Secure In-Band communication using Programmable switches

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Archit Bhatnagar
ID No. 2019A7TS0133P

Under the supervision of:

Prof. Mun Choon Chan
&
Prof. Virendra Singh Shekhawat

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
September 7, 2023

Declaration of Authorship

I, Archit Bhatnagar, declare that this Undergraduate Thesis titled, 'Secure In-Band communication using Programmable switches' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Archit Bhatnagar

Date: May 8, 2023

Certificate

This is to certify that the thesis entitled, "*Secure In-Band communication using Programmable switches*" and submitted by Archit Bhatnagar ID No. 2019A7TS0133P in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

Supervisor	*Co-Supervisor*
Prof. Mun Choon Chan	Prof. Virendra Singh Shekhawat
National University of Singapore	BITS-Pilani

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

Bachelor of Engineering (Hons.)

Secure In-Band communication using Programmable switches

by Archit Bhatnagar

Conventionally, the control channel on network switches has always been out-of-band. With the emergence of high-performance systems built upon programmable switches, the out-of-band control channel has become the bottleneck. Thus, there is an emerging trend of implementing the control channel in the data path (i.e., in-band) on programmable switches to achieve high throughput and low latency control actions. However, the use of in-band control channels comes with the risk of security vulnerabilities that have not been explored in prior literature. In this thesis, we present P4EAD, a cryptographic primitive to secure the in-band communication(primarily the control channels) on programmable switches entirely in the data plane. This ensures the integrity, authenticity, and confidentiality of in-band control messages. Extensive evaluations are performed to benchmark P4EAD. Through these evaluations and integration with an existing in-band high-performance control framework, we demonstrate the effectiveness of P4EAD in securing the in-band control channel without incurring any significant overhead.

Acknowledgements

I would like to take this opportunity to thank my supervisor Prof. **Mun Choon**, for providing me an opportunity to work at this institution, allowing me to work with hardware programmable switches. I would also like to thank my mentor Xin Ze Khooi, without his support this would not have been possible. I came here a bit confused about how real-world systems & network research works, and I think I have answers to many questions I had before getting started. I would like to thank Prof. **Virendra Singh Shekhawat**, my supervisor at BITS for his support throughout the semester. At last, I would like to thank my family and friends, who are always my constant source of support and motivation.

Contents

List of Figures

Abbreviations

AEAD	**A**uthenticated **E**ncryption with **A**ssociated **D**ata
AES	**A**dvanced **E**ncryption **S**tandard
IoT	**I**nternet **o**f **T**hings
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
P4	**P**rogramming **P**rotocol-independent **P**acket **P**rocessors
P-RND	**P**ermutation **R**ound
SDN	**S**oftware **D**efined **N**etworks

Dedicated to my family.

Chapter 1

Introduction

Modern networks are becoming more flexible and reconfigurable while yet supporting high line-rates. This has led to a shift from traditional networking to Software-Defined Networking(SDN), with there essentially being a centralized brain-like controller operating using flexible software to guide all the hardware switches. The onset of SDNs was supported by the rise of programmable switches and data plane programming. The rise of programmable switches, commonly known as PISA (Protocol Independent Switch Architecture) switches, has enabled faster and more accurate methods to process and monitor all packets entirely in the data plane while maintaining Terabit per sec-scale packet processing performance.

The basic difference between a normal switch ASIC and a programmable switch ASIC, as shown in Fig. 1.1 is the way the packets are processed initially in the internal pipeline, while a fixed function switch has a fixed pipeline for parsing the packet headers in a fixed manner allowing no flexibility. In PISA-based switches, the logic for each pipeline stage(or Match Action Unit) and the packet parsing logic are defined using a high-level programming language called **P4**. The onset of programmable switches has improved flexibility in terms of functions and protocols supported by the network. The commoditization of programmable switches has ignited the "Cambrian explosion" of innovations in the network data plane. This has induced a paradigm shift in designing a new breed of high-performance mission-critical systems in the data plane(running at unprecedented speeds (e.g., line-rate processing at 12.8 Tbps), including stateful load balancers [17, 14, 32], cloud gateways [22], application key-value store caches [16], 5G radio access network front-haul slicing mechanisms [8], and 5G user-plane functions (UPF) [6, 19].

While the packet processing capacity in the data plane has seen extensive growth over the years (e.g., up to 12.8 Tbps), the I/O performance of the communication channel between the control plane and the data plane has largely fallen behind. Here, the control plane can either be *(i)* locally on the switch – talk to the OS kernel driver to communicate with the ASIC over PCI-E,

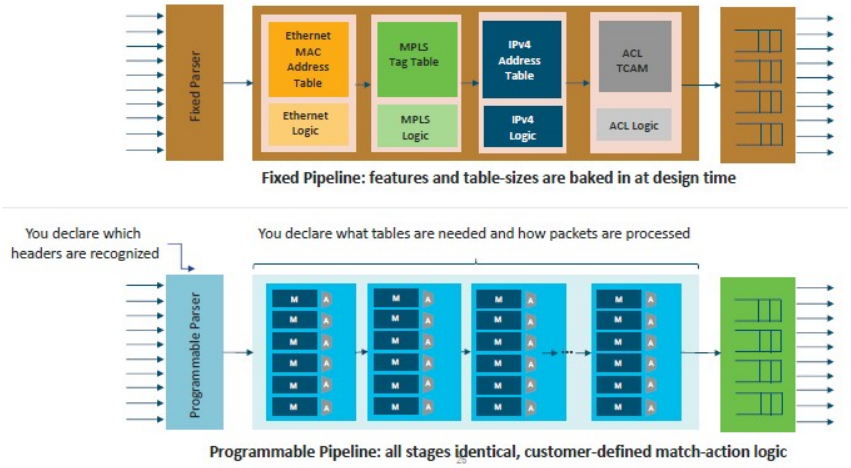


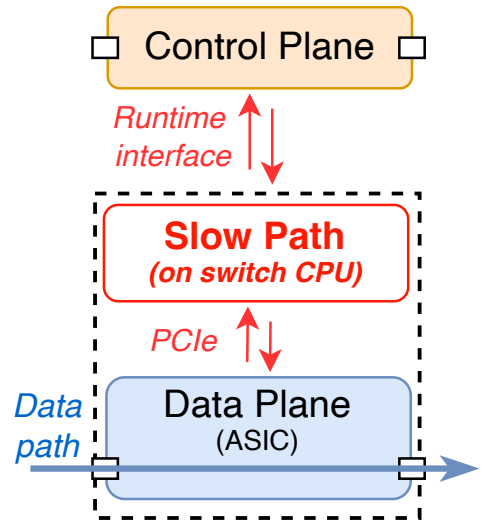
FIGURE 1.1: Pipeline difference between a fixed function and programmable switch. [24]

or *(ii)* situated on a centralized server that interacts with multiple switches' local control plane over secure TLS sessions (see Fig. 1.2a). communication channel refers to either locally, or remotely through the switch's OS kernel driver to communicate with the ASIC over PCI-E. In these *out-of-band* communications, the controller either talks to the switching ASIC directly over the local PCI-E bus or to other controllers running on the switch CPU over secure TLS sessions (see Fig. 1.2a). We also generalize the use case for inter-switch communication where essentially the local control planes of both the switches interact using the shared data planes instead of even going to the controller at all. Prior literature (DySO [28], AccelUPF [6]) have highlighted the significant difference in data plane forwarding rates (10^9 operations per second [10]) and control channel rates (10^4 to 10^5 operations per second [32]), and the impact of these differences on system performance.

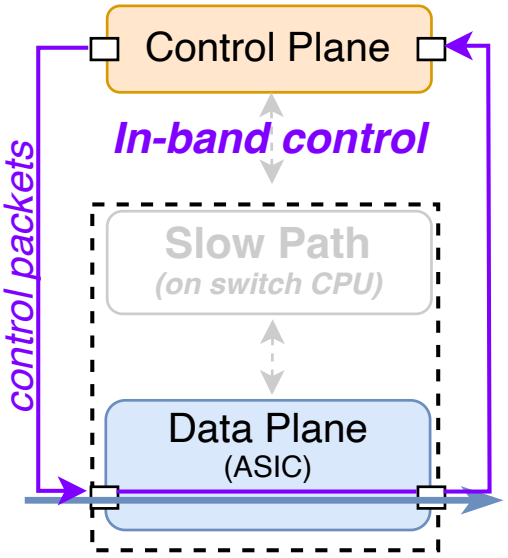
Similarly, poor throughput and high latency in configuring the data plane to set up 5G user sessions over the conventional control channel is also observed.

To overcome the aforementioned issue, many have argued for the use of *in-band control channels* [28, 32, 6, 8]. Essentially, the control plane directly communicates with the data plane over the data channel by sending dedicated *control packets* alongside normal traffic, which contains the corresponding instructions and/or states (see Fig. 1.2b). Basically, the instructions of how to react to different packets ingressing for each switch are based on and change with these updates. This enables one to attain high throughput and low latency control operations in updating/configuring the data plane, bypassing the conventional control channel. As a result, greater system performance can be achieved with a tighter control loop.

Many of these systems share either of the following traits: they depend on *(i)* near-real-time feedback between switches (e.g., via probe packets) or *(ii)* tight control loops between the control plane and the data plane to timely react to network events [18], or configure the appropriate data plane forwarding states [8, 17, 6]. Prior literature [28, 32] has highlighted the huge disparity



(A) Out-of-band.



(B) In-band.

FIGURE 1.2: Overview on out-of-band/in-band control.

between the conventional out-of-band PCI-E-based control channel and the multi-Tbps data plane forwarding rates. Thus, many have argued for the case of implementing the control channel in an in-band manner (i.e., via control packets) alongside data traffic.

Unsecure In-band Control Channels: There has been extensive research on securing the out-of-band control channels [5, 1]. Typically. state-of-the-art control plane implementations established secure control channels, e.g. using TLS, by default. The introduction of *in-band control channels* did not put performance as the priority over security. Now that data plane behaviour can be controlled through probe or control packets that are processed entirely in the plane, there is a need to ensure that the forwarding states can only be modified by the "rightful" party. At the same time, this departs from the conventional model of out-of-band control

channels and thus necessitates immediate attention to safeguard emerging high-performance networked systems. On the other hand, less care is given to securing in-band signalling. However, greater use of in-band signalling opens up greater attack surfaces for adversaries to manipulate the network behaviour through the injection of maliciously crafted or spoofed control packets. However, techniques used for securing out-of-band signalling in the control plane are not suitable for securing in-band signalling running in the data plane without any significant impact on system performance.

Threat model We assume an attacker that is present in the network and is able to intercept, modify and replay the control packets. The attacker's aim is to hamper the system's performance and/or cause denial of service

Securing the In-band Control Channels: Intuitively, the threat can be addressed by establishing a *secure* in-band control channel to protect the authenticity of control packet source and the integrity of the control plane update. This necessitates the data plane to implement necessary cryptographic primitives to support it. However, the restrictive pipeline programming model and limited hardware resources and limited instruction sets on commodity programmable switches make it challenging to implement conventional algorithms like the ones used in TLS i.e. AES-GCM or ChaCha20-Poly1305[20] as they require complex operations such as modular arithmetic involving large numbers (of order 2^{130},) and large number of loops operations(due to bit-by-bit procedure) that are not amenable to run on commodity programmable switches. Even if we are able to explicitly compromise on speeds, the minimum block lengths/configurations for which these cryptographic algorithms are considered "secure" is infeasible for programmable switches. To the best of our knowledge, this has not been attempted in the context of commodity programmable switches.

To that end, we identify ASCON, a lightweight cipher suite recently standardized by NIST [21] as a suitable candidate given their hardware-friendly design – requiring only binary operations like XOR, ROR, and AND. We design and implement ASCON-based authentication for the Intel Tofino and Tofino2 ASIC by adapting to the pipeline programming model.

We present P4EAD[1], an implementation of the ASCON-AEAD algorithm that runs entirely in the data plane to guarantee the authenticity and integrity of the control packets transmitted over the in-band control channel. In this thesis, the detailed implementation challenges and solutions applied to implement ASCON-AEAD on the Intel Tofino switches are discussed. Secure control channels aside, we envision our effort to be a key building block for many emerging secure in-network applications running on commodity programmable switches. The main contributions of the thesis are as follows:

[1] pronounced as "paid".

1. P4EAD: An implementation of ASCON-AEAD on both the Intel Tofino and Tofino2 switches.

2. Extensive performance evaluations of P4EAD with different pipeline configurations, input length alongside their hardware resource trade-offs to act as a guideline for data plane developers.

3. To demonstrate practicality, we integrate P4EAD to a state-of-the-art in-band control channel framework [28] and observe negligible performance degradation.

4. To motivate the need for securing in-band communication. To this end, P4EAD will publicly available on to facilitate the development of secure data planes.

Chapter Overview: The remainder of the thesis is structured as follows- Chapter 2 introduces the background and work related to cryptographic primitives on the programmable switches.In Chapter 3, we introduce our solutionP4EAD and discuss its implementation. Then we benchmark and evaluate P4EAD over different configurations in Chapter 4. This is followed by a case-study in Chapter 5 and a discussion on future work in Chapter 6.

Chapter 2

Background & Related Work

In this Chapter, we introduce authenticated encryption, ASCON cipher suite, and elaborate on ASCON-AEAD. We also discuss cryptographic primitives for hashing and standard encryption on programmable switches further on in the Chapter.

Authenticated Encryption: Traditional symmetric encryption is based on using a key to convert plaintext into ciphertext in a way that it can only be decrypted using the same key. We focus only on symmetric encryption because of the known immense complexity and extremely slow speeds of asymmetric encryption. There are two different types of encryption that offer different levels of security: standard encryption and authenticated encryption.

Standard encryption encrypts data using a key to ensure that only people with the same key can decrypt and read the data. It does not, however, offer any extra security guarantees for the authenticity or integrity of the encrypted data. This implies that even while the data may be

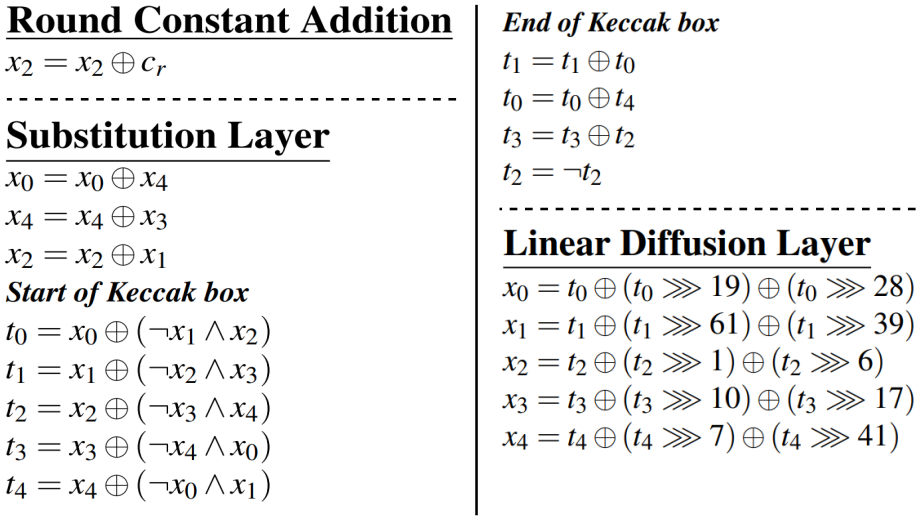


FIGURE 2.1: Overview on a P-RND in a PERM.

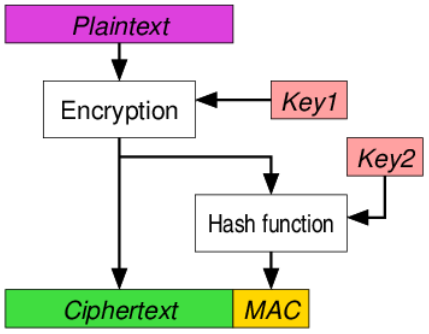


FIGURE 2.2: Encrypt then MAC approach [3]

shielded against unwanted access, there is no way to tell if it has been altered or tampered with and if it is from the intended source.

Whereas authenticated encryption guarantees the authenticity and integrity of the encrypted data along with its confidentiality. In order to make sure that the data hasn't been altered while in transit and that the sender is who they say they are, authenticated encryption uses a key for encrypting the data and uses a part of the unencrypted data to generate a message authentication code (MAC) which is sent in conjunction with the encrypted ciphertext thus binding the ciphertext to itself. This MAC is used to verify the integrity and authenticity of the ciphertext. Thus, authenticated encryption offers more protection than standard encryption since it guards against tampering and illegal access. A variation of authenticated encryption utilising the Encrypt-then-MAC structuring is depicted in Fig. 2.2.

AES-GCM [12] and ChaCha20-Poly1305 [20] are widely adopted and field-tested authenticated encryption schemes for secure communications, e.g., TLS [27]. They include associated data (AD), e.g., sequence numbers in the clear, as additional context bound to the ciphertext to prevent replay attacks. They provide three key security properties, namely confidentiality, integrity, and authenticity.

However, implementing these schemes on commodity programmable switches is infeasible, if not impractical. For instance, AES-GCM involves the Galois Field (GF) multiplication process, which has to be done bit by bit [29] and results in unacceptably slow performance. As for ChaCha20-Poly1305, it requires modular arithmetic performed on large numbers of the order $2^{1}30$. On one hand, modular arithmetic is unavailable on commodity programmable switches. On the other hand, even if possible through lookup tables, it requires a non-negligible amount of SRAM in the data plane which makes it inherently expensive. ASCON-AEAD being part of the lightweight ASCON [11] cipher suite designed for resource-constrained systems, presents a much more amenable option for implementation on programmable switches over AES-GCM and ChaCha20-Poly1305, while offering similar security properties using fewer resources and greater performance.

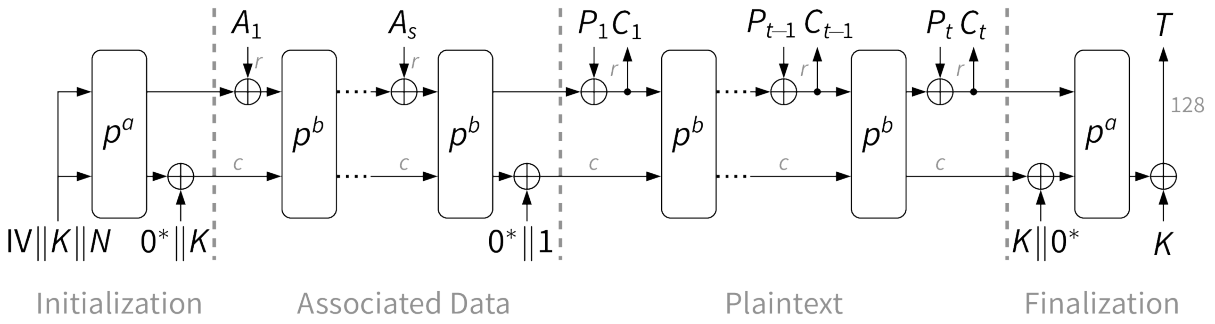


FIGURE 2.3: AEAD Encryption Side Overview [21]

ASCON cipher suite: ASCON is a family of lightweight authenticated encryption and pseudorandom functions (PRF) which was standardized for lightweight cryptography by NIST [21, 4] for resource-constrained systems. The suite consists of primitives like authenticated encryption with associated data (AEAD) and hashing functionality like ASCON HASH and ASCON HASHA. ASCON was the primary choice for lightweight authenticated encryption in the CAESAR competition.

ASCON-128, in particular, was recommended by NIST due to certain advantages like high cryptanalytic security, simplicity and the possibility of a cross-platform design. It is also provisioned to be secure against side-channel attacks and has room to implement counter-measures. ASCON provides room for security on extremely constrained devices which could not afford encryption before and is also pretty scalable for more conservative security or higher throughput. For this thesis, we are mainly focusing on the authentication variant of ASCON i.e. ASCON-AEAD. A general overview of the encryption side of ASCON is depicted in Fig. 2.3. Further, in Chapter 3, we go on to break down the ASCON functionality and discuss its implementation.

ASCON particularly relies only on simple operations (e.g., ADD, NOT, AND, ROR, and XOR), and the computations operate over a constant 320-bit state vector (VECT), i.e., it is memory efficient. At the core of ASCON, permutations (PERM) form the key building block, which consists of n × P-RNDs, denoted as PERM(n). A PERM is either PERM(6),PERM(8), or PERM(12). Each P-RND consists of three layers: *addition*, *substitution*, and *linear diffusion*, which operate on VECT iteratively. We depict the operations within a P-RND in Fig. 2.1.

In this paper, we focus on its variant for authenticated encryption – ASCON-AEAD. Fig. 2.4 illustrates the high-level workflow of ASCON-AEAD. To perform authenticated encryption, there are four phases, namely initialization, associated data (AD) absorption, plain text absorption, and tag generation & finalization. In the init phase, the keys, nonces, and initialization vector are copied to VECT before performing a PERM(12). Then, the associated data (AD), e.g., IP header/ sequence number in the clear, is "absorbed" in 64-bit blocks and goes through PERM(6) sequentially. After AD absorption, an additional PERM(6) is performed. The same process is

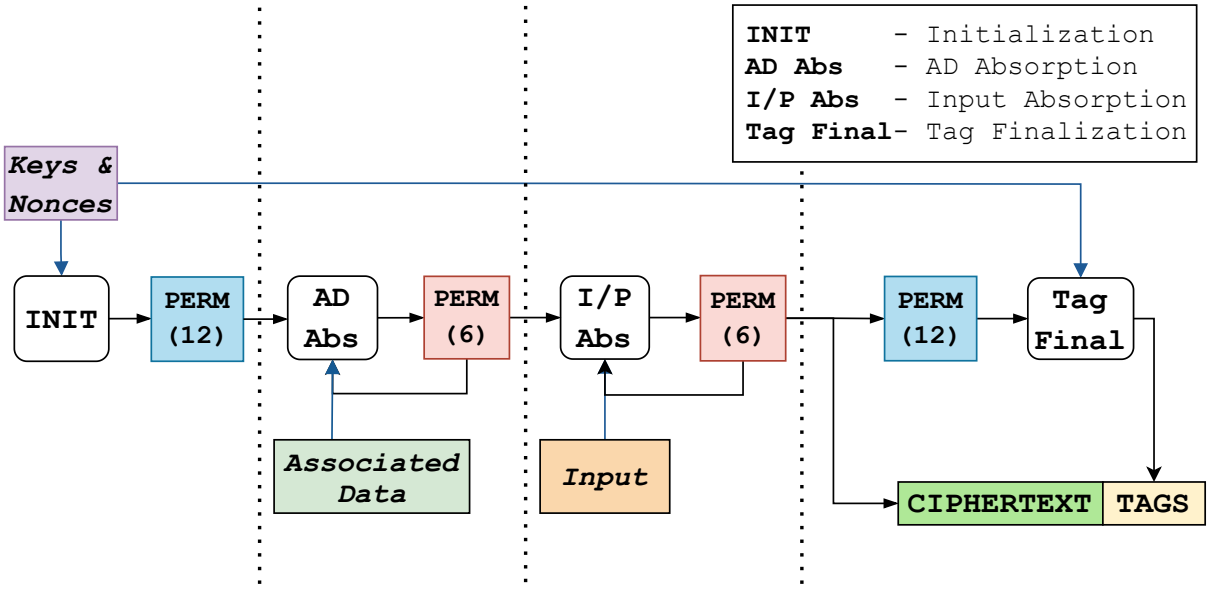


FIGURE 2.4: ASCON-AEAD encryption/ decryption flow.

repeated for the plaintext to get the ciphertext. To get the tag, VECT is sent through another PERM(12) and is retrieved after XORing with the keys. The decryption flow is symmetrical to encryption, except that it comes with an additional tag verification step at the end for integrity checks.

Implementing cryptographic algorithms on programmable switches: There have been several efforts on implementing cryptographic primitives on programmable switches like encryption schemes (e.g., P4-AES [9], ChaCha [31]), and secure keyed hash functions (SipHash [30]). However, existing schemes cannot provide the three necessary security properties to ensure end-to-end secure communication, and thus P4EAD is orthogonal with the aforementioned. This also makes ASCON-AEAD the first cryptographic primitive in the data plane to support secure communication channels. While one can compose P4-AES and SipHash to construct an authenticated encryption scheme, it is not proven to be secure [26] and it requires more dedicated hardware resources. Further, the P4-AES approach is not scalable, as the number of sessions is strictly limited by the memory available to maintain the per-key precomputed lookup tables. In contrast, ASCON-AEAD requires little memory (see chapter 3) to maintain the constants (i.e., 12×16-bit numbers) used for the P-RNDs. Given the low resource requirement of ASCON, it can be integrated into and co-exist with existing data plane programs to secure the in-band communication channels.

Chapter 3

P4EAD: Overview & Implementation

In this section, we present our implementation of P4EAD in P4, mainly discussing the version for the Intel Tofino switch. The discussion here also applies to the Intel Tofino2 switch.

3.1 Preliminaries

First, we discuss the preliminaries on programmable switches that relates to the P4EAD implementation. The switching pipeline consists of the Ingress and Egress blocks, which share common hardware resources [7] while remaining mutually exclusive. Thus, hardware resources are at a premium. A key challenge to implement P4EAD is that there is only a limited number of pipeline stages available (e.g., 12 [10] and 20 [15] in each block on Tofino and Tofino2, respectively). Within each stage, multiple operations (e.g., binary/ arithmetic) can be performed on mutually-exclusive/independent variables by the ALUs. The resulting output of any computation at a particular stage will only be available to the next stage of the pipeline. This chain of dependency determines the number of stages required for any particular program. If a program requires more stages than are available, the program cannot be compiled. Finally, if a computation cannot be completed within a pipeline pass, one common technique is to recirculate the packet back into the pipeline for another pass while retaining the overall state of operations as a part of the header. This is analogous to the concept of loops in general programming. However, packet recirculation consumes the available packet processing capacity, increases latency, and reduces throughput making it a trade-off we can take but not a luxury to build upon.

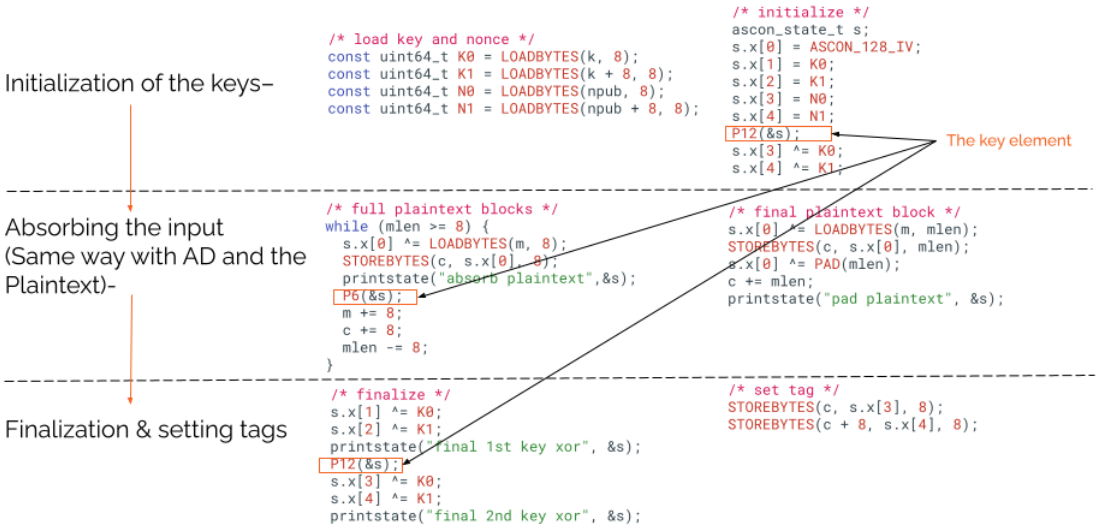


FIGURE 3.1: Encryption flow with C code for ASCON

3.2 Implementation

We have implemented both the HASH and ASCON-AEAD variants of ASCON on the switch. Here aligning with that thesis we only discuss our implementation of the ASCON-AEAD version of ASCON, which uses a duplex-sponge-based mode of operation for authenticated encryption. Since the building blocks for them are similar, this discussion can be extended to ASCON-based hashing as well. The recommended key, tag and nonce length are 8 bytes each.

General flow : The three major stages of ASCON around which the states for our implementation revolve are -

- Key and Nonce initialization
- Absorbing the Associated data and Plaintext(equivalent to Encryption in Fig. 2.2)
- Finalization and setting tags (equivalent to Hash function in Fig. 2.2)

The general flow of the encryption side of ASCON-AEAD in a C code representation is shown in Fig. 3.1.

PERM (as discussed in Chapter 2 is a core component of ASCON-AEAD. Naively implementing PERM would result in a long, interwoven dependency chain that would not fit within the switching pipeline. To that end, we employ packet recirculations to do PERM over multiple pipeline passes – all phases would share the same P-RND implementation. However, packet recirculation results in decreased throughput. We implement the P-RNDs in both Ingress and Egress pipelines to increase the number of P-RNDs that can be done within a single pipeline pass to reduce the number of recirculations required and its impact on throughput and latency.

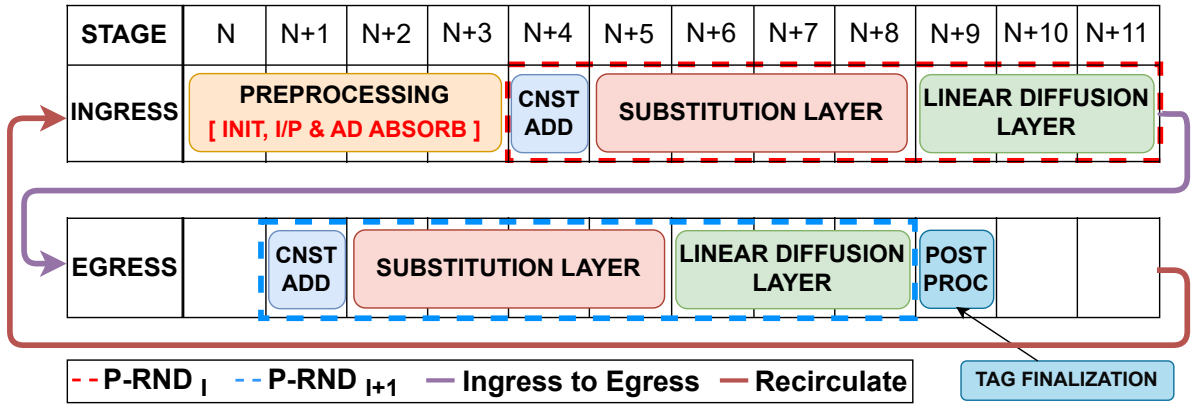


FIGURE 3.2: Overall stage layout(with recircs) of P4EAD on the Intel Tofino

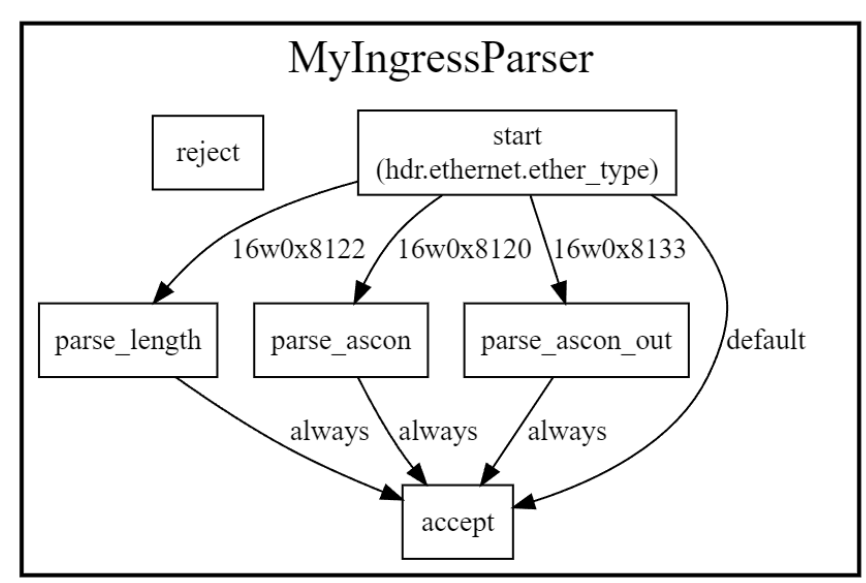
Fig. 3.2 illustrates how ASCON-AEAD is implemented on the Intel Tofino. In this example, we perform two P-RNDs within a pipeline pass, one each at the Ingress and Egress blocks[1]. Here, we leverage the fact that PERM always has an even number of P-RNDs (i.e., PERM(6), PERM(8), or PERM(12)), we expect PERM to terminate at the egress and proceed to the next phase at the ingress after the packet is recirculated. Thus, we perform preprocessing (i.e., initialization and absorption) at the Ingress and postprocessing (i.e., finalization) at the Egress. Note that the pre- and postprocessing blocks are bypassed when the packet is in the middle of a PERM.

Packet recirculation: Before recirculating packets, we add a special metadata header to the packet to store the current P-RND number and relevant states for stateful processing across recirculations. At the same time, we also save the original output port so that the packet can be forwarded accordingly at the end of the computation.

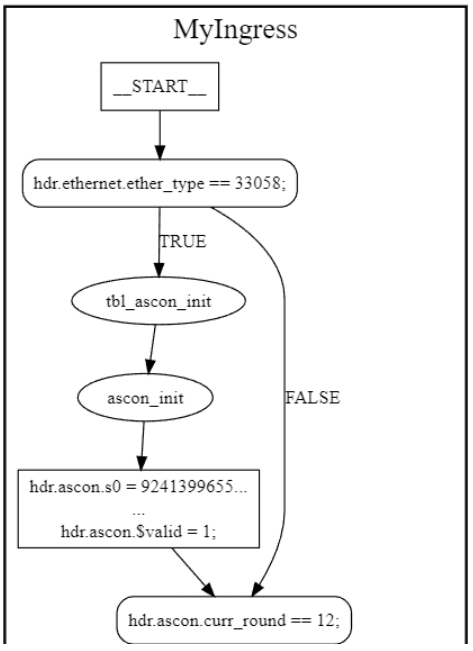
Preprocessing: The preprocessing block handles three kinds of packets: *(i)* newly received packets that needs to be encrypted/ decrypted, or *(ii)* recirculated packets in the middle of a PERM, or *(iii)* recirculated packets at the end of a PERM.

For *(i)*, we would initialize VECT, append it to the header stack, and then proceed to the initialization phase with the retrieved keys, nonces, and IV. Here, the keys are retrieved from match-action tables which are inserted after key exchange procedures[2]. In the case of *(ii)*, the preprocessing block is bypassed. For *(iii)*, absorption is performed by XORing the next input block with VECT, before continuing with the next PERM of the next phase. Handling these different states is a non-trivial task since a packet can be in any state each new recirculation. Thus to optimize the processing, we parse the payload only when required and only append new headers after they are completely generated. This is achieved by a combination of round numbers as a header and using a unique Ethernet type to signify the state a packet currently is in and then process it accordingly. The parser logic and a part of the ingress block are shown in Fig. 3.3 for reference.

[1] More P-RNDs can be performed within a single pipeline by committing more hardware resources, if available.
[2] In this paper, we consider the key exchange procedure out-of-scope.



(A) Ingress Parser



(B) Small part of the Ingress block

FIGURE 3.3: Overview of Tofino Parser and part of the Ingress logic

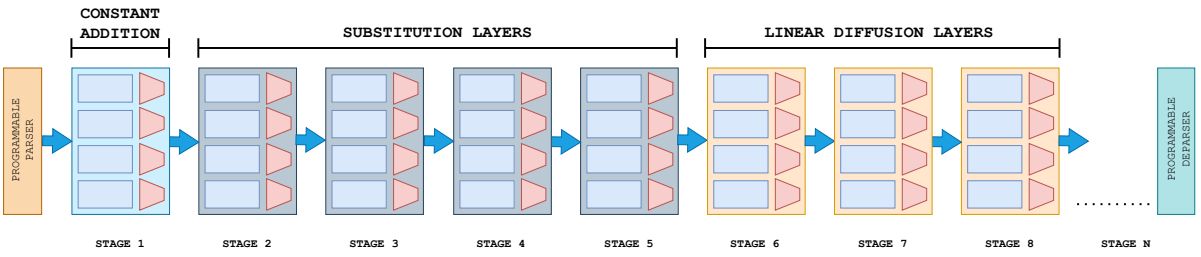


FIGURE 3.4: Basic layout of a P-RND in a PISA architecture based switch

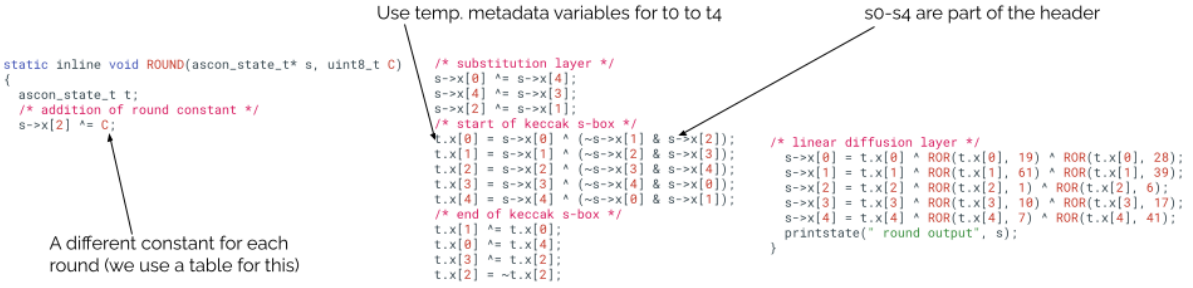


FIGURE 3.5: A P-RND's flow with C code as reference

P-RND: The key idea is that each step of ASCON involves permutation, which is iteratively basically applying an SPN-based round transformation 6 or 12 times. Extracting from [21], each round transformation consists of the following three steps, which operate on a 320-bit state divided into five words x_0, x_1, x_2, x_3 and x_4 of 64 bits each-

- Addition of Round Constants: XORs a round-specific 1-byte constant to word x_2.
- Nonlinear Substitution Layer: applies a 5-bit S-box 64 times in parallel in a bit-sliced fashion (vertically, across words).
- Linear Diffusion Layer: XORs different rotated copies of each word (horizontally, within each word).

The stage layout for a P-RND is shown in Fig. 3.4. A C code representation for the same is shown in Fig. 3.5.

Fig. 2.1 depicts the operations involved in a P-RND. To begin, each round starts with the addition layer. Here, a constant is added by XORing with x_2 from VECT which is predefined for each individual P-RND. We use a match-action table to implement a lookup table for the addition constants, as presented in Fig. 3.6. This is implemented within one pipeline stage.

It is then followed by the substitution layer which spans across 4 stages. Here, VECT is manipulated multiple times in a way that is non-trivial for the switch pipeline computation model. For example, take the operation $t_0 = x_0 \oplus (\neg x_1 \wedge x_2)$. This operation needs to be broken down into two parts. We have to compute $(\neg x_0 \wedge x_1)$ before getting the final result for t_0. This requires us to break down the operation into two separate stages and store the intermediate results as separate variables in the metadata.

The final layer is the linear diffusion layer which consists of XORs and ROR operations.We implement RORs using bit-slicing and concatenation [30]. Each operation involves two RORs and two XORs. Initially, we break down the operations in the diffusion layers into three parts by performing the RORs using the ALUs and PHVs naively and storing them using different intermediate variables during computation. This results in an extremely long dependency chain

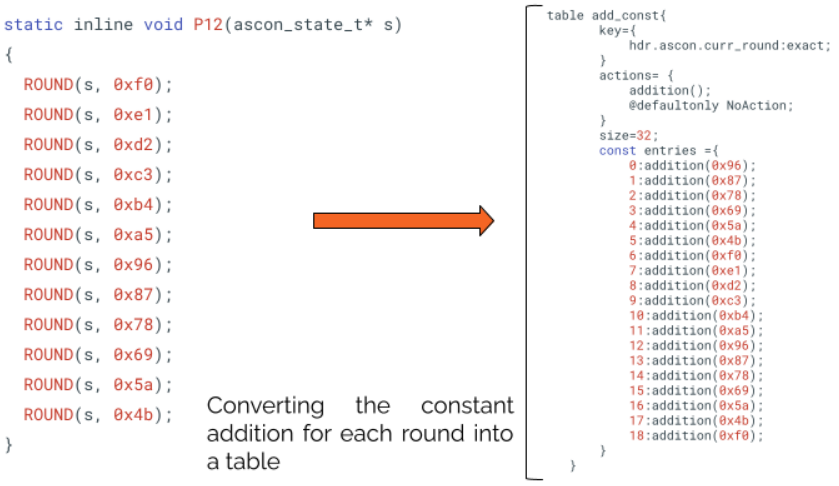


FIGURE 3.6: Round Constant Addition executed using a table in P4

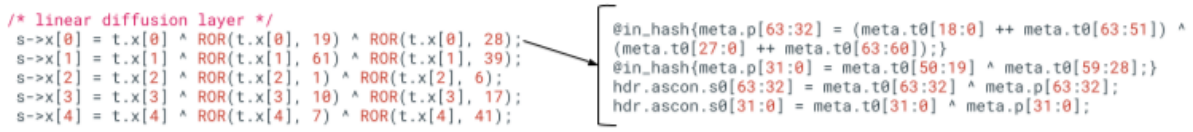


FIGURE 3.7: Linear Diffusion Layer in P4 using in_hash pragma.

for the linear diffusion layer and the program would not compile. Instead, we exploit the hash engines within the pipeline stages to perform the RORs using identity hashing and then feed the three parts to the ALUs to perform XOR within a single stage, as depicted in Fig. 3.7 This allows us to implement the linear diffusion layer within 3 stages.

In total, our implementation of P-RND requires 8 stages.

Postprocessing: At the end of the final PERM tag finalization (and tag verification for decryption) is(are) performed at the postprocessing block. After that, the header containing VECT is stripped off before forwarding the resulting packet.

Formula on number of recirculations Let p and q be the length of the input and associated data in bytes, respectively. Then, we let r be the number of P-RNDs per pipeline pass. The total number of pipeline passes, s, is given by:

$$s = \frac{12 + [6(\lfloor \frac{q}{8} \rfloor) + 6] + 6\lfloor \frac{p}{8} \rfloor + 12}{r} = \frac{30 + 6(\lfloor \frac{p}{8} \rfloor + \lfloor \frac{q}{8} \rfloor)}{r}$$

For example, for 8 byte input ($p = 8$), 4 byte AD length ($q = 4$) and 4 P-RNDs per pipeline pass using Tofino2 ($r = 4$), we require $\frac{30+6(0+1)}{4} = 9$ passes.

TABLE 3.1: P4EAD's hardware resource consumption on the Intel Tofino for different configurations.

Resource	**Tf1_(1rd/p).p4**	**Tf1_(2rd/p).p4**	**Tf2_(1rd/p).p4**	**Tf2_(2rd/p).p4**	**Tf2_(3rd/p).p4**	**Tf2_(4rd/p).p4**
SRAM	0.6%	0.8%	0.4%	0.6%	0.7%	0.8%
Hash Bits	6.6%	13.2%	4%	7.9%	11.9%	15.9%
Hash Dist Unit	27.8%	55.6%	16.7%	33.3%	50.0%	66.7%
VLIW Ins.	5.2%	6.5%	3.3%	3.8%	5.8%	6.3%
PHV	22.9%	29.3%	21.47%	25.6%	25.6%	33.9%

Hardware Resource Utilization We illustrate the hardware resource utilization for all the possible configurations of P4EAD, i.e., 1, 2 and 1, 2, 3 & 4 P-RNDs per pipeline pass for Tofino and Tofino2, respectively in Table 3.1. Notably, our implementation heavily uses the hash distribution units (HDUs) for the linear diffusion layers in a P-RND. Here, each *individual* P-RND consumes 27.8% and 16.7% of the HDUs on the Tofino and Tofino2. As for the other H/W resources, they do not vary much except for the hash bits, which increase linearly with the number of P-RNDs. Depending on the resources available, a data plane developer can decide on the corresponding configuration to integrate P4EAD to secure their applications.

Chapter 4

Evaluation & Benchmarks

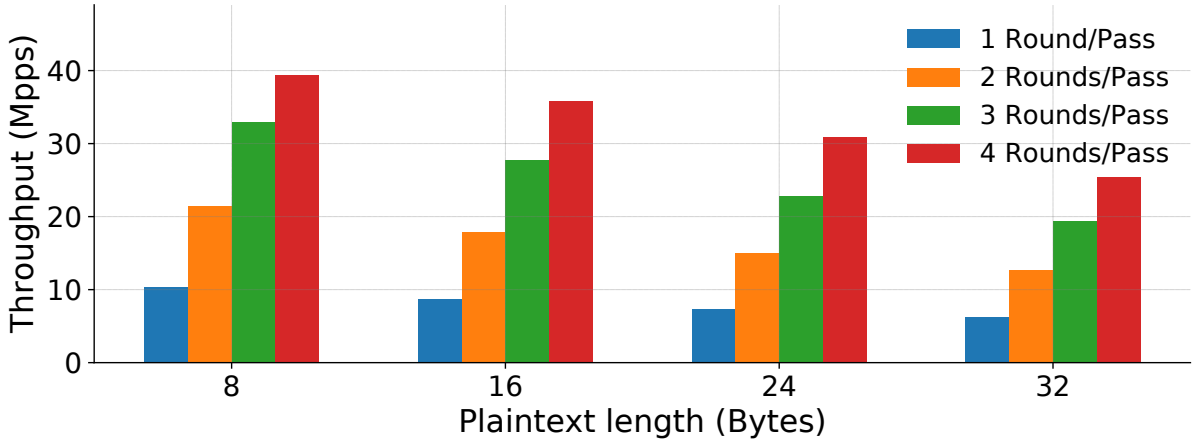
In this section, we evaluate the performance of P4EAD. First, we verify the correctness of our implementation against the reference implementation written in C [2] and the official test vectors. Next, we evaluate how the pipeline configurations (i.e., the number of P-RNDs per pipeline pass), and the input length (i.e., plaintext/ ciphertext) affect the throughput and latency. We also evaluate the scalability of our implementation by varying the number of dedicated recirculation ports. We also evaluate the deterministic latency and the impact of changing associated data lengths. Based on this in chapter 5, we present a case study by integrating P4EAD with an in-network KV store and study the performance impact.

Configurations: Our implementation allows flexibility in the number of P-RND s per pipeline pass. To this end, there are different set of configurations possible for both Tofino and Tofino2 switches. We vary the input plaintext size which indirectly changes the total number of P-RNDs. We also showcase the change in throughput and latency with changing Associated Data.

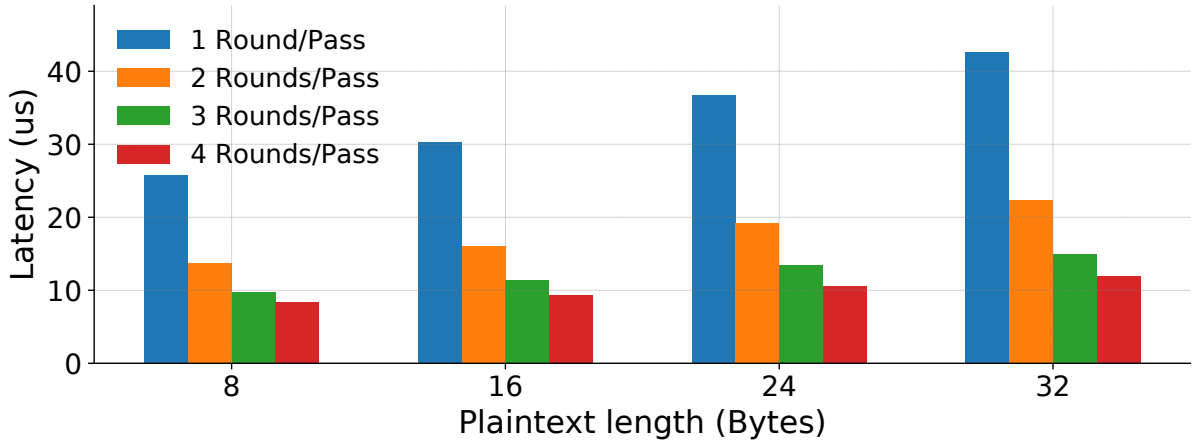
4.1 P4EAD Benchmarks

Setup: Our setup consists of three Intel Tofino switches. We compile and run the different variants of P4EAD using the Intel P4 Studio v9.11.2 on an Intel Tofino and Tofino2 switch. For traffic generation, we make use of the packet generator feature on a separate Intel Tofino switch which is inter-connected with the other two switches using two 100 Gbps copper DACs for evaluation.

We show only the encryption performance (with no packet loss) and omit decryption given that they are symmetrical. The results shown represent the best possible throughput without any packet drops. Unless otherwise stated, we use only one port for recirculation. The length of associated data (AD) is fixed at 4 bytes.Fig. 4.1 provides a comprehensive overview of the



(A) Throughput



(B) Latency

FIGURE 4.1: Combined overview- Impact of the number of P-RNDs per pipeline pass and plaintext length for the Intel Tofino2.Tofino 1 follows a similar pattern.

change in throughput and latency for Tofino 2 and acts as a brief guideline for the variation trend. Tofino1 follows a similar suit, we discuss the trends in detail next.

Impact of #P-RNDs per pipeline pass: We depict the results in Fig. 4.1 for Tofino 2. To ease discussion, we first look at the case when the input length is 8 bytes. We observe that the P4EAD throughput increases linearly from ∼10 Mpps to ∼40 Mpps when the number of P-RNDs per pipeline pass increases from 1 to 4. As for the latency, we see a corresponding decrease from 25.6μs to 8.5μs since there are fewer recirculations when more P-RNDs can be done within a pipeline pass. Similarly, for Tofino1, the throughput increases from 2.5 Mpps to 5 Mpps as P-RNDs increase from 1 to 2. As expected, the latency drops from 23.3μs to 12.23μs with 2 P-RNDs per pipeline pass.

Using one recirculation port in the pipeline, one can achieve up to ∼5 million packets per second (Mpps) on the Intel Tofino with 2 ASCON permutation rounds per-pipeline pass and up to ∼40 Mpps on the Intel Tofino with 4 ASCON permutation rounds per-pipeline pass. For brevity yet

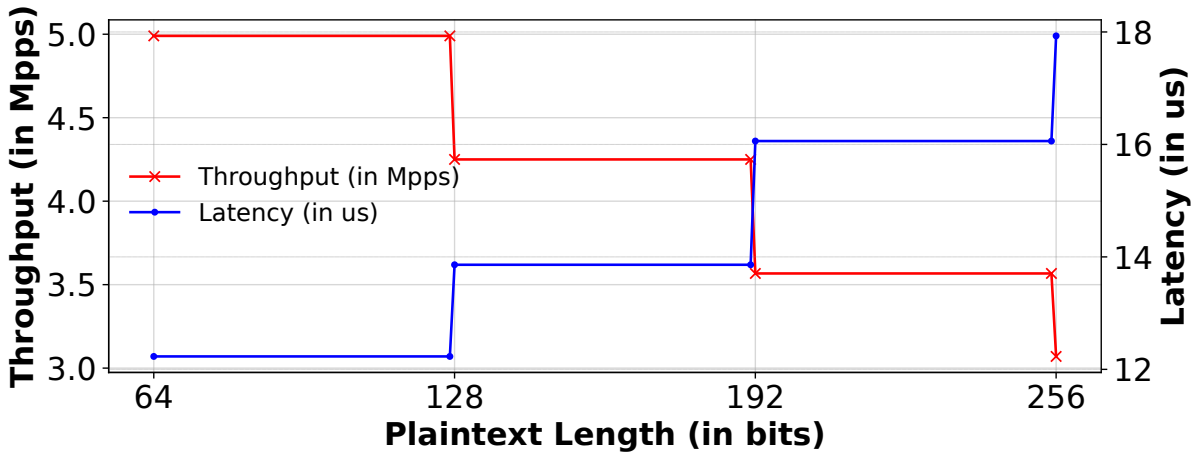


FIGURE 4.2: P4EAD throughput and latency on the Intel Tofino with **2** PERM rounds/pass. Assuming the header fields are of the same size as the maximum case and only one port for recirculation.

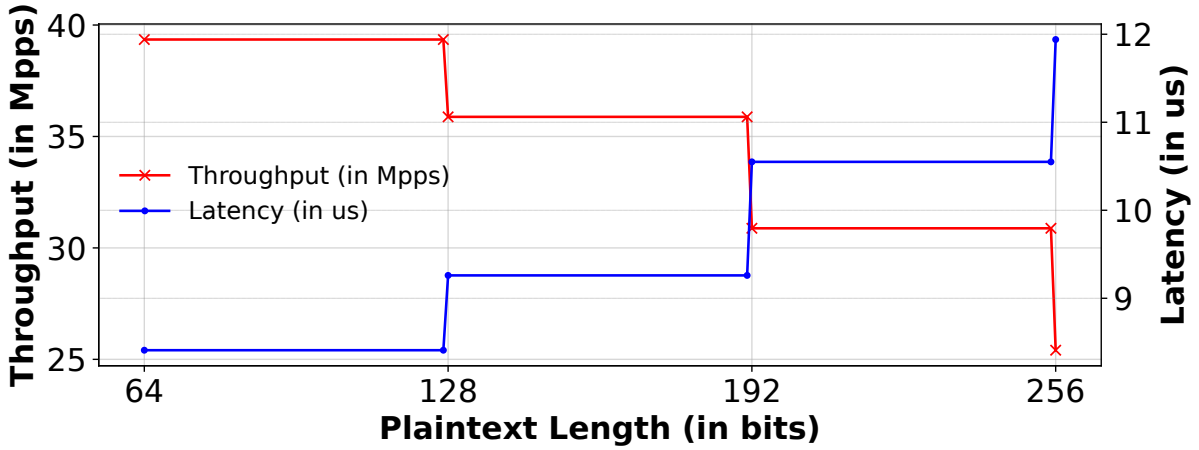


FIGURE 4.3: P4EAD throughput and latency on the Intel Tofino 2 with **4** PERM rounds/pass. Assuming the header fields are of the same size as the maximum case and only one port for recirculation.

without the loss of generality, we will be using the above-mentioned configuration for the rest of the discussion.

Impact of input length: Next, we evaluate the effect of different input lengths[1]. Here, we fix the length of the associated data at 4 bytes. We consider the maxed-out configurations i.e. 2 and 4 P-RNDs/pass for Tofino 1 and 2 resp. For every increase in 8 bytes, one additional PERM (6) is needed (see chapter 2). This leads to another interesting observation since the number of P-RNDs is the same until there is a jump in input length exactly with a multiple of 8. As an example, the throughput(and correspondingly the latency) remains constant(considering the header appended already had output fields corresponding to the increased payload size and

[1]This also applies to the associated data.

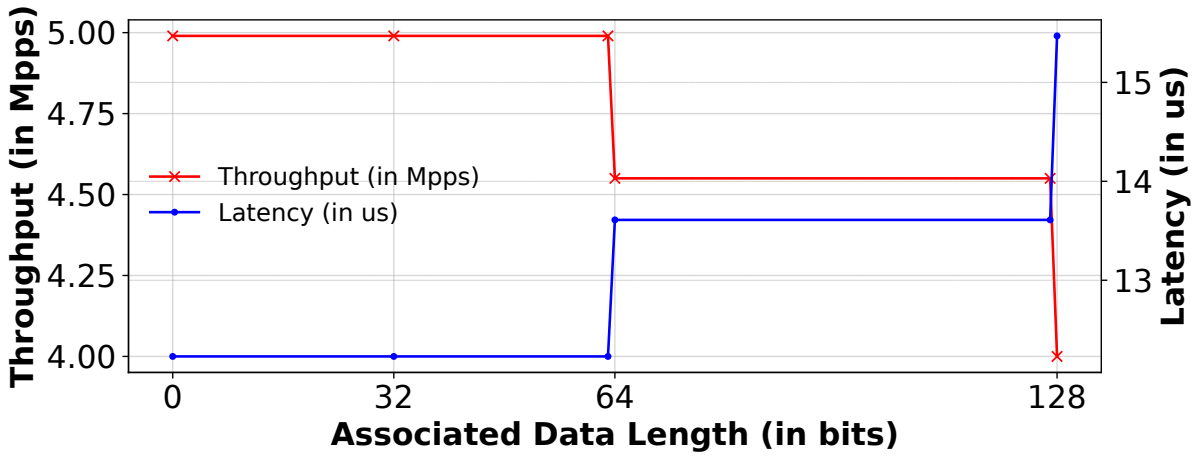


FIGURE 4.4: P4EAD throughput and latency on the Intel Tofino with **2** PERM rounds/pass with different Associated Data lengths. Assuming the header fields are of the same size as the maximum case and only one port for recirculation.

ciphertext length) from 64 bits(8 bytes) until 127 bits input and only decreases at 128 bit(16 bytes) input length.

This behaviour is inherent to ASCON in particular due to the inherent pad then encrypt nature of the algorithm. This trend is depicted in Fig. 4.3 for Tofino2 and Fig. 4.3 for Tofino1. This translates to lower throughput with atleast every 8-byte increase. Observing the overall trend from Fig. 4.1a, throughput reduces from ∼40 Mpps for an 8-byte input to ∼25 Mpps when the input is 32 bytes. A reverse trend is seen for the latency in Fig. 4.1b from 8.5μs to 12μs. A similar trend is observed for Tofino1.

Impact of associated data length: Similar to the case for increasing plaintext sizes, a similar trend of step-function-like trend throughputs and latencies follows with changing the associated data length, as depicted in Fig. 4.4. This is because, as explained in chapter 3, the number of P-RNDs increases for absorbing longer associated data. There is an increase of a PERM6 round after every 8-byte increase in AD length.

Impact of #recirculation ports: Here, we demonstrate how P4EAD scales to support higher throughput by increasing the number of dedicated recirculation ports (up to 8) in Fig. 4.5 using the 2 P-RND and 4 P-RND variants for Tofino and Tofino2, respectively. As more recirculation ports are used, the throughput increases linearly up to ∼320 Mpps for Tofino2.

However, in the case of Tofino2, when the dedicated recirculation ports all belong to the same pipeline (analogous to a CPU core), the throughput tapers off at ∼146 Mpps due to the pipeline being saturated at around 1.2 Bpps [15] with recirculated packets. This corresponds to the clock speed on the Tofino ASICs which is at around 1.2 GHz [23, 15]. Note that the throughput for Tofino with 8 ports is approximately the throughout for Tofino2 with only 1 port. By dedicating the entire Tofino pipeline with 16 ports (not shown in Fig. 4.5), the throughput saturates at ∼66

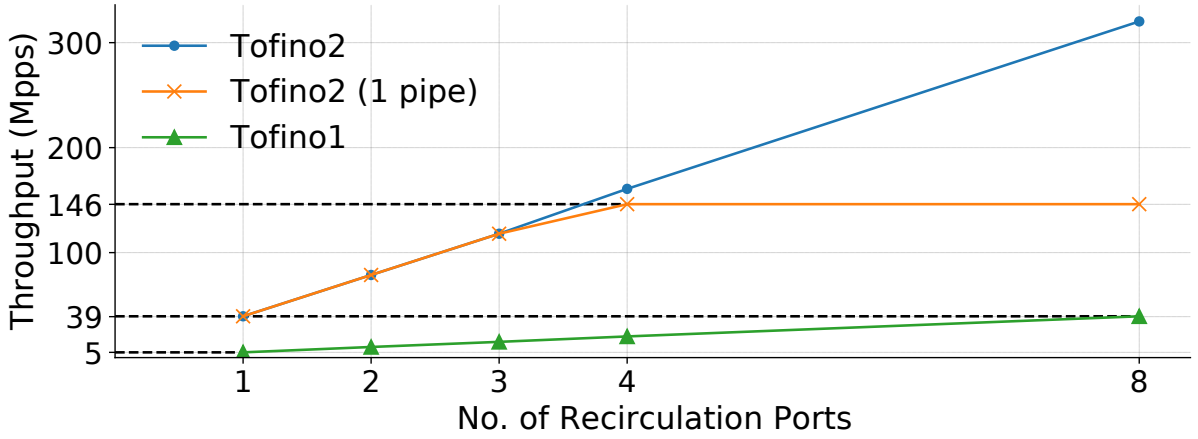


FIGURE 4.5: Impact of the number of dedicated recirculation ports on throughput.

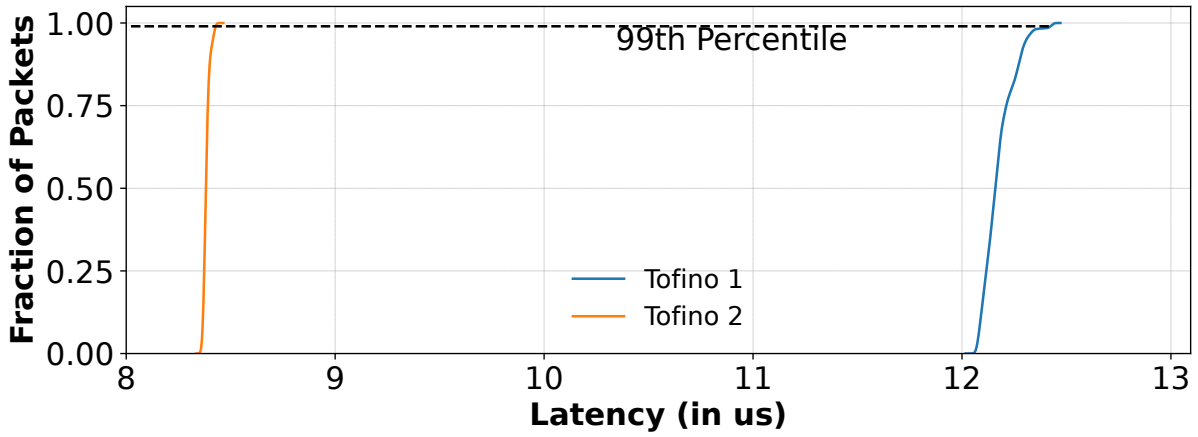


FIGURE 4.6: End-to-end latency of P4EAD on the Intel Tofino 1 and 2. We only dedicate one port for recirculation.

Mpps. If more switching pipeline is dedicated for P4EAD, it can even go beyond these values and almost always scale linearly.

In Fig. 4.5, we demonstrate how P4EAD can be scaled up to support higher throughputs by increasing the number of dedicated recirculation ports. While we observe that the throughput increases linearly with the number of recirculation ports dedicated, the throughput is eventually bounded by the packet processing capacity of the switching pipeline (analogous to a CPU core) – around 1.2 Bpps. By dedicating an entire pipeline, the maximum throughput achievable is ∼66 Mpps and ∼146 Mpps on Tofino and Tofino2, respectively.

As ASCON requires multiple pipeline passes to complete the rounds, the performance is thus dependent on the number of available ports dedicated for recirculation. Thus, if one requires greater throughput, it can be done by simply adding more recirculation ports.

Deterministic latency with short tail: On the computation time of P4EAD, we observe the minimum latency at the 99th percentile of 12.43μs and 8.41μs for Tofino and Tofino2, respectively.

There is an expected short tail due to the queuing in the traffic manager when almost saturating the recirculation ports. When compared to the traditional PCI-E control channel, the overhead of P4EAD can be considered negligible. The results are depicted in Fig. 4.6.

Chapter 5

Case Study: In-Network Key-Value Store

In this Chapter, we showcase the utility of P4EAD implemented alongside an existing application making use of in-band control signals. Though, in this case, the communication is between the central and local controllers, we showcase the minimal impact of P4EAD on system performance, thus, justifying a case for secure inter-switch communication. We evaluate the performance impact of P4EAD by integrating it with the publicly available implementation [13] of a control plane accelerated variant of NetCache [16]. NetCache maintains hot entries in the data plane to load balance the key-value store servers in order to speed up data lookups. As the workload can change rapidly in data centres, it is crucial that the *stale* key entries can be replaced swiftly to minimize the data key miss ratio, and thus there is a need for in-band control channels –

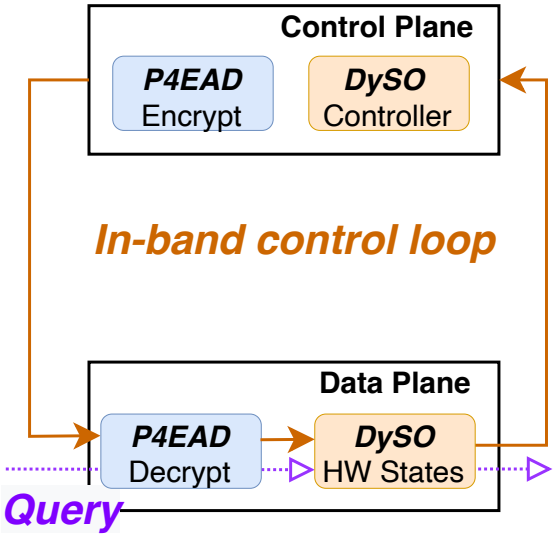


FIGURE 5.1: Evaluation setup for Dynamic state offload(DySO)

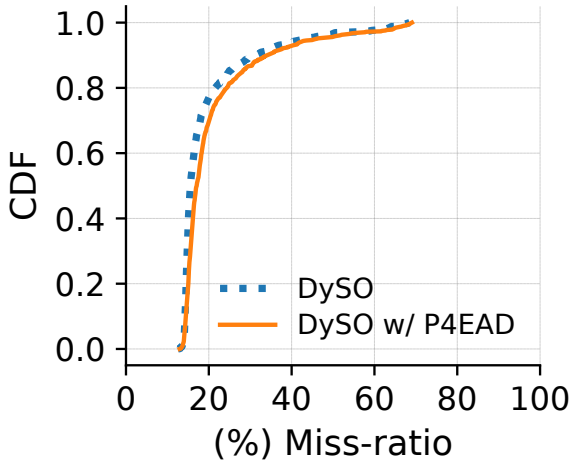


FIGURE 5.2: CDF of miss-ratio during 30ms after workload changes.

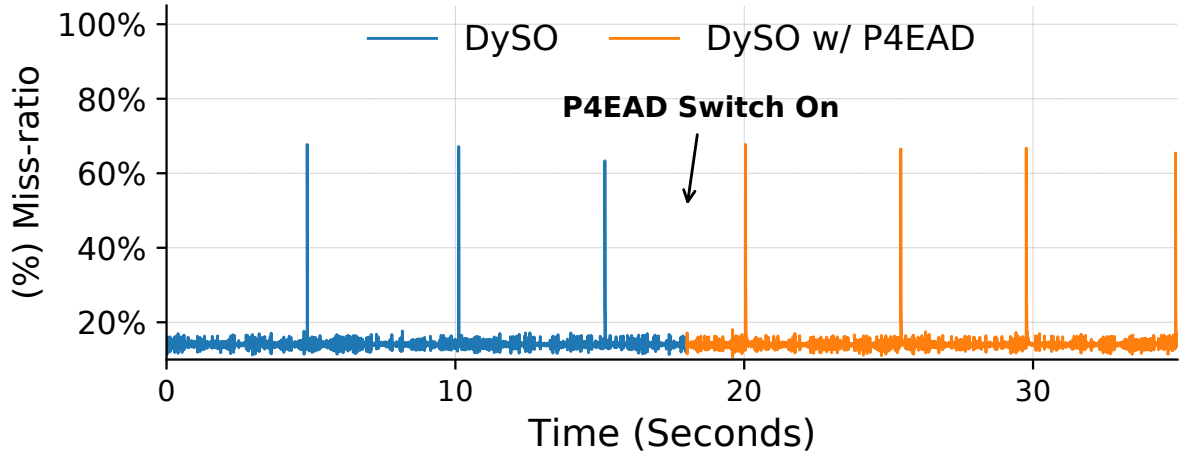


FIGURE 5.3: Time series of miss-ratio where P4EAD is on at 18s.We sample the cache miss-ratio every $1ms$.

DySO [28]. Here, we configure P4EAD with 1 P-RND per pipeline pass to run alongside the main program. We generate a key-value query stream with sudden workload changes, i.e., shifting unpopular items to top-rank popularity every 5 seconds for 100 seconds. The settings used are identical to the evaluation in [13]. The setup is depicted in Fig. 5.1.

Performance of in-network caching w/ and w/o P4EAD to workload changes is shown in Fig. 5.3. For this evaluation we sample the cache miss-ratio every $1ms$.

Thus we can conclude that despite P4EAD incurring additional latency, it is still comparable with the state-of-the-art [28] and remains orders of magnitude smaller than out-of-band control channels. This explains the minimal difference in terms of the cache-miss ratio with and without P4EAD in Fig. 5.2. This is an extremely justifiable tradeoff for securing overall in-band communication.

Chapter 6

Discussion and Future Work

Secure inter-switch communication: In this thesis, the focus was mainly on securing the in-band control channel, but we envision P4EAD to become a crucial aspect of emerging in-network applications including inter-switch communication. Generally, the control states and rules are forwarded through the control plane itself, but as networks evolve there is a need for better state synchronization and redundancy in case of failures. A secure inter-switch channel can help improve load balancing and prevent congestion with improved availability in case of any failure or disruptions. Since both the encryption and decryption processes are almost identical for ASCON, except for the tag verification part in decryption, a switch can be used to both send and receive encrypted messages from nearby switches using the shared data plane.

Associated Data alternatives for existing applications: Another interesting design decision is what fields in the clear text(un-encrypted part of the packet header) passed in a packet does one choose as the associated data(or basically the context) for authentication. Deciding a good context in general is the only way for one to prevent a replay/craft attack [25]. One can either have IP addresses or sequence numbers as the AD for most applications. But in cases where more security is desired, the sequence numbers can be synchronized using the dedicated control channel and updated frequently. This prevents a one-off chance for a craft attack by not leaking sequence numbers as a part of the clear text. Since AD can be of any length a combination of AD with a part of the clear text and sequence numbers will be secure against all kinds of malicious attacks.

Chapter 7

Conclusion

In this thesis, we present P4EAD, an implementation of ASCON-AEAD (for authentication with context) on a programmable switch for securing the emerging use of in-band communication with a focus on in-band control channels. We benchmark P4EAD on different configurations in terms of throughput and latency, as well as demonstrate its scalability. Finally, we implement P4EAD on top of an existing in-band control channel framework. We notice P4EAD does not incur noticeable overhead when integrated with the framework. Needless to say, it is much faster than a dedicated control channel while still being secure against malicious attacks including replay attacks. Beyond in-band control channels, we envision P4EAD to be a key building block in securing emerging in-network applications, e.g., inter-switch communications.

Appendix A

P4 Implementation on Tofino: Code

The code for the algorithm is attached. An excerpt of the P4 code for a P-RND is shown below
-

```
// Constant addition at the start of perm round using a table
action addition(bit<64> const_i) {
    hdr.ascon.s2 = hdr.ascon.s2 ^ const_i;
}
// Substitution layer
action substitution() {
    hdr.ascon.s0 = hdr.ascon.s0 ^ hdr.ascon.s4;
    hdr.ascon.s4 = hdr.ascon.s4 ^ hdr.ascon.s3;
    hdr.ascon.s2 = hdr.ascon.s2 ^ hdr.ascon.s1;
}

action start_sbox_0 () {
    meta.t0 = ~hdr.ascon.s1 & hdr.ascon.s2;
    meta.t1 = ~hdr.ascon.s2 & hdr.ascon.s3;
    meta.t2 = ~hdr.ascon.s3 & hdr.ascon.s4;
    meta.t3 = ~hdr.ascon.s4 & hdr.ascon.s0;
    meta.t4 = ~hdr.ascon.s0 & hdr.ascon.s1;
}

action start_sbox_1 () {
    meta.t0 = hdr.ascon.s0 ^ meta.t0;
    meta.t1 = hdr.ascon.s1 ^ meta.t1;
    meta.t2 = hdr.ascon.s2 ^ meta.t2;
    meta.t3 = hdr.ascon.s3 ^ meta.t3;
    meta.t4 = hdr.ascon.s4 ^ meta.t4;
}

action end_sbox() {
    meta.t1 = meta.t1 ^ meta.t0;
    meta.t0 = meta.t0 ^ meta.t4;
    meta.t3 = meta.t3 ^ meta.t2;
    meta.t2 = ~meta.t2;
}
```

```
table add_const{
    key={
        hdr.ascon.curr_round:exact;
    }
    actions= {
        addition();
        @defaultonly NoAction;
    }
    size=64;
    const entries ={
        0:addition(0xf0);
        1:addition(0xe1);
        2:addition(0xd2);
        3:addition(0xc3);
        4:addition(0xb4);
        5:addition(0xa5);
        6:addition(0x96);
        7:addition(0x87);
        8:addition(0x78);
        9:addition(0x69);
        10:addition(0x5a);
        11:addition(0x4b);
        ....
        ....
    }
}
```

```
apply{
    add_const.apply();
    substitution();
    start_sbox_0();
    start_sbox_1();
    end_sbox();

    // Linear Diffusion Layer
    // for hdr.s0
    @in_hash{hdr.ascon.s0[63:32] = meta.t0[63:32] ^ (meta.t0[18:0] ++ meta.t0[63:51]) ^ (meta.t0[27:0] ++ meta.
    @in_hash{hdr.ascon.s0[31:0] = meta.t0[31:0] ^ meta.t0[50:19] ^ meta.t0[59:28];}

    // for hdr.s1
    @in_hash{hdr.ascon.s1[63:32] = meta.t1[63:32] ^ (meta.t1[60:29]) ^ (meta.t1[38:7]);}
    @in_hash{hdr.ascon.s1[31:0] = meta.t1[31:0] ^ (meta.t1[28:0]++ meta.t1[63:61]) ^ (meta.t1[6:0]++ meta.t1[63

    // for hdr.s2
    @in_hash{hdr.ascon.s2[63:32] = meta.t2[63:32] ^ (meta.t2[0:0]++meta.t2[63:33]) ^ (meta.t2[5:0]++meta.t2[63:
    @in_hash{hdr.ascon.s2[31:0] = meta.t2[31:0] ^ meta.t2[32:1] ^ meta.t2[37:6];}

    // for hdr.s3
    @in_hash{hdr.ascon.s3[63:32] = meta.t3[63:32] ^ (meta.t3[9:0]++meta.t3[63:42]) ^ (meta.t3[16:0]++meta.t3[63
    @in_hash{hdr.ascon.s3[31:0] = meta.t3[31:0] ^ meta.t3[41:10]^ meta.t3[48:17];}

    // for hdr.s4
    @in_hash{hdr.ascon.s4[63:32] = meta.t4[63:32] ^ (meta.t4[6:0]++meta.t4[63:39]) ^ (meta.t4[40:9]);}
    @in_hash{hdr.ascon.s4[31:0] = meta.t4[31:0] ^ meta.t4[38:7] ^ meta.t4[8:0]++ meta.t4[63:41];}

}
```

Bibliography

[1] AbdelRahman Abdou, Paul C van Oorschot, and Tao Wan. "Comparative analysis of control plane security of SDN and conventional networks". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3542–3559.

[2] *Ascon - Lightweight Authenticated Encryption & Hashing*. https://github.com/ascon/ascon-c [Commit: ba61330]. 2023.

[3] *Authenticated encryption*. https://en.wikipedia.org/wiki/Authenticated_encryption.

[4] Jeffrey Avery et al. "Analysis of Practical Application of Lightweight Cryptographic Algorithm ASCON". In: (2022).

[5] Kevin Benton, L Jean Camp, and Chris Small. "OpenFlow vulnerability assessment". In: *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. 2013, pp. 151–152.

[6] Abhik Bose et al. "AccelUPF: accelerating the 5G user plane using programmable hardware". In: *Proceedings of the Symposium on SDN Research*. 2022, pp. 1–15.

[7] Pat Bosshart et al. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 99–110.

[8] Nishant Budhdev et al. "FSA: fronthaul slicing architecture for 5G using dataplane programmable switches". In: *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 2021, pp. 723–735.

[9] Xiaoqi Chen. "Implementing AES encryption on programmable switches via scrambled lookup tables". In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 8–14.

[10] Intel Corporation. *Intel Tofino*. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html [Accessed: Mar. 2023]. 2023.

[11] Christoph Dobraunig et al. "Ascon v1. 2: Lightweight authenticated encryption and hashing". In: *Journal of Cryptology* 34 (2021), pp. 1–42.

[12] Morris Dworkin(NIST). "Recommendation for Block Cipher Modes of Operation: Galois/-Counter Mode (GCM) and GMAC". In: (2007).

[13] *DySO P4*. https://github.com/dyso-project/dyso_p4. 2023.

[14] Kuo-Feng Hsu et al. "Contra: A programmable system for performance-aware routing". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 2020, pp. 701–721.

[15] *Intel® Tofino 2 12.8 Tbps, 20 stage, 4 pipelines*. https://www.intel.sg/content/www/xa/en/products/sku/218648/intel-tofino-2-12-8-tbps-20-stage-4-pipelines/specifications.html [Accessed: May 2023]. 2023.

[16] Xin Jin et al. "Netcache: Balancing key-value stores with fast in-network caching". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 121–136.

[17] Naga Katta et al. "Hula: Scalable load balancing using programmable data planes". In: *Proceedings of the Symposium on SDN Research*. 2016, pp. 1–12.

[18] Xin Zhe Khooi et al. "DIDA: Distributed In-Network Defense Architecture Against Amplified Reflection DDoS Attacks". In: *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. 2020, pp. 277–281.

[19] Robert MacDavid et al. "A p4-based 5g user plane function". In: *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*. 2021, pp. 162–168.

[20] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. June 2018. DOI: 10.17487/RFC8439. URL: https://www.rfc-editor.org/info/rfc8439.

[21] NIST. *Lightweight Cryptography Standardization Process: NIST Selects Ascon*. https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon. 2022.

[22] Tian Pan et al. "Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 194–206.

[23] Bosshart Pat. *Programmable Forwarding Planes at Terabit/s Speeds*. https://old.hotchips.org/hc30/2conf/2.02_Barefoot_Barefoot_Talk_at_HotChips_2018.pdf [Accessed: Mar. 2023]. 2018.

[24] *Programmable networks get a bigger foot in the datacenter door*. https://www.nextplatform.com/2018/12/04/programmable-networks-get-a-bigger-foot-in-the-datacenter-door/.

[25] *Replay attacks and anti-replay protocol*. https://www.techtarget.com/searchnetworking/definition/anti-replay-protocol.

[26] Phillip Rogaway. "Authenticated-encryption with associated-data". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 98–107.

[27] Yaron Sheffer, Peter Saint-Andre, and Thomas Fossati. *Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC 9325. Nov. 2022. DOI: 10.17487/RFC9325. URL: https://www.rfc-editor.org/info/rfc9325.

[28] Cha Hwan Song et al. "DySO: Enhancing application offload efficiency on programmable switches". In: *Computer Networks* 224 (2023), p. 109607. ISSN: 1389-1286.

[29] Tushar Swamy et al. "Taurus: a data plane architecture for per-packet ML". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022, pp. 1099–1114.

[30] Sophia Yoo and Xiaoqi Chen. "Secure keyed hashing on programmable switches". In: *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable network INfrastructure*. 2021, pp. 16–22.

[31] Yutaro Yoshinaka et al. "On implementing ChaCha on a programmable switch". In: *Proceedings of the 5th International Workshop on P4 in Europe*. 2022, pp. 15–18.

[32] Chaoliang Zeng et al. "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association. 2022, pp. 1345–1358.