Securing In-Network Traffic Control Systems with P4Auth

Ranjitha K*, Medha Rachel Panna*†, Stavan Nilesh Christian‡, Karuturi Havya Sree*†, Sri Hari Malla*†,
Dheekshitha Bheemanath*†, Rinku Shah§ and Praveen Tammana*
*IIT Hyderabad, India. ‡New York University, USA. §IIIT-Delhi, India.

***Abstract*—In-network traffic control systems built on programmable data planes enhance network performance. However, these systems also increase the attack surface and are vulnerable to attacks not seen before. We focus on a problem that stems from the fact that a programmable switch data plane trusts and processes the messages from upper layers in the switch software (OS, SDK, drivers) and from neighbor nodes in the network. Since these messages can update the state maintained in the data plane, which can influence traffic control decisions, it is important to protect such messages from adversaries aiming to degrade performance, compromise privacy, bypass security, or, worst case, network outage.**

In this paper, we present P4Auth, a key-based protection mechanism that ensures the authenticity and integrity of such messages in in-network systems making fast traffic control decisions. Our key idea is to move key-based security primitives to the switch data plane so that it reduces the trusted computing base and exposure to switch software vulnerabilities while enabling faster checks in the data plane. To realize this idea, we design and develop an authentication protocol, secure key exchange mechanism, and associated data plane primitives. We prototype P4Auth for Intel Tofino and understand the overheads of P4Auth. We also demonstrate how P4Auth protects two in-network systems from man-in-the-middle (MitM) adversaries.

***Index Terms*—programmable data plane security, in-network authentication, key exchange protocol**

I. Introduction

High-speed programmable data planes provide opportunities to design in-network systems that give better performance for various network functions such as fast reroute [1]–[3], load balance [4], intrusion detection [5]–[7], in-network compute [8], and measurement [9], [10]. At the core, these in-network systems have a fast control loop; they monitor traffic and maintain a network state, infer network conditions from the state, and act by updating table rules or registers in the switch data plane. To keep decision-making fast, some systems implement all three tasks entirely in the data plane, and some have analysis at the control plane at the cost of slow decision-making.

Despite the benefits of such in-network systems, they increase the attack surface and are vulnerable to network attacks not seen before. Attackers can influence control decisions either by sending adversarial traffic that pollutes the state [11]–[14] or by tampering with messages that update/report the state [15], [16].

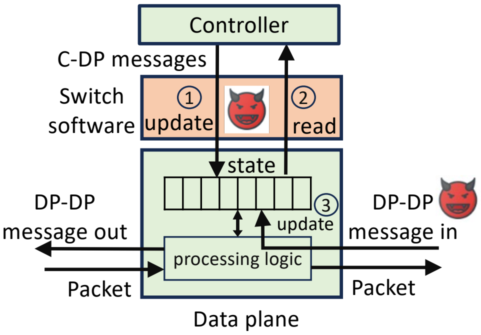


Fig. 1: **Threat model**

In this paper, we focus on the latter scenario: we consider an *adversary at the switch control plane* [16]–[21] making unauthorized modifications to the switch state in the data plane so that control decisions are manipulated. More specifically, as shown in Fig. 1, we consider two message types that update/report the state: (a) C-DP: messages from/to a controller (C) to data plane (DP) as in [3], [4], [8], and (b) DP-DP: network feedback messages that cross one or more links in the network (*e.g.*, probes) and processed entirely in the data plane as in [1], [22], [23]. In both cases, the switch data plane trusts the messages, processes them, and updates its state (*e.g.*, registers).

We investigate the following question: *how to protect fast traffic control decisions from MitM adversaries with the capability to manipulate the switch state using C-DP or DP-DP messages?*

The existing key-based security solutions [24]–[28] for authentication and message integrity have one main limitation. They trust the switch software in their solution design and are thus limited in protecting from an adversary at this layer. SSL/TLS protocol-based solutions (*e.g.*, P4Runtime [24]) secure communication between a controller and switch control plane but are not sufficient to protect from an adversary tampering arguments of function calls (*e.g.*, table rules and register values) passed to low-level switch software such as libraries, SDKs, OS, and drivers [16], [19], [21]. The existing approaches to secure BGP communication (advertisements) [29]–[31] use sophisticated cryptographic primitives, which may not be feasible to run on a programmable data plane with restrictions on per-packet operations and no for-loop support. Without these primitives in the data plane, in-

† Work done while at IIT Hyderabad

network DP-DP feedback messages must be forwarded to the control plane, which introduces delays for in-network fast reroute systems [1], [23]. To secure DP-DP communication in the data plane, recent work proposes key-based authentication techniques designed to run in the data plane [25]–[28]. However, their key management protocol exchanges messages over untrusted switch control planes or network links; thus, they are insecure.

P4Auth. In this paper, we propose P4Auth, a key-based protection mechanism that ensures the authenticity and integrity of C-DP and DP-DP messages that update/report the switch state. Our key idea is to run the key-based security operations in the switch data plane to reduce the trusted computing base and exposure to vulnerabilities at the switch software [21], [32]. Also, it enables faster checks for in-network DP-DP messages expected to be processed in the switch data plane.

Challenges. To realize this idea, we have two main challenges. The first challenge stems from the fact that the key-based protection approaches are only as good as the secrecy of the key. This means the secret keys between the controller and switch data plane (C-DP) and between two switch data planes (DP-DP) should be established without trusting the switch control plane. Inspired by TLS security protocol design [33], we propose an authenticated key exchange protocol (§VI) using the Diffie-Hellman (DH) algorithm to derive a pre-master secret, followed by master secret derivation using a key derivation function (KDF). This approach ensures secure key exchange and the confidentiality of the shared key. The second challenge pertains to implementing the data plane modules (*i.e.*, DH, KDF, HMAC) for secure key exchange and authentication checks on a data plane with restrictions on supported per-packet operations and a lack of security primitives. We address them by (1) carefully integrating a modified DH [25], [34], replacing exponentiation with *AND* and *XOR* operations while preserving the confidentiality of keys; (2) designing a custom KDF, inspired from TLS1.3's *Extract-and-Expand* principle [33], [35] that is proven to be secure [36]; and (3) realizing key-based authentication using hash-based message authentication code (HMAC) which can be implemented using simple arithmetic operators such as AND, XOR, and $rotate$ [28], [37].

The key contributions of this paper are:

- We motivate the need for protecting in-network systems from adversaries making unauthorized modifications to the switch state in the data plane using C-DP and DP-DP messages that update/report the state (§II).
- We design a key-based authentication protocol and associated data plane primitives; together, they authenticate and preserve the integrity of C-DP and DP-DP messages (§V).
- We design a key management protocol and associated data plane primitives using which C-DP and DP-DP securely share secret keys and automatically update the keys at regular intervals or whenever there is a change in the network topology (§VI).
- We develop a prototype of P4Auth for two switch targets: BMV2 software switch [38] and Intel Tofino [39]. We evaluate P4Auth prototype's overhead and effectiveness in terms of protecting against attacks on two in-network systems (Hula [1], RouteScout [3]) performing fast control decisions (§IX).

II. Attacking In-network fast control decision systems

A. Threat model and attacker goals

Goals. As shown in Fig. 1, an attacker at a compromised switch OS influences traffic control decisions either (1) by altering a C-DP update message (*e.g.*, a message with traffic split ratio among next hops in RouteScout [3]) or (2) by altering a C-DP report message with traffic statistics from the switch data plane (*e.g.*, a message with suspicious flow stats as in Netwarden [5]), or (3) by altering a DP-DP message (*e.g.*, a message carrying network path status as in HULA [1]). The work in this paper aims to prevent unauthorized modifications to switch state using C-DP and DP-DP messages so that traffic control decisions are not influenced by adversaries.

Attacker capabilities. Though the threat model is strong, it is feasible in practice. An attacker can install a backdoor application (*e.g.*, LD_PRELOAD trick [40]) which preloads a malicious library that can update the messages or parameters of function calls between the gRPC server agent in the control plane and the SDK APIs or driver [16]. This way, the attacker can alter the parameters of function calls related to register operations (read/write). Similarly, for the DP-DP case, the malicious library installed at a neighbor switch configures table rules [41]–[43] such that the rules reroute network feedback messages to the attacker's host, which alters the content and puts the messages back into the network. Another possible approach is advertising fake network links [44] to a benign switch, redirecting traffic to the attacker's host.

The attacker can gain the ability to install backdoor applications [16], [18]–[21] either by exploiting the switch OS vulnerabilities (details in §II-B), by malware infection, or by social engineering a benign operator. For instance, the attacker may exploit stack buffer overflow vulnerability [45] and perform remote code execution [20]. In another case, a malware-infected network administrator system gains access to one of the network switches via keylogging. Then, the malware establishes a reverse connection to the adversary server, installs a backdoor binary, and becomes the pivot for the backdoor binary [17]. This allows the adversary's remote server to access an active shell on the compromised switch. In yet another case, with BYOD (Bring Your Own Device) organizational policy, a malicious insider can also exploit and install backdoor binary compromising the switch [46].

Let us understand the impact of altering C-DP and DP-DP messages on fast traffic control decisions.

Attack1 - Altering report message. Many in-network systems configure programmable data planes to monitor network traffic and maintain a state representing traffic characteristics in the data plane. A controller in these systems updates/reads

TABLE I: Impact of altering C-DP update/report messages

	System	**Update/report messages between C-DP**	**Impact of altering update/report messages**
FRR	Blink [2], RouteScout [3]	- Blink: C updates per-prefix next hop list maintained in registers. - RouteScout: C periodically reads per-path latency from registers.	Poisoning of fast rerouting decision
LB	SilkRoad [4]	C clears the transit table (bloom filter) holding old DIPs after all the pending connections are added to the connection table.	Manipulating the data plane to use the wrong VIP during LB
IDS/IPS	Netwarden [5], FlowLens [6]	Netwarden: DP reports inter-packet delays (IPD) of suspicious connections to C. C updates the connection state in the data plane.	Evasion of malicious traffic detection
In-network cache	NetCache [8]	- C periodically clears query statistics maintained in compact data structures (bloom filter, count-min sketch) in the DP. - C updates hot keys in the DP.	Inflates time to retrieve the hot key's value
Measurement	FlowRadar [9], LossRadar [10]	DP periodically exports encoded flowlet information stored in FlowRadar and traffic digest in LossRadar to C.	Manipulates monitoring decisions, poison loss analysis

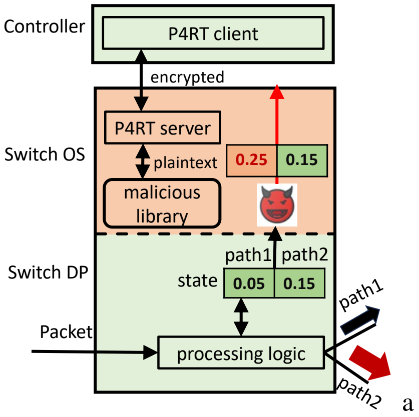


Fig. 2: **Altering path latency**

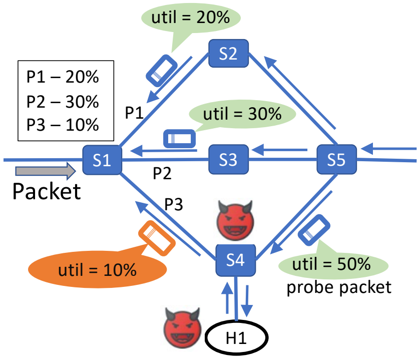


Fig. 3: **Altering path utilization**

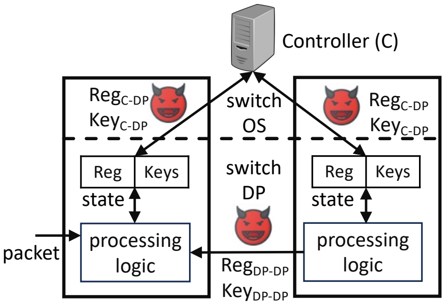


Fig. 4: **C-DP and DP-DP messages**

the state (in the registers) periodically for various network management tasks: traffic engineering [2], [3], load balance [4], intrusion detection [5], [6], in-network compute [8], and measurement [9], [10]. Table. I lists how the controller in various in-network systems updates/reads the state in the switch data plane and the impact of adversarial modifications' to update/read messages. For instance, RoutScout, an ISP scale performance-aware routing system, runs at the network edge and controls the outgoing traffic path. RouteScout's [3] controller (Fig. 2) periodically pulls aggregated latency information from the data plane, analyzes the information, and calculates the traffic split ratio to be followed by the data plane in the next epoch (*e.g.*, send more traffic to the best path). The attacker aiming to congest Path 2 may inflate latency on Path 1 such that the controller diverts more traffic to Path 2.

Attack2 - Altering feedback message. Switches in in-network load-balancing systems [1], [23] and in-network aggregation systems [47]–[49] process control messages from other switches (or hosts) entirely in the data plane. Altering the content in control messages can trick the packet-processing algorithm, leading to degradation of network performance (*e.g.*, inflates flow completion time (FCT) or job completion times (JCT)). For instance, consider a sample network topology implementing HULA [1] as shown in Fig. 3. An adversary on the S4-S1 link modifies path utilization information in probe messages traversing the link. Here, the S1 switch is informed that the path utilization to the destination via S4 is low (10%), though the actual utilization is relatively high (50%) compared to the utilization via S2 (20%) and S3 (30%). By doing so, S1 prefers S4 over S2 or S3 and forwards more traffic to S4, inflating flow completion times.

Summary. Adversarial modifications to C-DP and DP-DP messages can (1) mislead fast reroute systems by forwarding more traffic on a non-best path, (2) reduce the accuracy of telemetry systems, (3) fail to detect malicious flows in intrusion detection systems, (4) increase flow completion times (FCT) in in-network compute systems, or (5) network outage in the worst case.

B. Vulnerability assessment of existing NOS

To understand the feasibility of the threat models, using Trivy [50], we did a security scan of SONiC [51], a Linux-based open-source network operating system (NOS), 202111.3 2023 Mar. version installed on Edgecore Wedge 100BF 32X Tofino switch [52]. Interestingly, we found a few critical CVEs which an adversary with non-root access can exploit to achieve privilege escalation (root access) and execute arbitrary code, allowing the adversary to snoop, alter, or drop control messages. For instance, CVE-2017-14159 [53] exploits grant of insecure permissions, whereas CVE-2023-26604 [54] and CVE-2019-19882 [55] exploit system misconfigurations. CVE-2022-0563 [56], CVE-2018-7169 [57], and CVE-2016-

2781 [58] exploit other software bugs to achieve privilege escalation followed by backdoor installation on a compromised switch.

More specifically, we identified critical bugs with 15 SONiC docker containers. These containers provide services such as P4Runtime, route advertisements, network address translation, switch data store, telemetry, and link discovery. Across these 15 containers, we identified 382 CVEs, with 139 CVEs having severity levels as CRITICAL or HIGH. Software upgrades can fix 79 of these CVEs, but the remaining 60 CVEs cannot be fixed. Also, a total of 22 libraries were identified to have critical bugs. These libraries provide services such as system configuration (libc-bin), data transfer (libcurl4, curl), pattern matching (libpcre2-8-0), and a module for HTTP-based file upload that uses encoding and compression (urllib3).

III. Requirements and existing solutions

Given the threat model, in the section, we present the requirements for secure C-DP and DP-DP communication. Also, we explain the gaps in existing solutions to meet the requirements and strive to address the gaps.

A. Requirements

[R1] Authentication and integrity of C-DP messages. Consider $Reg_{C\text{-}DP}$ is the actual register read/write message or response message via untrusted switch OS from an authorized sender *S*, and the receiver observes $Reg'_{C\text{-}DP}$ from a sender *S'*. We define requirement R1 as guaranteeing:

$$S = S' \quad and \quad Reg_{\text{C-DP}} = Reg'_{\text{C-DP}} \tag{1}$$

Otherwise, detect and prevent processing messages that do not hold (1) and raise an alert.

[R2] Authentication and integrity of DP-DP network feedback messages. Consider $Reg_{DP\text{-}DP}$ is the actual feedback message received by a switch data plane via an untrusted network from an authorized sender *S*, and the receiver observes $Reg'_{DP\text{-}DP}$ message from a sender *S'*. We define requirement R2 as guaranteeing:

$$S = S' \quad and \quad Reg_{\text{DP-DP}} = Reg'_{\text{DP-DP}} \tag{2}$$

Otherwise, detect and prevent processing messages that do not hold (2) and raise an alert.

[R3] Secure key management protocol over untrusted network. Most common key-based solutions to secure communication, such as authentication and message integrity, require a key management protocol to establish secret keys and update keys periodically. However, as discussed earlier, a MitM adversary at a switch control plane or on a network link can alter the key exchange messages in the protocol, compromising the confidentiality of the shared secret. More specifically, consider $Key_{C\text{-}DP}$ is the original key exchange message sent by an authorized sender $S_{C\text{-}DP}$ and the receiver observes $Key'_{C\text{-}DP}$ from a sender $S'_{C\text{-}DP}$. Similarly, it is ($S_{DP\text{-}DP}$, $Key_{DP\text{-}DP}$) and ($S'_{DP\text{-}DP}$, $Key'_{DP\text{-}DP}$) between two switch data planes. We define requirement R3 guaranteeing:

$$\begin{aligned} S_{\text{C-DP}} = S'_{\text{C-DP}} \quad and \quad Key_{\text{C-DP}} = Key'_{\text{C-DP}} \\ S_{\text{DP-DP}} = S'_{\text{DP-DP}} \quad and \quad Key_{\text{DP-DP}} = Key'_{\text{DP-DP}} \end{aligned} \tag{3}$$

If (3) does not hold, we detect and raise an alert.

[R4] Process DP-DP control messages at line rate. In-network feedback messages should be processed fast (at line rate); otherwise, the processing delays may lead to inaccurate reroute decisions, especially in load balancing systems that adapt fast to congestion on network paths. Therefore, it is desired to have security primitives with minimal impact on packet processing delays while doing necessary checks.

B. Existing solutions and limitations

In this section, we discuss the existing approaches in the SDN security literature and their limitations.

[A1] Detection approaches using rule enforcement verification techniques are insufficient. We can consider table rule enforcement verification techniques (*e.g.*, REV [19], SDNSec [59]), which check whether table rules are correctly enforced in the switch data plane. They do so by comparing the expected network path with a packet's actual path at runtime. Similar to our threat model, these works consider an attacker at the switch control plane with privileges to manipulate table rules and divert traffic. However, unlike these works, we consider attacker privileges to manipulate the state maintained in registers, which do not always alter the packet's path but influence the control decisions (*e.g.*, traffic split ratio, block malicious flow). Moreover, these works try to detect violations at runtime but do not prevent them before they happen.

[A2] Authentication of DP-DP messages requires novel security primitives. BGP security works (such as bgpsec [29], s-bgp [30], psbgp [60]) are designed to process BGP advertisements (*i.e.*, control message) by upper layers such as switch control plane or a central controller, where sophisticated cryptographic primitives required for certificate verification and signature checks are feasible to implement. In contrast, DP-DP network feedback messages are processed in a data plane without support for loops, restrictions on allowed per-packet operations, and memory accesses. This motivates the need for key-based solutions to authenticate DP-DP messages under these constraints while meeting the requirements.

[A3] Key management protocols are insecure. Recent work (DH-AES-P4 [25], P4MACsec [26], Spine [27], secINT [28]) propose key-based protection solutions to authenticate and verify message integrity in the data plane (R4) and prevent unauthorized modifications to messages (R1 and R2). However, these solutions do not provide a secure key management protocol over untrusted networks (R3). More specifically, with a MitM attacker at a switch control plane or on a network link (see Fig. 4), the key exchange messages in the protocol can be altered or exposed to an attacker. For instance, DH-AES-P4 [25] proposed a customized Diffie-Hellman (DH) key

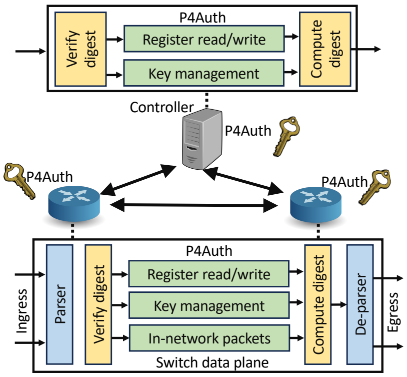
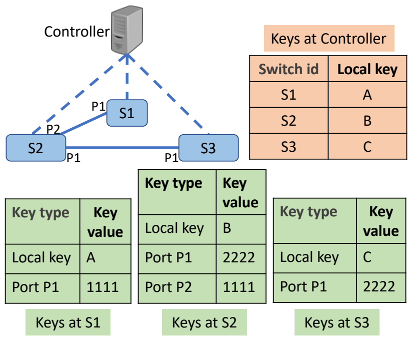


Fig. 5: **P4Auth architecture**



Keys at Controller

Switch id	Local key
S1	A
S2	B
S3	C

Key type	Key value
Local key	A
Port P1	1111

Keys at S1

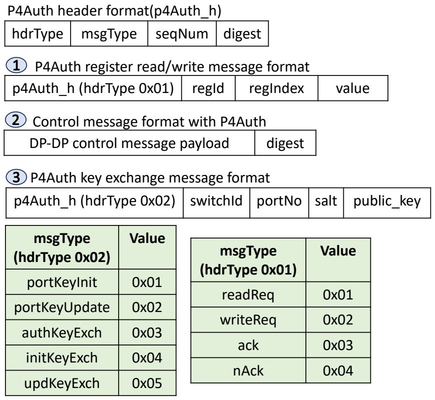
Key type	Key value
Local key	B
Port P1	2222
Port P2	1111

Keys at S2

Key type	Key value
Local key	C
Port P1	2222

Keys at S3

Fig. 6: **Keys maintained at the controller and switches**



P4Auth header format(p4Auth_h)

hdrType	msgType	seqNum	digest

1 P4Auth register read/write message format

p4Auth_h (hdrType 0x01)	regId	regIndex	value

2 Control message format with P4Auth

DP-DP control message payload	digest

3 P4Auth key exchange message format

p4Auth_h (hdrType 0x02)	switchId	portNo	salt	public_key

msgType (hdrType 0x02)	Value
portKeyInit	0x01
portKeyUpdate	0x02
authKeyExch	0x03
initKeyExch	0x04
updKeyExch	0x05

msgType (hdrType 0x01)	Value
readReq	0x01
writeReq	0x02
ack	0x03
nAck	0x04

Fig. 7: **P4Auth protocol message formats**

exchange algorithm between two data planes and the algorithm is designed to run entirely in the data plane, but it does not consider MitM attacker with a privilege to compromise the key exchange messages.

IV. P4Auth Design Overview

We design P4Auth with two key components to meet the requirements. Fig. 5 shows P4Auth architecture.

Authentication protocol. To address [R1, R2, and R4], we design an authentication protocol and associated data plane primitives; together, they authenticate and preserve the integrity of messages between C-DP and DP-DP by doing necessary checks completely in the data plane. We leverage existing security primitives (such as message digest) and design our authentication protocol operating in the data plane. Our key idea is that a sender tags each message with a digest computed on the message's data using a shared secret key. The receiver verifies the digest and raises an alert if the verification fails, indicating the possibility of a MitM adversary (§V).

Key management protocol. An adversary with the privilege to access secret keys (R3) can compromise digest verification, making our detection as good as the confidentiality of the shared keys. To protect P4Auth from such adversaries, we design an in-network and secure key management protocol and associated data plane primitives; together, they securely share and update secret keys automatically either at regular intervals or whenever there is a change in the network topology (more details in §VI).

Design choice. One design choice is to run the P4Auth's digest verification/computation module and the associated secret keys in the switch control plane. However, this will not protect the module from the same adversaries trying to manipulate the state in the data plane; thus, it does not satisfy R1, R2, and R3. As an alternative, we propose to design this module to run entirely in the data plane. This will reduce the trusted computing base (facilitating R1, R2, and R3) and enable faster checks (R4). Specifically, this ensures the module is not exposed to vulnerabilities at switch OS, SDKs, or other low-level switch firmware from a third party. Also, checking DP-DP messages in the data plane is fast and avoids C-DP communication delays.

V. Authentication Protocol

P4Auth's authentication protocol facilitates the authentication and integrity of C-DP and DP-DP messages that update/report the state maintained in the switch data plane.

One option to realize a key-based digest verification module in the data plane is digital signatures, but they pose implementation challenges on programmable data planes. Digital signature algorithms (*e.g.*, Rivest-Shamir-Adleman (RSA) [61], Elliptic Curve Digital Signature Algorithm (ECDSA) [62]) use modulo, exponentiation, hash, and multiplication operators for signature generation, which may not be feasible to implement on a multi-stage PISA-based switch pipeline architecture due to lack of security primitives and restrictions on per-packet operations. Inspired by prior works [19], [59], [63], we design the digest verification module using a standard hash-based message authentication code (HMAC). HMAC algorithm is amenable to implementation on high-speed programmable switches [28], [37] using simple arithmetic operators (AND, XOR, and $rotate$).

Each switch data plane has two types of key: (1) local key, a secret key shared with the controller, and (2) port key, a secret key for each port connected to a neighbor switch. These keys are securely shared using our key management protocol (§VI). Fig. 6 illustrates per-switch local key (K_{local}), used for authenticating messages between controller and switch data plane, and port keys (K_{port}), used for authenticating messages between two data planes.

Protocol messages. Fig. 7 shows the header format of messages in our authentication protocol. *hdrType* field specifies one among register read/write request message, an alert message, or a key exchange message. The definition of *msgType* depends on *hdrType* (more details in later subsections). *seqNum* maps a response to its corresponding request. *digest* carries the digest computed over two header groups: (1) *p4Auth_h* fields, excluding *digest* field, and (2) the message's *p4Auth_payload* as shown in Eqn. 4. Key K is either K_{local} or K_{port}.

$$digest = HMAC_{\mathrm{K}}(p4Auth\_h \mid\mid p4Auth\_payload) \quad (4)$$

Authentication of C-DP messages. For register read/write request-response messages, we define four messages: *readReq,*

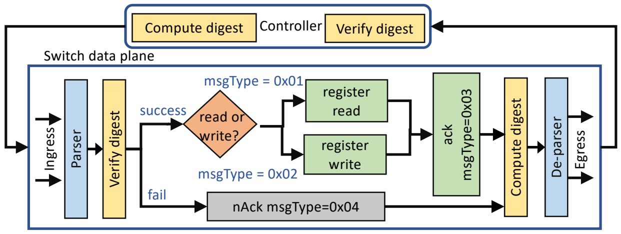


Fig. 8: **Register read/write request workflow**

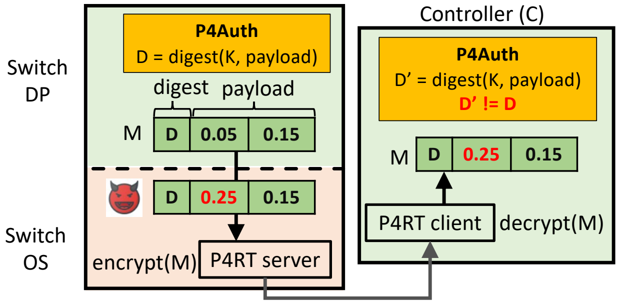


Fig. 9: **Detection of unauthorized manipulation using P4Auth.**

writeReq, ack, nAck (Fig. 7). For instance, consider a read register request (*readReq*) from a controller to the switch data plane. The detection module in the data plane computes an HMAC-based digest using its local key. If it matches the digest in the incoming message, an *ack* response message with the register value and associated digest is sent back to the controller. Otherwise, the data plane responds with a *nAck* message. The workflow for *writeReq* (Fig. 8) is similar to *readReq* except that the request message carries the value to be written to a register index.

Authentication of DP-DP control messages. In-network control messages are processed entirely in the data plane and then advertise the original or updated message to neighbor switches. In general, the switch state in registers is updated while processing the control messages [1], [23], [64]. P4Auth's detection module in the data plane authenticates such control messages before updating the state. More specifically, the sender of a control message computes a digest using the sender's egress port key. The digest is verified at the receiver using the receiver's ingress port key. P4Auth's key management protocol ensures that both ports use the same key.

Detection of misreported statistics. Consider that C and switch DP agree on a secret key (K) and the DP wants to share statistics with C. Fig. 9 shows the detection of unauthorized modification to latency statistics (Fig. 2) using P4Auth.

VI. Securing P4Auth

An adversary at the switch control plane with the capabilities mentioned in §II-A can also view the key exchange messages between the controller and the switch data plane and compromise the communication. The existing works [25], [26], [28] do not consider the MitM adversary at the control plane while sharing keys, thus insecure. To secure P4Auth from such adversaries, we propose a secure mechanism to (a) share the local/port keys, (b) periodically refresh the shared keys, and (c) automatically update the keys whenever the network topology changes.

Strawman approaches. Before discussing our solution, we present two strawman approaches and show why they do not work. The first naive approach uses static keys (local/port keys) programmed at compile-time as part of the switch binary. However, as network topology changes dynamically (*e.g.*, events such as switch boot up, port active, and port inactive), the local/port keys require reconfiguration. Therefore, we need to change the keys in the P4 binary as per the new topology, recompile it, stop the switch(es), reload the P4 binary, and start the switch. Such manual interventions are error-prone and could result in frequent network downtime. Another approach is using key exchange protocols such as Diffie-Hellman (DH) [65] to exchange keys between C-DP and DP-DP securely. However, the data plane restrictions do not allow us to implement complex operations such as exponentiation and modulus required to implement state-of-the-art cryptographic primitives. A modified DH algorithm [25], [34] replaces exponentiation with *AND* and *XOR* operations without compromising security guarantees, making it amenable to be implemented on a high-speed programmable switch. However, the modified DH proposal is based on the assumption of a secure communication channel between C-DP and DP-DP. A MitM adversary at the switch OS and between two data planes (as discussed in §II) makes it essential to authenticate every message between C-DP and DP-DP.

Our approach. Inspired by TLS1.2 [33] security protocol, we propose an authenticated key exchange protocol using the modified DH algorithm [25], [34] to derive a pre-master secret (*i.e.*, K_{pms}) followed by master secret (*i.e.*, local/port key) derivation using a custom key derivation function. Note that the novelty of this paper is not in the DH algorithm but in carefully integrating it into the proposed key exchange protocol. We design P4Auth to meet three functional requirements for enabling secure key sharing and management. **[F1]** Generate a shared secret between C-DP for protecting C-DP communication during the generation of a master secret key; **[F2]** After establishing the shared secret, C-DP and DP-DP should securely generate the master secret key (local or port key); and **[F3]** Automate key initialization and key exchange to enable periodic key rollover and to handle network topology changes.

In the rest of the section, we present design components that help us achieve the functional requirements: (1) Exchange of Authentication Key (EAK) satisfies F1 (see §VI-A), (2) Authenticated DH exchange and key derivation (ADHKD) satisfies F2 (see §VI-B), (3) Key management protocol (KMP) satisfies F3 (see §VI-C), and (4) Key Derivation Function (KDF) satisfies both F1 and F2 (see §VI-D).

A. Exchange of Authentication Key (EAK)

EAK protocol exchange, as shown in Fig. 11, derives an initial authentication key K_{auth} which is used for protecting C-

$msgType_K(v)$	A message with value v, authenticated and digest computed using key K
K_{seed}	Pre-shared secret shared between the data plane and controller during switch bootup
KDF (K,salt)	Key derivation function with key K and salt as parameters
P	Prime number
G	Generator
DH'	Modified Diffie-Hellman algorithm that takes P, G, and the random secret (R) as input and generates a public key (PK) to be shared as part of the exchange. PK = DH' (P, G, R) = G·R ⊕ P·R
DH"	Modified Diffie-Hellman algorithm that takes P, G, and the received public key (PK) as input and generates a shared pre-master secret using the random secret R and PK. K = DH"(P, R, PK) = PK·R⊕ P

Fig. 10: Notations

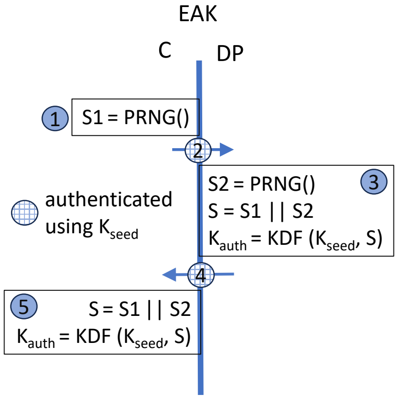


Fig. 11: EAK

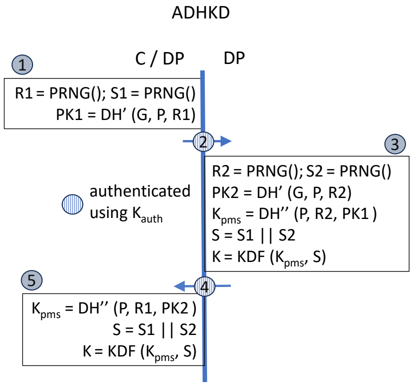


Fig. 12: ADHKD

DP communication during the master secret generation phase. This phase is triggered whenever the switch boots up. All messages in this step are authenticated using K_{seed}[1].

1) C generates a random salt, *S1*.
2) C transmits *S1* to DP.
3) DP receives *S1*, generates a random salt *S2* and *S* (by concatenating *S1* and *S2*). The authentication key, K_{auth}, is generated using a custom KDF as shown in Fig. 13, with the input parameters, K_{seed} and *S*.
4) DP transmits *S2* to C.
5) C receives *S2* and generates *S*. The authentication key K_{auth} is generated from K_{seed} and *S* using the KDF.

B. Authenticated DH exchange and key derivation (ADHKD)

ADHKD protocol exchange generates a master secret, *K* (K_{local} or K_{port}), which is used to authenticate messages between C-DP and DP-DP (see Fig. 12). Note that all key exchange messages in this phase are authenticated, and the details of keys used for authentication are explained in §VI-C.

1) C/DP generates a random salt (*S1*) and a random private key (*R1*). A modified DH algorithm generates the public key (*PK1*).
2) C/DP transmits *PK1* and *S1* to DP.
3) DP receives *PK1* and *S1*, and generates its random private key (*R2*) and a random salt (*S2*). Next, using the modified DH, the DP generates its public key (*PK2*) followed by the pre-master key (K_{pms}) from the received public key (*PK1*) and DP's private key (*R2*). The final master secret (*K*) is derived using a custom KDF with K_{pms} and *S* (concatenation of *S1, S2*) as inputs.
4) DP sends *PK2* and *S2* to C/DP.
5) C/DP generates the pre-master key, K_{pms}, using its private key (*R1*) and DP's public key (*PK2*). The final master secret, *K*, is derived from K_{pms} and *S* using a custom KDF.

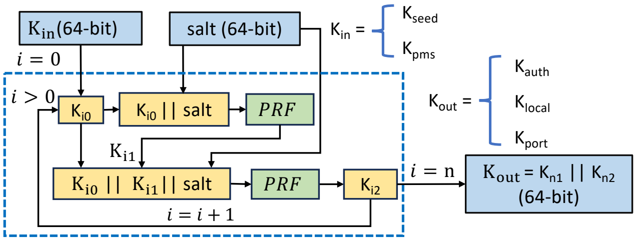
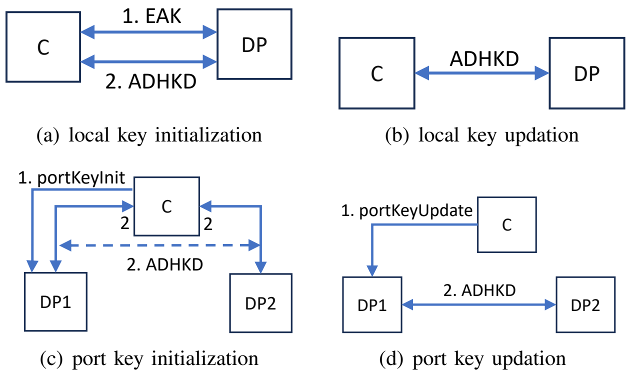


Fig. 13: **Key derivation function**



(a) local key initialization

(b) local key updation

(c) port key initialization

(d) port key updation

Fig. 14: **Local and port key initialization and updation**

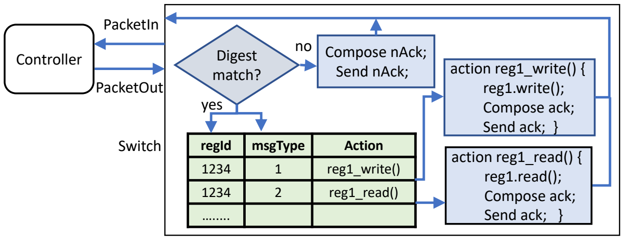
C. Key management protocol (KMP)

We propose KMP to automate key initialization[2] and periodic key rollover. The steps in EAK and ADHKD (Fig. 14) are realized using five types of messages (Fig. 7).

Key initialization. The key initialization process is triggered whenever a switch reboots or a port activation event is observed by the controller (*e.g.*, via LLDP message [26]). More specifically, local key initialization invokes EAK exchange to derive a common authentication key, K_{auth}, followed by ADHKD exchange that derives the local key, K_{local}, (*i.e.*, master secret). The workflow is shown in Fig. 14(a). Meanwhile, port key initialization comprises only ADHKD exchange and the exchange messages are authenticated using

[1] The initial secret (K_{seed}) is shared with the data plane through the P4 program binary loaded while booting the switch

[2] during events like switch boot up, port active/inactive



regId	msgType	Action
1234	1	reg1_write()
1234	2	reg1_read()
....		

```
action reg1_write() {
    reg1.write();
    Compose ack;
    Send ack; }
```

```
action reg1_read() {
    reg1.read();
    Compose ack;
    Send ack; }
```

Fig. 15: **Reg. read/write work flow**

K_{local}. As shown in Fig. 14(c), the controller initiates the port key initialization using *portKeyInit* message. P4Auth redirects ADHKD messages between two data planes via the controller using *initKeyExch* message. Then, each data plane derives a port-specific key K_{port} for the activated port.

Updating keys. To ensure the security of the secret keys (local key and port keys), the key must be periodically updated, securely. For the local key update, as shown in Fig. 14(b)), the controller initiates the exchange, and the messages are authenticated using the current local key ($updKeyExch_{K_{local}}$). At the end of the ADHDK exchange, the new local key (K'_{local}) is derived between the controller and data plane.

To update a port key (see Fig. 14(d)), the controller sends $portKeyUpdate_{K_{local}}$ message. But, unlike port key initialization, ADHKD exchange messages for updating port keys are directly managed by the data planes without involving the controller, as the data planes already share their port key. That is, ADHKD exchanges are authenticated using the current port key shared between DP1-DP2 (K_{port}). After ADHKD exchange, a new port key, K'_{port}, is shared between DP1-DP2.

Consistent key updates. The key update mechanism must ensure that messages are authenticated using the same key. To implement consistent key updates, the ideas from an existing work [66] can be borrowed. The control and data planes maintain two versions (old and new) of the local and port keys. The sender tags each control message with the key version used for message authentication, and the receiver uses the tagged key version to validate the message.

D. Key Derivation Function (KDF)

To generate local and port keys, P4Auth uses a custom KDF kept private to the controller and the switch data planes (as part of the P4 binary). The KDF implementation is based on TLS1.3's *Extract-and-Expand* principle [33], [35] that is proven to be secure [36] and it produces "close-to-random" keys. As shown in Fig. 13, the KDF takes two inputs, a 64-bit secret (*i.e.*, K_{in}) and a 64-bit public material (*i.e.*, salt), and generates keys by ensuring randomness using a pseudorandom function (PRF). The PRF can be a hash function such as MD5, SHA, etc. Since our PRF generates a 32-bit output, the KDF executes the PRF executed twice to produce the final 64-bit secret (K_{auth} or K_{local} or K_{port}).

VII. Implementation

P4Auth prototype is developed on two targets, BMv2 [38] and Intel Tofino [39]. We use PacketOut messages for C-DP register read/write requests (*readReq*, *writeReq*) and PacketIn messages for DP-C response or alerts (*ack*, *nAck*, and *alert*). PacketOut messages are processed in the P4 data plane. P4Auth data plane has 400 lines of code written in P4 and the controller is implemented in Python3.

Register read/write request authentication. For register read/write requests, the controller identifies a register using the associated identifier (from the p4Info file), and the data plane identifies the same register by its name. While processing a request in the data plane, we find which register to read/write using a table that maps register identifiers to the associated register name [3]. We also instrument P4 code to perform three tasks: (1) an action routine for reading from or writing to a register; (2) converts *readReq/writeReq* message to ack message (as discussed in §V); and (3) computes digest and sends response in a PacketIn message. For example, as shown in Fig. 15, on receiving a PacketOut message with *regId* as 1234 and *msgType* as 1, the data plane performs look-up on *reg_id_to_name_mapping* table and apply associated action *reg1_read*.

Key management protocol (KMP). We implement KMP using modified DH exchange [25]. P4Auth data plane uses P4's *random()* to generate the random secrets and salt at the controller and data plane. We define a register with $N + 1$ entries to store the local key and N port keys, where N is the number of ports. The local key is stored at index zero, and port keys at port number as the index. We implement our KDF with CRC32 as PRF and set the rounds to one.

Digest computation. Among the most common hash algorithms, such as MD5, SHA family, and Sip-Hash, we consider that MD5 is vulnerable to collisions [67]–[70], and the performance of Sip-Hash family of functions [71] for shorter inputs is better than SHA family. But Sip-Hash has implementation challenges [72]. A recent work [37] studies the implementation feasibility and performance of HalfSipHash on Intel Tofino [39] target is promising. So, we pick the HalfSiphash function as the HMAC algorithm. HalfSipHash starts with four 32-bit state variables. These 32-bit words are initialized by XORing with the lower and upper 32-bits of the 64-bit key and four other 32-bit constants. Then, c compression and d finalization rounds are performed on SipHash-c-d. On the BMV2 target, we implement HalfSipHash as an extern function named as *compute_digest*. *compute_digest* takes two input parameters: a 64-bit secret key and a variable list of arguments over which digest needs to be computed. On the Tofino target, we use CRC32 as the hash algorithm.

VIII. Security Analysis

P4Auth system assumes that the secret key shared between data planes and the controller is secure and no adversary can access them. With this assumption, we discuss the security boundaries and guarantees of P4Auth and other possible attacks adversaries can launch against P4Auth system.

[3] We use *reg_id_to_name_mapping* table (Fig. 15) with keys as register identifier and register operation (read/write). Each register has two entries, one each for read and write with action as *regName_read* and *regName_write*, respectively.

Security of key mgmt. protocol. We highlight two main design choices for securing the proposed key management protocol. First, the protocol is built on top of the modified DH key exchange [34], which is proven to be secure and preserves the confidentiality of the pre-master keys. Second, since the authentication key (K_{auth}) is derived from K_{seed} which is part of the switch binary, a MitM adversary with read privilege (§II-A) may obtain K_{seed} and possibly compromise DH exchange via reverse engineering [73]–[75]. We handle this by deriving K_{auth} using a KDF with custom logic in the binary, and the logic is kept secret between C and DP. Reverse engineering the logic can be further made hard by using binary obfuscation techniques [75]–[78].

Secret key size. An adversary at the switch control plane can see the message payload and associated digest and perform brute force on all 2^{64} possible key values to find the secret key. A recent work [79] has shown that a 56-bit key can be broken in only 215 days using commercially available computational hardware. Therefore, a brute-force attack on a 64-bit key is feasible. P4Auth periodically changes keys (local and port) to mitigate such attacks. Setting the periodicity of key updates to 180 days or lesser can prevent such brute force attacks.

Digest size. The output digest size is 32 bits. An attacker intending to send a crafted message can try at most 2^{32} different possibilities for the actual digest corresponding to a crafted message. However, during these adversarial trials of guessing the digest, an alert is raised, indicating a malformed digest encountered at the switch data plane or the control plane, revealing the possibility of the adversary. Thus, P4Auth is safe from such brute force attacks for guessing the digest.

Denial-of-service (DoS) attack. A denial-of-service attack can be launched on the controller by manipulating many messages to the controller or the data plane. There are two possibilities here. First, if an adversary modifies many request messages (*readReq, writeReq*) sent to the data plane, the data plane finds digests do not match and sends a stream of alert messages (one for every message) to the controller. This can jam the controller to the data plane communication link and overwhelm the controller, causing a DoS attack. Second, many modified response messages (*ACK, nACK, alert*) sent to the controller can overwhelm the controller and lead to a DoS attack. In both cases, upon detecting unauthorized modification, ideally, the network operator should take appropriate action (*e.g.*, isolate suspicious switch). If no action is taken, it can lead to a DoS attack on the controller. To mitigate the first attack, one can set a threshold on the number of alert messages sent to the controller in a specific period from the data plane. To mitigate the second type of attack, the controller should keep a threshold on the difference in the number of requests sent and responses received in a specific period. Moreover, a list of not yet acknowledged sequence numbers can be used to further defend against such DoS attacks.

Replay attack. An adversary may record *writeReq* messages and replay them, leading to an unauthorized modification to the data plane packet processing behavior. P4Auth is robust towards such replay attacks; the sequence number field in the P4Auth header is used to defend against replay attacks. More specifically, the sequence number in the replayed message and the one expected by the controller do not match. The switch data plane can detect this mismatch and send an alert message to inform the controller. A corner possibility for the attacker to succeed is if the sequence number wraps around to the same value as in the recorded message. This can be further mitigated by allocating more bits (16-bit or 32-bit) to this field and changing the local and port keys within the wrap-around time so the replayed message's digest becomes invalid.

IX. Evaluation

We evaluate P4Auth for two primary aspects: (1) P4Auth's functional evaluation and (2) P4Auth's performance evaluation in terms of throughput, latency, scalability, and resource overheads. Our evaluation aims to answer five important questions: (1) How well can P4Auth defend against attacks on in-network systems making fast traffic control decisions? (2) How does P4Auth affect C-DP message completion times and throughput? (3) What are the resource overheads of P4Auth's data plane implementation? and (4) How long does it take to initialize and update keys? (5) How does P4Auth affect packet processing time as the system scales across multiple switches?

A. Preventing attacks on fast reroute systems

We demonstrate P4Auth's effectiveness in preventing attacks on two data plane systems, RouteScout [3] and Hula [1].

Experimental setup. We assume that the attacker has privileges as per the threat model discussed in section II. Since RouteScout's source code is unavailable, we implement it as a software simulation. We used Hula's open-source bmv2 switch implementation [80] to test P4Auth's effectiveness.

Protecting RouteScout system from an adversary at the switch control plane. We consider an adversary modifying register read response messages from the data plane such that the traffic splitting decision by RouteScout's controller is influenced (as shown in Fig. 2). For instance, between two paths, path1 and path2, the controller decides to send more traffic on path2, though the average delay on path2 is high, which degrades the performance. Using P4Auth, the controller detects unauthorized modification to the response message, refrains from changing the current traffic splitting ratio, and raises an alert to the operator. We experiment on the RouteScout system (1) without an adversary, (2) with an adversary, and (3) with an adversary and P4Auth. We send packets to the RouteScout system by replaying CAIDA PCAP traces [81] for 60 seconds. Fig. 16 shows the distribution of traffic across path1 and path2. Without an adversary, RouteScout splits based on the aggregate delay computed on each path. With the adversary, around 70% of the traffic is rerouted to path 2. With P4Auth, RouteScout detects unauthorized modifications and retains the original splitting ratio. This demonstrates P4Auth defence against adversarial modifications.

Protecting HULA system from a MitM adversary in the network. We consider a MitM adversary eavesdrops and

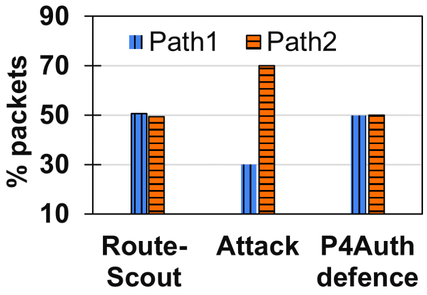


Fig. 16: **P4Auth prevents imbalance**

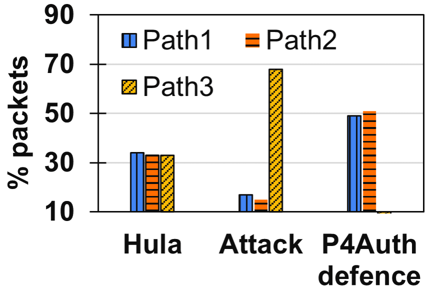


Fig. 17: **Preventing congestion on Path3**

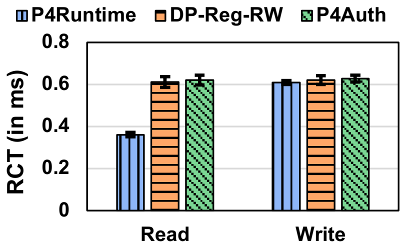


Fig. 18: **Reg. read/write RCT**

modifies the probe packets sent over the S1-S4 link (as shown in Fig. 3), forcing the HULA system to reroute all the traffic to the S1-S4 link. If the adversary modifies the *probeUtil* value, S1 detects this modification, ignores the message, and raises an alert to the controller. We evaluate P4Auth defence by sending probe packets from S5 towards S1 and data packets from S1 towards S5. We experimented on the HULA system (1) without a MitM adversary, (2) with the MitM adversary (as mentioned in II-A), and (3) with P4Auth and the MiM adversary. Fig. 17 shows the distribution of traffic across three paths, S1-S2, S1-S3, and S1-S4. The HULA system distributes traffic equally among the three available paths when there is no adversary. With the adversary, the HULA system reroutes more than 70% of the traffic through the compromised link (S1-S4). With P4Auth, S1 ignores the probe packets upon detecting unauthorized modification and blocks traffic on the compromised link.

B. Performance evaluation

Experimental setup. We deployed P4Auth on Aurora 610 Intel Tofino [82] switch with ONL1.4 NOS. We compiled P4Auth using Barefoot's software development environment, bf-sde 9.9.0. As a base for our evaluation, we add P4Auth's data plane modules (DH, KDF, HMAC, register read/write) to a P4 program that performs destination-based layer-3 port forwarding with two match-action tables and one register. We use the packet test framework (PTF) provided by SDE to generate messages and receive responses at the controller.

Parameters and metrics. We evaluate the impact of P4Auth on register read-write performance using two metrics: (a) throughput in terms of requests completed per sec and (b) end-to-end latency in terms of request completion time. During the experiment, the control messages were crafted and sent sequentially. All experiments ran for 30 seconds, 10 times, and the reported metrics averaged over these different runs. We compare these metrics for three P4 versions of basic L3-layer port forwarding. Their implementation varies in how they perform register read and write: (1) *P4Runtime* performs register read and write using P4Runtime API stack, (2) *DP-Reg-RW* performs register read and write using PTF's python library, and (3) *P4Auth* extends *DP-Reg-RW* and include security primitives proposed by P4Auth. The third version helps to understand the actual overheads introduced by P4Auth.

	TCAM	SRAM	Hash Units	PHV
Baseline	8.3%	2.5%	1.4%	11%
With P4Auth	8.3%	3.6%	51.4 %	23.1%

TABLE II: **Hardware resource overhead**

Impact on read-write performance. Fig. 18 and Fig. 19 show register read/write request completion time (RCT) and throughput, respectively. An interesting observation is that *P4Runtime*'s register read throughput is 1.7 times better than write throughput. This is because the read includes composing the index initialization, whereas the write includes composing both the register index and the data. There is not much difference in register write throughput among *P4Runtime, DP-REG-RW* and *P4Auth*. We observe that *P4Auth* has minimal impact on register read/write throughput and RCT. Compared to *DP-REG-RW*, *P4Auth's* read throughput is decreased by 4.2% and write throughput by 2.1%. This drop is due to digest computation and verification in the data plane.

Resource overhead. Table. II shows the switch resource utilization for the P4 programs, *Baseline* and *With P4Auth*. P4Auth adds code for (1) authentication protocol – requires extra *PHV* resources, (2) digest computation and verification – requires extra *Hash Units*, (3) key management – requires both *PHV, Hash Units*, (4) key register (64-bit) to store local key and port keys – consumes *SRAM* resources, and (5) register mapping table to map register id to the associated register name (see §VII) – consumes *SRAM* resources. Digest computation, verification, and KDF increase hash unit consumption and PHV utilization. However, this increase is constant, *i.e.*, the usage does not vary based on the P4 program or network topology. The SRAM consumption depends on the key register and register mapping table size. The size of the key register is proportional to the number of switch ports (*i.e.*, neighbor switches, see §V), M, which caps the SRAM usage for key registers to $64*(M+1)$ bits (additional 1 is for the local key). The number of entries in the register mapping table depends on the number of registers in the P4 program. For instance, if a P4 program has K registers, the register mapping table will have $2*K$ entries, each consuming 40 bits (32-bit regId and 8-bit msgType). It scales well, as the increase in SRAM consumption is proportional to the number of registers and neighbor switches, which is linear in practice.

Key management protocol (KMP) RTT. Key management

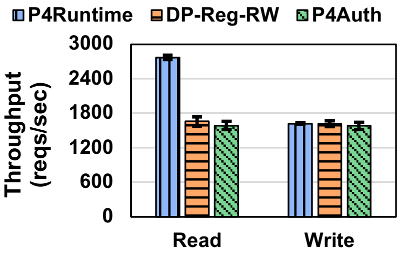


Fig. 19: **Read/write throughput**

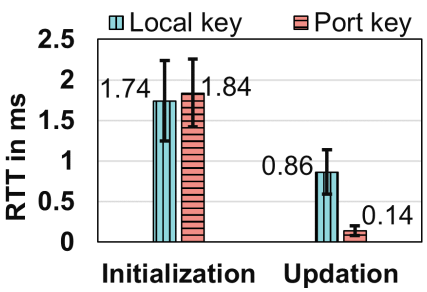


Fig. 20: **KMP RTT**

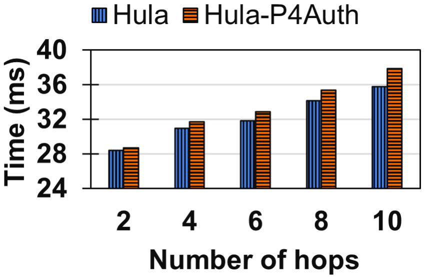


Fig. 21: **In-network control message processing time**

round-trip time (RTT) is the time elapsed from the first message exchange of key initialization/updation until the key derivation. For local key initialization, we execute EAK followed by ADHKD sequentially for 30 seconds and collect RTT samples, and for local key updates, we execute only ADHKD and collect RTT samples. For port key initialization and update, we require at least two Tofino switches, which we do not have, so we limit ourselves to an approximated RTT computed by adding ADHKD computation time in the data plane with the propagation time between C-DP or DP-DP.

Fig. 20 shows the average key management RTT that includes 4 and 5 message exchanges for local key and port key initialization, respectively. It is 2 and 3 message exchanges for updating keys. P4Auth takes 1-2 milliseconds for key initialization and less than a millisecond for key updation. Since few message exchanges are part of key management, P4Auth does not impact the ongoing traffic nor overwhelm the C-DP and DP-DP bandwidth. Though port key initialization comprises only ADHKD, it takes the longest as the key exchange messages between DP-DP are redirected via the controller (see §VI-C), which checks digest in both directions as part of redirection. Key updation takes less time than initialization as fewer messages are exchanged. It is worth noting that even though port key updation exchanges three messages, it takes less time than local key updation as the exchange between DP-DP is faster than the C-DP exchange.

C. Scalability across multiple switches

We deployed P4Auth with Hula on the BMV2 switch. As Hula probe message carrying link utilization traverses multiple switches, P4Auth implemented on each on-path switch performs digest verification. We study the impact of P4Auth on probe message traversal time (time the probe packet spends in the network traversing multiple hops) by varying the number of switches in the network. We evaluate P4Auth on BMV2 target by running Hula with and without P4Auth. Fig. 21 shows the impact of P4Auth on Hula probe packet traversal time as the number of hops increases. We observe that P4Auth's overhead increases linearly as the number of hops increases. With 2 hops, P4Auth increases the traversal time by 0.95%, whereas with 10 hops, it becomes 5.9%. This increase is minimal and does not pose any significant overhead. On a single hardware switch, the data packet processing time is only 6% more for P4Auth compared to the base case.

X. Related work

Vulnerabilities of P4 data planes. Prior work [11], [15], [16], [21], [84] highlights vulnerabilities of P4 programmability and statefulness to various security attacks. Agape et al. [21] identifies potential attacks and vulnerabilities related to the P4 language, compiler, controller, P4 Runtime, and switch firmware. Also, a small fraction of adversarial input traffic can degrade the performance of data-driven data plane systems [15], [16]. [84] demonstrates attacks and undefined system behavior due to buggy P4 programs. [11] highlights that resource-constrained programming models of hardware switches lead to vulnerable system designs. P4Auth complements these studies towards securing in-network control systems from adversarial manipulation of the data plane state.

P4 verification. P4 verification works leverage static analysis techniques [49], [85]–[93] to find properties under violation for a given P4 program and switch state (*i.e.*, table rules, registers). This approach is computationally expensive to predict all possible states at runtime and distinguish between legitimate and attacker-influenced control decisions. Also, modifications to table rules and registers do not always alter the packet's path. P4Auth complements these works by protecting adversarial manipulation to the switch state at runtime.

Key-based protection solutions. Prior research (e.g., DH-AES-P4 [25], P4MACsec [26], Spine [27], secINT [28]) focuses on securing DP-DP in-network messages by providing authentication, confidentiality, and message integrity using key-based protection solutions. However, these works do not address the need for a secure key management protocol to facilitate key agreement over untrusted networks (C-DP and DP-DP). Our work acknowledges the presence of a MitM adversary at C-DP and DP-DP and implements an authenticated key exchange mechanism over the untrusted network.

Detection approaches using packet path validation. REV [19] and SDNSec [59] detect compromised switches by comparing the expected network path with a packet's actual path at runtime. They employ cryptographic techniques to validate packet data paths, send probes to test forwarding behavior [94], [95], or monitor statistics collected across multiple switches [96], [97] to find misbehaving switches. An attacker can manipulate register state, without altering the packet's path or statistics collected across switches, but can influence control decisions and go undetected. These solutions

Operation	Single key initialization and update				Maximum controller overhead in a network with m switches and n links		Controller overhead when m=25, n=50 [83]	
	#messages exchanged		#bytes					
	local key	port key	local key	port key	#messages exchanged	#bytes	#messages exchanged	#bytes
Key initialization	4	5	104	138	4m + 5n	104m + 138n	350	9.5KB
Key update	2	3	60	78	2m + 3n	60m + 78n	125	5.4KB

TABLE III: **P4Auth scalability with multiple simultaneous key initializations/updates**

attempt to detect violations at runtime, whereas P4Auth is designed to prevent manipulation at runtime.

XI. Discussion

P4Auth scalability. The key initialization/update process, as described in Fig. 14, triggers 2-5 message exchanges (§IX-B) between C-DP and C-DP1-DP2. Consider a network of m switches and n links. In this network, P4Auth triggers at most $4m + 5n$ message exchanges for key initialization and $2m + 3n$ for key update (Table. III). Production networks often adopt multi-node logically centralized but physically distributed controller architecture [83], [98], [99], where one primary controller node is responsible for processing events from a specific subset of switches. Consider an example WAN topology with 205 [83] switches, 414 links, and 8 ONOS controllers. Simultaneous key initialization[4] triggers up to 350 messages with 9.5KB of data in total, at a single controller; it is 125 messages with 5.4KB of data for key updates. Considering 2ms per key initialization (Fig. 20), it takes 150ms to finish (improves significantly when done in parallel), and it is 75ms to update all keys with 1ms per key update. This analysis indicates that overheads are small and thus scale well. Additionally, controllers can carefully batch the key updates to control the number of concurrent updates.

Enhancing security by extending P4Auth. We design P4Auth as a generic authentication and key management framework to enable the plugging of various cryptographic primitives for enhanced security. More specifically, we cover three pluggable primitives in P4Auth: (1) asymmetric key cryptographic primitive for sharing the common secret (§VI), (2) hash function for authentication (§V), (3) PRF for key-material generation (§VI-D). These primitives in P4Auth framework can be replaced with complex and more secure hardware-offloaded target-specific native implementations [100], [101] such as cryptographic hash functions (*e.g.*, MD5, SHA1) for digest computation and as the PRF, and for public-key key exchange algorithms (*e.g.*, DH, RSA). Additionally, P4Auth can be extended to support symmetric key encryption and decryption of C-DP and DP-DP communication by deriving more symmetric keys from the master secret using KDF.

Digest size and computation overhead Since the tofino switch natively supports 32-bit operations, as the digest size increases (*e.g.*, 64-bit to 256-bit), the digest computation and verification require more compute cycles (multiplied by a factor of 2) and more hardware resources. For instance, compared to a 32-bit digest, the hash distribution units and the pipeline stages required for a 256-bit digest are increased by 560% and 100%, respectively. More pipeline stages mean more packet recirculations, which increases C-DP and DP-DP authentication time (100s of ns per recirculation).

Pre-master secret key enhances security. P4Auth uses modified DH as the asymmetric cryptographic primitive for exchanging the secret key, that is, pre-master secret (§VI and Fig. 12). The modified DH [25], [34] replaces modulo arithmetic by XOR operations to be amenable to implementation on switch hardware. XOR-based cryptography is considered secure if the private key is random and never reused. But, on the Tofino hardware, we may not guarantee that the private key (PK2) generated using PRNG is cryptographically strong and never reused. To strengthen the secret key, P4Auth uses KDF that leverages the PRF to randomize the secret key further (Fig. 13), at the cost of additional computation and switch resources (*i.e.*additional Hash Units) for KDF execution. Moreover, P4Auth key exchange security can be enhanced with futuristic switch hardware that natively supports cryptographically strong asymmetric and symmetric key cryptographic primitives; the KDF primitive can derive multiple cryptographically unrelated keys for authentication and encryption and derive initial values and nonces (§VI-D).

XII. Conclusion

In this paper, we propose P4Auth to protect fast traffic control decisions in in-network systems from MitM adversaries at switch software influencing the decisions by manipulating the switch state in the data plane. We design and develop (1) a key-based authentication protocol to ensure the authenticity and integrity of C-DP and DP-DP messages that update/report the state, and (2) a key management protocol using which C-DP and DP-DP securely share and manage secret keys. We prototype P4Auth for two P4 targets, BMV2 and Intel Tofino hardware switch, and evaluate its effectiveness, performance, and resource overheads.

XIII. acknowledgement

We thank our shepherd, Fernando Pedone, and other reviewers for their insightful feedback. We thank Devansh, Prashanth, Shiv, and Harish for their feedback and help on the earlier drafts. This work is supported by the National Security Council Secretariat (NSCS), India, and the Prime Minister Research Fellowship (PMRF).

[4] Assuming each controller is responsible for 25 switches and 50 links on average

References

[1] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *ACM SOSR*, 2016.

[2] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *USENIX NSDI*, 2019.

[3] M. Apostolaki, A. Singla, and L. Vanbever, "Performance-driven internet path selection," in *ACM SOSR*, 2021.

[4] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *ACM SIGCOMM*, 2017.

[5] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *USENIX Security*, 2020.

[6] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[7] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *USENIX Security*, 2021.

[8] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM SOSP*, 2017.

[9] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *USENIX NSDI*, 2016.

[10] ——, "Lossradar: Fast detection of lost packets in data center networks," in *ACM CoNEXT*, 2016.

[11] L. Wang, P. Mittal, and J. Rexford, "Data-plane security applications in adversarial settings," *ACM SIGCOMM Computer Communication Review*, vol. 52, no. 2, pp. 2–9, 2022.

[12] D. Pathak, S. Harish, S. P. Chinta, D. K. Reddy, and P. Tammana, "Anomaly detection in in-network fast reroute systems," in *IEEE IFIP Networking*, 2024.

[13] A. Sanghi, K. P. Kadiyala, P. Tammana, and S. Joshi, "Anomaly detection in data plane systems using packet execution paths," in *ACM SIGCOMM workshop on secure programmable network infrastructure*, 2021.

[14] H. SA, K. S. Kumar, A. Majee, A. Bedarakota, P. Tammana, P. G. Kannan, and R. Shah, "In-network probabilistic monitoring primitives under the influence of adversarial network inputs," in *Proceedings of the 7th Asia-Pacific Workshop on Networking*, 2023, pp. 116–122.

[15] R. Meier, T. Holterbach, S. Keck, M. Stähli, V. Lenders, A. Singla, and L. Vanbever, "(self) driving under the influence: Intoxicating adversarial network inputs," in *ACM HotNets*, 2019.

[16] C. Black and S. Scott-Hayward, "Adversarial exploitation of p4 data planes," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 508–514.

[17] G. Pickett, "Staying persistent in software defined networks," *Black Hat Briefings*, 2015.

[18] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to detect a compromised sdn switch," in *IEEE NetSoft*, 2015.

[19] P. Zhang, H. Wu, D. Zhang, and Q. Li, "Verifying rule enforcement in software defined networks with rev," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 917–929, 2020.

[20] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking control of sdn-based cloud systems via the data plane," in *ACM SOSR*, 2018.

[21] A.-A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "Charting the security landscape of programmable dataplanes," *arXiv preprint arXiv:1807.00128*, 2018.

[22] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, L. J. Wobker *et al.*, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.

[23] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *USENIX NSDI*, 2020.

[24] (2021) P4Runtime. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[25] I. Oliveira, E. Neto, R. Immich, R. Fontes, A. Neto, F. Rodriguez, and C. E. Rothenberg, "Dh-aes-p4: on-premise encryption and in-band key-exchange in p4 fully programmable data planes," in *IEEE NFV-SDN*, 2021.

[26] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-macsec: Dynamic topology monitoring and data layer protection with macsec in p4-based sdn," *IEEE Access*, vol. 8, pp. 58 845–58 858, 2020.

[27] T. Datta, N. Feamster, J. Rexford, and L. Wang, "{spine}: Surveillance protection in the network elements," in *USENIX FOCI*, 2019.

[28] D. Kong, Z. Zhou, Y. Shen, X. Chen, Q. Cheng, D. Zhang, and C. Wu, "In-band network telemetry manipulation attacks and countermeasures in programmable networks," in *IEEE/ACM IWQoS*, 2023.

[29] G. Huston and R. Bush, "Securing bgp with bgpsec," in *The Internet Protocol Forum*, vol. 14, no. 2, 2011.

[30] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (s-bgp)," *IEEE Journal on Selected areas in Communications*, vol. 18, no. 4, pp. 582–592, 2000.

[31] T. J. G. P. d. Vale, "Securing the internet at the exchange points," Ph.D. dissertation, 2022.

[32] (2022) Software-Defined Networks: A Systems Approach. [Online]. Available: https://sdn.systemsapproach.org/future.html

[33] (2008) The Transport Layer Security (TLS) Protocol Version 1.2. [Online]. Available: https://dl.acm.org/doi/pdf/10.17487/RFC5246

[34] S. H. Jeon and S. K. Gil, "Optical secret key sharing method based on diffie-hellman key exchange algorithm," *Journal of the Optical Society of Korea*, vol. 18, no. 5, pp. 477–484, 2014.

[35] (2010) HMAC-based Extract-and-Expand Key Derivation Function (HKDF). [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5869

[36] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Annual Cryptology Conference*. Springer, 2010, pp. 631–648.

[37] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," in *ACM SIGCOMM SPIN*, 2021.

[38] (2016) BMV2 software switch. [Online]. Available: https://github.com/p4lang/behavioral-model

[39] (2020) Intel Intelligent Fabric Processors. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html

[40] R. Cieslak. (2013) Dynamic linker tricks: Using ld preload to cheat, inject features and investigate programs. [Online]. Available: https://rafalcieslak.wordpress.com/2013/04/02/dynamic-linker-tricks-using-ld_preload-to-cheat-inject-features-and-investigate-programs/

[41] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an sdn with a compromised openflow switch," in *Secure IT Systems: 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings 19*. Springer, 2014, pp. 229–244.

[42] E. Christensson, "Man in the middle attacks on software defined network," 2023.

[43] M. Brooks and B. Yang, "A man-in-the-middle attack against opendaylight sdn controller," in *Proceedings of the 4th Annual ACM Conference on Research in Information Technology*, 2015, pp. 45–49.

[44] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures." in *NDSS*, 2015.

[45] (2016) Nist - national vulnerability database. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-2074

[46] A. Shaghaghi, S. S. Kanhere, M. A. Kaafar, E. Bertino, and S. Jha, "Gargoyle: A network-based insider attack resilient framework for organizations," in *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, 2018.

[47] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arXiv preprint arXiv:1903.06701*, 2019.

[48] H. Ke, P. Li, S. Guo, and M. Guo, "On traffic-aware partition and aggregation in mapreduce for big data applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 818–828, 2015.

[49] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift, "Atp: In-network aggregation for multi-tenant learning." in *USENIX NSDI*, 2021.

[50] (2019) Trivy. [Online]. Available: https://aquasecurity.github.io/trivy/v0.42/

[51] (2023) Software for Open Networking in the Cloud. [Online]. Available: https://sonicfoundation.dev/

[52] (2024) Wedge 100bf-32x 32 x 100g qsfp28 switch ports with tofino 32d. [Online]. Available: https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=335

[53] (2017) NIST - National Vulnerability Database, CVE-2017-14159. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2017-14159

[54] (2023) NIST - National Vulnerability Database, CVE-2023-26604. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-26604

[55] (2019) NIST - National Vulnerability Database, CVE-2019-19882. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-19882

[56] (2022) NIST - National Vulnerability Database, CVE-2022-0563. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-0563

[57] (2018) NIST - National Vulnerability Database, CVE-2018-7169. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-7169

[58] (2016) NIST - National Vulnerability Database, CVE-2016-2781. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-2781

[59] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "Sdnsec: Forwarding accountability for the sdn data plane," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–10.

[60] T. Wan, E. Kranakis, and P. C. van Oorschot, "Pretty secure bgp, psbgp." in *NDSS*, 2005.

[61] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, p. 120–126, feb 1978.

[62] (2009) P416 Intel Tofino Native Architecture – Public Version. [Online]. Available: https://csrc.nist.gov/csrc/media/publications/fips/186/3/archive/2009-06-25/documents/fips_186-3.pdf

[63] F. Pereira, N. Neves, and F. M. Ramos, "Secure network monitoring using programmable data planes," in *IEEE NFV-SDN*, 2017.

[64] W. Wang, X. C. Wu, P. Tammana, A. Chen, and T. E. Ng, "Closed-loop network performance monitoring and diagnosis with spidermon," in *USENIX NSDI*, 2022.

[65] (2001) Diffie–Hellman key exchange. [Online]. Available: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[66] N. P. Katta, J. Rexford, and D. Walker, "Incremental consistent updates," in *Second ACM SIGCOMM HotSDN*, 2013.

[67] X. Wang and H. Yu, "How to break md5 and other hash functions," in *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*. Springer, 2005, pp. 19–35.

[68] M. Stevens, A. Lenstra, and B. De Weger, "Chosen-prefix collisions for md5 and colliding x. 509 certificates for different identities," in *Advances in Cryptology-EUROCRYPT 2007: 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007. Proceedings 26*. Springer, 2007, pp. 1–22.

[69] A. Sotirov, M. Stevens, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, "Md5 considered harmful today, creating a rogue ca certificate," in *25th Annual Chaos Communication Congress*, no. CONF, 2008.

[70] F. Mendel, C. Rechberger, and M. Schläffer, "Md5 is weaker than weak: Attacks on concatenated combiners," in *Advances in Cryptology–ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings 15*. Springer, 2009, pp. 144–161.

[71] J.-P. Aumasson and D. J. Bernstein, "Siphash: a fast short-input prf," in *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer, 2012, pp. 489–508.

[72] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic hashing in p4 data planes," in *ACM/IEEE ANCS*, 2019.

[73] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *ACM SIGACT-SIGPLAN POPL*, 1977.

[74] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 337–340.

[75] C. Black and S. Scott-Hayward, "Defeating data plane attacks with program obfuscation," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–13, 2023.

[76] S. Banescu and A. Pretschner, "A tutorial on software obfuscation," *Advances in Computers*, vol. 108, pp. 283–353, 2018.

[77] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *J. ACM*, vol. 59, no. 2, may 2012. [Online]. Available: https://doi.org/10.1145/2160158.2160159

[78] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl, "Protecting software through obfuscation: Can it keep pace with progress in code analysis?" *ACM Comput. Surv.*, vol. 49, no. 1, apr 2016. [Online]. Available: https://doi.org/10.1145/2886012

[79] C. Tezcan, "Key lengths revisited: Gpu-based brute force cryptanalysis of des, 3des, and present," *Journal of Systems Architecture*, vol. 124, p. 102402, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762122000066

[80] (2018) Hula-hoop. [Online]. Available: https://github.com/rachitnigam/Hula-hoop

[81] (2022) CAIDA Macroscopic Internet Topology Data Kit (ITDK) . [Online]. Available: https://www.caida.org/catalog/datasets/internet-topology-data-kit/

[82] (2024) Aurora 610. [Online]. Available: https://netbergtw.com/products/aurora-610/

[83] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *ACM HotSDN*, 2014.

[84] M. V. Dumitru, D. Dumitrescu, and C. Raiciu, "Can we exploit buggy p4 programs?" in *ACM SOSR*, 2020.

[85] M. Neves, L. Freire, A. Schaeffer-Filho, and M. Barcellos, "Verification of P4 Programs in Feasible Time using Assertions," in *ACM CoNEXT*, 2018.

[86] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu, "Debugging P4 Programs with Vera," in *ACM SIGCOMM*, 2018.

[87] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, and N. Foster, "P4V: Practical Verification for Programmable Data Planes," in *ACM SIGCOMM*, 2018.

[88] D. Dumitrescu, R. Stoenescu, L. Negreanu, and C. Raiciu, "bf4: towards bug-free P4 programs," in *ACM SIGCOMM*, 2020.

[89] Y. Zhou, J. Bi, T. Yang, K. Gao, C. Zhang, J. Cao, and Y. Wang, "KeySight: Troubleshooting Programmable Switches via Scalable High-Coverage Behavior Tracking," in *IEEE ICNP*, 2018.

[90] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "Automatic Test Packet Generation," in *ACM CoNEXT*, 2012.

[91] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, and P. Athanas, "P4pktgen: Automated test case generation for p4 programs," in *ACM SOSR*, 2018.

[92] F. Ruffy, T. Wang, and A. Sivaraman, "Gauntlet: Finding Bugs in Compilers for Programmable Packet Processing," in *USENIX OSDI*, 2020.

[93] A.-A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "P4fuzz: Compiler fuzzer for dependable programmable dataplanes," in *ACM ICDCN*, 2021.

[94] K. Bu, X. Wen, B. Yang, Y. Chen, L. E. Li, and X. Chen, "Is every flow on the right track?: Inspect sdn forwarding with rulescope," in *IEEE INFOCOM, 2016*.

[95] Y.-C. Chiu and P.-C. Lin, "Rapid detection of disobedient forwarding on compromised openflow switches," in *IEEE ICNC*, 2017.

[96] A. Kamisiński and C. Fung, "Flowmon: Detecting malicious switches in software-defined networks," in *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, 2015, pp. 39–45.

[97] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: detecting security attacks in software-defined networks." in *NDSS*, 2015.

[98] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks," in *USENIX OSDI*, 2010.

[99] Y. Ganjali and A. Tootoonchian, "{HyperFlow}: A distributed control plane for {OpenFlow}," in *2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking (INM/WREN 10)*, 2010.

[100] (2016) Netronome Agilio CX SmartNICs. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[101] (2023) AMD Pensando™ Networking. [Online]. Available: https://www.amd.com/en/products/accelerators/pensando.html#featured-products