Research Article

Constant-Size Credential-Based Packet Forwarding Verification in SDN

Ping Wu,1,2 Chao-Wen Chang,1,2 Ying-Ying Ma,1,2 and Zhi-Bin Zuo3

1Zhengzhou Information Science and Technology Institute, Zhengzhou, Henan Province 450001, China
2Henan Key Laboratory of Information Security, Zhengzhou, Henan Province 450001, China
3Henan University of Technology, Zhengzhou, Henan Province 450001, China

Correspondence should be addressed to Ping Wu; wpieucs@sina.com and Zhi-Bin Zuo; zzbin_2000@163.com

Received 22 March 2022; Accepted 10 September 2022; Published 11 October 2022

Academic Editor: Leandros Maglaras

Copyright © 2022 Ping Wu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The emerging software-defined networking (SDN) technology lacks tools to proactively ensure that policies will be followed or to inspect the behavior of the network. The network is vulnerable to sophisticated attacks against packets, such as alteration, injection, dropping, and hijacking attacks. Accurate and efficient network packet forwarding verification is a critical issue to ensure the correctness of packet forwarding when confronting with malicious attacks, while most of the existing packet forwarding verification solutions were implemented by inserting linear-scale cryptographic tags that increased with path length, which introduced significant communication overhead. In this paper, we propose a constant-size credential based packet forwarding verification mechanism in SDN. In the scheme, the ingress switch of a flow embeds a tag credential of constant-size which is independent of the packet forwarding path, each downstream switch verifies packets basing on the constant-size credential, and the controller periodically acquires node forwarding statistics along the path and localizes anomaly. The header space communication overhead of the proposed scheme is less than existing linear-scale mechanisms. We further prototype and evaluate the proposed scheme. Experiments demonstrate that the scheme achieves efficient forwarding and effective anomaly localization with less than 11% of additional forwarding delays and no more than 10% of throughput degradation.

1. Introduction

As a new network paradigm, SDN [1] decouples the control plane and data plane and reshapes the ossified network architectures. The open network programming interface of SDN promotes network innovation, reduces the cost of network operation and maintenance, provides a new experimental way for the research of the new network architecture, and also greatly promotes the development of the next generation of Internet technology. However, the "three layers and two interfaces" (application plane, control plane, data plane, southbound interface, and northbound interface) of the network architecture enlarges network attack surfaces to facilitate malicious attackers. SDN faces both typical security threats of traditional IP networking and those unique to its architecture. These typical or unique security issues, such as legality and consistency of flow rules, vulnerabilities of the southbound interface protocol, e.g., OpenFlow protocol, vulnerabilities of the data plane and controller, and lack of trust mechanism between controller and applications, are constraints for large-scale deployment of SDN [2].

Network nodes or routers should verify packets to ensure that the full-link security from source to destination is an important part of network security [3]. However, the current architecture of SDN does not guarantee that a packet of a flow does follow the rules or path specified by the controller. The SDN controller knows nothing about the real forwarding path and behavior of data plane switches. The main reason is that SDN lacks tools to proactively ensure that policies will be followed or to reactively inspect the behavior of the network. Malicious switch nodes controlled by an adversary in the network can inject, drop, and alter packets, or even redirect packets of a flow to deviate from the controller authorization path and violate the crucial network policies [4]. The general methodology of packet forwarding
verification in traditional IP networking is to embed linear-scale cryptographic tags in packets to verify packets hop-by-hop. In SDN, most of the existing packet forwarding verification solutions leverage the methodology that is in traditional IP networking. However, embedding linear-scale cryptographic tags as forwarding path lengths introduces significant header space communication overhead.

Embedding linear-scale cryptographic tags in packet of a flow as path lengths is a fundamental efficiency barrier. The primary reason is as follows. As forwarding path length increases, the validation tags inserted take up more header space overhead as well as network transmission bandwidth. What is more, longer validation tags inserted need to take more time to produce and verify. Existing packet forwarding verification mechanisms enforce \( O(n) \) size validation tags for an \( n \)-hop forwarding path, which introduces significant header space overhead and excessive computation overhead, and degrade network transmission performance. The packet forwarding verification solutions should strive to provide verification ability with insignificant overhead and locate the abnormal link.

Making use of the traits of centralized control and network programmability in SDN, we present a constant-size credential based packet forwarding verification. To summarize, the contributions of this paper are threefold.

1. We design CSCPV, a constant-size credential based packet forwarding verification in SDN. The CSCPV limits the header space overhead of cryptographic tags embedded. It outperforms the existing linear-scale mechanisms; even the flow path length is a very small value. CSCPV can effectively detect malicious forwarding behavior and achieve efficient packet forwarding with negligible header space overhead.

2. We design an efficient tag credential for packet forwarding verification. The size of credential is independent of path length; it is constant-size and easy to implement. And we also conduct theoretical analysis of credential security.

3. Leveraging P4 [5] (Programming Protocol-independent Packet Processors), we further prototype CSCPV. We also extensively evaluate our CSCPV prototype on Mininet simulations. The analyses and experiment results demonstrate that the proposed mechanism introduces limited computation and header space communication overhead. Concretely, CSCPV only introduces less than 10% of throughput degradation and no more than 11% of additional forwarding delays. And due to constant-size credential, CSCPV significantly reduces the header space overhead over the SDN forwarding verification mechanisms OPT [6] and SDNsec [7].

The rest of the article is organized as follows. Section 2 introduces related works. Section 3 briefly describes the problem and attack model. In Section 4, we present CSCPV, the constant-size credential based packet forwarding verification mechanism in SDN. In Section 5, we analyze the security of CSCPV, and we further prototype CSCPV leveraging P4 switch and present the performance evaluation results based on the CSCPV prototype with Mininet simulations in Section 6. We conclude the paper in Section 7.

2. Related Works

Embedding or inserting linear-scale cryptographic tags approaches have been widely used in packet forwarding verification in traditional IP networking as well as SDN. OPT [6] presents a lightweight source and path verification mechanism, the sender embeds all validation tags of nodes along the forwarding path in packets, and each intermediate router performs two message authentication code calculation operations to verify packets. The OPT’s cryptographic tags increase linearly as path lengthens and OPT is unable to tackle malicious node dropping or hijacking attacks. In literature [7], SDNsec, as a type of network packet path validation mechanism, was proposed. The controller checks whether each packet of a flow has followed the correct path, which introduces a large bandwidth overhead on the control channel. Also, the disadvantage of SDNsec is that the inserted cryptographic tags increase linearly as path lengthens. ICING [8] uses aggregated message authentication code technology to enable source and path authentication; however, ICING requires each intermediate router to store shared keys with other routers on the path and has a higher header space overhead. More precisely, ICING has a per-hop overhead of 42 bytes. PrivNPV [9] proposes path verification target for path privacy and index privacy security; similarly, the packet header space overhead increases linearly with the path length. The main goal of data path credentials architecture [10] is to identify valid node and thus not to allow the transmission of attack traffic, but the system requires \( 4L \) times \( (L \text{ is path length}) \) of handshake interactions on the credential initialization stage, which introduces significant network communication overhead, and the credential fails to protect packet integrity.

Many researches attempt to localize malicious switches by checking the consistency of flow statistics [11–14] in SDN; these approaches can only achieve coarse granularity verification. Verifying the correctness of data plane configuration [15–17], ATPG [18], VeriDP [19], and [20, 21] proactively verify path consistency by using probe packets or checking pairwise reachability; however, all of these mechanisms assume switches or routers are benign or trustable.

Inspired by the literature [10], in this paper, we propose CSCPV, a packet forwarding verification mechanism based on constant-size credential, which can effectively compress the packet header space overhead, introduce negligible computation and communication overhead, and forward packets efficiently.

3. Problem Statement

In SDN, controller issues forwarding policies, and the data plane switches follow issued policies to forward packets. However, malicious switches controlled by an adversary can commit attacks such as alteration, dropping, injection, and
even hijacking packets. An attacker is to control switches on a path and interrupt the normal packets forwarding of a flow.

Packet alteration attack: a malicious attacker controls a switch on a path and tampers with any part of the data packet of a flow, such as packet header information and packet payload.

Packet injection attack: a malicious attacker controls a switch on a path and inserts forged packets of a flow and sends it towards a downstream switch.

Packet hijacking attack: a malicious attacker controls a switch on a path and redirects packets of a flow to another path that has not been authorized by the controller.

Packet dropping attack: a malicious attacker controls a switch on a path and drops packets of a particular flow.

In this paper, we focus on packet forwarding verification, detecting malicious attacks, and localizing network anomaly with limited computation and communication overhead. We assume that the controller in SDN is security; also SDN enables Transport Layer Security (TLS) and communication between the controller and switches in data plane are secured. And we assume that, along a forwarding path, the ingress and egress switch are benign nodes, because there is no meaning to verify packets if packets even do not enter or leave the network.

4. Design of CSCPV

The challenge of designing CSCPV is to abandon the methodology of existing packet forwarding verification solutions which embed linear-scale cryptographic tags in packets as path lengths. Instead, we perform packets verification by introducing negligible constant-size tag credential which is independent of path length. In this section, we will present detailed description to solve the design challenge. Table 1 shows some of the relevant notations used in this paper.

Section 4.1 outlines the CSCPV mechanism and Sections 4.2–4.4, respectively, illustrate the initialization of constant-size credential, packet forwarding verification, and anomaly localization in CSCPV.

4.1. Overview. In the network, each switch node is associated with an identity (say, \( N_i \)), and there exists a trustable party, called the Key Generation Center (KGC). The KGC chooses a group \( G \), and \( G = \langle g \rangle \) is a multiplicative group of prime order \( q \), where \( g \) is a generator of the group \( G \). KGC chooses two hash functions, \( H_1: \{0, 1\}^* \rightarrow Z_q^* \) and \( H_2: Z_q \times Z_q \rightarrow \Omega \), where \( \Omega \) is the output space of \( H_2 \), and \( \Omega \in \{0, 1\}^r \). Then KGC picks a random element \( x \) from \( Z_q \) and sets \( y = g^x \). Finally, the KGC outputs the master secret key \( \text{MSK} = x \) and the public parameters \( \text{MPK} = \{ G, q, g, y, H_1, H_2 \} \). For each node with identity \( N_i \), the KGC selects a random element \( a_i \in Z_q \) and sets \( b_i = g^{a_i} \); using the master secret key \( \text{MSK} = x \), the KGC computes \( c_i = a_i + H_1(N_i, b_i), x \) and sends the partial secret key \( (b_i, c_i) \) to the node \( N_i \). Node \( N_i \) has obtained a partial secret key \( (b_i, c_i) \) from KGC, and then it picks a random element \( x_i \) from \( Z_q \) and sets \( y_i = g^{x_i} \). Finally, node \( N_i \) outputs the public key \( \text{PK}_i = (b_i, y_i) \) and the secret key \( \text{SK}_i = (c_i, x_i) \).

When a new flow enters the network, the controller calculates the path of the flow and issues the flow forwarding path \( \text{PATH} \) and the session identifier \( \text{SID} \) to the source ingress switch \( N_0 \) through a secure channel. Here, \( \text{PATH} \) is the authorized forwarding path. The secure channel can be implemented with OpenFlow on top of SSL/TLS. At the same time, the controller installs flow table entries for switches on the authorized forwarding path. \( N_0 \) generates a tag credential of constant-size (say, \( \text{CSC} \)). As shown in Figure 1, the tag CSC locates in-between the IP and TCP/UDP header.

The fixed length of CSC is 32 bytes (the length of CSC is 160 bytes when a packet is the first packet of a flow), and the meaning of each field is as follows.

| Notation | Meaning |
|----------|---------|
| SID | Flow identifier |
| PATH | Flow path information |
| PKV | Field enabling node \( N_i \) to verify the packet |
| PktHash | Digest of the packet’s payload |
| L | Flow path length |
| \( h_1, h_2, \ldots, k_i \) | \( k \) independent hash operation functions |
| Payload | Payload in a packet |
| PVF | Field enabling node \( N_i \) to verify the packet |
| sk | Anonymous key shared between \( N_0 \) and \( N_i \) |

Table 1: Some major notations used in the paper.
For a flow, the source ingress switch $N_0$ uses anonymous key-agreement protocol [22] to generate a temporary shared key $sk_i$ with each switch node $N_i$ on the path and computes the message authentication code MAC based on the key $sk_i$ and constructs the packet validation field $PVF$ based on Bloom Filter data structure [23]. Here, the core idea is that the field $PVF$ can contain multiple MAC items at the same time. The $L$ (path length) MAC elements of switch nodes on the path are then superimposed in the field $PVF$ by the controller to verify the packet. The controller periodically obtains statistics of each switch node on the forwarding path and localizes network anomaly. The overall architecture of CSCPV is shown in Figure 2.

In the control layer, there are two modules.

Path Computation and Flow Rule Module: when a new flow enters the network, the control layer computes the path of flow transmission and installs flow table entries for the switches of the authorized path.

Anomaly Detection and Localization Module: the controller obtains packet statistics of a flow from the switches of authorized path and detects and localizes the anomaly link.

And in the data layer, there are four modules.

Session Key Negotiation Module: this module implements key negotiation when a new flow enters the network.

Packet Verification Module: the module that incorporates a Bloom Filter data structure verifies integrity of a packet.

Packet Verification Counter Module: according to the verification result, the module updates the counters of valid packets of a flow.

Flow Table Entries: the module stores flow rules installed by the controller, and the switch forwards packets based on the rules.

4.2. Credential Initialization. As described above, when a new flow enters the network, the source ingress switch $N_0$ sends a request to the controller, and controller calculates packet forwarding path $PATH$ of the flow. Controller sends authorized path $PATH$ and the session identifier (say SID) to the source ingress switch $N_0$. Leveraging anonymous key-agreement protocol [22], $N_0$ generates a temporary shared key $sk_i$ with each switch node $N_i$ on the path, here path is $PATH = <N_0, N_1, ..., N_L>$. $N_0$ picks a random element $\omega \in Z_q$, set $\sigma = g^\omega$, here $\sigma$ is the shared key negotiation parameter. For each node $N_i$ on the path $PATH$, according to section 4.1, its public key is $PK_i = (b_i, y_i)$. $N_0$ computes $z_{i,0} = [h_{i} y^H (N_0, b_0)]^\omega$ and $z_{i,0} = y_i^\omega$, then the anonymous shared symmetric key between $N_0$ and $N_i$ is $sk_i = H_2(z_{0, i} \| z_{i, i})$. $N_0$ only computes the shared symmetric key $sk_i$ with each node on the path when the first packet of a new flow enters the network, no need to repeat computing in the subsequent packets transmission. $N_0$ generates a tag of constant-size as shown in Figure 1. As described in Section 4.1, when a packet is the first packet of a flow, the flag $F$ in CSC is set to 1, and then $\sigma$/NULL field is set to $\sigma$, which is the shared key negotiation parameter as above. All the subsequent packets of the flow have a flag $F$ set to 0 in tag CSC, and the tag CSC will no longer contain the field $\sigma$/NULL. For each switch node $N_i$ on the path $PATH = <N_0, N_1, ..., N_L>$, $N_0$ calculates the message authentication code MAC based on the shared symmetric session key $sk_i$ as follows:

$$MAC_i \leftarrow MAC_{sk_i}(\text{CSC}_8 \| \text{PktHash} \| \text{TTL}_i \| \text{IPINVAR}) \quad (1)$$

Here, PktHash is digest of the IP packet’s payload and PktHash $\leftarrow$ Hash (payload), CSC$_8$ is the first 8 bytes of the constant-size credential, and TTL$_i$ denoted the expected TTL at switch node $N_i$ during the IP packet transmission. The IPINVAR is the invariant portion of the original IP header at

**Figure 1:** CSC format in the packet header space.

**Figure 2:** The overall architecture of CSCPV.
each router during forwarding. \(N_0\) constructs the packet validation field PVF based on Bloom Filter [23]. As shown in Figure 3, here, \(N_0\) computes \(MAC_i (1 \leq i \leq L)\) items and superimposes them in the field PVF. When a packet of a flow is transmitted, each downstream switch node \(N_i\) can query or check if its own MAC exists in the field PVF based on the Bloom Filter and thus verifies the packet.

The Bloom Filter is a type of data structure containing an array of \(m\)-bits. An empty Bloom Filter data structure (the field PVF) is initialized with all bits set to zero. Based on the Bloom Filter, \(N_0\) constructs the field PVF. When inserting an element, e.g., MAC, one bit corresponding to the value of a single hash function for the element MAC is set to 1. Concretely, for each message authentication code MAC\(_i\) \((1 \leq i \leq L)\), the Bloom Filter maps MAC\(_i\) to \(k\) positions in field PVF (corresponding bits will be set to 1) with \(k\) independent hash functions \(h_1, h_2 \ldots k\). Finally, the \(L\) (path length) MAC elements of all switch nodes along the path are then superimposed in the field PVF based on Bloom Filter data structure.

Making use of \(k\) independent hash functions \(h_1, h_2 \ldots k\), a message authentication code MAC\(_i\) is mapped to \(k\) positions (of course, maybe there exists positions collision) in the PVF. Supposing the set of indices \(v_i\ [j] \ (1 \leq j \leq k)\) of bits of a MAC\(_i\) element mapped in the PVF, there must be the following (2) to be established:

\[
\prod_{j=1}^{k} PVF[v_i\ [j]] = 1. \tag{2}
\]

When the tag CSC is constructed completely, as shown in Figure 1, the tag CSC locates in-between the IP and TCP/UDP header. According to the flow table entry installed, the ingress switch \(N_0\) sends the packet to the next hop node \(N_1\).

### 4.3. Packet Verification

For each switch node \(N_i\) on the path \(PATH = \langle N_0, N_1, \ldots, N_L \rangle\), \(N_i\) receives a packet and parses fields in the tag CSC. If the flag \(F\) is set to 1, this indicates that the packet is the first packet of a flow. Based on the field of shared key negotiation parameter \(\sigma = g^w\) in tag CSC, \(N_i\) computes the temporary shared symmetric key \(sk_i\) with the source switch node \(N_0\). According to Section 4.1, node \(N_i\) possesses private key \(SK_i = (c_i, x_i)\), \(N_i\) sets \(z_{i,0} = \sigma^w\) and \(z_{i,j} = \sigma^x_i\), and then the temporary shared symmetric key between \(N_i\) and \(N_0\) is \(sk_i = H_2(z_{i,0}||z_{i,j})\). As follows in (3) and (4), we can deduce \(sk_i = sk_j\), which means the symmetric key \(sk_i\) computed by \(N_0\) and \(sk_i^{\prime}\) computed by \(N_i\) are the same.

\[
z_{i,0} = h_i^yH_i(N_i||b_i)^w = \left[ g^{\sigma_wx_1H_1(N_i||b_i)} \right]^w = \left[ g^{\sigma^w} \right]^w = g^{wzc_i} = \sigma^x_i, \tag{3}
\]

\[
z_{i,j} = y_i^w = g^{x_iw}, \tag{4}
\]

Switch node \(N_i\) saves the temporary shared symmetric key \(sk_i\) and there is no need to repeat calculation in the subsequent packets transmission. When a packet tagged with constant-size credential is received by switch node \(N_i\), firstly, \(N_i\) checks if the value of field \(N_{na}\) exceeds the presetting threshold \(L \cdot k\) (L is path length, and \(k\) denotes the number of independent hash functions) and also verifies whether the real count of bits set to 1 in the field PVF of the tag CSC exceeds the presetting threshold value. If validation fails, the packet will be dropped. Secondly, \(N_i\) calculates the message authentication code \(MAC_i\) according to (1) and verifies checks if the message authentication code \(MAC_i\) exists in the field PVF based on Bloom Filter as shown in Figure 3.

If \(\prod_{j=1}^{k} PVF[v_i\ [j]] = 0\), it indicates that packet verification has failed, and node \(N_i\) will drop the packet. And if \(\prod_{j=1}^{k} PVF[v_i\ [j]] = 1\), the packet is verified to be valid with a higher probability, and then \(N_i\) forwards the packet to the next node \(N_{i+1}\).

### 4.4. Anomaly Localization

A packet of a flow arrives at the egress switch node \(N_L\), if the packet is the first packet of the flow, basing on the temporary shared symmetric key \(sk_L\) computed via the anonymous key negotiation protocol depicted in Section 4.3, \(N_L\) computes the message authentication code and verifies the packet based on the field PVF. If verification succeeds, \(N_L\) forwards the packet to the destination.

The controller periodically collects forwarding packet’s statistics (say, \(count[i]\)) of each switch \(N_i\) on the authorized transmission path. If the difference of count of packets forwarded between adjacent connection switch node along the path exceeds a threshold, e.g., the natural packet loss rate \(\theta\), there must be a malicious node dropping packets or tampering packets. The controller will locate the link as abnormal link. Anomaly localization algorithm is shown below.

For any two adjacent nodes \(N_i\) and \(N_{i+1}\) along the path \(PATH = \langle N_0, N_1, \ldots, N_L \rangle\), the controller compares counts of packets received and verified to be valid by the two nodes, e.g., \(count[i]\) and \(count[i+1]\). If \(count[i] > count[i+1]\) and \((count[i] - count[i+1]) / count[i] < \theta\) (here \(\theta\) is natural packet loss rate for the link), the link is normal. Otherwise, if \(count[i] > count[i+1]\) and \((count[i] - count[i+1]) / count[i] > \theta\), the link between \(N_i\) and \(N_{i+1}\) will be judged as an abnormal link.

As shown in Figure 4, the controller periodically acquires the node forwarding statistics on the transmission path. The counts of valid packets of nodes \(N_i\), \(N_{i+1}\) and \(N_{i+2}\) forwarding are 10000, 9000, and 8998, respectively. In Figure 4, \(count[i] - count[i+1] > 0\) and \((count[i] - count[i+1]) / count[i] > \theta\), the link is therefore judged to be an abnormal link, while \(count[i+1] - count[i+2] > 0\) and \((count[i+1] - count[i+2]) / count[i+1] < \theta\), so the count difference can be considered as natural packet loss during network packets transmission, instead of a hijacking/
dropping or tampering attack performed by a malicious node, and link $l_{k+1}$ is a normal link.

5. Analyses and Discussions

In CSCPV, each switch node holds the public key $PK_i = (b_i, y_i)$ and private key $SK_i = (c_i, x_i)$. Based on anonymous key negotiation protocol, the source ingress switch negotiates the shared session key $sk_i$ ($1 \leq i \leq L$) with nodes along the forwarding path via the first packet of a flow. The source ingress node generates the tag CSC of constant-size which includes packet validation field PVF based on Bloom Filter. Using the shared symmetric key, downstream switch node includes packet validation field PVF based on Bloom Filter. The ingress node generates the tag CSC of constant-size which includes packet validation field PVF based on Bloom Filter data structure to verify packet.

However, the attacker can simply modify or fabricate the field $N_{bit}$ of the tag CSC and set all the bits to 1 in field PVF to 1; it would result in the idea that any packet injected or tampered by the attacker would be verified to be valid by all downstream nodes. In order to make the tag CSC immune to these attacks, we need to limit the size of field $N_{bit}$ and check if the count of bits set to 1 in the field PVF exceeds a presetting value. Here, we introduce two additional concepts to the field PVF.

**Definition 1.** Function $\Gamma(pvf)$ represents the total count of bits set to 1 ($PVF[j] = 1, 1 \leq j \leq m$) in field PVF of $m$-bits length array in CSC, as shown in

$$\Gamma(pvf) = \sum_{j=1}^{m} PVF[j]. \quad (5)$$

**Definition 2.** For any packet of the flow, function $pvf(m, k, L)$ reflects the expectation value of $\Gamma$’s number in field PVF based on Bloom Filter data structure of size $m$ with $k$ independent hash operation functions and $L$ (path length) inserted MAC items ($1 \leq i \leq L$) items.

The source ingress node $N_0$ calculates the MAC ($1 \leq i \leq L$) based on the temporary shared session key $sk_i$, using $k$ independent hash functions; for each MAC item, $N_0$ sets $k$ bits to 1 in field PVF (there may be cases of position collision between and within each MAC item).

For a single MAC item, the probability that one bit in field PVF is set to 1 by a single function in $k$ independent hash functions is $P$.

$$P = \frac{1}{m}. \quad (6)$$

On the contrary, the probability that a bit in field PVF is not set to 1 by a single hash function in $k$ independent hash functions is $P^c$:

$$P^c = 1 - P = 1 - \frac{1}{m}. \quad (7)$$

And the probability that a bit in field PVF is not set to 1 by none of $k$ independent hash functions is $P^c_k$:

$$P^c_k = \left(1 - \frac{1}{m}\right)^k. \quad (8)$$

Then, along the path of length $L$, in field PVF of aggregated $L$ validation MAC items, the probability that a bit will not be set to 1 by none of $k$ independent hash functions is $(P^c)^{k \cdot L}$.

$$\left(P^c\right)^{k \cdot L} = \left(1 - \frac{1}{m}\right)^{k \cdot L}. \quad (9)$$

So, on the path of length $L$, in field PVF of aggregated $L$ validation items, the probability that a bit is set to 1 by $k$ independent hash functions is $1 - (P^c)^{k \cdot L}$.

$$1 - \left(P^c\right)^{k \cdot L} = 1 - \left(1 - \frac{1}{m}\right)^{k \cdot L}. \quad (10)$$

Accordingly, the expectation value of bits set to 1 in field PVF based on Bloom Filter of size $m$ with $k$ independent hash functions and $L$ (path length) inserted MAC items is

$$pvf(m, k, L) = m \cdot \left(1 - \left(P^c\right)^{k \cdot L}\right) = m \cdot \left(1 - (1 - \frac{1}{m})^{k \cdot L}\right). \quad (11)$$

Assuming there are no bit position collisions between and within each MAC in field PVF of aggregated $L$ validation MAC items via $k$ independent hash functions mapping, the maximum value of function $\Gamma(pvf)$ is $\Gamma(pvf)_{max} = L \cdot k$, which means the maximum expectation value of $pvf(m, k, L)$ does not exceed $L \cdot k$. So, it is no doubt that function $pvf(m, k, L)$ is met with (12) as follows:

$$pvf(m, k, L) \leq \Gamma(pvf)_{max} = L \cdot k. \quad (12)$$

In Figure 1, for a packet of a flow, the value of field $N_{bit}$ in the tag CSC of constant-size can be preset to $\Gamma(pvf)_{max} = L \cdot k$. When a switch node receives the packet, firstly, it verifies whether the value of field $N_{bit}$ is greater than $\Gamma(pvf)_{max}$; secondly, it counts the real number of bits
set to 1 in field PVF. If the real count is greater than \( N_{\text{bit}} \), the switch node would drop the packet, which indicates that there exists malicious node attacking against the tag CSC of constant-size.

Considering the presence of malicious node \( N_{i-1} \) on the forwarding path, node \( N_{i-1} \) implements attacking via injecting or tampering packets. Node \( N_i \) receives a packet and calculates the message authentication code \( MAC_i \) of the packet and checks and verifies \( MAC_i \) in field PVF. False negatives of checking and verification are not possible (one \( MAC_i \) that is an item of the field PVF may never be reported as not being an item). However, checking and verification are of a probabilistic nature and false positives are possible (one \( MAC_i \) that is not an item of the field PVF may be reported to an item); according to [23], the false positive rate is shown as follows:

\[
P_{\text{FPR}} = \left(1 - e^{-\frac{K}{L}}\right)^n.
\]

Thus, a packet injected or tampered by node \( N_{i-1} \) will be verified to be invalid with probability \( 1 - (1 - e^{-\frac{K}{L}})^n \) by the next hop node \( N_i \) and will be dropped. A packet of malicious node \( N_{i-1} \) injecting or tampering is verified to be valid by all the downstream nodes \( N_i \) \( (i \leq L) \) along the path, which indicates that the malicious node has successfully implemented an attack, and the probability is denoted as

\[
P_{\text{FPR}} = \prod_{j=i+1}^{L-1} \left(1 - e^{-\frac{K}{L}}\right)^n.
\]

Even if an attacker has controlled a switch node completely, it is difficult to fabricate the field PVF based on Bloom Filter since the result of cryptographic hash function \( MAC_K \) cannot be guessed without availability of other node’s shared session key \( K \). It is reasonable to assume that the function \( MAC_K(\cdot) \) with shared key \( K \) generates pseudorandom outputs. Via \( k \) independent hash functions \( h_1, h_2, \ldots, k \) mapping, the adversary needs to determine which bits should be set to 1 or set to 0 in the field PVF of CSC, so the attacker can not exploit.

### 6. Experiment and Evaluation

In this section, we further prototype CSCPV with Mininet simulation network environment to evaluate the effectiveness of proposed scheme by behavioral-model version 2 (BMV2) and programmable P4 switch. Experiments include the number of bits set to 1 in field PVF of tag CSC at different path lengths, malicious node tampering attacking success rate, anomaly localization accuracy, and evaluation of the proposed scheme network performance.

#### 6.1. Experiment Setup

With 64-bit Ubuntu 16.04 operating system, the simulation platform is configured with Intel (R) Core (TM) i7-8550 CPU, 1.8 GHz, 8 GB of memory. Our experiments are performed on Mininet, programmable P4 software switch, and controller components based on P4Runtime interface. In this paper, we extend the switch behavior model BMV2 using C++ to implement the anonymous key negotiation and packet forwarding verification of CSCPV. The process for P4 switch of CSCPV is shown in Figure 5, which includes Input, Parse, Ingress, Egress, Output, etc. For more details about P4, please refer to the literature [5]. And the virtual network simulation prototype is composed of 30 virtual P4 switches and several virtual host terminals.

#### 6.2. The Number of Bits Set to 1

**Experiment 1.** The experiment tests the number of bits set to 1 in PVF at different path lengths and different numbers of \( k \) independent hash functions. According to the description and discussion in Section 4 and Section 5, the ingress switch of a flow embeds a tag CSC of constant-size in packets, and aggregated \( L \) (path length) MAC items based on temporary shared key with node on the forwarding path are then superimposed in the field PVF based on Bloom Filter data structure. Each downstream switch node verifies packets based on field PVF. The value of field \( N_{\text{bit}} \) in CSC is the maximum number of bits set to 1 in field PVF. In order to make the tag CSC of constant-size immune to the attack that attacker sets all bits to 1 in field PVF which make all packets injected or tampered verified to be valid by all downstream switch nodes, we should limit the value of field \( N_{\text{bit}} \) to restrain these attacks. Different path lengths and the number of independent hash functions result in different numbers of bits set to 1 in PVF. The experiment results of number of bits set to 1 in PVF under different path lengths circumstances are shown in Figure 6.

Figure 6 depicts the results for the number of bits set to 1 in PVF. We performed experiments for different path lengths (\( L \) varying from 2 to 20) and different numbers of \( k \) (\( k = 3 \) and \( k = 4 \)) independent hash functions. Figure 6 shows that when the path length \( L \) is 20 hops and the number of hash functions is \( k = 3 \) and \( k = 4 \) respectively, the number of bits set to 1 in PVF based on Bloom Filter is about 50 and 62, respectively, less than half of size \( m \) of the PVF (here \( m = 128 \)). When the path length is almost the average value of the internet transmission path, i.e., \( L = 13 \) hops [24], \( k = 3 \), and \( k = 4 \), respectively, the number of bits set to 1 in
PVF is about 34 and 43, respectively. So, the field PVF of size \( m \) (128 bits) in tag CSC really meets the network packet transmission path requirement. And the experiment results are in line with (11).

### 6.3. Malicious Tampering Success Rate

**Experiment 2.** This experiment tests the success rate of tampering packet by malicious nodes. We selected a path whose length is 12 hops and picked switch node \( N_5 \) as an adversary node. Node \( N_5 \) tampers network packets via probabilities \( \lambda = 0.1 \) and \( \lambda = 0.2 \), respectively. We performed experiment using different numbers of \( k \) (\( k = 3, k = 4 \)) independent hash functions.

As shown in Figure 7, when \( k = 3 \), node \( N_5 \) performs tampering attack via probabilities \( \lambda = 0.1 \) and \( \lambda = 0.2 \), respectively, node \( N_6 \) receives and verifies packets, and the false positives rates (packets tampered be verified to be valid) are about 1.7% and 1.68%, respectively. And when \( k = 4 \), the false positives rates are about 1.42% and 1.38%, respectively, at the node \( N_6 \). When the packets tampered by \( N_5 \) are verified to be valid by \( N_6 \) and forwarded to node \( N_7 \), the false positives rates are about ranging from 0.01% to 0.02% at node \( N_7 \) and 0% at node \( N_8 \). According to (14), for a packet tampered by the malicious node \( N_5 \), the probability that the packet tampered by \( N_5 \) and verified to be valid along the downstream nodes \( N_7 \) to \( N_{12} \) trends toward 0.0%. So, there is hardly any possibility of a packet tampered by a malicious node and passed through the network to destination.

### 6.4. Localization Accuracy

**Experiment 3.** We performed experiments for testing controller anomaly localization accuracy for malicious node tampering or dropping attack. According to problem description in Section 3, tampering a packet is equivalent to dropping an original packet and injecting a forged packet at the same time, while hijacking attack is equivalent to
dropping the forwarded packets on the authorized path. So, this experiment mainly considers the two types of anomaly localization for malicious nodes tampering and dropping against packets. In Section 4.4, controller localizes anomaly link by comparing the statistics of valid packets received by adjacent nodes on the forwarding path. The natural packet loss rate of the link is about $\theta \approx 0.001$; we evaluate the anomaly localization accuracy with variation of packet tampered or dropped probability of misbehaved switch nodes. And the controller performs an anomaly detection per 300 ms.

As the following, Figure 8 depicts the experiment results. We make a further evaluation for the localization accuracy of tampering and dropping against packets, respectively, in terms of different values of attacking rate. We pick node $N_5$ as an adversary, and $N_5$ continuously alters and drops packets, respectively, with probability varying from 0.03% to 0.3%. As Figure 8 shows, when the malicious node $N_5$ performs tampering or dropping attacks with trivial probability varying from 0.03% to 0.09%, the anomaly localization accuracy ranges from 40% to 80%; when the attacking probability increases from 0.15% to 0.18%, the anomaly localization accuracy is about 90%, where localization accuracy becomes more accurate when the value of attacking rate increases. And while the attack probability varies from 0.21% to 0.3%, localization accuracy is above 95%.

6.5. Performance Evaluation. In this subsection, we evaluate the performance of proposed mechanism, including the computation overhead during packets transmission, packets transmission round trip time (RTT), the network throughput, and header space communication overhead.

6.5.1. Computation Overhead. The computation overhead of packets transmission is the major factor which affects forwarding delay and network throughput. The computation overhead of existing typical schemes such as OPT [6] and SDNsec [7] is shown in Table 2.

From Table 2, we learn that the computation overhead of CSCPV is less than the existing typical linear-scale counterparts. OPT [6] presents a lightweight packet verification mechanism where the sender embeds all validation tags of intermediate nodes along the packet forwarding path, each intermediate router performs two times of message authentication code computation operations, the destination receivers need $L$ times, and the total computation overhead is $4L \times M$. SDNsec [7] verifies the path compliance and consistency of switches in SDN. However, to check whether each packet of a flow has really followed the authorized path, the egress switch needs to report each packet with cryptographic tags to the controller, and the total computation overhead is $2L \times (E + M)$. In CSCPV, the ingress switch node needs to perform $L$ message authentication code computation operations, and each downstream node only computes one time, so the total overhead of CSCPV is $2L \times M$.

Experiment 4. We performed experiments for testing round trip time of running CSCPV and nonrunning CSCPV protocol (Baseline) at different path lengths. Moreover, we test round trip time of SDNsec and OPT under the same circumstances of network simulation environment, and the experiment results are shown in Figure 9.

From Figure 9, we learn that the average round-trip time of the Baseline of 6 hops and 8 hops is about 14.5 ms and 20 ms, respectively, while CSCPV is approximate to 15.5 ms and 22 ms; CSCPV introduces less than 11% of additional forwarding delays on average. SDNsec and OPT are slightly higher than CSCPV, which introduces a range from 13% to 14% of additional forwarding delay on average.

Experiment 5. We performed experiments for testing network throughput of running CSCPV and nonrunning...
CSCPV protocol (Baseline) on the path of 8 hops. Packet payload size varies from 300 bytes to 1200 bytes. Moreover, we test the SDNsec and OPT network throughput under the same circumstances of network simulation environment, and the experiments results are shown in Figure 10. What we need notice is that our experiment platform is not the real network environment but the virtual network, so there exists a big difference about the throughput with the real network.

From Figure 10, with the same payload size, we learn that SDNsec and OPT network throughput is about 13% degradation, while CSCPV is no more than 10% throughput degradation.

6.5.2. Communication Overhead. The header space communication overhead is an extra portion of the normal IP packet (i.e., SDNsec header, OPT header, and CSCPV header). From SDNsec and OPT, we know the embedded tags lengths of SDNsec and OPT increase as forwarding path lengthens, with the formulation as $22 + 8 \times L$ and $52 + 16 \times L$, respectively. Table 3 shows the header space communication overhead of SDNsec, OPT, and CSCPV at different path lengths. When path length varies from 4 to 16, SDNsec’s header space overhead increases from 54 to 150 bytes, and OPT increases from 116 to 372 bytes, while the proposed scheme of CSCPV overhead is constant-size of 32 bytes. Here, we define header space communication overhead ratio $\kappa$ between header space tagged and entire packet payload size, e.g., for packet payload size of 1024 bytes, $\kappa_{\text{OPT}} = 52 + 16L/1024$, $\kappa_{\text{SDNsec}} = 22 + 8L/1024$, and $\kappa_{\text{CSCPV}} = 32/1024$. Specifically, Table 4 shows that when the path length is 10 hops (the average internet transmission path length is 13 hops [24]), packet payload size varying from 128 to 1024 bytes, the header space communication overhead ratio $\kappa$ of SDNsec varies from 79.6% to 9.96%, and

\begin{verbatim}
for (i = 1; i < L; i++) {
    if (count[i] > count[i+1]) {
        if ((count[i] − count[i+1])/count[i] < \theta) continue;
        if ((count[i] − count[i+1])/count[i] > \theta) {
            link{i} is abnormal link.
            break;
        }
    }
}
\end{verbatim}

**ALGORITHM 1**: Controller anomaly localization.
The analyses and experiment results, we can conclude that the data used to support the findings of this study can be obtained from the corresponding author upon request.

| Table 3: Header space overhead with different path length. | Header space communication overhead |
|----------------------------------------------------------|-----------------------------------|
| L = 4 hops | L = 8 hops | L = 12 hops | L = 16 hops |
| CSCPV | 32 B | 32 B | 32 B | 32 B |
| SDNsec | 54 B | 86 B | 118 B | 150 B |
| OPT | 116 B | 180 B | 244 B | 372 B |

| Table 4: Header space overhead ratio with path length L = 10. | Header space overhead ratio of different payload |
|----------------------------------------------------------|-----------------------------------|
| 128 B | 256 B | 512 B | 1024 B |
| CSCPV | 25% | 12.5% | 6.25% | 3.13% |
| SDNsec | 79.6% | 39.8% | 19.9% | 9.96% |
| OPT | 100% | 82.8% | 41.4% | 20.7% |

OPT varies from 100% to 20.7%, while only 25% and 3.13% communication overhead ratio κ of CSCPV. According to the analyses and experiment results, we can conclude that CSCPV outperforms existing SDNsec and OPT.

7. Conclusion

The existing packet forwarding verification solutions in SDN enable packet verification by embedding linear-scale cryptographic tags as forwarding path lengths, which introduce significant computation and communication overhead. We present CSCPV, a packet forwarding verification mechanism based on constant-size credential in SDN. In CSCPV, the ingress switch embeds a tag of constant-size credential which is independent of the length of packet forwarding path. Each downstream node verifies packets based on the constant-size credential. And the controller periodically obtains forwarding statistics of each node on the path to locate network anomaly. We further prototype and evaluate the proposed CSCPV. The analyses and experiments results show that CSCPV computation and communication overhead are less than similar linear-scale counterparts. With less than 11% additional forwarding delay and no more than 10% throughput degradation, CSCPV achieves efficient packet forwarding and can effectively detect and locate anomaly. With no doubt, the bigger the size of cryptographic data embedded, the higher the network communication overhead. In CSCPV, when the path length is a small value, the credential tags inserted are 32 bytes of fixed-size still. In future work, we will plan to focus on the credential tags that dynamically change at different path lengths. Concretely, according to the variation of path length, leveraging on the Bloom Filter, we will try to research self-adaptive-size packet verification credential to further reduce the network communication overhead without loss security.

Data Availability

The data used to support the findings of this study can be obtained from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the publication of this paper.

Authors’ Contributions

Ping Wu and Chao-wen Chang contributed equally to this work.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant no. 61572517) and the Science and Technology Project of Henan Province of China (Grant no. 222102210070).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008.
[2] D. Singh, A. Shiv, and S. K. Chamoli, “Software defined networking (SDN) challenges, issues and solution[1],” INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING, vol. 7, no. 1, pp. 884–889, 2019.
[3] C. Pappas, R. M. Reischuk, and A. Perrig, "FAIR: forwarding accountability for internet reputability," in Proceedings of the 2015 IEEE 23rd International Conference on Network Protocols (ICNP), November 2016.
[4] A. Akhunzada, A. Gani, N. B. Anuar et al., “Secure and dependable software defined networks,” Journal of Network and Computer Applications, vol. 61, no. C, pp. 199–221, 2016.
[5] P. Bosshart, D. Daly, G. Gibb et al., “P4,” ACM SIGCOMM - Computer Communication Review, vol. 44, no. 3, pp. 87–95, 2014.
[6] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," ACM SIGCOMM Computer Communication Review, 2014.
[7] T. Sasaki, C. Pappas, T. Lee, H. Torsten, and A. Perrig, “SDNsec: forwarding accountability for the SDN data plane,” in Proceedings of the International Conference on Computer Communication & Networks, Kauai, Hawaii, USA, February 2016.
[8] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra, "Verifying and enforcing network paths with ICING," Proceedings of ACM CoNEXT, Tokyo, Japan, December 2011.
[9] B. Sengupta, Y. Li, and K. Bu, "Privacy-preserving network path validation," ACM Transactions on Internet Technology, vol. 20, no. 1, p. 27, 2020.
[10] T. Wolf, S. Natarajan, and K. T. Vasudevan, "High-performance capabilities for 1-hop containment of network attacks," IEEE/ACM Transactions on Networking, vol. 21, no. 6, pp. 1931–1946, 2013.
[11] Z. Peng, S. Xu, Z. Yang et al., “FOCES: detecting forwarding anomalies in software defined networks,” in Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, Vienna, Austria, July 2018.
[12] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, “Sphinx: Detecting security attacks in software-defined networks,” in
[13] A. Kaminski and C. Fung, “FlowMon: Detecting malicious switches in software-defined networks,” in Proceedings of the ACM CCS Workshop on Automated Decision Making for Active Cyber Defense, pp. 39–45, Denver Colorado USA, October 2015.

[14] C. Pang, J. Yong, and L. Qi, “PADE: detecting forwarding anomaly in software-defined networks,” in Proceedings of the IEEE International Conference on Communications, IEEE, Kuala Lumpur, Malaysia, May 2016.

[15] H. Yang and S. S. Lam, “Scalable verification of networks with packet transformers using atomic predicates,” IEEE/ACM Transactions on Networking, vol. 25, no. 5, pp. 2900–2915, 2017.

[16] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, “VeriFlow: Verifying network-wide invariants in real time,” ACM SIGCOMM - Computer Communication Review, vol. 42, no. 4, pp. 467–472, 2012.

[17] A. Horn, A. Kheradmand, and M. R. Prasad, “Delta-net: real-time network verification using atoms,” in Proceedings of the Networked Systems Design & Implementation NSDI, Boston, MA, USA, March 2017.

[18] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, “Automatic test packet generation,” IEEE/ACM Transactions on Networking, vol. 22, no. 2, pp. 554–566, Apr. 2014.

[19] Z. Peng, L. Hao, C. Hu, L. Hu, and X. Lei, “Stick to the script: monitoring the policy compliance of SDN data plane,” in Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems, IEEE, Santa Clara California USA, March 2016.

[20] B. Kai, X. Wen, Y. Bo, Y. Chen, E. L. Li, and X. Chen, “Is every flow on the right track?: inspect SDN forwarding with RuleScope,” in Proceedings of the IEEE INFOCOM 2016 - IEEE Conference on Computer Communications, IEEE, San Francisco, CA, USA, April 2016.

[21] C. Hu, Z. Peng, and C. Zhang, “Fast testing network data plane with RuleChecker,” in Proceedings of the IEEE International Conference on Network Protocols, IEEE Computer Society, Toronto, ON, Canada, October 2017.

[22] C. Dario, F. Dario, and G. Rosario, “Certificateless onion routing,” in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS’09), pp. 151–160, IL, USA, April 2009.

[23] A. Broder and M. Mitzenmacher, “Network applications of Bloom filters: a survey,” Internet Mathematics, vol. 1, no. 4, pp. 485–509, 2004.

[24] B. Huffaker, M. Fomenkov, D. J. Plummer, M. David, and K. Claffy, “Distance metrics in the internet,” in Proceedings of the IEEE International Telecommunications Symposium (ITS), September 2002.