P4-sKnock: A Two Level Host Authentication and Access Control Mechanism in P4 based SDN

Aneesh Bhattacharya
Dept of CSE
IIIT Naya Raipur, India
aneesh19100@iiitnr.edu.in

Risav Rana
Dept of ECE
IIIT Naya Raipur, India
risav19101@iiitnr.edu.in

Suvrima Datta
Dept of CSE
IIIT Naya Raipur, India
suvrima@iiitnr.edu.in

Venkanna U.
Dept of CSE
IIIT Naya Raipur, India
venkannau@iiitnr.edu.in

***Abstract*—The adoption of Software-Defined Networks (SDN) and the shift towards programmable data planes have led to better network management. However, this has not been accompanied with the implementation of robust host authentication or access control mechanisms to improve network security and prevent unauthorized access to the network. The current literature has explored the implementation of the widely adopted authentication mechanism - port knocking in SDN to address the former. However, they suffer from two major drawbacks making them vulnerable to MITM (Man-In-The-Middle) attacks: unsecured transfer of the port knocking sequences between the SDN controller and hosts, and the lack of host identity verification mechanisms post port knocking authentication. This paper introduces P4-sKnock: a P4 based two level host authentication and access control mechanism. The first level introduces encrypted dynamic port knocking to secure the transfer of port knocking sequences over a compromised channel by encrypting them. Further, a challenge-response host identity verification mechanism is introduced as a second level authentication measure following which a host can be authorized, quarantined or blocked owing to the programmability of the P4 switch providing robust access control. Experimental analysis shows that P4-sKnock can authenticate a new SDN host within 500 ms and mitigate MITM attacks like IP spoofing and replay attacks making it significantly more secure than previous P4 based port knocking authentication systems.**

***Index Terms*—SDN, P4, port knocking, encrypted dynamic port knocking, MITM, replay attack, IP spoofing**

I. Introduction

The adoption of SDN has led to a significant increase in the manageability and programmability of networks [1]. Separation of the data plane and control plane has not only reduced the dependency of users on private vendors but has also provided opportunities for them to design, develop and test their own networking algorithms. Initially, OpenFlow [2] was introduced as one of the first SDN standards but it had its own drawbacks like limited support for a fixed number of header fields and new hardware requirement to support new versions being some of them. Keeping the original SDN architecture intact, P4 (Programming Protocol-Independent Packet Processors) [3] was proposed to offer more control in the data plane by specifying how data plane devices will process packets. However, SDN is still in its initial stages of implementation and its architecture has some pressing vulnerabilities [4], [5] which can be exploited by attackers. Some of the notable ones

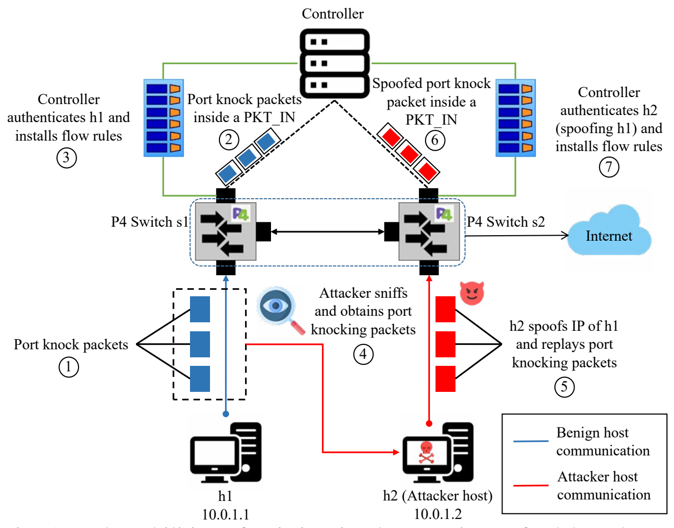

Controller
Controller authenticates h1 and installs flow rules
③
Port knock packets inside a PKT_IN
②
Spoofed port knock packet inside a PKT_IN
⑥
Controller authenticates h2 (spoofing h1) and installs flow rules
⑦
P4 Switch s1
P4 Switch s2
Internet
Attacker sniffs and obtains port knocking packets
④
Port knock packets
①
h2 spoofs IP of h1 and replays port knocking packets
⑤
h1
10.0.1.1
h2 (Attacker host)
10.0.1.2
Benign host communication
Attacker host communication

Fig. 1: Vulnerabilities of existing implementations of P4 based port knocking in SDN environments being exploited by an attacker (*h2*)

are the absence of reliable interconnections among network elements and exposed communications channels. Further, the lack of authentication mechanisms in SDN leads to adversaries gaining illicit access to the network [6]. Vulnerability to attacks like Denial of Service (DoS) and MITM attacks (spoofing) can be exploited by attackers to hamper the confidentiality and integrity of the network and compromise hosts and switches [7]. This in turns leads to the lack of trust on the network, making it unusable and resulting in poor QoS.

In the current literature, firewalls are used as a first line of defence against malicious SDN hosts [8]. However, to establish trust and reliability among hosts in the network, robust host authentication mechanisms must be present for SDN environments after the initial screening by firewalls. In traditional networks, a well known host authentication mechanism: port knocking is used [9]. In port knocking, a host sends packets (port knocks) in a specific sequence to the specified server ports. This sequence is a secret known only by the host and the server. The server verifies the sequence of port knocks and allows a connection to be established by the host if sequence is correct. In an SDN context, this technique has been recently adapted and implemented as a mechanism to authorise hosts [10] [11]. Further, the offloading of the port knocking mechanism to P4 switches has also been explored [12]. A key drawback of these approaches is that the same knocking sequence is used by the hosts to connect to the server. If this sequence is compromised, the authentication mechanism fails [13]. To overcome this, [14] introduced the concept of dynamic port knocking in SDN. However, there are still issues with the current literature on the implementation of port knocking in SDN which can be summarized using Fig. 1 as follows:

- The absence of encryption of the port knocking sequence during its transfer between the controller and hosts or vice-versa can lead to its compromise. The communication from a benign host to switches can be intercepted by attackers shown by label 4. Then attackers can spoof the IP of an authorized host and send the sniffed port knock sequence to gain access to the network.
- The lack of a challenge-response mechanism post port knocking authentication to verify a host's identity can lead to attackers establishing unauthorized connections by simply replaying the port knocking packets without having to explicitly verify their identity as demonstrated by labels 5, 6 and 7.

To overcome the above highlighted issues, this paper proposes P4-sKnock: a P4 based two level host authentication and access control mechanism for SDN environments. The proposed solution has two stages: (a) encrypted dynamic port knocking and (b) challenge-response host identity verification. The first part is enforced when a new host tries to communicate with another host in the network. The controller assigns a unique port knocking sequence to this new host which is RSA (Rivest-Shamir-Adleman) encrypted and sent as the payload of three consecutive packets. In this manner, even if the communication channel is compromised and the attackers sniff and intercept the packets, they cannot obtain the port knocking sequence as it is encrypted. The second part is enforced when this host tried to connect to the network using the issued knocking sequence. The encrypted knocking sequences are verified by the controller who then issues a challenge to this host for the verification of its identity. The challenge issued is a randomly generated 256 bit key which is RSA encrypted using the public key of the host. The host is required to respond with the decrypted key, re-encrypted with the controller's public key and digitally signed by it using the RSA digital signature algorithm to prevent the key from being exposed by MITM attacks. The host can only decrypt the key and correctly digitally sign it if it has the genuine private key. Thus, this step protects against replay and spoofing attacks by attackers as only genuine hosts can respond with the correct signed key and be allowed to connect to the network.

In summary, our paper makes the following significant contributions:

- Securing the transfer of port knocking sequences between the SDN controller and hosts over an unsecured communication channel and protecting the network against MITM attackers by using two step authentication.
- Enforcing access control via the SDN controller to allow, quarantine or block hosts from communicating in the network by dynamically updating Match-Action flow rules in the P4 programmable switches.
- Implementing such a two step solution for SDN host authentication and access control in mininet using BMV2 switches and analysing its latency to show the effectiveness of the solution in a real life deployment scenario.

The structure of the paper is as follows - Section II formulates the problem statement followed by section III which describes the proposed methodology of the P4-sKnock host authentication and access control mechanism. Section IV presents the implementation and result analysis of our proposed solution. Conclusion and areas of future research are lastly described in section V.

II. Problem Formulation

The existing port knocking authentication mechanisms in SDN follow an architecture which consider a host to host communication scenario. Packets from authenticated hosts $\{h_1, h_2, ..., h_i\} \in H_{auth}$ are forwarded through programmable switches S_w. An SDN controller C is used to update flow rules in the switches. There exist new hosts $\{n_1, n_2, ..., n_j\} \in N$ that are previously unseen by the switches S_w and the controller C, which need to be authenticated by port knocking before they can communicate in the network. Each host has a unique IP address $\{ip_1, ip_2, ..., ip_x\} \in IP_{id}$ which is used as their identifier. Let there exist an unseen host n_γ with IP ip_γ and an attacker host n_θ with IP ip_θ. Host n_γ tries to communicate to an authenticated host h_4 and n_θ sniffs the network. To join the network, n_γ performs port knocking and gets authenticated as,

$$n_\gamma \xrightarrow{\text{Sends port knocking packets}} S_w \tag{1}$$

$$n_\gamma \in H_{auth} \tag{2}$$

Attacker host n_θ sniffs these port knocking packets and further obtains the IP of n_γ and spoofs it,

$$n_\theta \xleftarrow{\text{Obtain } ip_\gamma} n_\gamma \tag{3}$$

$$ip_\theta = ip_\gamma \tag{4}$$

Now n_θ has the identifier IP as ip_γ and can simply replay the port knocking packets pretending to be n_γ. This will allow n_θ to be authenticated by port knocking and perform attacks on the desired host.

$$n_\theta \xrightarrow{\text{Replay port knocking packets}} S_w \tag{5}$$

$$n_\theta \in H_{auth} \tag{6}$$

$$n_\theta \xrightarrow{\text{Perform attacks}} h_4 \tag{7}$$

This shows how the existing port knocking implementations in SDN are vulnerable to IP spoofing and replay attacks.

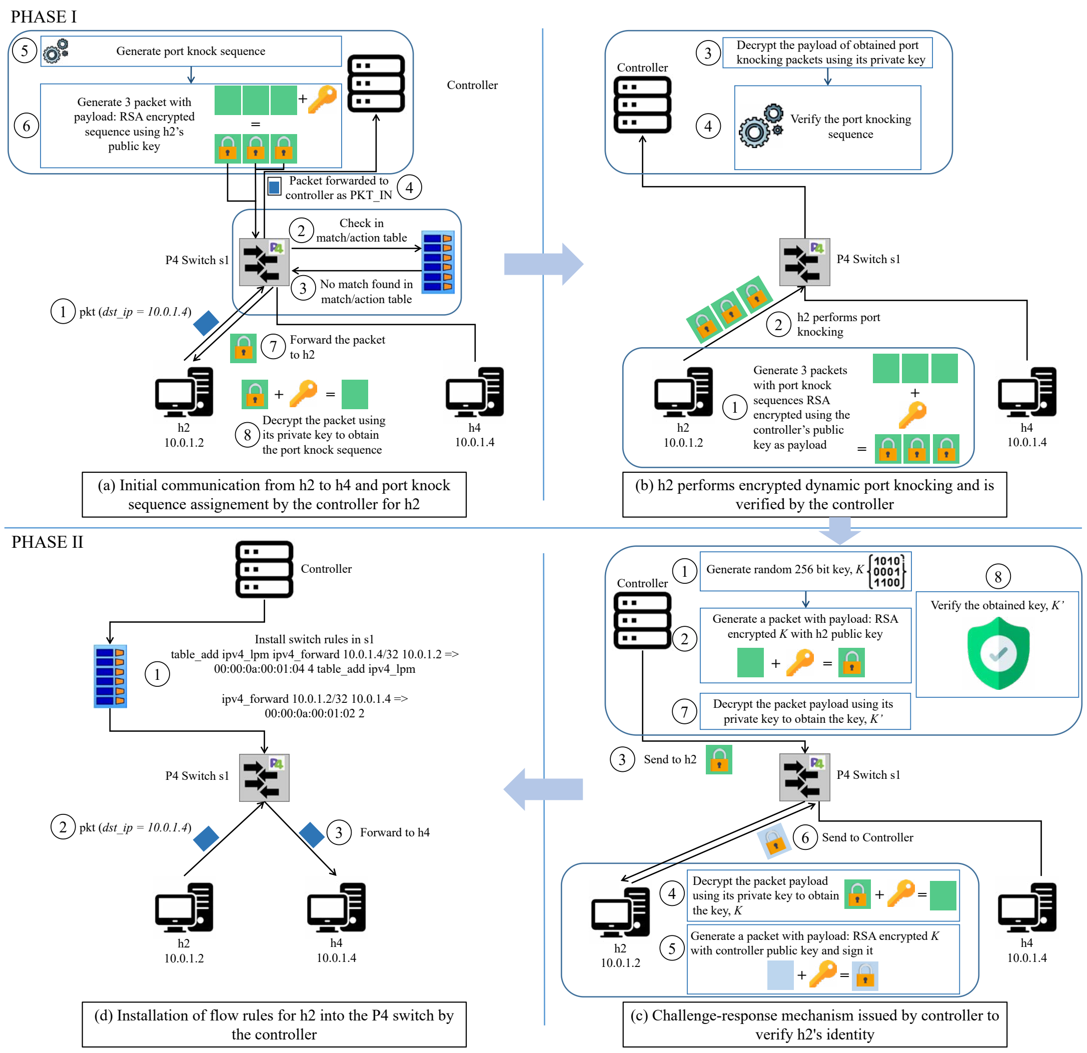

PHASE I
5 Generate port knock sequence
6 Generate 3 packet with payload: RSA encrypted sequence using h2's public key
Controller
Packet forwarded to controller as PKT_IN 4
2 Check in match/action table
3 No match found in match/action table
P4 Switch s1
1 pkt (dst_ip = 10.0.1.4)
7 Forward the packet to h2
8 Decrypt the packet using its private key to obtain the port knock sequence
h2 10.0.1.2
h4 10.0.1.4
(a) Initial communication from h2 to h4 and port knock sequence assignement by the controller for h2
3 Decrypt the payload of obtained port knocking packets using its private key
4 Verify the port knocking sequence
2 h2 performs port knocking
1 Generate 3 packets with port knock sequences RSA encrypted using the controller's public key as payload
(b) h2 performs encrypted dynamic port knocking and is verified by the controller
PHASE II
Install switch rules in s1
1 table_add ipv4_lpm ipv4_forward 10.0.1.4/32 10.0.1.2 => 00:00:0a:00:01:04 4 table_add ipv4_lpm ipv4_forward 10.0.1.2/32 10.0.1.4 => 00:00:0a:00:01:02 2
2 pkt (dst_ip = 10.0.1.4)
3 Forward to h4
(d) Installation of flow rules for h2 into the P4 switch by the controller
1 Generate random 256 bit key, K 1010 0001 1100
2 Generate a packet with payload: RSA encrypted K with h2 public key
7 Decrypt the packet payload using its private key to obtain the key, K'
8 Verify the obtained key, K'
3 Send to h2
6 Send to Controller
4 Decrypt the packet payload using its private key to obtain the key, K
5 Generate a packet with payload: RSA encrypted K with controller public key and sign it
(c) Challenge-response mechanism issued by controller to verify h2's identity

Fig. 2: Proposed P4-sKnock host authentication and access control methodology

III. Proposed Methodology

This section describes the host authentication and access control mechanism of P4-sKnock which is divided into two phases overall as illustrated in Fig. 2. The first phase is focused on encrypted dynamic port knocking based authentication of a new host followed by the second phase which initiates a challenge-response mechanism to verify the identity of the host. The flow of action in each phase is numbered numerically in the figure.

A. System Components

The details of each component of the P4-sKnock host authentication mechanism is as follows:

1) Controller: The controller is a central entity that keeps track of every host connected to the network. It contains a public key repository module, a port knock sequence generator module and a module to encrypt and decrypt the data transferred as packet payloads. It oversees the entire encrypted dynamic port knocking and challenge-response host identity verification mechanism in our solution. It can also insert or delete flow rules from the Match-Action table at the P4 switches.
2) P4 switch: The controller leverages the programmability of the switch to provide access control in our solution by issuing Match-Action rules to it to forward or drop packets.
3) Hosts: The hosts are edge devices that have access to a public key repository module and can encrypt or decrypt packet payloads and digitally sign them using the RSA algorithm.

B. Phase 1: Encrypted Dynamic Port Knocking Based Host Authentication

*1) **Initial communication from a new host:*** When a host tries to send a packet to another host in the network, the packet is forwarded through the P4 switch. Forwarding is only possible when there exists appropriate packet forwarding flow rules in the Match-Action tables of the switch. However, when a new host tries to communicate via the switch, there are no existing flow rules for this host in the Match-Action tables. In such a scenario, we forward the packet to the controller. Upon receiving the forwarded packet of a new host, the controller first checks its memory for existing port knocking rules for this host. If there are no such rules for the host, the controller generates a random port knocking sequence consisting of three ports e.g 1111, 2222, 3333. This is followed by the generation of three separate TCP SYN packets, each containing the encrypted port knocking sequences as the payloads. RSA encryption algorithm is used to encrypt the port knocking sequences using the public key of the host. These packets are then sent to the host. The host decrypts the port knocking sequence using its own private key. This step is shown in Fig. 2 (a).

*2) **Encrypted dynamic port knocking**:* To perform encrypted port knocking, a host is required to send three separate TCP SYN packets with each of the port knocking sequences, encrypted using the controller's public key as the payload. These packets are decrypted and the sequence is verified at the controller. If the correct sequence is sent, phase 2 is initiated. Else the host is quarantined. This process is explained in Algorithm 1 and also shown in Fig. 2 (b). Post completion of the first level authentication of a host using encrypted dynamic port knocking, the second level authentication of the host's identity using the challenge-response mechanism is initiated.

Algorithm 1: Encrypted Dynamic Port Knocking

```
Input: Host IP address: $ip$, Knock_Sequence_Counter:
       $seq\_count$ = 0, Controller private key: $K$
procedure PortKnock(pkt_in):
Receive packet at switch
At switch:
Do table lookup on present table
if match is not found then
    Send packet to controller via get_digest action
    At controller:
    if $seq\_count < 3$ then
        get knock sequence list $H\_seq$ of that $ip$ from
          memory
        Decrypt packet payload using $K$ as decrypted
          sequence $seq$
        if $seq == H\_seq[seq\_count]$ then
            continue
        end
        else
            Install drop rules for $ip$
        end
        if $seq\_count == 3$: then
            challenge_response_mechanism()
        end
    end
end
else
    Quarantine the host
end
```

C. Phase 2: Challenge-Response Host Identity Verification

An existing challenge with the current literature of port knocking in SDN environments is the absence of host identity verification mechanisms, making them vulnerable to IP spoofing and replay attacks. In our proposed solution, when a host authenticates itself using encrypted dynamic port knocking, we further verify its identity using a challenge-response mechanism.

*1) **Challenge-response mechanism**:* To verify the authenticity of a host, the controller generates a random 256 bit key and issues it as a challenge to the host. This key is encrypted with the public key of the host and communicated to it as the payload of a TCP SYN packet. The host is now required to send a packet as a response to the last destination port of the port knocking sequence. This packet must contain the 256 bit key, encrypted using the controller's public key and digitally signed by the host in its payload. These conditions require the host to decrypt the key using its own private key and further use it to perform the RSA digital signature. If the host is authentic and has the correct private key in its possession, the response will have a genuine digital signature and contain the correct 256 bit key. Else, the host will fail to reply with a packet meeting the set criteria and would in turn be classified as a malicious host. This step is shown in Fig. 2 (c) and explained in Algorithm 2.

*2) **Enforcing access control**:* Upon conclusion of the two level authentication process of the proposed P4-sKnock, access control is enforced. A new flow rule is inserted in the P4 switch. For an authenticated host, the flow rule inserted is a forwarding rule allowing the communication between this new host and its desired host in the network. However, upon failing to authenticate itself the host is quarantined and the controller inserts drop rules for it. Any further packets sent by this host are dropped at the P4 switch. This step is shown in Fig. 2 (d).

IV. Implementation and Result Analysis

A. Testing Environment

For implementation and demonstration of the solution, mininet has been used to develop the topology. The testing topology consists of eight hosts and two switches. The switches are of type BMV2 and execute the P4 program. All the hosts and switches are linked as illustrated in Fig. 3. Host *h2* is assumed to be an external host trying to communicate to an already authenticated host *h4*. Host *h5* is used as the controller. To craft packets Scapy [15] is used at the hosts and the controller. A Privacy Enhanced Mail (PEM) file containing the public keys of all the entities in the network has been maintained and is accessible by all the hosts and the controller, representing a public key repository.

B. Initial Communication From an Unauthenticated Host

For our use case, we have kept host *h2* as an unauthenticated host. Since *h2* is unauthenticated, any packet it sends to another host in the network is dropped. Fig. 4 (a) shows that the packet from *h2* to an internal host *h4* never reaches it. The P4 switch

```
Algorithm 2: challenge_response_mechanism()
Input: Controller public key: $C\_pk$, Controller private
       key: $C\_prk$, Host public key: $H\_pk$, Host
       private key: $H\_prk$
At controller:
Generate random 256-bit key: $k$
Message $Msg\_c$ = Encrypt($k$, $H\_pk$)
Send $Msg\_c$ to host
At host:
Decypted message $Msg\_h$ = Decrypt($Msg\_c$, $H\_prk$)
Response $Res$ = Digital_Sign(Encrypt($Msg\_h$, $C\_pk$))
Send $Res$ to controller
At controller:
Verify digital signature and decrypt response from
 host, $Res\_h$ = Decrypt($Res$, $C\_prk$)
if $Res\_h$ == $k$ then
    Install flow rules for the host
end
else
    Install drop rules for the host
end
```

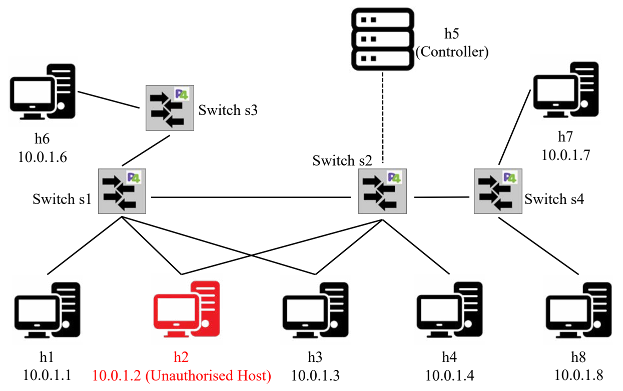
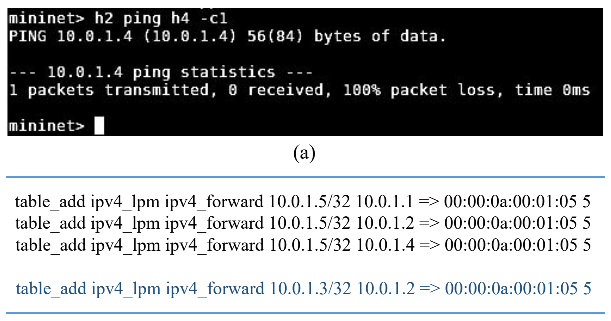


Fig. 3: Topology used for P4-sKnock

instead pushes the packet header to the controller and drops the packet. Once the controller receives the packet header, it processes the header and determines that *h2* is a new external host. The controller then installs appropriate rerouting rules for it as shown in Fig. 4 (b). Now, any future packets sent by *h2* to an internal host are rerouted to the controller, preventing this unauthorized host from communicating with any other host.

C. Encrypted Dynamic Port Knocking

Once the controller obtains the packet header from *h2*, it obtains its source and destination IPs. Next, it assigns a randomly generated port knock sequence for the device of the source IP: *h2* to communicate to the device of the destination IP: *h4*. This port knock sequence is further RSA encrypted using the public key of *h2* and sent to it as shown in Fig. 5 (a). Encrypting the port knocking sequence over the communication channel ensures that even if the sequence is intercepted by MITM attackers, it cannot be exposed without the attackers knowing the private key of *h2*. Host *h2* decrypts



```
mininet> h2 ping h4 -c1
PING 10.0.1.4 (10.0.1.4) 56(84) bytes of data.

--- 10.0.1.4 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

mininet> █
```

(a)

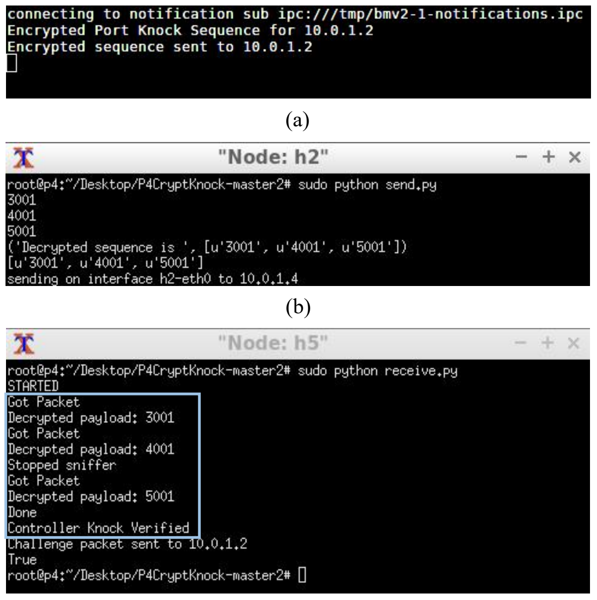
```
table_add ipv4_lpm ipv4_forward 10.0.1.5/32 10.0.1.1 => 00:00:0a:00:01:05 5
table_add ipv4_lpm ipv4_forward 10.0.1.5/32 10.0.1.2 => 00:00:0a:00:01:05 5
table_add ipv4_lpm ipv4_forward 10.0.1.5/32 10.0.1.4 => 00:00:0a:00:01:05 5

table_add ipv4_lpm ipv4_forward 10.0.1.3/32 10.0.1.2 => 00:00:0a:00:01:05 5
```

(b)

Fig. 4: Initial communication from *h2* (a) packet from *h2* to *h4* is dropped (b) rules for rerouting packet from *h2* to *h5*

the payload of the received packets using its private key and obtains the port knocking sequence. Now, to perform encrypted dynamic port knocking, *h2* generates three packets with each of the port knock sequences RSA encrypted with the public key of the controller as the payload. These packets are sent to the controller sequentially as shown in Fig. 5 (b). The controller receives and decrypts the encrypted payloads using its private key and verifies the port knocking sequence as seen in Fig. 5 (c). If any mismatch is found in the sequence or if any packet's payload fails to decrypt, the controller installs drop rules for host *h2* to quarantine it. Else, the challenge-response mechanism is initiated to verify the identity of *h2*.



```
connecting to notification sub ipc:///tmp/bmv2-1-notifications.ipc
Encrypted Port Knock Sequence for 10.0.1.2
Encrypted sequence sent to 10.0.1.2
█
```

(a)

"Node: h2"

```
root@p4:~/Desktop/P4CryptKnock-master2# sudo python send.py
3001
4001
5001
('Decrypted sequence is ', [u'3001', u'4001', u'5001'])
[u'3001', u'4001', u'5001']
sending on interface h2-eth0 to 10.0.1.4
```

(b)

"Node: h5"

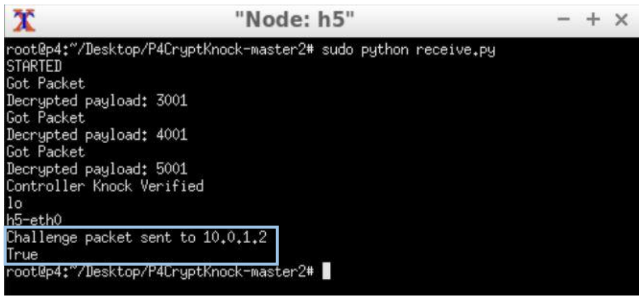
```
root@p4:~/Desktop/P4CryptKnock-master2# sudo python receive.py
STARTED
Got Packet
Decrypted payload: 3001
Got Packet
Decrypted payload: 4001
Stopped sniffer
Got Packet
Decrypted payload: 5001
Done
Controller Knock Verified
Challenge packet sent to 10.0.1.2
True
root@p4:~/Desktop/P4CryptKnock-master2# █
```

(c)

Fig. 5: Initial communication from *h2* (a) Encrypting the port knock sequence and sending to *h2* (b) sending encrypted sequence from *h2* (c) controller verifies the decrypted sequence

D. Challenge-Response Host Identity Verification

Once the port knocking sequence is verified by the controller, identity of *h2* is needed to be verified. For this, the controller first generates a random 256 bit key which is RSA encrypted using the public key of *h2*. This key is the challenge which is sent to *h2*. Host *h2* is required to respond to this challenge by sending the same key, encrypted with the public key of the controller after digitally signing it to the last server port in the port knocking sequence. To achieve this, host *h2* is required to decrypt the challenge key from the controller using its own private key as well as know the terminal server port of the decrypted port knocking sequence. Upon receiving the response, the controller verifies the digital signature as well as the key in the payload. If the correct response is received as a reply, the controller considers *h2*'s identity verified as shown in Fig. 6.



```
"Node: h5"
root@p4:~/Desktop/P4CryptKnock-master2# sudo python receive.py
STARTED
Got Packet
Decrypted payload: 3001
Got Packet
Decrypted payload: 4001
Got Packet
Decrypted payload: 5001
Controller Knock Verified
lo
h5-eth0
Challenge packet sent to 10.0.1.2
True
root@p4:~/Desktop/P4CryptKnock-master2#
```

Fig. 6: Sending challenge to *h2* and verifying the response

E. Access Control Mechanism and Latency Analysis

Once *h2* passes both the phases, the controller installs forwarding rules for that host in the switch. Host *h2* is thus authenticated and is able to communicate to the desired host in the network. The combination of both phases provides an end-to-end authentication and access control mechanism for SDN. However, for the solution to be feasible it must verify the authenticity of the host in a real time. To investigate the practicality of the proposed solution in an SDN environment, the latency of the solution is analysed which shows that the solution is able to allow, quarantine or deny communications from new hosts within an average of 500 ms. Fig. 7 (a) shows a new host being denied from communicating in the network post failing authentication. Fig. 7 (b) shows a new host being allowed to communicate in the network with the desired host after being successfully authenticated.

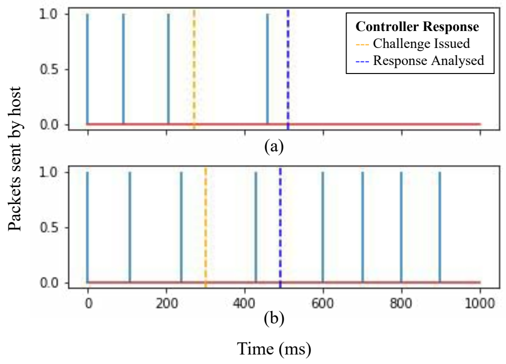


Fig. 7: Latency analysis (a) A malicious host is denied communication (b) A genuine host is allowed to communicate to the desired host

V. Conclusion and Future Work

This paper proposes a P4 based two level host authentication and access control mechanism in SDN environments and overcomes the highlighted vulnerabilities of existing implementations of port knocking in SDN environments. It secures the communication of port knocking sequences over compromised channels and further verifies the identity of hosts using a challenge-response mechanism providing 2 level security. P4-sKnock thus provides a secure and robust network security solution against Man in the Middle attacks in SDN environments. Currently, the solution is only implemented and tested in a virtual network with a limited number of hosts and switches. This work can be extended with real hardware implementation and testing for use in SDNs.

References

[1] G. P. Tank, A. Dixit, A. Vellanki, and D. Annapurna, "Software-Defined Networking-The New Norm for Networks," in *3rd National Conference on Recent Innovations in Science and Engineering*, 2012.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, mar 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," vol. 44, no. 3, p. 87–95, jul 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[4] A. Feghali, R. Kilany, and M. Chamoun, "SDN security problems and solutions analysis," in *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)*, 2015, pp. 1–5.

[5] M. B. Jiménez, D. Fernández, J. E. Rivadeneira, L. Bellido, and A. Cárdenas, "A Survey of the Main Security Issues and Solutions for the SDN Architecture," *IEEE Access*, vol. 9, pp. 122 016–122 038, 2021.

[6] A. Al Hayajneh, M. Z. A. Bhuiyan, and I. McAndrew, "Improving Internet of Things (IoT) Security with Software-Defined Networking (SDN)," *Computers*, vol. 9, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2073-431X/9/1/8

[7] A. Danping, M. Pourzandi, S. Scott-Hayward, H. Song, M. Winandy, and D. Zhang, "Threat Analysis for the SDN Architecture," https://www.opennetworking.org/, July 2016.

[8] J. Cao, Y. Liu, Y. Zhou, C. Sun, Y. Wang, and J. Bi, "CoFilter: A High-Performance Switch-Accelerated Stateful Packet Filter for Bare-Metal Servers," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[9] M. Krzywinski, "Port Knocking," https://www.linuxjournal.com/article/6811, [Online; accessed 25-March-2022].

[10] E. O. Zaballa, D. Franco, Z. Zhou, and M. S. Berger, "P4Knocking: Offloading host-based firewall functionalities to the network," in *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2020, pp. 7–12.

[11] A. Almaini, A. Al-Dubai, I. Romdhani, M. Schramm, and A. Alsarhan, "Lightweight edge authentication for software defined networks," in *Computing*, vol. 102, no. 2, 2021, pp. 291–311.

[12] A. Almaini, A. Al-Dubai, I. Romdhani, and M. Schramm, "Delegation of Authentication to the Data Plane in Software-Defined Networks," in *2019 IEEE International Conferences on Ubiquitous Computing Communications (IUCC) and Data Science and Computational Intelligence (DSCI) and Smart Computing, Networking and Services (SmartCNS)*, 2019, pp. 58–65.

[13] F. von Eye, M. Grabatin, and W. Hommel, "Detecting Stealthy Backdoors and Port Knocking Sequences through Flow Analysis," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 38, no. 3-4, pp. 97–104, 2015. [Online]. Available: https://doi.org/10.1515/pik-2015-0011

[14] A. Saxena, R. Muttreja, S. Upadhyay, K. S. Kumar, and D. V. U, "P4Filter: A two level defensive mechanism against attacks in SDN using P4," 2022. [Online]. Available: https://arxiv.org/abs/2205.12816

[15] P. Biondi, "Scapy," http://www.secdev.org/projects/scapy, [Online; accessed 25-March-2022].