermore, it

also registers and authenticates the DPDs and the service managers in the SDN environment.

- **Service Manager (SM):** This is responsible for handling the blockchain of a specific region and is associated with the witness peer. The SM is also authenticated by the AS and, once authenticated, can establish a secure connection with the DPD for secure service delivery.
- **Witness Peer (WP):** This writes the authentication outcome to the blockchain public ledger. Together, both WP and SM form a consortium blockchain and depend on PBFT (practical Byzantine fault tolerance) for consensus establishment.

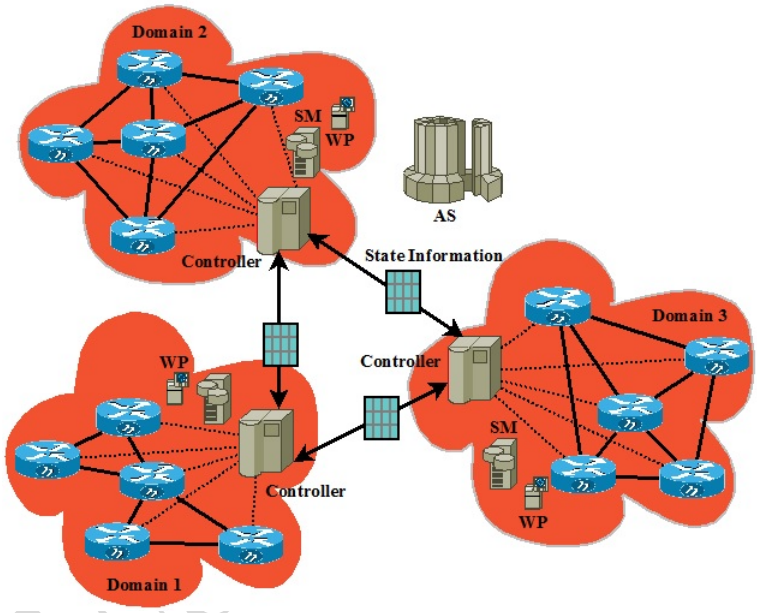

Figure 4. Multi-domain SDN environment

5.1. Proposed mechanism for SBI security

The entire process of authentication and key exchange can be categorized into the following phases: i. Initialization; ii. Registration; iii. Authentication and Key Exchange; iv. Consensus; and v. Service-Delivery. All of the five phases are elucidated below:

i) **Initialization Phase:** In this phase, the AS initializes the SDN environment for the forthcoming phases and involves the following steps:

1. The AS uses an elliptic curve E with public parameters G and n, where G is the base point of E with order n.
2. Using elliptic curve E, the AS determines its private key (S_{AS}) and public key (P_{AS}) as follows: $S_{AS} \in Z_P$ and $P_{AS} = S_{AS}.G$, where (.) represents elliptic curve cryptographic (ECC) multiplication.

3. At the end of the initialization, the AS publishes the following public parameters: $[E, G, n, H_1(), H_2(), P_{AS}]$, where $H_1()$ and $H_2()$ are the collision-resistant one-way hash functions that are used during the mutual-authentication and key-exchange phase.

ii) **Registration Phase:** In this phase, all of the entities (including the switches and SMs) register themselves with the AS over a secure channel. For source anonymity, the identities of these entities are never communicated in clear text (as shown in Figure 5).

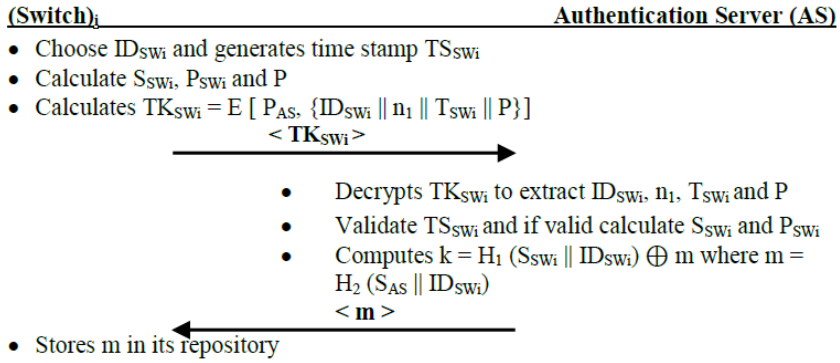

Figure 5. Registration of Data Plane Device

Here, we have elucidated the registration of a switch; likewise, the other entities can be registered.

1. Switch (SW_i) selects an identity ID_{SWi} for itself and concurrently generates a nonce or time stamp TS_{SWi} that protects from replay attack. Afterwards, it computes the private key and public key for itself as follows:
   - Selects secret key, $S_{SWi} = n_1$, and calculates public key, $P_{SWi} = n_1.G$
   - Calculates hash of following: $P = H_1(ID_{SWi}||n_1||TS_{SWi})$.
2. Switch (SW_i) calculates an intermediate-token, $TK_{SWi} = E[P_{AS}, \{ID_{SWi}||n_1||TS_{SWi}||P\}]$ and then relays it to AS.
3. The AS decrypts the received TK_{SWi} and extracts the values of ID_{SWi}, n_1, TS_{SWi}, and P. The AS validates the received time stamp; if it falls within the permissible limit, then the AS proceeds further – otherwise, the connection is terminated. The received P ensures message integrity.
4. Like switch (SWi), the AS generates the private and public keys for switch (SWi) as follows:
   Uses received secret key $S_{SWi} = n_1$ and calculates public key $P_{SWi} = n_1.G$.
5. Last, AS calculates, transmits, and stores the value of $k = H_1(S_{SWi}||ID_{SWi}) \oplus m$ in its distributed ledger, where $m =$

$H_2(S_{AS}||ID_{SWi})$. It also transmits 'm' to the i^{th} switch, which acts as an authentication token for future correspondence.

6. The i^{th} switch stores it in its repository upon receiving the value of 'm' .

iii) **Mutual Authentication and Key-exchange Phase:** In this phase, the switches mutually authenticate with the SM via the controller and share a session key for further communication.

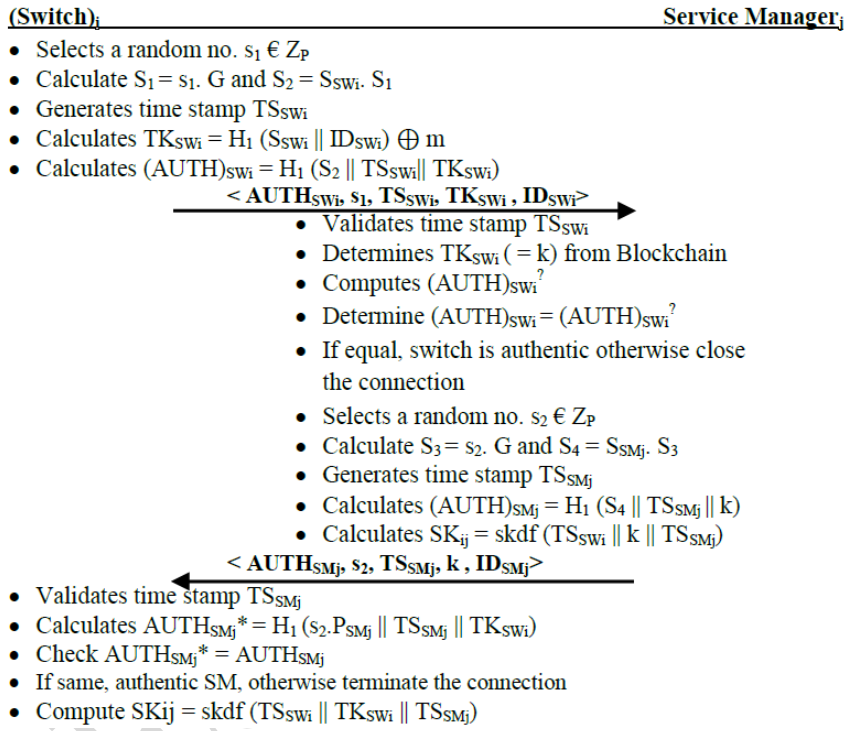

Figure 6. Authentication and Key Exchange

This process is shown in Figure 6 and explained as follows:

1. The i^{th} switch (SWi) generates random number s_1 ($s_1 \in Z_P$) and time stamp TS_{SWi}. Afterwards, it performs two ECC multiplicative operations over s_1 to compute S_1 and S_2 as follows:

$$S_1 = s_1.G; \text{ and } S_2 = S_{SWi}.S_1$$

.

2. The switch calculates the value of token $TK_{SWi}(= k) = H_1(S_{SWi}||ID_{SWi}) \oplus m$. Using TK_{SWi}, the switch calculates the authentication token as follows:

$$(AUTH)_{SWi} = H_1(S_2||TS_{SWi}||TK_{SWi}).$$

The values of $[(AUTH)_{SWi}, s_1, TS_{SWi}, TK_{SWi}, ID_{SWi}]$ are securely relayed to the service manager for source authentication and further processing.

3. Once the SM receives it, it validates the received time-stamp TS_{SWi}; upon its successful validation, it fetches the value of $TS_{SWi}(= k)$ from the registered list of switches from the blockchain. Using k, SM determines an authentication token to prove the truthfulness of the received $(AUTH)_{SWi}$ as follows:

$$(AUTH)_{SWi}? = H_1(s_1.P_{SWi}||TS_{SWi}||k).$$

If the values of $(AUTH)_{SWi}$ and $(AUTH)_{SWi}?$ are the same, this proves the authenticity of SWi to SMj; otherwise, the connection is terminated.

4. Likewise, the SMj authenticates itself to the switch by generating a random number s_2, a time stamp TS_{SMj} and authentication token $(AUTH)_{SMj}$. In addition to this, it calculates a session key with the help of a session key derivation function (skdf) to be used for further communication as $SK_{ij} = skdf(TS_{SWi}||k||TS_{SMj})$. Finally, the values of $[AUTH_{SMj}, s_2, TS_{SMj}, k, ID_{SMj}]$ are securely transmitted to the SWi for further processing. Here, we have not described function skdf(), as any mechanism for key derivation can be employed by the end parties.
5. Upon receiving the values of $[AUTH_{SMj}, s_2, TS_{SMj}, k,$ and $ID_{SMj}]$, the switch validates the time stamp; if valid, it will proceed further to determine the authenticity of SMj. The switch also determines session key $SK_{ij} = skdf(TS_{SWi}||TK_{SWi}||TS_{SMj})$. Therefore, both parties mutually authenticate each other and are now ready for secure data transmission that is encrypted with the session key.

iv) **Consensus Phase:** In the proposed approach, PBFT is used to form the public ledger, as it requires a relatively limited number of nodes to build the consensus. The authentication outcome is transmitted to the blockchain using the following steps:

1. In this setup, we assume 'n' witness peers (WPs) that can write a block to the distributed public ledger. Out of these 'n' WPs, one is designated "Speaker" while the others behave as "Congressmen" during the consensus. The speaker ensures a smooth consensus process and cannot join the voting process. Moreover, the speaker usually conducts 'm' rounds of consensus for saving the time that is required in the speaker selection. Here, speaker 'a' is nominated on the basis of following mechanism: $a = (h \bmod n) + 1$, where 'h' denotes the current block's height.
2. As discussed earlier, the result is broadcast to all WPs after a successful registration. The WPs store the authentication results before they can be updated in the distributed public ledger.
3. After a period of 't' intervals, the block that contains the authentication results is created, which is followed by the voting process. Initially, the speaker

broadcasts a request to all congressmen for vote using $[P_{req}, h, WP_i, block_h,$ and $Sig_{WPi}(block)]$, where P_{req} refers to the request to vote.

4. After receiving the voting request, the j^{th} WP shares its vote using $[P_{res}, h, WPj, block_h,$ and $Sig_{WPj}(block)]$, where P_{res} denotes the j^{th} WP's response.
5. After receiving responses from the WPs, the speaker builds a consensus to publish an authentication block to the public ledger.

v) **Service Delivery Phase:** This phase seamlessly provides DPDs with the necessary services without re-authentication if there is a change of controller or a switch migrates to another controller. If the controller changes, the Switch (SWi) simply sends $E(P_{SMj}, k)$ where $k = H_1(S_{SWi}||ID_{SWi}) \oplus H_2(S_{AS}||ID_{SWi})$ to the new controller. Once the new controller receives the request from a new switch, it validates the existence from the distributed public ledger. If it is found and not on the revocation list, the necessary services are provided to the switch without further re-authentication.

6. Analysis and discussion

In this section, we have evaluated the proposed approach in terms of the various security features and compared it with the TLS-based security solution. The evaluation was performed on the basis of (i) the supported security features and (ii) the communication and re-authentication overhead.

6.1. Supported security features

The proposed scheme provides confidentiality and forward secrecy by encrypting all of the messages in an entire workflow. The hash P with the internal encryption in the registration phase and the other hash values in the subsequent phases ensure message integrity. The timestamps that are used in various transcripts resist replay attacks. On the other hand, the source anonymity and impersonation is ensured, as no message-carrying device identity is sent in clear text. Furthermore, a device can select a pseudonym randomly and decide when it will expire during the authentication and key-derivation phases. Therefore, the responsibility of preserving the privacy lies with the corresponding device.

Non-interactivity ensures that the opponent has least amount of critical information when attempting to break the implemented security mechanism. In the proposed scheme, a data plane device merely sends one message (authentication/service request) to an SM and, hence, guarantees non-interactivity. Furthermore, the re-authentication of a DPD under a new controller requires only one message (as illustrated in the service-delivery phase).

If there is a misbehaving DPD in a domain, it can be reported to the AS, which will check its real identity and accordingly invalidate its public key. This ensures trace-

ability, as the AS will reveal its identity. Therefore, non-repudiation is also guaranteed in the proposed scheme.

The various security features of the proposed lightweight scheme are shown in Table 2. It is clear from the table that the proposed mechanism provides additional security features with respect to the existing TLS-based solution. Furthermore, the authentication blocks in the blockchain can be accessed by any user, and such records are immune to tampering/modification due to the intrinsic feature of the blockchain technology.

Table 2
Comparison of security features between TLS and Proposed Mechanism

Security Features	**TLS**	**Proposed Mechanism**
Confidentiality	Y	Y
Integrity	Y	Y
Immune to Record Tampering	Y	Y
Resists Impersonation	Y	Y
Mutual Authentication	Y	Y
Source Anonymity	N	Y
Forward Secrecy	Y	Y
Non-repudiation	Y	Y
Resists Replay	Y	Y
Key Exchange	Y	Y
MITM	N	Y
Non-interactivity	N	Y

6.2. Communication and re-authentication overhead analysis

Some SDN controllers have adopted TLS-based security solutions for different SDN interfaces [3]. The latest TLS version 1.3 has introduced a faster handshake process, new security features, and the restricted use of vulnerable cipher suites. The initial handshake is reduced from a four-way to three-way handshake. It begins with client_hello (CH), which includes version negotiation (Ver_c) (for backward compatibility with older versions), client nonce (N_c), and supported cipher suites (C_c) (as shown in Figure 7).

It also includes TLS_extension (Ext), which contains client_supported_version (sv_c) and key_share (ks_c). The client sets sv_c as per its priority and its elliptic-curve-based Diffie–Hellman key exchange (EC-DHKE) public key along with security parameters like DHE group pairs in ks_c. Upon receiving this message from the client, the server responds with server_hello (SH) that includes selected_version (Ver_s, for backward compatibility), server nonce (N_s), selected cipher suite (Cs), and extension (Ext), which includes selected supported_version (sv_s) and key_share (ks_s). Furthermore, the server also sends encrypted_extension (EE), server_certificate (CERT), and

certificate_verify ($CERT_V$). EE include extensions like signature algorithms and supported_group, which were sent as plaintext in TLS 1.2 earlier. $CERT_v$ is one of the major enhancements in TLS version 1.3, as it ensures strong resilience against MITM by performing digital signatures of all of the messages that are exchanged from CH to CERT. After CH and SH, both end parties can derive the session key by using the DHKE algorithm. Finally, the handshake is done by exchanging the FIN message.

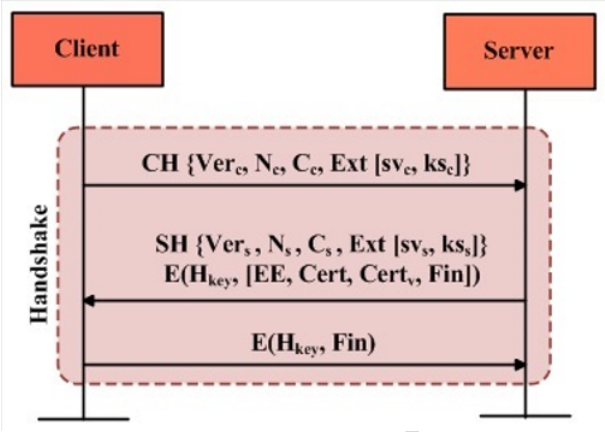

Figure 7. Three-way handshake in TLS 1.3

For simplicity, we have assumed EC-DHKE and unilateral authentication (i.e., client authenticates a server) in the three-way handshake (as shown in Figure 7). However, both the client and the server share various security parameters as well as their certificates for mutual authentication and key derivation. Most SDN controllers lack security measures [3]; those that use TLS-based security for SBI have not updated to the latest TLS version, thereby making them prone to various attacks.

In order to compare the proposed key-exchange and authentication mechanism with TLS, we have analyzed the TLS handshake by using the Wireshark packet-sniffer tool. It has been observed that version 1.2 usually involved more than 5000 bytes for a complete handshake, whereas version 1.3 involves more than 3000 bytes. In the proposed approach, on the other hand, the mutual authentication mainly involves two hash values: AUTH, and k/TK_{SWi} (as shown in Figure 6). If a strong hash function like SHA-256/SHA-384 is used, then the two messages in the authentication and key exchange do not exceed more than 1500 bytes. Therefore, the proposed approach reduces the overall control traffic. In addition to this, the re-authentication in the proposed scheme only involves a single message (as discussed in the service delivery phase), which further reduces the control traffic in SDNs.

Additionally, if we consider the fat-tree topology [4, 10] (which is very widely used in datacenter networks), the amount of control traffic for the authentication and key exchange increases rapidly with increased aggregation levels. So, if we are able

to reduce the control traffic of the initial authentication and key exchange to half, the overall bandwidth requirement and control latency can be reduced significantly in such use cases.

7. Conclusion

In this manuscript, we have attempted to safeguard SDN control traffic from adversaries and malicious devices. We have highlighted numerous security threats and issues that are associated with an unprotected southbound interface and have proposed a decentralized blockchain-based security mechanism for authentication and key exchange. The proposed scheme provides the following benefits: i) it involves the least amount of control traffic for authentication and key exchange; ii) it relieves the SDN controller from the burden of initial authentication and key determination; iii) it ensures non-interactivity; and iv) its blockchain-based distributed ledger ensures uninterrupted availability of authentication information and overcomes the security issues that are faced by a central authority. The proposed mechanism is compared to the widely used TLS-based security solution and provides adequate security with limited overhead. The security analysis of the proposed scheme shows that it is practical to safeguard centralized SDN control with a distributed blockchain technology.

This manuscript only provides informal proof of the various security features of the proposed scheme. In the future, we will try to investigate formal security proof of the proposed mechanism. Furthermore, we will try to explore the possibility of extending the proposed mechanism to the other SDN interfaces. Another possible direction is to further reduce the authentication and consensus overhead with more flexibility.

References

[1] Abdou A., Van Oorschot P.C., Wan T.: Comparative analysis of control plane security of SDN and conventional networks, *IEEE Communications Surveys & Tutorials*, vol. 20(4), pp. 3542–3559, 2018.

[2] Agborubere B., Sanchez-Velazquez E.: 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)*, pp. 560–566, IEEE, 2017.

[3] Ahmad S., Mir A.H.: Scalability, consistency, reliability and security in SDN controllers: a survey of diverse SDN controllers, *Journal of Network and Systems Management*, vol. 29(1), pp. 1–59, 2021.

[4] Al-Fares M., Loukissas A., Vahdat A.: A scalable, commodity data center network architecture, *ACM SIGCOMM computer communication review*, vol. 38(4), pp. 63–74, 2008.

[5] AlEroud A., Alsmadi I.: Identifying cyber-attacks on software defined networks: An inference-based intrusion detection approach, *Journal of Network and Computer Applications*, vol. 80, pp. 152–164, 2017.

[6] Alharbi T.: Deployment of blockchain technology in software defined networks: A survey, *IEEE Access*, vol. 8, pp. 9146–9156, 2020.

[7] Ali S.T., Sivaraman V., Radford A., Jha S.: A survey of securing networks using software defined networking, *IEEE transactions on reliability*, vol. 64(3), pp. 1086–1097, 2015.

[8] Alsmadi I., Xu D.: Security of software defined networks: A survey, *Computers & Security*, vol. 53, pp. 79–108, 2015.

[9] Alupotha J., Prasadi S., Alawatugoda J., Ragel R., Fawsan M.: Implementing a proven-secure and cost-effective countermeasure against the compression ratio info-leak mass exploitation (CRIME) attack. In: *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, 2017.

[10] Ashouri M., Setayesh S.: Enhancing the Performance and Stability of SDN Architecture with a Fat-Tree Based Algorithm, 2018. https://hal.archives-ouvertes.fr/hal-01858528. (working paper or preprint).

[11] Aviram N., Schinzel S., Somorovsky J., Heninger N., Dankel M., Steube J., Valenta L., Adrian D., Halderman J.A., Dukhovni V., *et al.*: {DROWN}: Breaking {TLS} Using {SSLv2}. In: *25th USENIX Security Symposium (USENIX Security 16)*, pp. 689–706, 2016.

[12] Benton K., Camp L.J., Small C.: OpenFlow vulnerability assessment. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pp. 151–152, 2013.

[13] Berde P., Gerola M., Hart J., Higuchi Y., Kobayashi M., Koide T., Lantz B., O'Connor B., Radoslavov P., Snow W., *et al.*: ONOS: towards an open, distributed SDN OS. In: *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, 2014.

[14] Bhargavan K., Leurent G.: On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 456–467, 2016.

[15] Bhargavan K., Leurent G.: Transcript collision attacks: Breaking authentication in TLS, IKE, and SSH. In: *Network and Distributed System Security Symposium–NDSS 2016*, 2016.

[16] Bianchi G., Bonola M., Capone A., Cascone C.: Openstate: Programming platform-independent stateful openflow applications inside the switch, *ACM SIGCOMM Computer Communication Review*, vol. 44(2), pp. 44–51, 2014.

[17] Bianchi G., Bonola M., Pontarelli S., Sanvito D., Capone A., Cascone C.: Open Packet Processor: a programmable architecture for wire speed platform-independent stateful in-network processing, *arXiv preprint arXiv:160501977*, 2016.

[18] Bosshart P., Daly D., Gibb G., Izzard M., McKeown N., Rexford J., Schlesinger C., Talayco D., Vahdat A., Varghese G., *et al.*: P4: Programming protocol-independent packet processors, *ACM SIGCOMM Computer Communication Review*, vol. 44(3), pp. 87–95, 2014.

[19] Chica J.C.C., Imbachi J.C., Vega J.F.B.: Security in SDN: A comprehensive survey, *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[20] Chole S., Fingerhut A., Ma S., Sivaraman A., Vargaftik S., Berger A., Mendelson G., Alizadeh M., Chuang S.T., Keslassy I., *et al.*: dRMT: Disaggregated Programmable Switching. In: *SIGCOMM'17: Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 1–14, 2017.

[21] Fisher D.: CRIME Attack Uses Compression Ratio of TLS Requests as Side Channel to Hijack Secure Sessions, *ThreatPost*, 2012.

[22] Floodlight Project. http://www.projectfloodlight.org/.

[23] Foster N., Harrison R., Freedman M.J., Monsanto C., Rexford J., Story A., Walker D.: Frenetic: A network programming language, *ACM Sigplan Notices*, vol. 46(9), pp. 279–291, 2011.

[24] Gao J., Zhai E., Liu H.H., Miao R., Zhou Y., Tian B., Sun C., Cai D., Zhang M., Yu M.: Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics. In: *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 435–450, 2020.

[25] He C., Feng X.: Pomp: protocol oblivious sdn programming with automatic multi-table pipelining. In: *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 998–1006, IEEE, 2018.

[26] Hong S., Xu L., Wang H., Gu G.: Poisoning network visibility in software-defined networks: New attacks and countermeasures. In: *NDSS*, vol. 15, pp. 8–11, 2015.

[27] Hsu K.F., Beckett R., Chen A., Rexford J., Walker D.: Contra: A programmable system for performance-aware routing. In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 701–721, 2020.

[28] Huang S., Zhao J., Wang X.: HybridFlow: A Lightweight Control Plane for Hybrid SDN in Enterprise Networks. In: *the Proceedings of IEEE/ACM Twenty Fourth International Symposium on Quality of Service*, pp. 1–2, 2016.

[29] Jin C., Lumezanu C., Xu Q., Mekky H., Zhang Z.L., Jiang G.: Magneto: Unified Fine-Grained Path Control in Legacy and Openflow Hybrid Networks. In: *Proceedings of ACM Symposium on SDN Research*, pp. 75–87, 2017.

[30] Jin C., Lumezanu C., Xu Q., Zhang Z.L., Jiang G.: Telekinesis: Controlling Legacy Switch Routing with Openflow in Hybrid Networks. In: *Proceedings of ACM SIGCOMM Symposium on Software Defined Networking Research*, pp. 1–7, 2015.

[31] Khan S., Gani A., Wahab A.W.A., Guizani M., Khan M.K.: Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art, *IEEE Communications Surveys & Tutorials*, vol. 19(1), pp. 303–324, 2016.

[32] Koponen T., Casado M., Gude N., Stribling J., Poutievski L., Zhu M., Ramanathan R., Iwata Y., Inoue H., Hama T., *et al.*: Onix: A distributed control platform for large-scale production networks. In: *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[33] Kreutz D., Ramos F.M., Verissimo P.E., Rothenberg C.E., Azodolmolky S., Uhlig S.: Software-defined networking: A comprehensive survey, *Proceedings of the IEEE*, vol. 103(1), pp. 14–76, 2014.

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

[35] Latif S.A., Wen F.B.X., Iwendi C., Li-li F.W., Mohsin S.M., Han Z., Band S.S.: AI-empowered, blockchain and SDN integrated security architecture for IoT network of cyber physical systems, *Computer Communications*, vol. 181, pp. 274–283, 2022.

[36] Lee S., Shin Y., Hur J.: Return of version downgrade attack in the era of TLS 1.3. In: *Proceedings of the 16th International Conference on Emerging Networking Experiments and Technologies*, pp. 157–168, 2020.

[37] McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.: OpenFlow: enabling innovation in campus networks, *ACM SIGCOMM computer communication review*, vol. 38(2), pp. 69–74, 2008.

[38] Meng W., Li W., Zhou J.: Enhancing the security of blockchain-based software defined networking through trust-based traffic fusion and filtration, *Information Fusion*, vol. 70, pp. 60–71, 2021.

[39] Merget R., Brinkmann M., Aviram N., Somorovsky J., Mittmann J., Schwenk J.: Raccoon Attack: Finding and Exploiting {Most-Significant-Bit-Oracles} in {TLS-DH (E)}. In: *30th USENIX Security Symposium (USENIX Security 21)*, pp. 213–230, 2021.

[40] Microsoft Research. https://mitls.org/pages/attacks/3SHAKE.

[41] Möller B., Duong T., Kotowicz K.: This POODLE bites: exploiting the SSL 3.0 fallback, *Security Advisory*, vol. 21, pp. 34–58, 2014.

[42] Monsanto C., Reich J., Foster N., Rexford J., Walker D.: Composing software defined networks. In: *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 1–13, 2013.

[43] Moshref M., Bhargava A., Gupta A., Yu M., Govindan R.: Flow-level state transition as a new switch primitive for SDN. In: *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 61–66, 2014.

[44] OpenDayLight Project. http://www.opendaylight.org/.

[45] OpenFlow Switch Specifications Version 1.0. https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.

[46] OpenFlow Switch Specifications Version 1.5. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[47] Pandya B., Parmar S., Saquib Z., Saxena A.: Framework for securing SDN southbound communication. In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1–5, IEEE, 2017.

[48] Pontarelli S., Bifulco R., Bonola M., Cascone C., Spaziani M., Bruschi V., Sanvito D., Siracusano G., Capone A., Honda M., *et al.*: {FlowBlaze}: Stateful Packet Processing in Hardware. In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 531–548, 2019.

[49] POX Controller. https://github.com/noxrepo/pox/.

[50] Shahbaz M., Choi S., Pfaff B., Kim C., Feamster N., McKeown N., Rexford J.: Pisces: A programmable, protocol-independent software switch. In: *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 525–538, 2016.

[51] Shin S., Song Y., Lee T., Lee S., Chung J., Porras P., Yegneswaran V., Noh J., Kang B.B.: Rosemary: A robust, secure, and high-performance network operating system. In: *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 78–89, 2014.

[52] Sivaraman A., Budiu M., Cheung A., Kim C., Licking S., Varghese G., Balakrishnan H., Alizadeh M., McKeown N.: Packet Transactions: High-Level Programming for Line-Rate Switches, 2016.

[53] Sivaraman A., Cheung A., Budiu M., Kim C., Alizadeh M., Balakrishnan H., Varghese G., McKeown N., Licking S.: Packet transactions: High-level programming for line-rate switches. In: *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 15–28, 2016.

[54] Song H.: Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane. In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 127–132, 2013.

[55] Voellmy A., Kim H., Feamster N.: Procera: A language for high-level reactive network control. In: *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 43–48, 2012.

[56] Wu X., Li P., Miskell T., Wang L.M., Luo Y., Jiang X.: Ripple: An Efficient Runtime Reconfigurable P4 Data Plane for Multicore Systems. In: *2019 International Conference on Networking and Network Applications (NaNA)*, pp. 142–148, IEEE, 2019.

[57] Yoon C., Lee S., Kang H., Park T., Shin S., Yegneswaran V., Porras P., Gu G.: Flow wars: Systemizing the attack surface and defenses in software-defined networks, *IEEE/ACM Transactions on Networking*, vol. 25(6), pp. 3514–3530, 2017.

[58] Zhang Y., Lin X., Xu C.: Blockchain-based secure data provenance for cloud storage. In: *International conference on information and communications security*, pp. 3–19, Springer, 2018.

[59] Zhu S., Bi J., Sun C., Wu C., Hu H.: Sdpa: Enhancing stateful forwarding for software-defined networking. In: *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*, pp. 323–333, IEEE, 2015.

Affiliations

Suhail Ahmad
Suhail Ahmad Mir, University of Kashmir, Department of Computer Science and Engineering, India, suhail.sam008@gmail.com

Ajaz Hussain Mir
Ajaz Hussain Mir, National Institute of Technology, Electronics and Communication Department, Srinagar, Jammu & Kashmir, India, ahmir@rediffmail.com

Received: 20.12.2020
Revised: 11.11.2022
Accepted: 01.01.2023