Hindawi
Security and Communication Networks
Volume 2020, Article ID 6302739, 18 pages
https://doi.org/10.1155/2020/6302739

Research Article

Attribute-Guard: Attribute-Based Flow Access Control Framework in Software-Defined Networking

Xianwei Zhu, ChaoWen Chang, Qin Xi, and ZhiBin Zuo

PLA Strategic Support Force Information Engineering University, Zhengzhou 45004, China

Correspondence should be addressed to Xianwei Zhu; zhu13939055330@163.com

Received 24 May 2019; Revised 18 October 2019; Accepted 16 November 2019; Published 10 January 2020

Academic Editor: Roberto Di Pietro

Software-defined networking (SDN) decouples the control plane from the data plane, offering flexible network configuration and management. Because of this architecture, some security features are missing. On the one hand, because the data plane only has the packet forwarding function, it is impossible to effectively authenticate the data validity. On the other hand, OpenFlow can only match based on network characteristics, and it is impossible to achieve fine-grained access control. In this paper, we aim to develop solutions to guarantee the validity of flow in SDN and present Attribute-Guard, a fine-grained access control and authentication scheme for flow in SDN. We design an attribute-based flow authentication protocol to verify the legitimacy of the validity flow. The attribute identifier is used as a matching field to define a forwarding control. The flow matching based on the attribute identifier and the flow authentication protocol jointly implement fine-grained access control. We conduct theoretical analysis and simulation-based evaluation of Attribute-Guard. The results show that Attribute-Guard can efficiently identify and reject fake flow.

1. Introduction

Software-defined networking (SDN) [1] is a new network structure proposed by Clean Slate team of Stanford University. It separates the control plane from the data plane and enables high programmability and dynamic orchestration. In the basic SDN architecture, as shown in Figure 1, there are three layers, the application plane, the control plane, and the data plane. The control plane dictates network behaviors and configures network devices via a set of flow rules that control the network traffic flows. The data plane only has the function of data forwarding, which makes it difficult to monitor the data source by controller and cannot achieve end-to-end data authentication. For SDN flow table forwarding, there is no effective access control framework [2], which can prevent forgery attacks.

A fully functional access and forwarding control framework should have three points: (1) preventing illegal users from accessing network services; (2) giving legitimate users appropriate permissions to access protected services or resources; (3) preventing legitimate users from accessing network services that do not give user permissions. Because OpenFlow can only be based on the first four layers of network protocols controlling the forwarding, it cannot divide network services and is unable to achieve fine-grained access control. Attackers attack SDN using legitimate devices as springboards. Therefore, ensuring the legality and correctness of flow access SDN and preventing the proliferation of malicious flow are clearly the main challenges in SDN security.

Digital signature as a tool for validation of data has been widely used in the operating system and network. FortNOX [3] encrypts and authenticates applications. Attribute signature [4, 5] enables users to achieve fine-grained access control without access list. By changing the access attributes, the access control structure is updated. Based on above these, the attribute signature meets diverse security needs [5].

1.1. Our Approach. This paper analyses the shortcomings of SDN's access control frameworks. According to lack of flow authentication and fine-grained network access control, we

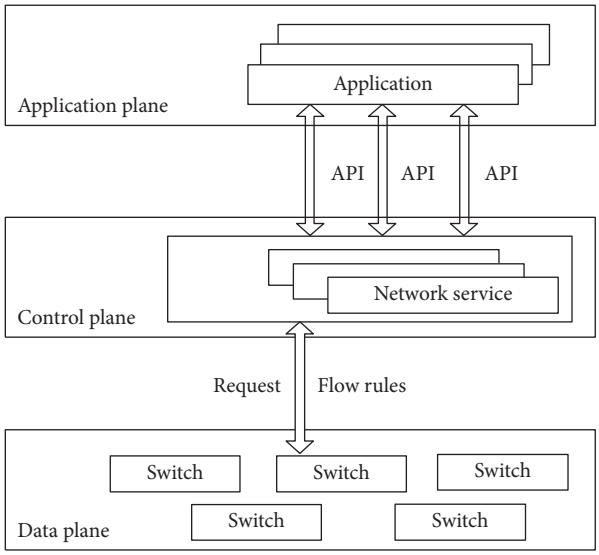

Figure 1: SDN architecture.

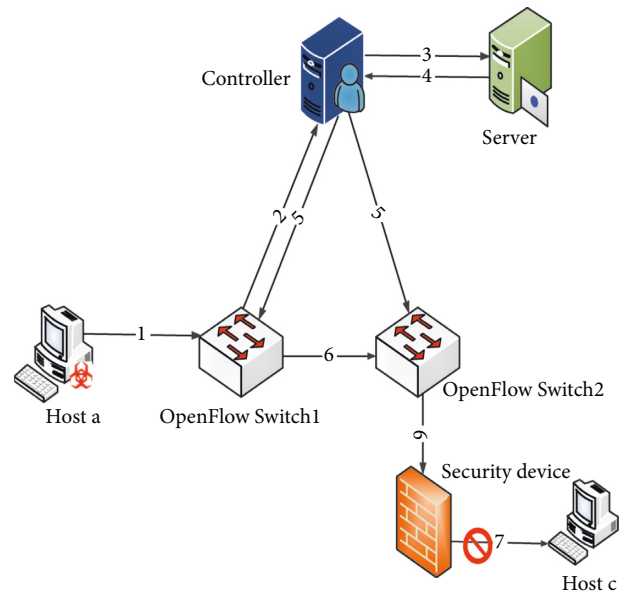

Figure 2: The abstract SDN communication model.

combine SDN with attribute signature and propose SDN security access control framework based on the attribute identifier—Attribute-Guard. It uses device attributes to generate attribute identifiers. Attribute-Guard manages attribute identifiers in each packet that defines network forwarding. To ensure authentication of each flow, we introduced an attribute-based signature scheme in switch to authenticate the flow based on their signature, thus to prevent invalid flow launching continuous malicious attacks to the network. As a result, the Attribute-Guard can implement fine-grained access control and data source identification based on network services.

To sum up, the main contributions of this paper are as follows:

(i) We propose Attribute-Guard, a fine-grained flow access framework. The proposed framework redefines the SDN forwarding framework that binds the flow with its device's attribute identifiers (AIDs).

(ii) We present flow authentication protocol that can effectively prevent fake flow and filter invalid flow created by an attacker in unauthorized manners, and it has a fine-grained management.

(iii) We prototype our approach in the OpenDaylight controller and evaluate the system performance. The theoretical analysis and experimental results demonstrate that the proposed framework can effectively prevent the forging flow attack and implement fine-grained access control.

1.2. Background and Motivation. The abstract SDN communication model contains the following elements: (1) hosts; (2) SDN controller; (3) application that provides flow rules for controllers; (4) OpenFlow switch; and (5) security devices, such as firewall and security gateways.

If host a (a malicious attacker) wants to attack host c, the attack proceeds as shown in Figure 2: (1) host a sends a request to OpenFlow Switch1; (2) if there is no flow rule matching the request from host a in OpenFlow Switch1, OpenFlow Swich1 sends the request to the controller and waits for the response; (3) application receives a flow rule production request from the controller; (4) the application produces a flow rule and sends it to the controller; (5) the controller receives the flow rule, and it saves the flow rule in the flow rule database and forwards the flow rule to OpenFlow Switch1 and OpenFlow Switch2; (6) the new flow rule matches the packets from host a and forwards the packets to the security device through OpenFlow Switch1 and OpenFlow Switch2, respectively; and (7) the security device detects the packet according to the rule, and it determines that host a cannot communicate with host c.

According to the SDN structure, we constructed two attack methods [6,7], which achieve illegal access by tampering with flow rules and controlling flow. There are other direct attack scenarios, such as DDos:

(i) As shown in Figure 3, the attacker tampers with the flow rule in OpenFlow Switch2, which allows malicious host a to access host c directly. Thus, the packets from host a can bypass the security device and scan host c.

(ii) As shown in Figure 4, Malicious Application B generates two new flow rules. The first flow rule is used to modify the source IP address of the packet from the malicious host a to the source IP address of host b that can access host c. Then, the second flow rule changes the destination address of the packet to the IP address of host a. If the packet is delivered from host c to host a, the security device simply allows forwarding the packet from host b to host c. In this way, the packets from host a can bypass the security device. The host can scan host c.

The two examples in Figures 3 and 4 show that an attacker can tamper with flow rules and control flow to

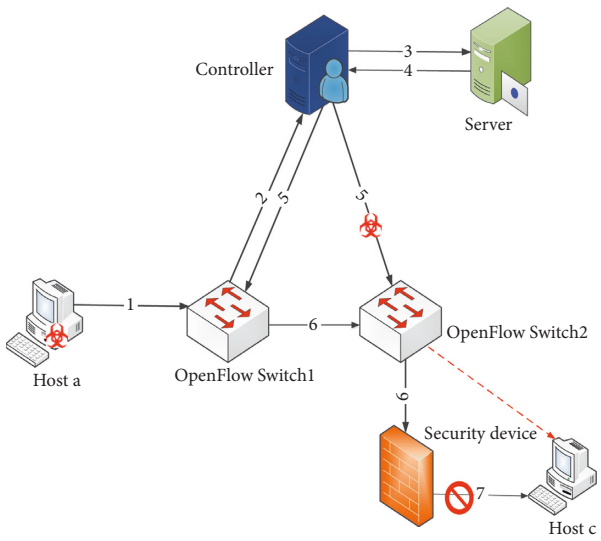

FIGURE 3: Example 1: an attacker tampers with flow rules and uses it to allow malicious packets bypass a security device.

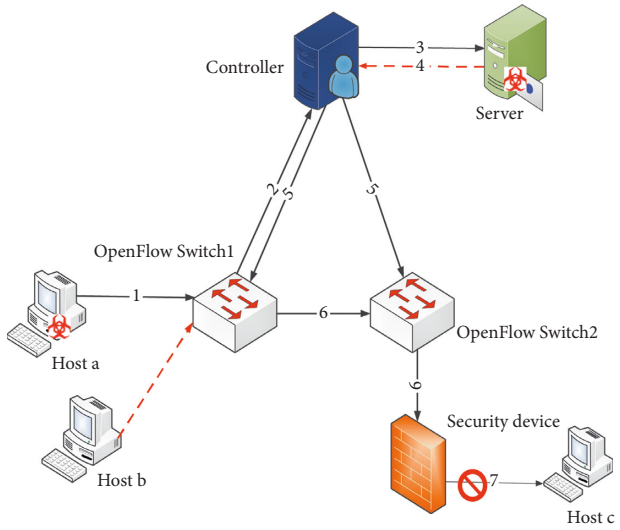

FIGURE 4: Example 2: an attacker adds flow rules to change the IP address of the packet and thus to mislead the security device.

circumvent the security device deployed on fixed paths. The essence of the aforementioned problems is the illegal access of the flow. Therefore, it is necessary to implement user authentication in the data plane. Our goal in this paper is to present a fine-grained flow access framework.

1.3. Related Work. There are two ideas to prevent illegal access in the network: (1) the controller authenticates the flow rules and (2) the controller authenticates the devices entering the SDN and sends network strategy. The first idea is mainly based on the flow rules of role authentication. Porras et al. introduced FortNOX [8], a security enforcement kernel on the NOX controller which provides a role-based authorization scheme for applications that produce flow rules, in response to perceived runtime requests. It manages applications that create flow rules by default three authorization roles, including the role of administrator, the role of security applications, and the role of nonsecurity applications. These roles are assigned to each application, and each application is required to sign its flow rules. Then, Porras extended the scheme and proposed a new security system named SE-Floodlight [9]. The system introduced security enforcement kernel to the Floodlight controller, implementing role-based flow rule management. Similarly, RoseMary [10] and FRESCO manage flow rules by using roles [11]. Although above methods can defend against illegal flow, these may assign the same role to different security applications and unable to achieve fine-grained management of flows. Thus, Wen et al proposed a set of 18 permissions based on the interface of the controller and used a system called PermOF to distribute privileges. Compared with SE-Floodlight [9], RoseMary [10], and FRESCO [11], WEN achieves fine-grained flow management, but it cannot prevent forgery attacks at terminals. Based on the second idea, Lopez-Millan et al. [12] introduced a terminal management system, which protects the terminal by using the IPsec protocol and proposed a solution to manage IPsec SAs using SDN. But it did not divide the role of the terminal and could not achieve the fine-grained management of the flow. However, these two ideas will greatly increase the load of the controller and reduce the performance of the controller.

Therefore, researchers had shifted their research direction from controllers to switches and SDN architectures. They implement data plane security by modifying the protocol stack. Lopez-Millan et al. [13] described the use case of providing IPsec-based flow protection in SDN, but it lacks fine-grained management. As a supplement, Wundsam et al. implemented OFRewind architecture [14], a new architecture to authorize the device in the network layer and support multiple granularity management. Halpern et al. studied the service function chain SFC [15], using flow rules to control the flow. Caprolu et al. designed FORTRESS: a stateful firewall for SDN networks that leverages the stateful data plane architecture to move the logic of the firewall from the control plane to the data plane [16]. Fayaz et al. implemented Bohatei, a flexible and elastic DDoS defense system [17].

According to security requirement, the flow is divided into different security levels assigning different function chains and function link paths to flow of different security levels. On the other contrary, IEEE 802.1X offers secure and flexible authentication mechanisms. Garay et al. [18] proposed flow-based network access control (FlowNAC) which is a modified version of IEEE 802.1X standard and has the advantage to authorize access on the basis of flow nature. Benzekki [19] introduced a secure SDN architecture with IEEE 802.1X port-based authentication, which devolves the access control capability to the data plane.

Southbound protocol is mainly an OpenFlow protocol, and it defines type and field limited to four network layer protocols. It provides limited forwarding granularity [20]. However, after OpenFlow v1.2 [21], matching fields use the TLV format of OXM architecture, which makes it possible to expand the scope of matching fields. Atting et al. designed the parsing language PPL [22], which introduced a packet

header parsing algorithm, and improved the packet performance in addition to the packet header parsing. Arashloo et al introduced SNAP that offers a simpler "centralized" stateful programming model, by allowing programmers to develop programs on top of one big switch [23].

1.4. Organization. This paper is organized as follows: in Section 2, we describe overall architecture of Attribute-Guard. Section 3 introduces the attribute-based group signature scheme. Section 4 proposes a flow table processing pipeline based on the attribute identifier. In Section 5, we evaluate Attribute-Guard on security functionalities, time consumption, performance, and availability. Finally, we conclude our study in Section 5.

2. Overview of the Attribute-Guard Framework

In this section, we describe the Attribute-Guard framework, which is a fine-grained flow access control mechanism to ensure the validity of flow in SDN.

2.1. Overall Architecture. As Figure 5 shows, the conceptual diagram of Attribute-Guard includes four components: an attribute identifier authority, an attribute identifier component, a control plane based on the attribute identifier, and a data plane based on the attribute identifier. Attribute-Guard provides following security functionalities: (1) managing the attribute cipher set of a valid host; (2) verifying the validity of flow; and (3) defining the forwarding behaviour based on the attribute identifier.

(i) Attribute identifier authority: the attribute identifier authority generates system public parameters and access public parameters for the destination device. On the contrary, it generates an attribute identifier for the source device and uses the attribute identifier to generate the attribute private key.

(ii) Attribute identifier component: the attribute identifier component is an application installed on the host. First, it is responsible for generating an attribute set for the new source/destination device. Second, it obtains the attribute private key and the attribute identifier from the attribute identifier authority and generates a packet of flow authentication. The attribute identifier encapsulation is implemented by modifying the protocol stack of the host, and the host is not physically extended. Finally, it stores access structure T (attribute set for verifying the signature, as described in detail in Section 3.1).

(iii) Control plane based on the attribute identifier: the control plane includes packet parsing module, attribute identifier invalidation module, and a flow rule generation module. By default, there are two main functions: (1) Obtaining network topology by collecting data plane information and (2) generating flow rules based on attribute identification to implement forwarding control in the data plane.

(iv) Data plane based on the attribute identifier: the data plane based on the attribute identifier is mainly composed of an authentication switch and a forwarding switch. The authentication switch located between the backbone network and the access network authenticates the identity of flow. It mainly consists of flow authentication module and flow table module. The flow authentication module uses the signature generated by the attribute identifier to authenticate the validity of flow. The flow table module verifies the validity of AID by means of flow table matching and forwards legitimate flow to a specified location. The forwarding switch located in the backbone network only has the flow table module, and the module directly matches and forwards the received flow.

2.2. Communication Using Attribute-Guard. We use the source host H1 to access the destination host H2 as an example to describe the communication process of the Attribute-Guard in Figure 6.

(i) Before accessing the network, H1 needs to be initialized by a locally installed attribute identifier component and H1 sets its attribute and destination address upload to attribute identifier authority. Then, it receives the attribute private key and generates the attribute signature from the attribute identifier authority.

(ii) When H1 needs to communicate with H2, it sends a flow authentication request to the source authentication switch (SAS) that connects H1. It forwards the authentication request to the destination authentication switch (DAS) that connects H2 through the controller. DAS verifies the validity of the flow and returns the results to the controller. Then, the controller generates the flow rules with attribute identification.

(iii) The forwarding switch receives the packet from DAS, and it directly performs the matching and forwarding.

(iv) The destination authentication switch receives the packet from the forwarding switch and uses the flow table module to authenticate the validity of the attribute identifier.

(v) The destination H2 receives the legal packet to generate a general IP packet by using the local attribute identifier component to decapsulate.

2.3. Generate Pocket with Attribute Identification. The new host H1 needs to define the identity of the device before accessing the network. The traditional method is to generate a unique device ID for each device and use an ID identity table to verify the ID, resulting in an increase in system overhead. In fact, the network needs a few attributes to determine the identity of the device, and the user is a legitimate user as long as certain attributes of the user meet the

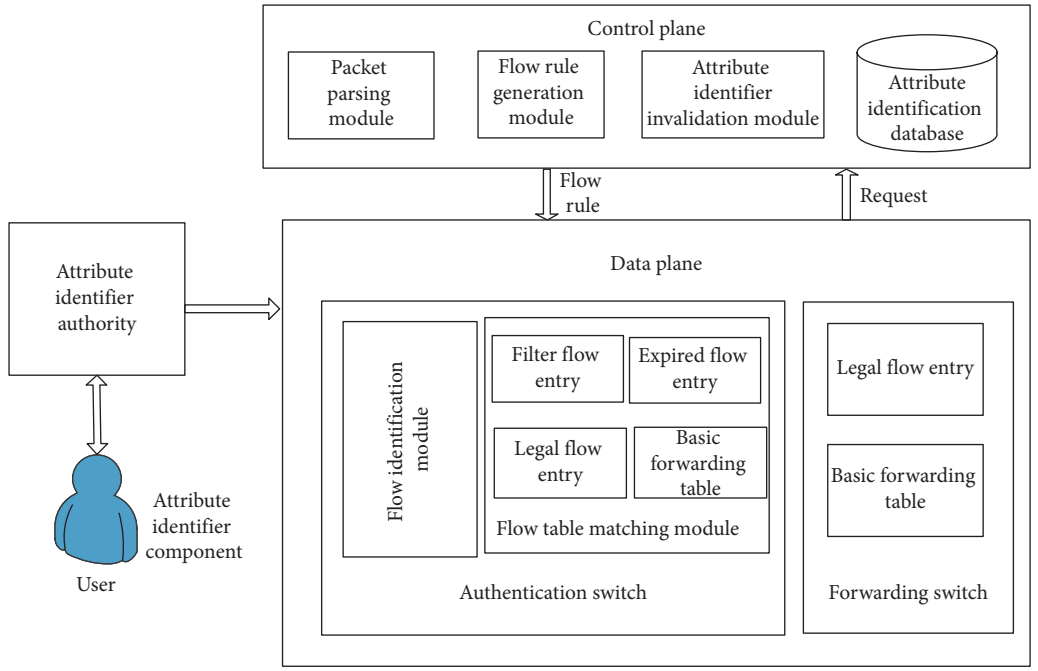

Figure 5: The conceptual diagram of Attribute-Guard.

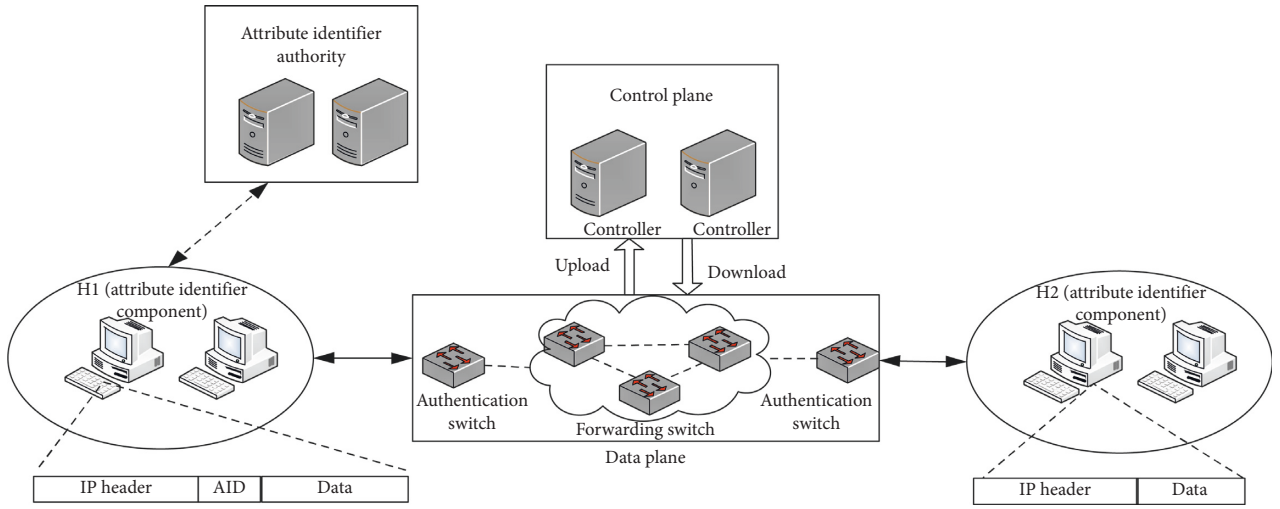

Figure 6: The communication model of the Attribute-Guard.

requirements. The authenticator does not care about the signer's name, address, and other irrelevant information. The platform defines device attributes from the department where the device is located, the role of the owner, and the business involved. These attributes are represented by Boolean functions. For example, Dan is an IT department engineer who needs to write to a host. His attribute set is IT department ∧ enginer ∧ write.

Assuming that the attribute identifier component receives the attribute that is not forged, we define hash functions to create attribute's length as we just desire: H: $\{0,1\}^* \longrightarrow Z_q^*$. We use an attribute string as the attribute identifier (AID). The AID is used as a license for packets to enter and leave the network. It is located between the network layer and the transport layer. We will encapsulate AID for all packets. The message structure is shown in Table 1.

(i) Version (4 digits): the version of the attribute identifier

(ii) Secondary protocol (8 bit): the type of the protocol immediately following the attribute identifier, such as 6 (TCP), 17 (UDP), or 50 (ESP)

(iii) Length (8 bits): the value is the length of the entire attribute identifier (including the header and metadata) in units of 32 bits (4 bytes)

Table 1: Attribute identifier message format.

Version (4 bit)	Secondary protocol (8 bit)	Length (8 bit)	Reserved (12 bit)
Serial number (32 bit)			
AID (64 bit)			
Data			

(iv) Reserved (12 bit): to be used in future extensions to attribute identification, and the current protocol specifies that this field should be set to zero

We modify the protocol type of the IP header and add AID after the IP header. For example, if the upper protocol without AID is TCP, the protocol type of the IP header is changed to 6.

3. Attribute-Based Group Signature Scheme

Our attribute-based signature scheme is an extension of the attribute-based group signature scheme presented by Dalia Khader et al. [24].

3.1. Access Structure. The access structure T is an authorization set of attributes for verifying the signature, defined by a verifier. When a verifier requests a signature of a host who satisfies certain attributes, a host will use his different private keys to generate the signature according to the verifier's access structure. The table is a linear structure. If the verifier uses a table to represent the access structure, the verification algorithm will be run as many times as the number of attributes in the signature, thus compromising efficiency. We use the attribute tree Γ that is a nonlinear structure to describe the access structure, and its constructor is based on the constructor presented by Goyal et al. [25]. Each root node in the attribute tree has a threshold value, and each attribute is connected to it as its leaf node. Each threshold value indicates the number of conditions that needs to be met in the leaf node to which it is connected; that is, the number of attributes required under the root node. The access tree is shown in Figure 7.

We use the attribute tree to generate the public key. Only the signature of the user who meets the requirements of the attribute tree can pass the verification. As shown in Figure 7, the administrator of the IT department needs to perform the read operation. The user satisfies the attribute tree, so the signature can be verified. Engineers in the IT department want to perform a read operation that does not meet the requirements of the attribute tree and cannot be verified.

3.2. Authentication Process. The attribute signature verification and update the access control structure are implemented by the attribute identifier component, the attribute identifier authority, and the authentication switch. Access to services under Attribute-Guard architecture requires two basic stages:

Step 1: it includes five processes and is shown in Figure 8. Firstly, the attribute identifier authority performs initialization to generate system public parameters; the authentication switch creates an access control structure T according to requirements, and it uploads T to the attribute identifier authority. The attribute identifier authority uses the primary private key and the system public parameter to generate the public parameter of T and stores it in the authentication switch; the attribute identifier component uploads the host's attribute set to the attribute identifier authority. According to the host's attribute, the attribute identifier authority generates the attribute identifier and attribute private key and returns it to the attribute identification component. Therefore, different hosts have different attribute identifiers and attribute private keys. If the attacker controls a legitimate host, DAS will update the access structure T to generate the new public parameter of T. The original valid attributes and the signature acquired by the attacker will be invalid.

Step 2: flow identification. Based on the parameters generated by step 1, the authentication switch completes the flow identification, and the process is shown in Figure 9.

(1) The host initiates an authentication request to the controller. The request packet contains the following message: {host's AID, signature, mac, destination mac, source IP, destination IP, source port, destination port}.

(2) The controller successfully receives the authentication request packet from the source authentication switch (SAS) port. To prevent malicious users from using the authentication request packet to initiate a malicious DDoS attack, the default flow table will be installed on the SAS and the authentication packet from the port will be discarded in the T period.

(3) According to the authentication request information, the controller forwards the request to the destination authentication switch (DAS). The destination authentication switch acquires the access structure T of the destination host according to the request.

(4) The destination authentication switch verifies the signature and sends the result protected by SSL to the controller.

(5) The controller receives the result; if the flow is legal, it sends the flow rule with the attribute identifier to switch; otherwise, the connection is refused.

3.3. Formalization of the Scheme. The relevant definitions of attribute-based group signatures are given below.

Definition 1. The attribute tree Γ is used to represent the access structure T, and the attribute tree uses the top-down-left-right order. The root node is represented as (m, n), where m indicating $n$the threshold value and n indicating the number of leaf nodes. κ indicates the number of leaves in the attribute tree, as shown in the attribute tree in Figure 5:

$$\Gamma = \{(2,2), \text{IT}, (1,2), (2,2), (2,2), \text{administrator}, (1,2), \text{engineer}, (1,2), \text{read}, \text{write}, \text{notification}, \text{write}\}. \quad (1)$$

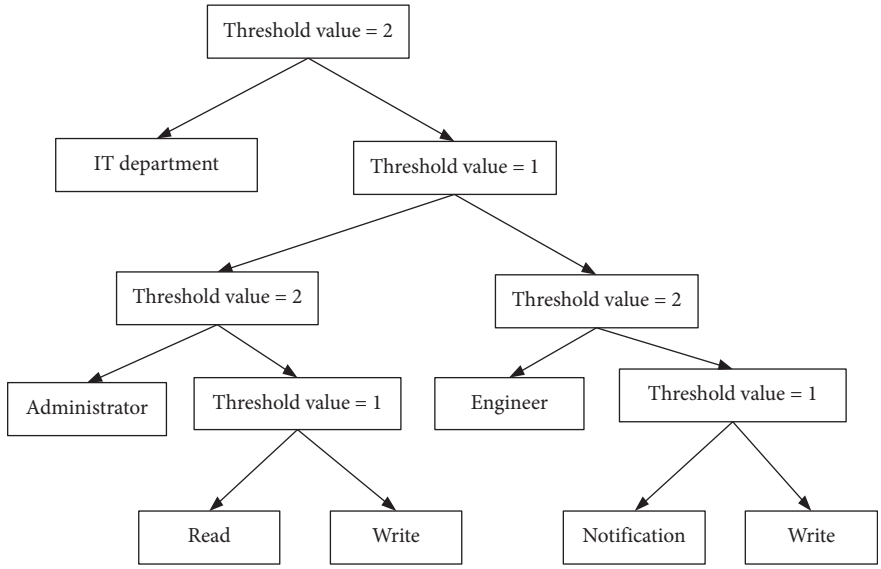

Figure 7: Attribute tree structure.

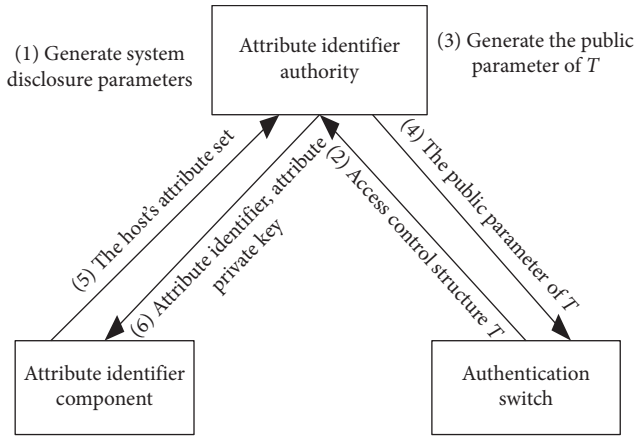

Figure 8: System initialization.

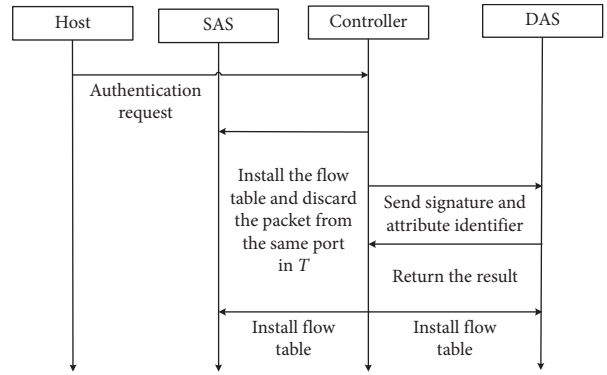

Figure 9: Flow identification process.

Definition 2. γ_i represents the private key owned by each user. μ indicates the number of private keys.

Definition 3. User uses the attribute to signature, and the element in ζ_i satisfies γ_i, that is, $\zeta_i \subseteq \gamma_i$, and the number of ζ_i is τ, for example, $\Gamma = \{(1, 2), \text{administrator}, \text{engineer}\}$, that the employee i can use $\zeta_i = \{\text{engineer}\}$ to verify.

The algorithm includes the following: Setup, KeyGen, Sign, and Verify.

Setup: the attribute identifier authority chooses a bilinear pair e: $G_1 \times G_2 \longrightarrow G_T$, where G_1, G_2, and G_T are multiplicative loop groups and of prime order p. g_2 is the generator of G_2, and there exists homomorphic mapping $g_1 \longleftarrow \psi_2$. The system chooses an open hash function: H: $\{0,1\}^* \longrightarrow Z_P^*$. Choose $h \in G_1$ and ξ_1, ξ_2 ($\xi_1, \xi_2 \in Z_P^*$). Set $u, v \in g_1$ such that $u^{\xi_1} = v^{\xi_2} = h$, and randomly select $\omega \in Z_P^*$. We set $W = g_2$. Define a universe of attributes $U = \{1, 2, \ldots, n\}$ and each $j \in U$. Select $t_f \in Z_p^*$, and then calculate the public parameters and system secret parameters:

$$PK = \{G_1, G_2, G_T, g_1, g_2, e, H, h, u, v, W\},$$
$$MK = \langle \{t_i\}_{i \in U}, \omega, \xi_1, \xi_2 \rangle. \quad (2)$$

KeyGen: the process generates an attribute private key and a public parameter of the access control structure T. The system generates a base private key $gsk[i]_{\text{base}} = (A_i, x_i)$ for user $i (1 \le i \le n)$ through γ and it is an SDH pair, i.e., $A^i = g_1^{1/(\omega + x_i)} \in G_1$.

(1) $\text{KeyGen}_{\text{public}}(\Gamma)$: to generate a public key for an attribute tree Γ, we select a polynomial q_x for each nonleaf node in the middle and use a top-down construction method for the root node. The node x in the tree has a polynomial q_x whose number d_x is less than its threshold k_x, i.e., $d_x = k_x - 1$. The root node is q(0) = q(index (x)), and select the polynomial q to construct attribute tree polynomials recursively. Finally, the public parameters of the attribute tree Γ

are obtained by $D_x = g_2^{q_x(0)/t_j}$, $h_n = h^{t_i}$, and $i = \mathrm{att}(x)$. And send the system public parameters $TPK = \langle \{D_x, h_x\}_{x \in \gamma_\Gamma} \rangle$ to the authentication switch.

(2) $\mathrm{KeyGen}_{\mathrm{private}}(gsk[i]_{\mathrm{base}}, \gamma_i)$: we use attribute γ_i owned by user i (i.e., $j \in \gamma_i$) to generate private key $SK = \langle A_i, x_i, \{T_{i,j}\}_{j \in \gamma_i} \rangle$ by computer $\Gamma_{i,j} = A^{t_j}$.

Sign: when the system enters the attribute set $\gamma_i \subseteq U$, $j \subseteq \gamma_i$, the public parameters of the attribute tree, and the message m, it is calculated as follows:

(1) Select attribute $\zeta \subseteq \gamma_i$ and random number α, β, and $\mathrm{rnd} \in Z_p^*$.

(2) Calculate the linear encryption of A_i and T_{ij}, where the formula is as follows: $C_1 = u^a$, $C_2 = v^\beta$, $C_3 = A_i h^{\alpha+\beta}$, and $CT_j = (T_{i,j} h_j^{\alpha+\beta})^{\mathrm{rnd}}$.

(3) Let $\delta_1 = x_i \alpha$ and $\delta_2 = x_i \beta$. Select the random number r_β, r_x, r_{δ_1}, and r_{δ_2}. Calculate $R_1 = u^{\gamma_\partial}$, $R_2 = v^{\gamma_\beta}$, $R_5 = C_2^{\gamma_x} v^{-\gamma_{\delta_2}}$, $R_4 = C_1^{\gamma_x} u^{-\gamma_{\delta_1}}$, and $R_3 = \hat{e}(C_3, g_2)^{\gamma_x} \hat{e}(h, \omega)^{-\gamma_\alpha - \gamma_\beta} \hat{e}(h, g_2)^{-\gamma_{\delta_1} - \gamma_{\delta_2}}$.

(4) $c = H(M, C_1, C_2, C_3, R_1, R_2, R_3, R_4, R_5) \in Z_p^*$

(5) Construct $s_\alpha = (\gamma_\alpha + c\alpha)$, $s_\beta = (\gamma_\beta + c\beta)$, $s_x = (\gamma_x + cx)$, $s_{\delta_1} = (\gamma_{\delta_1} + c\delta_1)$, and $s_{\delta_2} = (\gamma_{\delta_2} + c\delta_2)$

(6) Let $\eta = \omega^{\mathrm{rnd}}$ and calculation signature $\sigma = (m, C_1, C_2, C_3, c, \{CT_i\}_{i \in \zeta}, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}, \eta)$. Finally, the signature and attribute identifier (σ, AID) are sent to the verifier, where the AID is the hash of the attribute set ζ.

Verify: it includes two steps. Firstly, we define a recursive algorithm Verifynode. For leaf nodes, calculate as follows:

$$\mathrm{Verifynode}(x) = \begin{cases} \hat{e}(CT_j, D_j) \text{if } j = \mathrm{att}(x)\ j \in \zeta, \\ \text{otherwise, return} \perp. \end{cases} \quad (3)$$

The result is $\hat{e}(CT_j, D_j) = \hat{e}(A_i h^{\alpha+\beta}, g_2^{\mathrm{rnd}})^{q_j(0)}$.

If the node x is not a leaf node, we perform the following steps: all child value z of node x are stored in function F We recursively calculate the value of the root node F_x using Lagrangian interpolation. Let $\Delta s_{x,\mathrm{index}(z)} = \prod(-j/(\mathrm{index}(z) - j))$, where $j \in \{\mathrm{index}(z): z \in s_x - \mathrm{index}(z)\}$, and compute

$$\begin{aligned} F_x &= \prod_{z \in s_x} F_z^{\Delta s_{x,\mathrm{index}(z)}}, \\ F_x &= \prod_{z \in s_x} \left(\hat{e}\left(A_i h^{\alpha+\beta}, g_2^{\mathrm{rnd}}\right)^{q_z(0)} \right)^{\Delta s_{x,\mathrm{index}(z)}}, \\ F_x &= \prod_{z \in s_x} \left(\hat{e}\left(A_i h^{\alpha+\beta}, g_2^{\mathrm{rnd}}\right)^{q_{\mathrm{parent}(z)}(\mathrm{index}(z))} \right)^{\Delta s_{x,\mathrm{index}(z)}}, \\ F_x &= \hat{e}\left(A_i h^{\alpha+\beta}, g_2^{\mathrm{rnd}}\right)^{q_x(0)}. \end{aligned} \quad (4)$$

Let $F_x = \hat{e}(C_3, \eta)$.

If it is established, it means that the signature satisfies the attribute tree Γ, and calculate $\overline{R}_1 = u^{s_\alpha} C_1^{-\kappa}$, $\overline{R}_2 = v^{s_\beta} C_2^{-\kappa}$, $\overline{R}_3 = \hat{e}(C_3, g_2)^{s_x} \hat{e}(h, W)^{-s_\alpha - s_\beta} \hat{e}(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} (\hat{e}(C_3, W)/\hat{e}(g_1, g_2))^\kappa$, $\overline{R}_4 = C_1^{s_x} u^{-s_{\delta_1}}$, and $\overline{R}_5 = C_2^{s_x} v^{-s_{\delta_2}}$. If $\kappa = H(m, C_1, C_2, C_3, \overline{R}_1, \overline{R}_2, \overline{R}_3, \overline{R}_4, \overline{R}_5)$, then accept the signature; otherwise, the signature is rejected.

3.4. Flow Authentication Protocol Design. We modify 802.1x protocol, which is widely used at present, and design a flow authentication protocol to support the above process.

The standard Ethernet frame must have the destination MAC address, DST MAC, and Ethernet frame type, Ether Type. When the flow authentication protocol is running, the host client program cannot obtain the destination host MAC address. Therefore, we adopt the default multicast address 01-80-C2-00-00-03, and the frame type is defined as 0X888F, which is different from 802.1x. The flow authentication protocol is carried out in the frame data portion. Since the maximum length of the Ethernet frame data is currently set to 1 518 bytes, the maximum length of the authentication protocol data is 1 500 bytes.

As illustrated in Table 2, the semantic of each field in the flow authentication protocol format is as follows:

Version: the version number of the current protocol

Type: type field. This field is mainly used to indicate the stage of the current data frame: 00 means registration and 01 means flow identification

Sequno: this field indicates the sequence number of the current packet, which prevents the loss and out of order of the packet

Length: this field is used to indicate the length of the data body

Data body: according to different identification stages, the corresponding data is carried.

4. Flow Matching Based on Attribute Identifier

The attribute identifier is designed as a match field that the OpenFlow switch can recognize, and a processing pipeline is designed based on the multiple flow tables. The validity of the attribute identifier can be authenticated by the processing pipeline, and the legal data are transferred to the specified location.

4.1. Structure of Flow Table. Due to the addition of the attribute identifier in the packet, the original OpenFlow needs to be extended as shown in Figure 10. According to OpenFlow 1.3 [18], we use the TLV format of the OXM architecture to define a new field called the attribute identifier. We add the attribute identifier in the Flow-Mod message so that the flow rule with the attribute identifier can be accepted by the authentication switch and forwarding switch. In this paper, the southbound protocol is compatible with the OpenFlow1.3 protocol. The controller that supports the OpenFlow1.3 protocol can generate the flow rule containing the attribute identifier.

Table 2: Flow authentication protocol format.

Version (8 bit)	Type (8 bit)	Sequno (8 bit)	Length (8 bit)
Data body (40–1498 byte)			

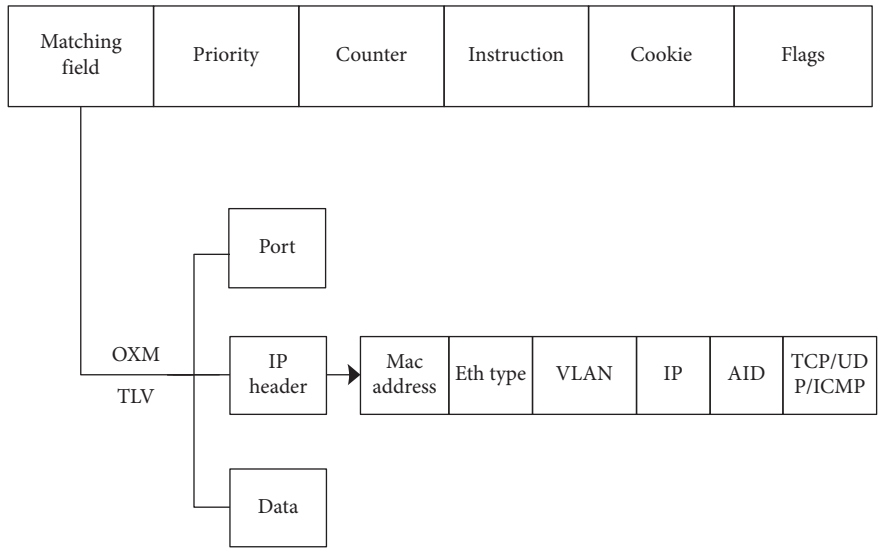

Figure 10: SDN flow entry structure based on attribute identification.

4.2. Multiple Flow Table Processing Pipeline. As illustrated in Figure 11, multiple flow table processing pipeline consists of two flow tables. The verification flow table is a level 0 flow table. It classifies the packets according to the type of the flow entry and selects an appropriate processing manner.

(1) Filter flow entry: the filter flow entry drops the packet without the attribute identifier (Ether-Type = 0x0800 or 0x86dd). If the packet does not have an attribute identifier, it will be sent to the expired flow entry.

(2) Expired flow entry: the expired flow entry is used to authenticate the validity of the user and quickly filters the packet with the expired attribute identifier. The field type is the invalid attribute identifier, and the flow table action is "drop." If the matching succeeds, the SDN switch will discard the packet.

(3) Legal flow entry: the legal flow entry will match the attribute identifier and forward the valid packet to the basic forwarding table for further matching.

(4) Table-miss flow entry: the table-miss flow entry is used to be compatible with non-IP packets and directly forwards non-IP packets to the basic forwarding table. The field type is ∗ (arbitrary), and the action is to jump to the basic forwarding table.

The basic forwarding flow table is the Level 1 flow table, which is used for direct matching forwarding of packets. It supports all matching fields and all types of instructions/actions. The matching process is shown in Figure 9.

5. Experiment and Evaluation

We have implemented and deployed an Attribute-Guard system. The system is based on OpenDaylight, we extend the function of the OpenDaylight controller and the OVS switch to complete the function deployment and use SFlow to monitor network traffic. Finally, the attribute-based group signature scheme is implemented through bilinear encryption library and C++ code. Next, we will evaluate it in terms of functional effectiveness, time consumption, and performance.

5.1. Experimental Environment. The experiment uses 10 computers as the experimental environment and its configuration is shown in Table 3. Six of them are used to simulate authentication switches and forwarding switches.

Its network topology is shown in Figure 12.

5.2. Functional Effectiveness Testing. We test the two attack scenarios in Section 1.2, indicating that the malicious host a cannot attack host c with the protection of Attribute-Guard:

(i) As shown in Figure 2, host a directly accesses the destination device c by tampering with the flow rule. However, Attribute-Guard uses the authentication switch to verify the flow of host a. Since the host does not satisfy the access structure of host c, the packets cannot reach host c.

(ii) An attacker can change the address of the packet by generating two new flow rules to scan host c. However, Attribute-Guard forwards the flow by

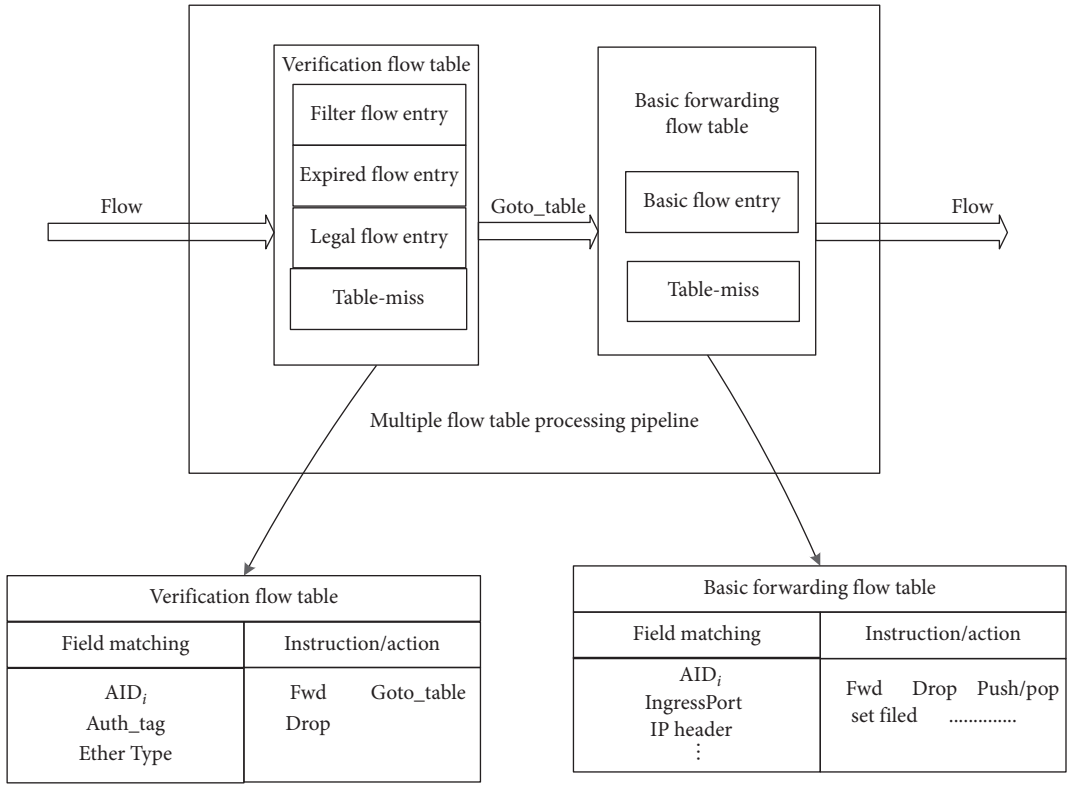

Figure 11: SDN flow entry structure based on attribute identification.

Table 3: Experimental environment configuration.

Function	Number	Configuration
Controller	1	i7-8400cpu, 16G DDR4, 4 network cards
Authentication switch	3	i7-8400cpu, 16G DDR4, 6 network cards
Forwarding switch	3	i7-8400cpu, 16G DDR4, 6 network cards
Host	6	i5-8400cpu, 8G DDR4, 2 network cards

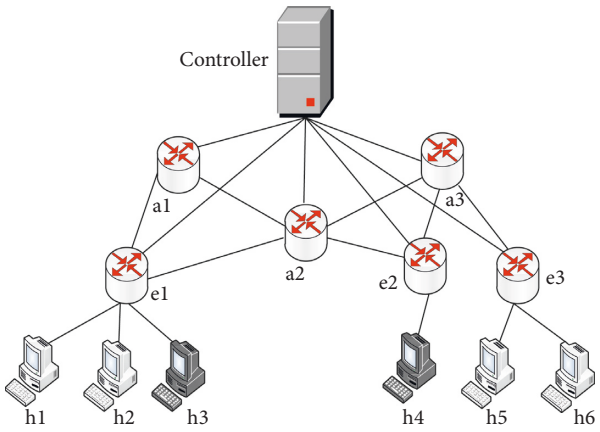

Figure 12: Experimental network topology.

multiple flow table processing pipeline and determines the flow path based on the attribute identifier. Therefore, the OpenFlow Switch1 cannot forward flow from host c to host a, although the flow is valid.

We test the function of SDN network based on the attribute identifier: (1) whether the system can identify the validity of flow and (2) whether the system can realize the access control based on the attribute identifier.

(1) Host h1's authentication request carries a valid attribute identifier and signature. Host h2's authentication request carries the attribute identifier and signature that do not conform to the h6 access structure. h3 is that the general packet does not carry any attribute identifier and signature. The above three hosts continuously send data to h8 at the rate of 50 packages/s. The controller makes the path h1/h2/h3 ⟶ e1 ⟶ a2 ⟶ a3 ⟶ e3 ⟶ h6. We use SFlow to monitor the traffic of e1 and e3, repeating 10 times for 12 seconds each time, and the result is averaged. The results are shown in Figures 9 and 10. The X-axis is the time, and its unit is seconds; the Y-axis indicates the number of packets, and its unit is packages/s, and the positive direction represents the inflow packet; that is, the authentication

switch receives the packets of h1, h2, and h3, and the negative direction represents the outflow. The packet from (h1, h2, and h3) is forwarded to h6.

As it is illustrated in Figure 13, the authentication switch e1 receives packets from h1, h2, and h3. Since h1 and h2 carry the attribute identifier, e1 only forwards h1 and h2 packets, and h3 packets are discarded directly. However, h2 started with a small amount of forwarding and then discarded.

As shown in Figure 14, the packets of h3 are discarded. The packets of h2 are forwarded in the first second and then discarded. The packets of h1 are still forwarded. We check the authentication switch. e3 do not add new flow rules, and e1 generates an invalid flow rule to discard the packets of h2.

From the experimental results, it can be seen that when the packet enters the network, the authentication switch first discards the packet that does not conform to the specification; i.e., the attribute identification is empty. The destination authentication switch verifies the validity of the authentication request. Invalid flow tables are sent to the source authentication switch to discard its subsequent packets. Therefore, the network can reject the packet with invalid attribute identification and the illegal packets without the attribute identification into the network.

(2) h1 sends the packets that carry the attribute identification conforming to the h6's access structure to e1 at the rate of 50 data packets per second, and then the path of h1 is modified to h4 by the controller in the next 12 seconds. SFlow is used to monitor e2 and e3 traffic, repeating 10 times for 12 seconds each time, and the results are averaged as shown in Figures 15 and 16.

In Figures 15 and 16, if the attribute identifier in the switch flow entry is the same as the attribute identifier of the packet. The packet would allow entering the device. When it accesses another device, the authentication switch determines that it is illegal and discards it. Therefore, fine-grained access control based on attribute identification can be implemented.

5.3. Scheme Comparison. We compare the Attribute-Guard with the six most recent schemes; as illustrated in Table 4, SE-Floodlight [9], RoseMary [10], FRESCO [11], and FortNOX [3] offer a role-based source authentication for flow rules, and PERM- GUARD [26] and PermOF [27] offer a source authentication based on the access list.

Each new flow rule generated in the SDN network represents a change of the flow and the flow needs to be authenticated. The number of flow rules is positively related to the number of authentications, which reflects the granularity of data flow management. Figure 17 illustrates the number of authenticated identities comparison between Attribute-Guard and the most recent schemes. Because SE-Floodlight, RoseMary, FRESCO, and FortNOX provide only three roles for authentication, their bottleneck of identity authentication first appears. PERM-GUARD and PermOF are able to provide a better granularity due to the access lists.

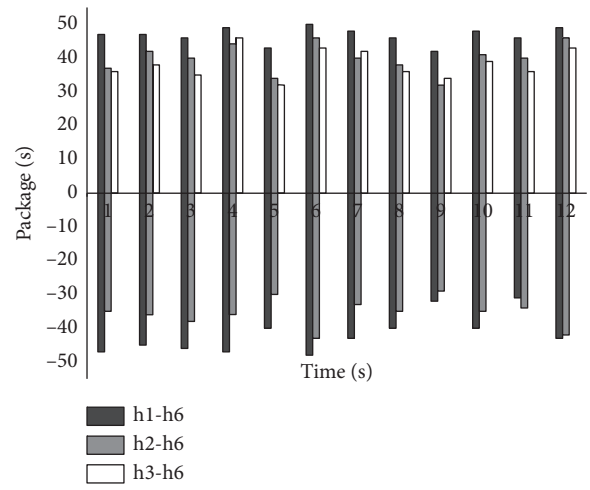

Figure 13: Authentication switch e1 traffic statistics.

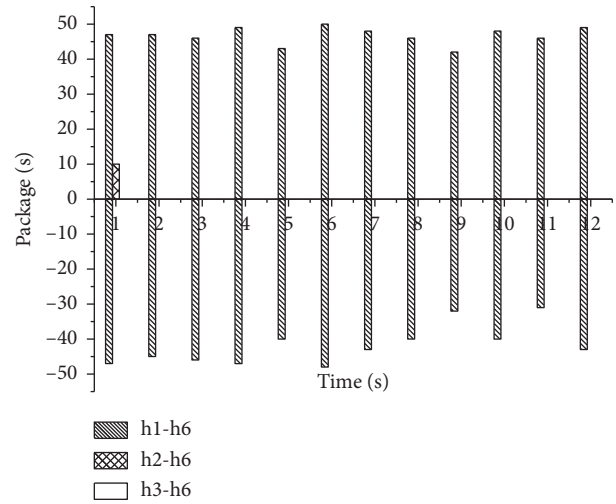

Figure 14: Authentication switch e3 traffic statistics.

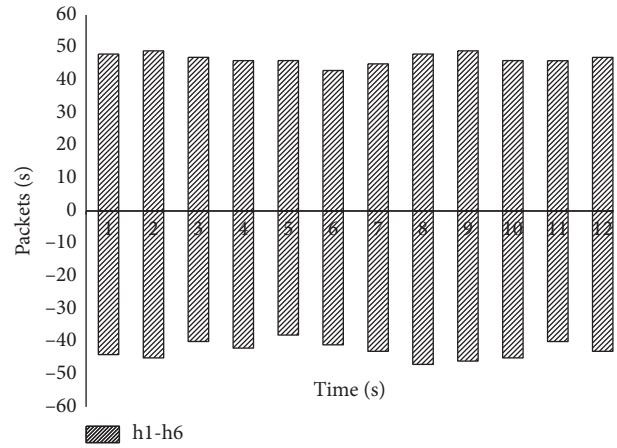

Figure 15: Authentication switch e3 traffic statistics under the experiment of access control.

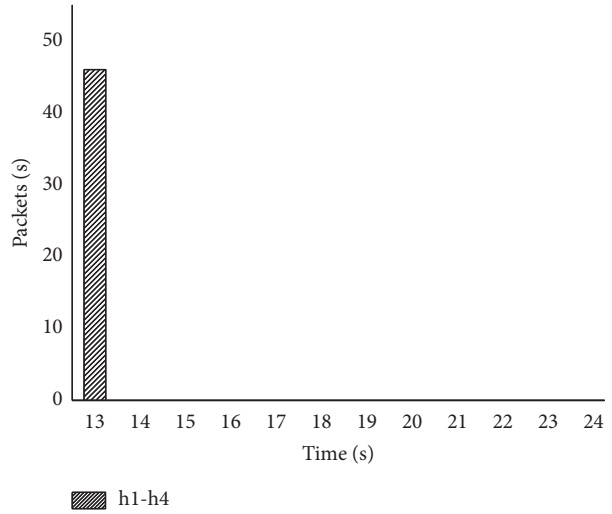

Figure 16: Authentication switch e2 traffic statistics.

Table 4: The granularity of flow authentication comparison.

Schemes	Signature	Permission management	The granularity of authentication
SE-Floodlight	Role-based	Role-based	Three authorization roles
RoseMary	Role-based	Role-based	Three authorization roles
FRESCO	Role-based	Role-based	Three authorization roles
FortNOX	Role-based	Role-based	Three authorization roles
PERM-GUARD	Identity-based	Identity-based and access list	16 permission
PermOF	No	Access control list and PKI	18 permission
Attribute-Guard	Attribute-based	Attribute-based	Custom the number of access structure

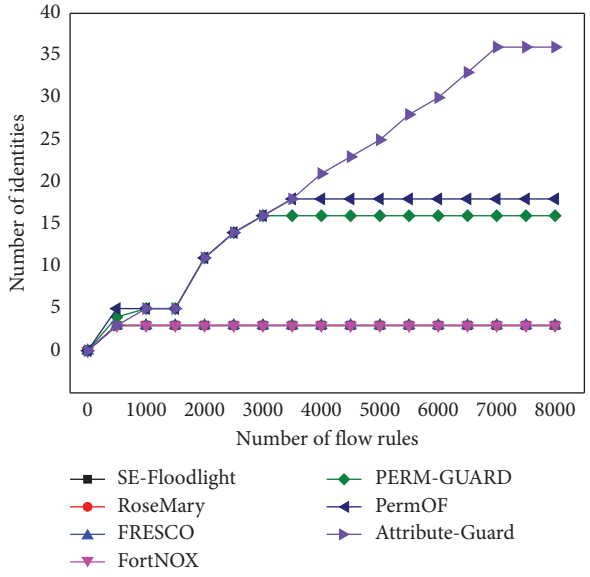

Figure 17: Granularity comparison of identity authentication.

But their granularity is limited by the size of the access list. Our scheme provides a fine-grained, Attribute-Guard, which adopts the attribute signature. The scheme defines the access structure according to requirements, and only the signature that satisfies its access structure can be verified successfully. The signature algorithm implements access control and does not need to create an access list, so it is not limited by the access list and achieves better fine-grained management.

As SDN design enables to push all the control functionality to a centralized controller, SE-Floodlight, RoseMary, FRESCO, FortNOX, PERM-GUARD, and PermOF develop network control applications and enforcing policies. Thus, controllers might potentially become a bottleneck for the network operations. Attribute-Guard reduces the request and the overhead on the SDN controller by delegating the access control capability to the data plane and has the lowest CPU utilization and memory utilization. The result is shown in Figures 18(a) and 18(b). Attribute-Guard has a better scalability than other schemes.

5.4. Performance Analysis. In this section, we will evaluate the performance of Attribute-Guard: SDN network packet processing capability based on attribute identification.

For the problem, we measure the time consumption of generating signature and verification signature to estimate the performance of Attribute-Guard. These two features determine how many flows could be handled by Attribute-Guard. However, because the length of the authentication request sent by the host is different and the traffic of the host is different at different times, the measurement results are different. To measure the maximum performance of Attribute-Guard, we need to pay attention to the two points: (1) increase the proportion of signature verification in packet generation and forwarding time and (2) measure time-

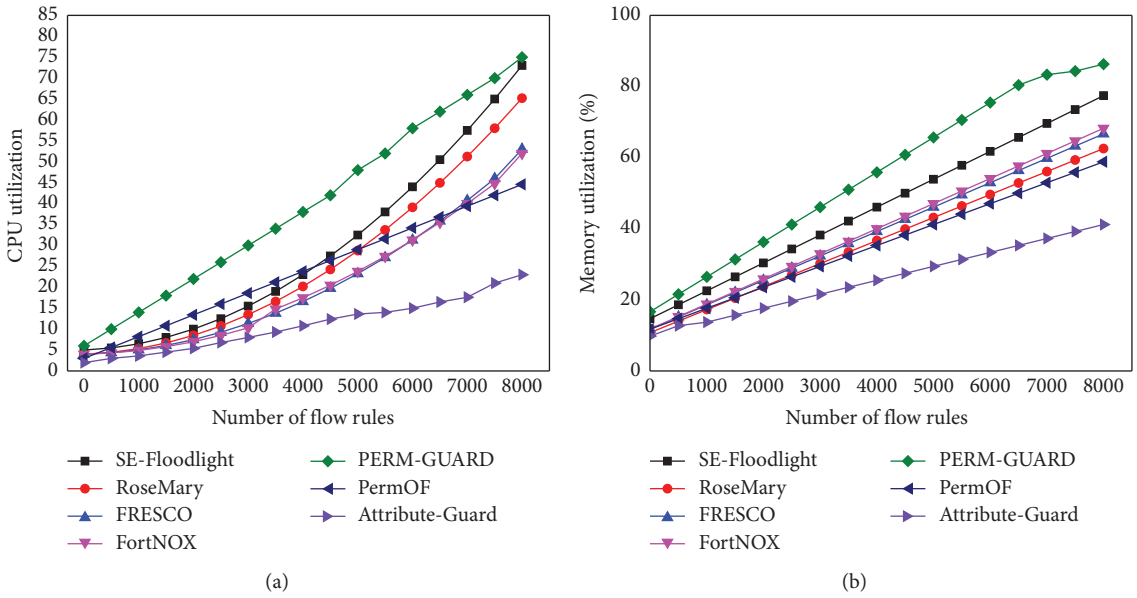

Figure 18: (a) CPU utilization comparison of schemes. (b) Memory utilization comparison of schemes.

consumption when the network is most active. For (1), we continue to send short messages with a fixed length of 64 bytes on the host. For (2), we refer to Stanford 300 users [25] and 22000 [28] user SDN experiments and Berkeley Lab [8] statistics on more than 8000 user SDN networks, and they indicate that the number of packets forwarded by each active host mainly ranges from 9 to 25 minutes. Therefore, first we count the generating time of 200 authentication requests within the 9 to 25 minutes of the host running. As shown in Figure 19, the X-axis corresponds to the number of authentication request, and the Y-axis corresponds to the time-consumption of generating authentication request. From the figure, we can calculate that the average time to generate a new authentication request with attribute identification is 25.76 ms. That is, 38 sets of authentication requests with different attribute identifiers and signatures can be sent per second at a rate of about 760 m/s.

We test 200 authentication requests to measure the average time for the authentication switch verifying each signed authentication request. As shown in Figure 20, the average time for the authentication switch to verify the signed authentication request is about 51.77 ms, which means that the authentication switch can handle 386 m per second.

According to the literature [6, 8, 9, 28, 29], the switch needs to handle the traffic in daily use as shown in Table 4. According to Table 5, the Attribute-Guard system performance can meet the basic network requirements.

When the authentication switch receives more packets, the authentication time will increase. It becomes a problem whether the verification efficiency will decrease as the packets increase becomes the bottleneck of data throughput. Figure 21 illustrates the authentication time required for the authentication switch under different packets, with the X-axis corresponding to the number of packets, of which 20M each. According to the result, the relationship between the number of authentication packets and the time of the

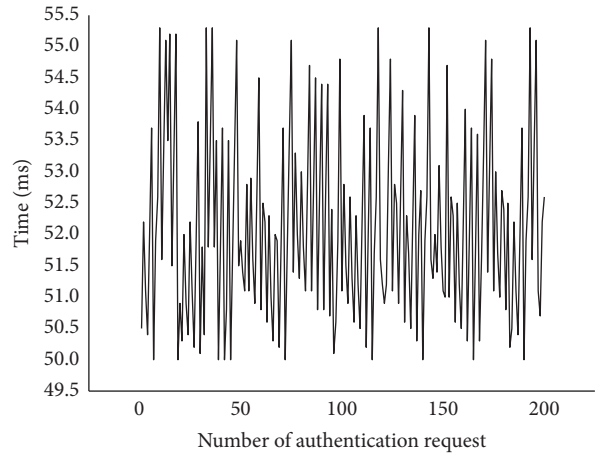

Figure 19: The time for the host to generate authentication request with attribute identifier and signature.

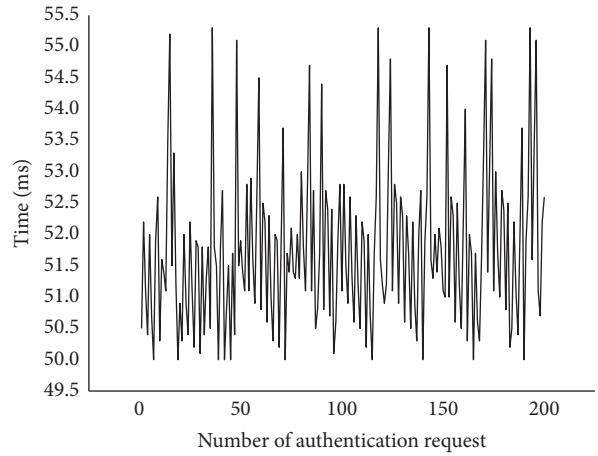

Figure 20: The time for the authentication switch to verify the authentication request.

Table 5: The number of packets forwarding per second in different networks.

Schemes	Size of network	The number of packets per second (m/s)
[25]	Over 300 hosts	246
[28]	22000 hosts	497
[8]	8000 hosts	356
[9]	4 hosts	196
[26]	100 hosts	320

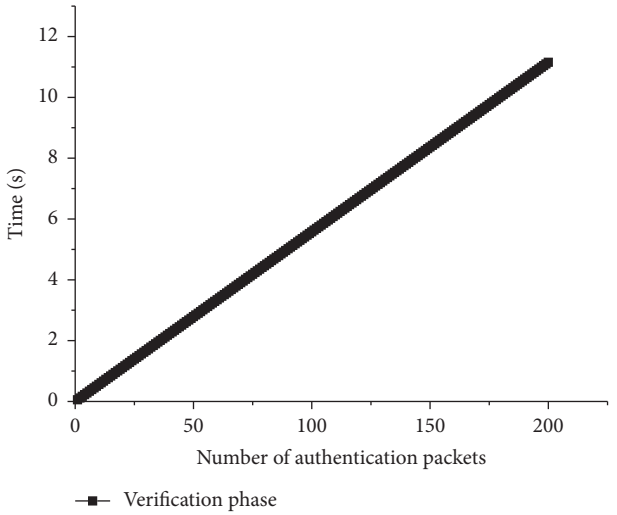

Figure 21: Time cost in the verification phase.

authentication switch is linear. The rate of the authentication packet does not change with the increase of the packet.

5.5. Network Overhead. Attribute-Guard introduces a new single point of failure in the "attribute identifier authority" and deploys security modules in the controller and data plane, and the limitation of the Attribute-Guard is needed to be discussed and the additional performance overhead of the controller, host, and forwarding device should be evaluated.

For the controller, we continuously send legal packets at different locations on the host and count controller additional performance overhead at different Pack_in rates.

Firstly, we compare the CPU utilization of the native OpenDaylight controller with the CPU utilization of the Attribute-Guard's controller. In Figure 22, the CPU utilization of the controller in the Attribute-Guard system is slightly higher than that of the native. OpenDaylight controller at the same Pack_in rate. Secondly, we compare the memory utilization of the native OpenDaylight controller with Attribute-Guard's controller. As illustrated in Figure 23. The memory utilization of Attribute-Guard's controller is slightly higher than the native OpenDaylight controller with a difference about 13%. It indicates that the Attribute-Guard system does not increase the controller load too much under the premise of implementing the security function.

We use hping3 to inject data packets into the network at 50 Mbit/s, 100 Mbit/s, 150 Mbit/s, 200 Mbit/s, 250 Mbit/s,

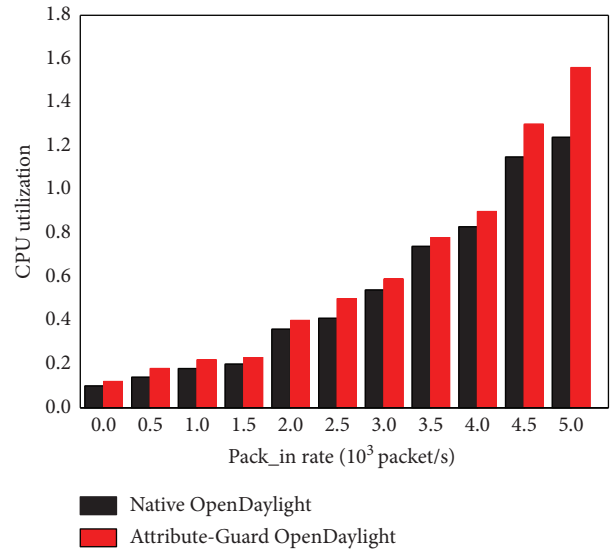

Figure 22: CPU utilization of controller comparison.

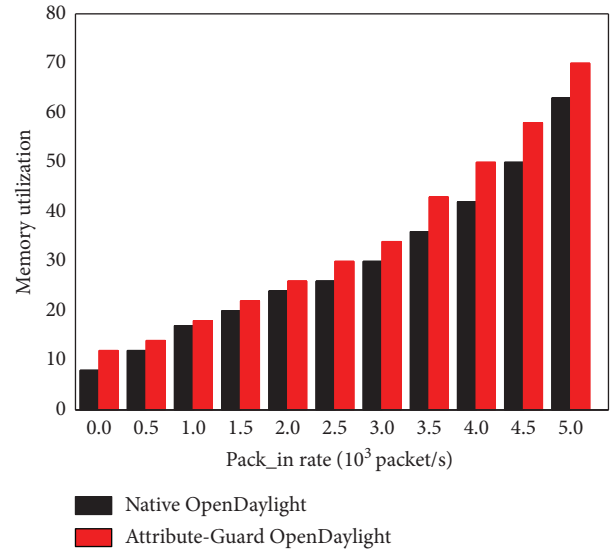

Figure 23: Memory utilization of controller comparison.

300 Mbit/s, 350 Mbit/s, 400 Mbit/s, 450 Mbit/s, and 500 Mbit/s. The CPU utilization and memory utilization of host and data plane are measured in different scenarios. It is repeated 10 times at different rates, and the average value is taken each time.

For the host, as shown in Figure 24, the CPU utilization of the host with the attribute identification component is slightly higher than that of the standard host, and both increase with the increase in the traffic. As the network traffic increases, the number of packets requiring authentication increases, and the difference in CPU usage gradually increases because the host with the attribute identification component needs to generate a signature and encapsulate the attribute identifier before sending the packet. Although the computational cost of generating a signature is large, it is only related to device attributes and is independent of the packet. It only needs to perform one operation, and the

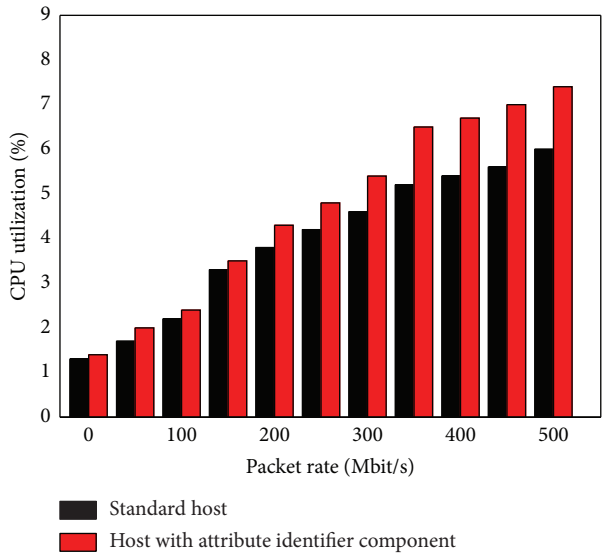

Figure 24: CPU utilization of host comparison.

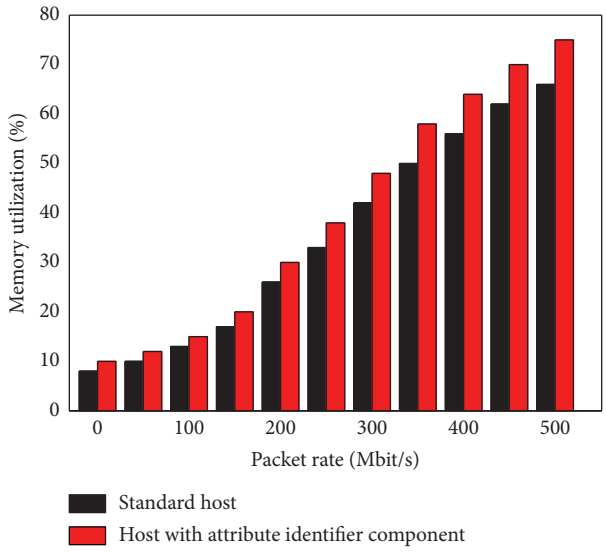

Figure 25: Memory utilization of host comparison.

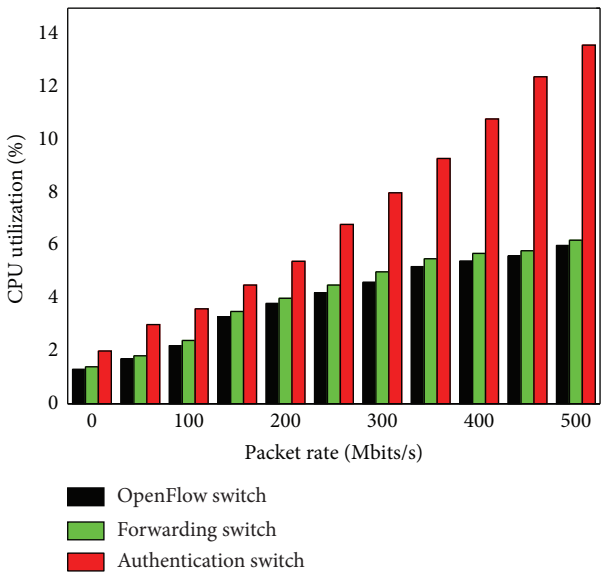

Figure 26: CPU utilization of switch comparison.

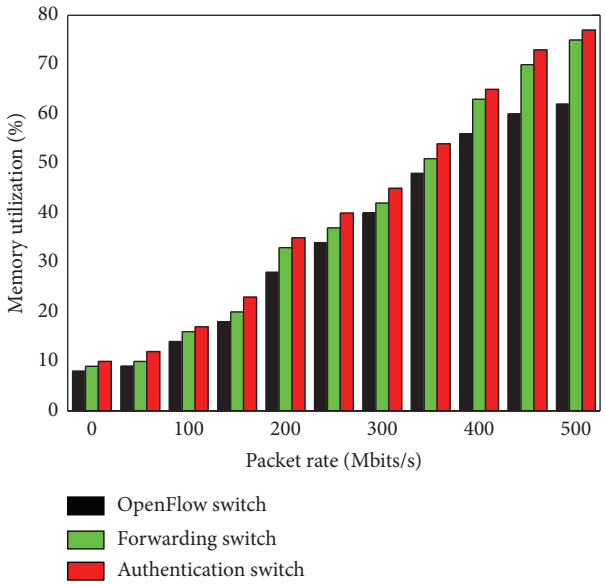

Figure 27: Memory utilization of switch comparison.

computational overhead of encapsulating signatures and attribute identifiers is small. Therefore, the difference in CPU usage between the two schemes is small as the rate increases, and the maximum difference is 0.8%. Similarly, in Figure 25, the more the network traffic, the more the packets needed to be encapsulated, and thus, the more the memory utilization. The difference between the standard host and the host with attribute identifier component is not very large, which is within the acceptable range.

For the data plane, the Attribute-Guard's data plane is composed of the authentication switch and forwarding switch. Therefore, we compare the network overhead between the authentication switch, the forwarding switch, and the OpenFlow switch. The authentication switch deploys a flow identification module and a processing pipeline based on the attribute identifier. When the flow enters the authentication switch, it needs to be authenticated. This process increases the computing overhead. In Figure 26, we can see that as the packet rate increases, the CPU utilization of the forwarding switch is the same as that of the OpenFlow switch, and the authentication switch suffers from a linear growth of CPU utilization rapidly. As shown in Figure 27, the authentication switch and the forwarding switch have higher memory utilization than the OpenFlow switch because the authentication switch and the forwarding switch add a pipeline based on attribute identification. The authentication switch and the forwarding switch require more flow tables than the OpenFlow switch. Finally, we compare the delay of the authentication switch, the forwarding switch, and the OpenFlow switch. The result is shown in Figure 28. The delay of the forwarding switch is slightly higher than that of the OpenFlow

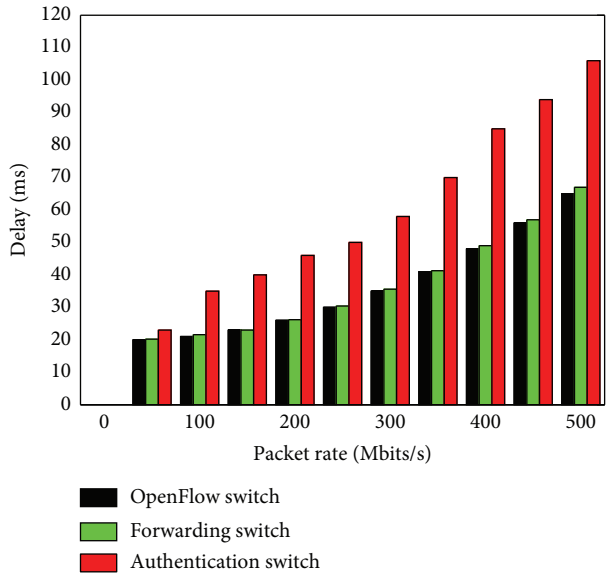

Figure 28: Delay of switch comparison.

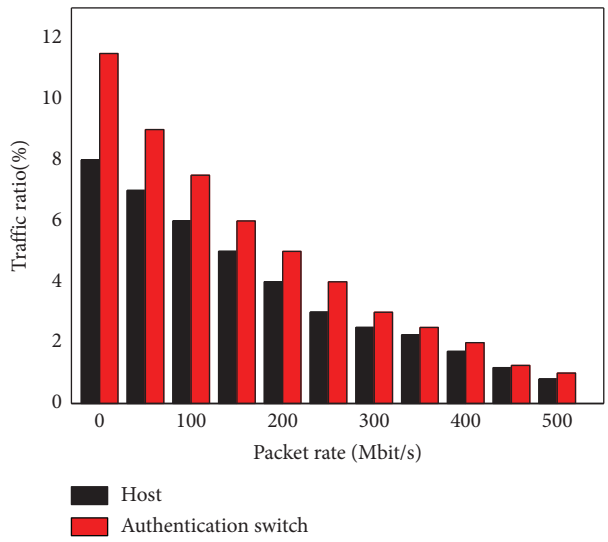

Figure 29: Traffic ratio of the authentication switch and host.

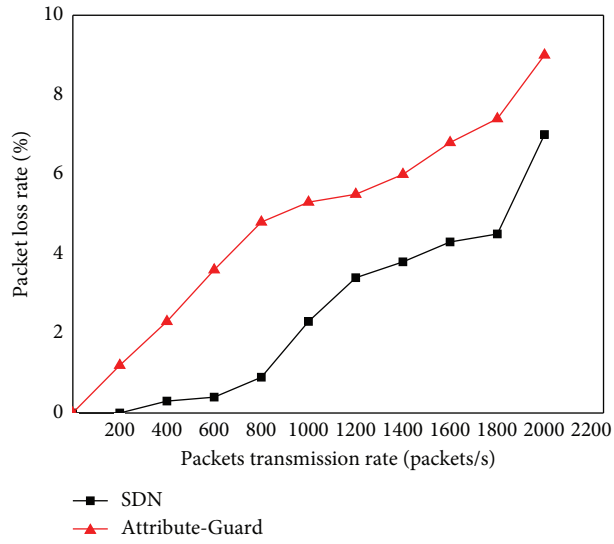

Figure 30: Packet loss rate comparison.

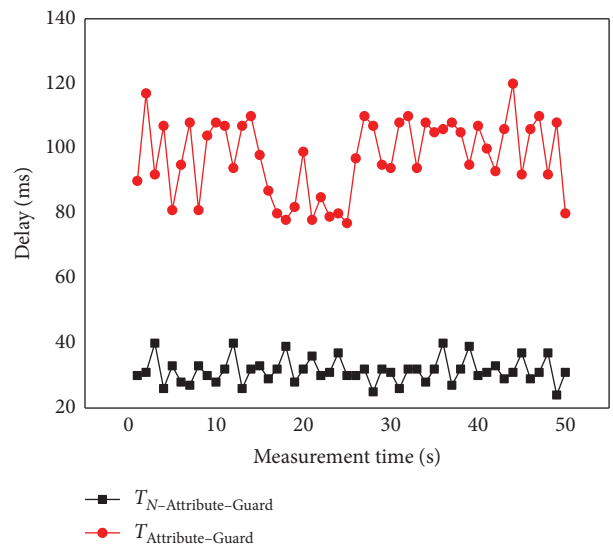

Figure 31: Network delay comparison.

switch, and the delay of the authentication switch is larger than the others. The authentication switch needs flow authentication and forwarding, while the forwarding switch and the OpenFlow switch only need forwarding. Therefore, the algorithm complexity of the signature is main network overhead, and the pipeline based on attribute identification has little effect on network overhead.

We have extended the SDN by adding an attribute identifier authority that needs to communicate with the authentication switch and host. In order to describe the impact of the attribute identifier authority on the authentication switch and the host, we count the traffic between the authentication switch and the attribute identifier authority, and the traffic between the host and the attribute identifier authority. Figure 29 shows the traffic ratio of the attribute identifier authority to the authentication switch and the traffic ratio of the attribute identifier authority to the host. The attribute identifier authority's traffic accounts for less than 12% of the total traffic and decreases as the packet rate increases.

5.6. Availability Analysis. We compare the packet loss rate and delay between Attribute-Guard and SDN networks and analyse network availability. First, we send packets with a length of 1514 bytes at different rates and different ports and count the packet loss rate as shown in Figure 30.

As shown in Figure 30, as the packet-receiving rate increases, the packet loss rate of the network increases because the packet with the attribute identifier needs to be verified and the flow table adds the math field based on the attribute identifier in the switch. The packet rate is larger than the network without attribute identifier and signature.

The result shows that the network has an extra packet loss rate of 2.56%.

Then, the average delay is obtained by counting 50 times of ACK return. In the Attribute-Guard scheme, the delay of each ACK return is $T_{\text{Attribute–Guard}}$. The delay of the system without the attribute identifier and signature is $T_{N\text{–Attribute–Guard}}$.

From Figure 31, the average network delay of $T_{N\text{–Attribute–Guard}}$ is 31.48 ms, while the average network delay of $T_{\text{Attribute–Guard}}$ is 103.2 ms, and the average delay is increased by 71.72 ms. The analysis shows that the time to generate signature and verify the signature accounts for 65.4% of the total delay. Therefore, the algorithm complexity of the signature directly determines the network delay, but it is still within the feasible communication delay.

6. Conclusion

In this paper, we propose Attribute-Guard, an access control framework based on the attribute identifier. The goal of Attribute-Guard is to make flow in the data plane more credible. This framework implements flow authentication and fine-grained access control, which enables the data plane to shield the host from varieties of malicious flow attacks. We have implemented and deployed Attribute-Guard on the OpenDaylight and OVS while verifying the functionality and usability of the network, and the verification results show that the framework ensures the availability of the network. In the future, we will make more efforts in flow verification to improve the performance of flow forwarding.

Data Availability

All data included in this study are available upon request from the corresponding author.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was funded in part by the National Natural Science Foundation of China.

References

[1] Z. Sun, J. Li, and K. Yang, "Software-defined networking," *Zte Communications*, vol. 56, no. 9, pp. 16–19, 2013.

[2] E. Al-Shaer and S. Al-Haj, "Flowchecker: configuration analysis and verification of federated openflow infrastructures," in *Proceedings of the 3rd ACM workshop on assurable and usable security configuration*, pp. 37–44, ACM, Chicago, IL, USA, 2010.

[3] N. Gude, T. Koponen, J. Pettit et al., "Nox," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[4] R. Guo, H. Shi, Q. Zhao, and D. Zheng, "Secure attribute-based signature scheme with multiple authorities for blockchain in electronic health records systems," *IEEE Access*, vol. 6, pp. 11676–11686, 2018.

[5] H. Xiong, Y. Bao, X. Nie, and Y. I. Assor, "Server-aided attribute-based signature supporting expressive access structures for industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 6, pp. 1–5, 2019.

[6] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, vol. 1–17, pp. 121–126, ACM, Helsinki, Finland, August 2016.

[7] A. Abdou, P. C. Van Oorschot, and T. Wan, "Comparative analysis of control plane security of sdn and conventional networks," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3542–3559, 2018.

[8] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A first look at modern enterprise traffic," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, p. 2, USENIX Association, Berkeley, CA, USA, October 2005.

[9] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer," in *Proceedingsof the 2015 Network and Distributed System Security Symposium*, San Diego, CA, USA, February 2015.

[10] S. Shin, Y. Song, T. Lee et al., "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, ACM, Scottsdale, AZ, USA, November 2014.

[11] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, and M. Tyson, "Fresco: modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*, San Diego, CA, USA, February 2013.

[12] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework," *Computer Standards & Interfaces*, vol. 66, Article ID 103357, 2019.

[13] G. Lopez-Millan, R. Lopez, and A. Abadcarrascosa, *Software-Defined Networking (Sdn)-Based Ipsec Flow Protection*, Internet Engineering Task Force, Fremont, CA, USA, 2016.

[14] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: enabling record and replay troubleshooting for networks," in *Proceedings of the USENIX Annual Technical Conference*, pp. 327–340, USENIX Association, Portland, OR, USA, June 2011.

[15] J. Halpern and C. Pignataro, "Service function chaining (SFC) architecture," No. RFC 7665, 2015.

[16] M. Caprolu, S. Raponi, and R. Di Pietro, "Fortress: an efficient and distributed firewall for stateful data plane sdn," *Security and Communication Networks*, vol. 2019, Article ID 6874592, 16 pages, 2019.

[17] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: flexible and elastic ddos defense," in *Proceedings of the 24th {USENIX} Security Symposium ({USENIX} Security 15)*, pp. 817–832, Washington, DC, USA, August 2015.

[18] J. M. J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *Proceedings of the Third European Workshop on Software-Defined Networks, EWSDN 2014*, pp. 1–3, IEEE, London, UK, September 2014.

[19] K. Benzekki, *Devolving IEEE 802.1X Authentication Capability to Data Plane in Software-Defined Networking SDN Architecture*, John Wiley & Sons, Hoboken, NY, USA, 2016.

[20] B. Jun, "SDN architecture and future network architecture innovation environment," *Telecommunications Science*, vol. 29, no. 1, pp. 6–15, 2013.

[21] Open Network Foundation, *OpenFlow Switch Specification Version 1.2.0*, Open Network Foundation, 2014, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-vl.2.0.pdf.

[22] M. Attig and G. Brebner, "400 Gb/s programmable packet parsing on a single FPGA," in *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pp. 12–23, IEEE, Brooklyn, NY, USA, October 2011.

[23] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: stateful network-wide abstractions for packet processing," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 29–43, ACM, Florianópolis, Brazil, August 2016.

[24] D. Khader, "Attribute based group signatures," *IACR Cryptology ePrint Archive*, vol. 159, 2007.

[25] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pp. 89–98, ACM, Alexandria, VA, USA, October 2006.

[26] M. Wang, J. Liu, J. Chen, X. Liu, and J. Mao, "Perm-guard: authenticating the validity of flow rules in software defined networking," *Journal of Signal Processing Systems*, vol. 86, no. 2-3, pp. 157–173, 2017.

[27] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a secure controller platform for openflow applications," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pp. 171-172, ACM, Hong Kong, China, August 2013.

[28] M. Casado, T. Garfinkel, A. Akella et al., "SANE: a protection architecture for enterprise networks," in *Proceedings of the USENIX Security Symposium*, vol. 49, p. 50, Vancouver, Canada, August 2006.

[29] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures," in *Proceedings of the Annual International Cryptology Conference*, pp. 41–55, Springer, Santa Barbara, CA, USA, August 2004.

[30] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "A distributed and robust SDN control plane for transactional network updates," in *Proceedings of the IEEE INFOCOM 2015—IEEE Conference on Computer Communications*, IEEE, Kowloon, China, April 2015.