AC-PROT: An Access Control Model to Improve Software-Defined Networking Security

Wei Wu*, Renping Liu*, Wei Ni†, Dali Kaafar†, Xiaojing Huang*
*University of Technology Sydney, Australia
†Data 61, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia
Email: {Alven.Wu, Wei.Ni, Dali.Kaafar}@csiro.au, {RenPing.Liu, Xiaojing.Huang}@uts.edu.au

***Abstract*—The logically-centralized controllers have largely operated as the coordination points in software-defined networking(SDN), through which applications submit network operations to manage the global network resource. Therefore, the validity of these network operations from SDN applications are critical for the security of SDN. In this paper, we analyze the mechanism that generates network operations in SDN, and present a fine-grained access control model, called Access Control Protector(AC-PROT), that employs an attribute-based signature scheme for network applications. The simulation result demonstrates that AC-PROT can efficiently identify and reject unauthorized network operations generated by applications.**

***Keywords*: software-defined networking, attribute-based signature, access control.**

I. Introduction

Software-defined networking (SDN) paradigm decouples network control from data forwarding, offering high programmability of control plane and a global view of the network. The basic SDN architecture consists of three layers including the application layer, the control layer and the data-plane layer. The adoption of this paradigm promotes the development of logically centralized and integrated security policies, which simplifies the solution of complex network security problems [1].

SDN network operations, like flow rule insertion, port configuration and network state inquiry, are generated dynamically by a variety of network applications. The controller receives these network operations from network applications and delivers them to switches in the data-plan layer. Switches in the data-plan layer will subsequently operate according to these network operations and absolutely trust them. However, the current design of SDN does not provide an efficient access control mechanism to mediate these network operations between the application layer and the control layer. SDN network is vulnerable to various attacks such as denial of service, information disclosure attacks and flow-rule forging attacks caused by the compromised network operations from malicious applications [3], [6], [8]. Even worse, an adversary may tamper with a set of network operations to control the states of all switches by masquerading as a valid application. Therefore, ensuring the validity of network operations from network applications is definitely one of the main challenges in SDN security.

Many techniques for enhancing the network application security in SDN have been developed in recent years [2]–[5], [7], [8]. Porraset et al. proposed a new security system called SE-Floodlight [3], which extends the Floodlight controller with a role-based authorization scheme, with three default authorization roles among applications, i.e., administrator, security application, and traffic application. Therefore, it may simply treat different applications as the same role. Son et al. implemented FLOVER [5], a model checking system to verify the aggregation of flow polices instantiated within an SDN network, while Mattos et al. presented a new framework for control applications, enabling software-defined network controllers to use the host identity as a new flow field to define forwarding rules [1]. However, as the network status in SDN changes dynamically, these methods could protect the flow rules in its execution process, but could not prevent malicious applications from taking abnormal network operations into the controller. Compared with them, our work contributes a fine-grained access control model to verify the legality of network operations from network applications with multiple attributes in SDN.

In this paper, we present a fine-grained access control model called Access Control Protector(AC-PROT) for SDN applications. AC-PROT manages the access control policies for each network operation defined in SDN and detects the applications whose network operations are against these policies. To achieve these security functionalities, we take advantage of the feature of the logically-centralized SDN structure and introduce an attribute-based signature scheme in AC-PROT. Each application uses this scheme to sign its network operations with its unique attributes in SDN. As a result, the controller can authenticate these applications network operations based on the access control policies and their signatures, thus to prevent malicious applications launching unauthorized network operations to switches in the data-plane layer.

II. Access Control Model for SDN Applications

In this section, we present our design of a fine-grained access control model for network applications running on the controller.

A. System Model

1) Overall Structure: Fig.1 illustrates the conceptual diagram of AC-PROT, which extends the classical SDN architecture with three additional security components: i) an attribute-based signature module, and ii) a trusted authority(TA), iii) an application verifier(App-Verifier). It provides two security functionalities: i) managing the access control policy for applications network operations, and ii) verifying the validity of applications network operations.

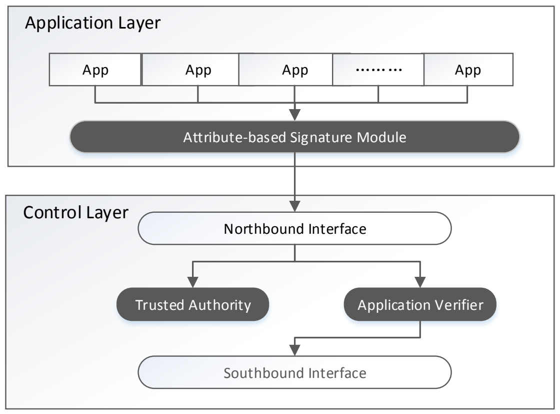

Fig. 1. The conceptual architecture of AC-PROT.

2) Access Structure: AC-PROT provides digital signature validation for applications to provisionally limit their network operations based upon their attributes. To support the security functionality, we summarize a set of 7 application attributes in the Table I, which may be augmented by the SDN administrator with additional attributes when necessary.

TABLE I
ATTRIBUTE SET FOR NETWORK APPLICATIONS

Notation	Attribute Description	Attribute Id
App-Developer	*The developer of App*	a_1
App-SerialNumber	*The serial number of App*	a_2
App-Name	*The name of App*	a_3
App-Function	*The main function of App*	a_4
App-Version	*The current version of App*	a_5
App-CreateTime	*The create time of App*	a_6
App-UpdateTime	*The update time of App*	a_7

Let $\{a_1, a_2, \ldots, a_n\}$ be a set of application attributes. **An access structure** is a collection A of non-empty subsets of $\{a_1, a_2, \ldots, a_n\}$.

In this paper, the set of all applications that produce network operations for the controller is denoted as APP. An application $App_i \in APP$ will be assigned a user access structure A_u to generate its attribute-based signature by the network administrator.

3) Access Control Policy: The access control policies define the granted network operations of the application, e.g., read network states, write flow rules, etc. According to the categorization of message communication between applications and the controller presented in previous work [3], [7], we define a set of 16 access control policies that would be enforced by AC-PROT on the controller and classify them into three categories, as illustrated in Table II.

TABLE II
CATEGORY OF ACCESS CONTROL POLICY

Category	Policy Name	Policy Id
Read policies	Read-Topology	p_1
	Read-Flow	p_2
	Read-Statistics	p_3
	Read-Packet-in-Payload	p_4
Notification Policies	Packet-in-Event	p_5
	Flow-Removed-Event	p_6
	Error-Event	p_7
	Topology-Event	p_8
Write Policies	Packet-in-Event	p_9
	Flow-Mod-Route	p_{10}
	Flow-Mod-Drop	p_{11}
	Set-Flow-Priority Event	p_{12}
	Flow-Mod-Modify	p_{13}
	Modify-All-Flows	p_{14}
	Set-Packet-Out	p_{15}
	Set-Device-Configuration	p_{16}

An attribute tree is used to describe an access control policy. Each non-leaf node of the tree represents a threshold gate. If num_x is the number of children of a node x and k_x is its threshold value, then $0 < k_x < num_x$. Each leaf node x of the tree is described by an attribute and a threshold value $k_x = 1$. $prt(x)$ represents the parent of the node x in the tree. The children of every node are numbered from 1 to num. The function $index(x)$ returns such a number associated with the node x, where the index values are uniquely assigned to nodes in the attribute tree in an arbitrary manner. The function $attr(x)$ denotes the attribute associated with the leaf node x.

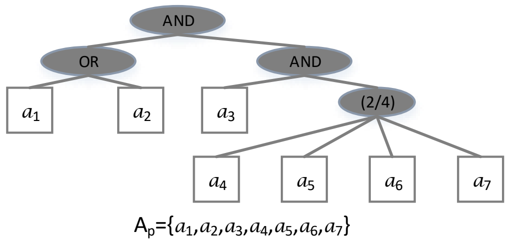

Fig. 2. Logic Structure of An Attribute Tree.

We illustrate the logic structure of an attribute tree in Fig.2. A policy access structure A_p represents the attribute collection set of all the leaf nodes in an attribute tree.

Let T_x be the subtree of T rooted at the node x. If an access structure A satisfies the access tree T_x, we denote it as $T_x(A) = 1$. We compute $T_x(A)$ recursively by the **Satisfying an Attribute Tree (SAT)** Algorithm defined as follows. If x is a non-leaf node, evaluate $T_x(A)$ for all children z of node x, returns 1 if at least k_x children return 1; If x is a leaf node, then $T_x(A)$ returns 1 if $att(x) \in A$.

B. Attribute-Based Signature Scheme

Our attribute-based signature scheme is an extension of the signature scheme proposed by Cao et al. [9] and the attribute-based encryption scheme presented by Junbeom et al. [10]. It consists of the following five polynomial time algorithms which is based on bilinear groups and computational Diff-Hellman(CDH) problem: *Setup*, *Public-Key-Generation*, *Private-Key-Generation*, *Operation-Signing*, and *Operation-Verification*.

TABLE III
NOTATIONS

Notation	Description
$params$	*Public parameters*
msk	*Master secret key*
A	*A system access structure*
A_u	*A user access structure*
A_p	*A policy access structure*
A_s	*A signing access structure*
A_v	*A verifying access structure*
T_p	*An attribute tree for a policy*
pk_p	*The public key for a policy*

First, we assume that operation messages produced by valid applications are n-bit strings. Then, Table III summarizes several notations and their corresponding meanings that will be used in the paper. Our attribute-based signature scheme includes the following steps.

Setup: TA generates two groups $\mathbb{G}$, and $\mathbb{G}_T$ of prime order p, and constructs a bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$, and randomly chooses a generator g of $\mathbb{G}$. Then, $\alpha \in \mathbb{Z}_q$ is randomly chosen by TA as a secret. TA sets $g_1 = g^{\alpha}$ and picks g_2 randomly in $\mathbb{G}$, and computes $V = e(g_1, g_2)$. After that, TA defines the system access structure $A = \{a_1, a_2, \ldots, a_n\}$ for some integer n and a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$. For each $a_i \in A$, TA chooses a random element $r_i \in \mathbb{Z}_q$ and calculates $R_i = g^{r_i}$. Therefore, the public parameters are $param = (q, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, e, g, g_1, g_2, V, H, \{R_i\}_{i=1}^{n})$, and the master secret key consists of the following components $msk = (\alpha, \{r_i\}_{i=1}^{n})$. We summary the setup algorithm as Algorithm 1.

Algorithm 1 Setup

```
Input: $\kappa$: the security parameter
Output: $params$, $msk$
 1) generate two groups: $\mathbb{G}$ and $\mathbb{G}_T$
 2) construct a bilinear map $e$
 3) randomly choose a generator $g$ of $G$ and $\alpha \in \mathbb{Z}_q$
 4) compute $g_1 = g^{\alpha}$
 5) randomly choose $g_2 \in \mathbb{G}$
 6) compute $V = e(g_1, g_2)$
 7) define a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}$
 8) for each $a_i \in A$, where $A$ is the system access structure, do
 9)     randomly choose $r_i \in \mathbb{Z}_q$
10)     compute $R_i = g^{r_i}$
11) end for
12) return $param = (q, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, e, g, g_1, g_2, V, H, \{R_i\}_{i=1}^{n})$
13) return $msk = (\alpha, \{r_i\}_{i=1}^{n})$
```

Public-Key-Generation: An applications network operation is regarded as authorized when its access structure satisfies the access control policy. Receiving an attribute tree defined by the network administrator, TA uses Public-Key-Generation algorithm to compute the corresponding policy public key. The Public-Key-Generation algorithm of our approach is shown in Algorithm 2, which proceeds as follows.

For a defined attribute tree, TA chooses a polynomial p_x of degree $d_x = k_x - 1$ for each node x in an attribute tree, where k_x is the threshold value. That is done in a top-down manner. Starting from the root $p_{root}(0) = \alpha$ and d_{root}, other points in the polynomial will be random. For any other node x, set $p_{x(0)} = p_{prt(x)}(index(x))$ and choose d_x for other points randomly. Then, let Y be the set of leaf nodes in T_p. Once the polynomials have been decided, the public key for the attribute tree is $pk_p = \{D_y = g^{p_y(0)}, D_i = R_i^{p_y(0)}\}$, and the policy access structure is $A_p = \{a_i\}$, where i is the subscript of $a_i = att(y)$, and y is a leaf node.

Algorithm 2 Public-Key-Generation

```
Input: $T_p$
Output: $pk_p$, $Ap$
 1) for each node $x \in T_p$ in a top-down manner, do
 2)     choose a polynomial $p_x$ of degree $d_x = k_x - 1$, where
        $k_x$ is the threshold value
 3)     if node $x$ is the root node
 4)         set $p_{root}(0) = \alpha$
 5)     else
 6)         set $p_x(0) = p_{prt(x)}(index(x))$
 7)     end if
 8) for each leaf node $y \in T$, do
 9)     compute $D_y = g^{p_y(0)}$, $D_i = R_i^{p_y(0)}$, where $i$ is the
        subscript of $a_i = att(y)$
10) end for
11) return $pk_p = \{D_y, D_i\}$ and $A_p = \{a_i\}$, where $i$ is the
    subscript of $a_i = att(y)$, and $y$ is the leaf node
```

Let P denotes the policy set defined in Table II. When all the attribute trees are constructed by the Public-Key-Generation algorithm, TA will send $\{p_i, pk_{p_i}, T_{p_i}, A_{p_i}\}_{p_i \in P}$ to the APP-Verifier, and then distribute $\{p_i, A_{p_i}\}_{p_i \in P}$ to each application $App_i \in APP$.

Private-Key-Generation: TA generates the private keys for each applicaiton $App_i \in APP$. The private key includes two components: (1)the base component, and (2)the attribute component. The algorithm is described in Algorithm 3, which works as follows.

Algorithm 3 Private-Key-Generation

```
Input: $msk$, $params$ and $A$
Output: $sk$: the private key
 1) randomly choose $u \in \mathbb{Z}_q$
 2) compute $k_0 = g_2^{\alpha(u-1)}$
 3) for each attribute $a_i \in A$, do
 4)     randomly choose $\lambda_i \in \mathbb{Z}_q$
 5)     compute $k_{i1} = g_2^{u} R_i^{\lambda_i}$
 6)     compute $k_{i2} = g^{\lambda_i}$
 7) end for
 8) return $sk = (k_0, \{k_{i1}, k_{i2}\}_{a_i \in A})$
```

TA takes a random $u \in \mathbb{Z}_p$ and compute the base component of the secret key $k_0 = g_2^{\alpha(u-1)}$. Then for each attribute $a_i \in A$, TA chooses $\lambda_i \in \mathbb{Z}_q$ randomly and set $k_{i1} = g_2^{u} R_i^{\lambda_i}$, $k_{i2} = g^{\lambda_i}$ as the attribute components of the secret key. Therefore, the secret key $sk = (k_0, \{k_{i1}, k_{i2}\}_{a_i \in A})$, where A is the system access structure.

After that, TA chooses the attribute components of the private key for each $App_i \in APP$ based on its user access structure A_{u_i}, and delivers $sk_{u_i} = (k_0, \{k_{i1}, k_{i2}\}_{a_i \in A_{u_i}})$ to it securely.

Operation-Signing: An application should sign its operation with the attribute-based signature scheme before it sends the operation message to the controller. The operation-sign algorithm of our approach is shown in Algorithm 4, which works as follows.

Let m be the n-bit string that represents the operation message. The application firstly chooses a random $s \in \mathbb{Z}_q$ and computes $\sigma_0 = g^s$ and $\sigma_1 = H(m)^s k_0$. Then for each $a_i \in A_s$, where $A_s = A_p \bigcap A_u$ as a signing access structure, a random $\lambda'_i \in \mathbb{Z}_p$ is chosen. TA computes $\sigma_{i1} = k_{i1} R_i^{\lambda'_i}$, $\sigma_{i2} = k_{i2} g^{\lambda'_i}$. Therefore, a signature on the operation message m is constructed as $\sigma = (\sigma_0, \sigma_1, \{\sigma_{i1}, \sigma_{i2}\}_{a_i \in A_s})$.

Algorithm 4 Operation-Signing

```
Input: $m$, $sk$, $params$, $A_p$ and $A_u$
Output: $\sigma$: the signature on $m$
 1) randomly choose $s \in \mathbb{Z}_q$
 2) compute $\sigma_0 = g^s$
 3) compute $\sigma_1 = H(m)^s k_0$
 4) compute $A_s = A_p \bigcap A_u$
 5) for each attribute $a_i \in A_s$, do
 6)     randomly choose $\lambda'_i \in \mathbb{Z}_q$
 7)     compute $\sigma_{i1} = k_{i1} R_i^{\lambda'_i}$
 8)     compute $\sigma_{i2} = k_{i2} g^{\lambda'_i}$
 9) end for
10) return $\sigma = (\sigma_0, \sigma_1, \{\sigma_{i1}, \sigma_{i2}\}_{a_i \in A_s})$
```

Operation-Verification: It is run by the APP-Verifier on the controller, when the controller receives an operation message. The Operation-Verification algorithm takes a signature σ, an operation message m and the public key pk_p as inputs. If it is a valid signature on m, the algorithm will output *accept*. Otherwise it will output *reject*.

To verify the signature, a recursive function $VerNode(\sigma, pk_p, x)$ is defined in the Operation-Verification Algorithm, where x is a node in the attribute tree. It outputs a group element of $\mathbb{G}_{\mathbb{T}}$ or *fail*, which works as follows.

For a node x in the attribute tree, the SAT algorithm $T_x(A_v)$ is firstly run to check if the node satisfies the attribute tree, where $A_v = A_p \bigcap A_u$ as a verifying access structure. When $T_x(A_v) = 1$ and x is a leaf node:

$$\begin{aligned}
&VerNode(\sigma, pk_p, x)\\
&= e(\sigma_{i1}, D_x)/e(\sigma_{i2}, D_i)\\
&= e(k_{i1} R_i^{\lambda'_i}, g^{p_x(0)})/e(k_{i2} g^{\lambda'_i}, R_i^{p_x(0)})\\
&= e(g_2^u, g^{p_x(0)}) e(R_i^{\lambda_i + \lambda'_i}, g^{p_x(0)})/e(g^{\lambda_i + \lambda'_i}, R_i^{p_x(0)})\\
&= e(g, g_2)^{up_x(0)}.
\end{aligned}$$

When $Tx(Au) = 1$ and x is a non-leaf node, the $VerNode(\sigma, pk_p, x)$ proceeds as follows: For all nodes z that are children of x, it calls $VerNode(\sigma, pk_p, z)$ and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z, $i = index(z)$, $S_z = \{index(z) : z \in S_x\}$. Then it calculates:

$$\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S_z}(0)}\\
&= \prod_{z \in S_x} (e(g, g_2)^{up_z(0)})^{\Delta_{i,S_z}(0)}\\
&= \prod_{z \in S_x} (e(g, g_2)^{up_{prt(z)}(index(z))})^{\Delta_{i,S_z}(0)}\\
&= \prod_{z \in S_x} e(g, g_2)^{up_x(i)\Delta_{i,S_z}(0)}\\
&= e(g, g_2)^{up_x(0)}
\end{aligned}$$

where
$$\Delta_{i,S_z}(x) = \prod_{j \in S_z, j \neq i} (x - j)/(i - j)$$

as a lagrange interpolation item. otherwise, it returns *fail*.

To verify the signature, the Operation-Verification Algorithm runs the function $VerNode(\sigma, pk_p, x_{root})$ to compute F_{root}, where x_{root} is the root node of the attribute tree. Then it checks if

$$V == F_{root} e(H(m), \sigma_0)/e(g, \sigma_1).$$

The Operation-Verification Algorithm is described in Algorithm 5 as follows.

Algorithm 5 Operation-Verification

```
Input: $m$, $\sigma$ and $pk_p$
Output: $Result$: the verification result
 1) choose the root node $x_{root}$ in the attribute tree $T$
 2) call function $F_{root} = VerNode(\sigma, pk_p, x_{root})$
 3) if $F_{root} =$ fail
 4)     $Result =$ reject
 5) else
 6)     compute $Value = F_{root} e(H(m), \sigma_0)/e(g, \sigma_1)$
 7) end if
 8) if $(Value == V)$
 9)     $Result = accept$
10) else
11)     $Result = reject$
12) end if
13) return $Result$
```

If the Operation-Verification Algorithm returns *accept*, App-Verifier will accept the network operation message and send it to network devices in the data-plan layer. Otherwise, the App-Verifier will reject the network operation request.

III. Implementation and Evaluation

We have deployed a simple prototype of AC-PROT with the Pairing-Based Cryptography Library, and evaluated it on its granularity and actual time consumption.

A. Granularity Analysis

We compare AC-PROT to the current existing schemes [3], [8], which are also authentication schemes in SDN. As illustrated in Table IV, SE-Floodlight [3] offers a role-based source authentication for network operations, which recognizes all applications with only three roles. OperationCheckpoint [8] provides a permission management system, which can also manage the flow rule permissions of each application for the SDN controller.

TABLE IV
THE GRANULARITY OF NETWORK OPERATIONS SOURCE AUTHENTICATION COMPARISON

Schemes	Permission Management	Granularity
SE-Floodlight	*Role-based*	3 roles
OperationCheckpoint	*Access control list*	15 permissions
AC-PROT	*Attribute-based*	16 policies

Our scheme represents a fine-grained access control model based on attribute-based signature scheme, which dynamically provides a more flexible way to manage the network operations of each application by access control policies. Thus, the granularity of network operation authentication is more fine-grained compared with other schemes. In addition, it also achieves higher flexibility because the network administrator can modify these policies according to the dynamic SDN network state.

B. Time Consumption Analysis

We calculate the overhead of AC-PROT by two criteria: (i) the time consumption for an application to sign its network operations with the Attribute-based Signature Module in the signing phase, and (ii) the time consumption for the App-Verifier to verify signed network operations in the verification phase. The time consumption in both the signing and the verification phase is an important measure to estimate the performance of AC-PROT, because it determines how many network operations could be handled by AC-PROT.

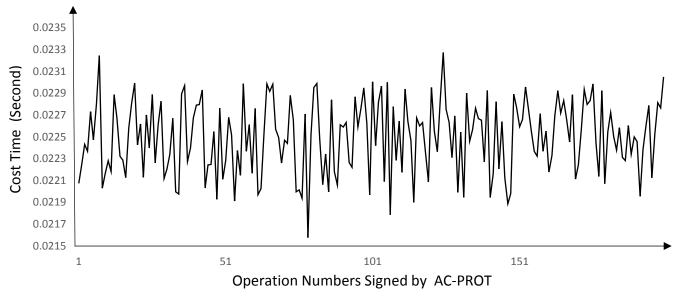

Fig. 3. The time to sign each network operation.

In our experiment, we construct the attribute tree for the access control policy as showed in Fig.2, then run the signing and verification algorithm on a Linux machine with an Intel Core 4.0 GHz i7-6700K CPU. We test 200 network operation messages to measure the average time for the machine to sign and verify each operation message, respectively.

As shown in Fig.3, the horizontal-axis corresponds to new operation to be signed, and the vertical-axis corresponds to the time cost to sign each message in signing phase. We can observe that the average time the machine takes to sign each operation message is about 22.4ms, which means that 2,678 new operation messages can be signed every minute with the Attribute-based Signature Module.

As illustrated in Fig.4, the average time to verify each signed operation message is about 44.5ms. It means 1,348 newly network operations per minute can be handled on the machine. Therefore, AC-PROT is very efficient and absolutely acceptable by networks described in [1], [3], [4], [7].

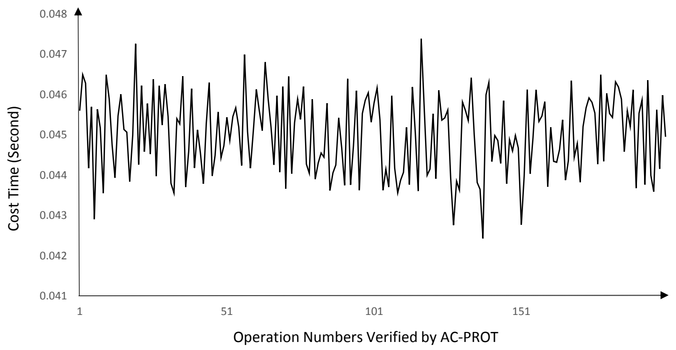

Fig. 4. The time to verify each network operation.

IV. CONCLUSION

In this paper, we propose AC-PROT, an access control model to authenticate the validity of network operations from applications in SDN. The key challenge that we address is the lack of trust introduced by the open interface between the application plane and the control plane that knowledgeable adversaries can exploit. As there are no standard security definitions for the northbound API, a serious concern is whether a network operation that produced by an application can be trusted or not. Our proposed AC-PROT enables the control plane to shield the controller from varieties of malicious network operation attacks launched by applications without the granted access structure.

REFERENCES

[1] D. M. F. Mattos, & O. C . M. B. Duarte, "AuthFlow: authentication and access control mechanism for software defined networking," in *annals of telecommunications*, pp.1-9, 2016.
[2] F. Klaedtke, G. O. Karame, R. Bifulco, & H. Cui, "Access control for sdn controllers," *Ecosystems*, vol.16, pp.1325-1335, 2014.
[3] P. Porras, S. Cheung, & M. Fong, "Securing the software-defined network control layer," *Network and Distributed System Security Symposium*, 2015.
[4] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, & S. Shenker, "Sane: a protection architecture for enterprise networks," *Conference on Usenix Security Symposium*, vol.15, pp.1-15, 2006.
[5] S. Son, S. Seungwon, V. Yegneswaran, P. Porras, & G. Guofei, "Model checking invariant security properties in openflow," *IEEE International conference on communications*, pp.1794-1799, 2013.
[6] X. Wen, Y. Chen, C. Hu, C. Shi, & Y. Wang, "Towards a secure controller platform for openflow applications," *the second ACM SIGCOMM workshop on hot topics in software defined networking*, 2013.
[7] M. Wang, J. Liu, & J. Chen, "PERM-GUARD: Authenticating the Validity of Flow Rules in Software Defined Networking," *IEEE, International Conference on Cyber Security and Cloud Computing*, pp.1-17, 2016.
[8] S. Scott-Hayward, C. Kane, & S. Sezer, "Operationcheckpoint:Sdn application control," *IEEE, International Conference on Network Protocols*, pp.618-623, 2014.
[9] D. Cao, X. Wang, B. Zhao, J. Su, & Q. Hu, "Mediated attribute based signature scheme supporting key revocation," *IEEE, Information Science and Digital Content Technology*, vol.2, pp.277-282, 2012.
[10] H. Junbeom, C. Park, & S. Hwang, "Fine-grained user access control in ciphertext-policy attribute-based encryption," *Security & Communication Networks*, vol.5, pp.253-261, 2012.