2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) | 978-1-6654-3983-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/NFV-SDN53031.2021.9665012

dh-aes-p4: On-premise encryption and in-band key-exchange in P4 fully programmable data planes

Isaac Oliveira†, Emídio Neto*, Roger Immich*,
Ramon Fontes* and Augusto Neto‡
DIMAp‡/IMD*, UFRN and UERN†
Natal, Rio Grande do Norte, Brazil

Fabrício Rodriguez
and Christian Esteve Rothenberg
FEEC, UNICAMP
Campinas, Sao Paulo, Brazil

***Abstract*—Software-Defined Networking (SDN) fostered unprecedented advances over legacy networks by employing a central-logic control plane to coordinate data-plane nodes in a net-programmable manner. From the security view, control applications that run atop the SDN controller are in charge of establishing secure data-plane connections between pairs of data-plane forwarding nodes. The Diffie–Hellman (DH) is a widely used solution for cryptographic key exchange between endpoints. However, traditional DH implementations impose high computational costs and key management hazards, leading to issues in the SDN central-logic control plane. This paper introduces the `dh-aes-p4`, which tackles the penalties of legacy SDN security solutions by turning the data plane into fully programmable P4 nodes. The proposed solution allows P4-enabled data plane nodes to establish secure channels between each other. In doing that, it is possible to harness in-band DH key exchange with AES encryption, enclosing on-site features to generate keys dynamically and enforcing them autonomously and high-agile without SDN controller central-logic intervention. A prototype was designed to validate the feasibility and estimate performance impacts of `dh-aes-p4` concerning regular SDN central logic.**

I. Introduction

Cryptography is essential to delivering secure communication service between networked endpoints. However, most existing mechanisms operate on Layer 3 or above the network architecture (e.g., IPSec, TLS, and VPN), requiring operating system and/or application support. Lack of such support and faulty application designs or system administration errors eventually jeopardize the data protection and cause privacy issues, among other hazards.

The secure exchange of cryptographic keys, so as two networked endpoints can communicate securely over a public channel, stands to a well-known challenge in symmetric cryptography systems. In this context, Diffie-Hellman (DH) [1] is one of the most prominent, widely-used, and consolidated method to settle secure communications. Incidentally, the advent of Software-Defined Networking (SDN) [2] revolutionized the traditional networking scenario by opening up a plethora of new possibilities to implement and exercise new protocols and services flexibly. In legacy SDN, data plane nodes are introduced with a well-known southbound interface along with a fixed set of open functions, whereby a software-programmable SDN controller central-logic enforces new settings in data-plane devices. From the security point of view, the SDN controller allows atop-running control plane applications to (re)program the data-plane nodes to cater to secure communication paths. Unlikely, traditional security solutions rely on a network administrator to configure a set of running tools (e.g., firewall and Detection and Prevention Systems - IDS/IPS) statically, which is time-consuming.

However, the traditional DH cryptography key management approach is computational-costly (in terms of CPU and memory consumption), whereby the central-logic coordination of legacy SDN limits the system concerning time-response and complexity. We claim that decoupling DH facilities from the SDN controller central-logic is a viable strategy to achieve a performance-enriched and secure approach. The proposed idea is to introduce security control plane facilities for running at data plane node on-site premise. Furthermore, including security functions at fully programmable data-path nodes will fundamentally reduce the attack surface since private keys are kept in the embedding system memory, and they can be generated on a flow basis for a limited duration. To achieve this, the Programming Protocol-independent Packet Processors (P4) [3] paradigm is explored, which enables a non-centralized key management control plane. Notably, the biggest driver for P4 is possibly disaggregation. While currently, devices from different vendors can be orchestrated by a customized controller, P4 may have the potential to extend disaggregation towards specialized appliances based on off-the-shelf programmable hardware.

Inspired by the innovative power and rapid data plane prototyping possibilities that P4 technology offers we propose `dh-aes-p4`, the first DH key-exchange implementation with Advanced Encryption Standard (AES) encryption tailored to P4 targets. To demonstrate the feasibility of `dh-aes-p4`, a prototype was designed based on the Behavioral Model version 2 (`BMv2`)[1] P4 software target where cryptographic parameters (e.g., private key) can be generated through invoking native P4 functions. As a proof of concept, a simple to understand network topology was built, and the `dh-aes-p4` proposal was assessed through practical and fully reproducible experiments.

The reminder of this paper is organized as follows. Section 2 reviews the technical background in support of this work. Section 3, describes the related work on P4-based network security applications. A detailed description of `dh-aes-p4`

[1]https://github.com/p4lang/behavioral-model

is given in Section 4. Section 5, shows the validation settings of the prototype and reports the lessons learned. Finally, the conclusions can be found in Section 6.

II. Background

This section overviews the fundamentals of both AES and DH algorithms approach to protect data communications.

A. Advanced Encryption Standard

Advanced Encryption Standard (AES) constitutes a cryptographic system specified by the National Institute of Standards and Technology (NIST) in 2001 to protect sensitive data [4]. AES is widely used for information security by corporations, governments, and government agencies worldwide to secure and transmit data without fear of it being compromised. A set of research endeavors even indicate that the 256-bit version has been suggested to be quantum-safe [5], [6].

The AES algorithm is a symmetric block cipher that can process data blocks of 128 bits using three different cipher key lengths 128, 192, or 256 bits. AES algorithm is a typical substitution-permutation network encryption primitive based on symmetric block cipher structure. Depending on the key size, AES performs 10 to 14 encryption/decryption rounds where each round has your respective subkey. This subkey composed of 4-word (128-bits) is generated by **Key Expansion** where each round key is generated as a product of the previous round key. Each word is calculated by a constant that changes at each round and an S-Box, which will compose one of the 4-word (128-bits) set for each round. Encryption and decryption rounds of the AES (except the last round) consist of four steps:

(a) **SubBytes**. The first step where data in the plain text is substituted by pre-defined values from a substitution box (S-box) that is invertible;
(b) **ShiftRows**. Responsible for creating a simple permutation by cyclically shifting the table rows;
(c) **MixColumns**. Is the primary source of diffusion based on a matrix multiplication in Galois field using prime polynomial;
(d) **AddRoundKey**. Applies a simple bitwise XOR of the current block with a respective expanded key part in each round.

B. Diffie–Hellman

The DH key exchange protocol establishes a shared secret that can be used for secret communications by exchanging data over a public network. This key exchange can be used to support the subsequent encryption process required for communication between two hosts (e.g., with AES). According to [7], the DH algorithm uses two numbers that are publicly known: (i) a prime number q and (ii) an integer α, which is a primitive root of q. Thus, if Alice (A) and Bob (B) want to create a shared key, Alice selects a random integer $X_A < q$ and calculates $Y_A = \alpha^{X_A} \mod q$. Similarly, Bob selects a random integer independently $X_B < q$ and calculates $Y_B = \alpha^{X_B} \mod q$. In this way, each user turns X into a private value and makes Y the public value. Thus, X_A is Alice's private key, and Y_A is Alice's corresponding public key - the same for Bob. In this case, Alice needs to calculate the shared key as $K_A = Y_B^{X_A} \mod q$, and Bob calculates the same shared key-value through $K_B = Y_A^{X_B} \mod q$. This procedure produces identical results, allowing Alice and Bob to use their secret keys K_A and K_B for data encryption.

Although considered secure, some network devices such as optical devices may suffer from performance issues during the execution of the DH key exchange algorithm [8]. Hence, *dh-aes-p4* relies on a modified version of DH proposed by [8] that uses XOR operations instead of the arithmetic module used in the traditional DH algorithm. This modified version guarantees the same levels of security since operations are carried out with random numbers and never reused.

In summary, the modified DH workflow is as follows:

1) Alice and Bob make public two numbers G and P, where G and P are randomly generated;
2) Alice chooses a random number A and first calculates $G.A$ and $P.A$ by logical boolean AND operation;
3) Alice calculates a public key K_A by the XOR operation $G.A \oplus P.A$ and sends it to Bob;
4) Bob also chooses another random B and calculates $G.B$ and $P.B$ by logical boolean AND operation;
5) next, Bob calculates a public key K_B by the logical operation XOR $G.B \oplus P.B$ and sends it to Alice;
6) Alice first calculates $K_B.A$ by logical AND operation and calculates a secret key S_A by XOR operation with P by $S_A = K_B.A \oplus P = \{(G.B \oplus P.B).A\} \oplus P$;
7) Finally, Bob calculates $K_A.B$ by logical AND operation and calculates a secret key S_B by XOR operation with P in $S_B = K_A.B \oplus P = \{(G.A \oplus P.A)B\} \oplus P$.

In the end, Alice and Bob have the same shared secret key, where $S = S_A = S_B$. Both Alice and Bob store this shared secret key as a private key used in the cryptographic system.

The combined use of DH and AES bring to our proposal the ability to settle an insecure channel into a secure channel between a pair of communicating nodes with no prior knowledge of each other.

III. Related Work

It is notorious that programmable network nodes have sparked a lot of interest in academia and industry [9], [10]. In particular, the P4 language allows implementing network security applications that were not so accessible before, and a set of related works have been devoted to deploying programmable nodes to perform numerous security-related functions in the data plane (cf. Section XIII on network security in [10] and Section XI on cybersecurity in [9]). Noteworthy, most of the related works discussed next were recently published about a year ago, which shows the recent interest by the research community in using P4 for the data plane security-oriented implementations.

For example, [11] proposed a solution that allows a packet-header obfuscation system running entirely in the data plane of a programmable network node for clients sending and

receiving DNS packets securely. In [12], the authors proposed the first implementation of IPsec for P4-based nodes. However, these works feature layers above the data link layer, and they do not bring solutions for the automatic configuration of P4 switches, delegating the management of keys to an SDN controller. IPsec, in particular, still requires tunnel configuration; the encrypted traffic works with authentication, and it requires a central authority to introduce the configuration.

A technique for implementing AES through Scrambled Lookup Table [13] aiming to reduce the number of sequential arithmetic operations required for AES encryption. This was done by utilizing the table matching capability available on programmable switches. The author implemented AES on the Barefoot Tofino programmable switch and showed an experimental prototype evaluation. However, the code provided by the author does not include all the AES rounds for encryption, the rounds were not used in the decryption process, and there is no key agreement.

In P4-MACsec [14], the authors include AES-GCM encryption and decryption directly on P4 switches. However, the proposal requires an SDN controller, which is responsible for the MACsec setup. Another set of related work relates to the hardware implementations of DH and hardware-acceleration of security functions, an active research topic due to the high-performance and improved security compared to traditional software-only approaches. For example, [15] discusses scalable implementations of elliptic curve scalar multiplication and presents an implementation targeted to an FPGA device. In [16], the authors focused on facilitating the process of DH elliptic curve cryptography implementation in hardware. Modern SmartNICs commonly feature public key cryptographic engines[2] for hardware offloading of all asymmetric cryptographic operations. In contrast to related works, our proposal is based on the high-level P4 language and does not depend on target architecture-specific P4 extern functions.

IV. DH-AES-P4 Proposal Outlook

The `dh-aes-p4` proposal follows the [13] approach. However, `dh-aes-p4` differentiates itself among all elicited related works by the dynamic and in-band DH key-exchange approach. Hence, the data plane nodes themselves become capable of generating random private cryptographic keys. In contrast, the SDN Controller generates the cryptographic key to enforce targeting data-plane nodes afterward in legacy SDN systems. With `dh-aes-p4`, P4 functions themselves generate cryptographic parameters, such as the private key, according to the programmed behavior. Being data plane functions executed directly in the data plane hardware-premises pipeline reduces the computational cost of the cryptographic key generation methods and reduces the attack surface of the solutions. The reason is that the private cryptographic keys do not leave the system memory, and thus they can be generated on a traffic flow basis for a limited duration. Lowering the attack vectors is in alignment with the recommendation provided by the NIST [17] regarding the number of participating entities being proportionally related to an increased probability to expose the cryptographic key to non-authorized entities.

The main advantages of `dh-aes-p4` can be grouped into four technical benefits:

1) **Flow granularity** allows a flexible selection (e.g., using a rule-based on a pattern of packet header fields or metadata) of the traffic to be encrypted;
2) **Forward Secrecy** for each new flow, new private keys can be generated and held in device memory only during the duration of this flow;
3) **Opportunistic and endpoint transparency** based on edge-to-edge encryption methods to protect any data traffic without requiring specific pre-arrangements between the end-systems which remain unaware of the encryption service applied to their traffic;
4) **In-band distributed key management** between edge devices programmed to run an in-band security association exchange process and directly encrypt traffic edge-to-edge without additional entities (e.g., centralized key management servers).

A. Towards dh-aes-p4 Prototype Implementation

We implemented `dh-aes-p4` in a prototype, which is designed into three phases: (*i*) the DH key-exchange; (*ii*) AES encryption; and (*iii*) data transmission in a secure channel. Next sections provide details regarding the three phases.

1) Diffie-Hellman cryptographic key-exchange: As a fundamental part of our implementation, we initially considered the key exchange between two nodes (e.g., switches) based on DH. For this, we consider that a pair of nodes comprises an independent cryptographic system. Figure 1 illustrates how DH key-exchange works in our implementation. The procedures of creating the private, public, and secret keys proceed in a three-way handshake, and the steps are described below:

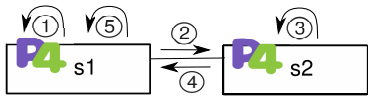

Figure 1: Diffie Hellman key-exchange.

1) the DH key-exchange may be triggered by either the number of packets or clock time. This makes `s1` to generate a random private key and the corresponding public key;
2) `s1` transmits its public key to `s2`;
3) `s2` receives and stores the s1's public key in a register. In addition, `s2` creates its private and public keys, and computes the secret key with the public key received from `s1`;
4) `s2` transmits its public key back to `s1`;
5) After receiving the `s2`'s public key, `s1` also computes the secret key and stores it in a register.

From now on, P4 nodes are ready to exchange packets securely once they are authenticated by themselves. `s1` trans-

[2]https://www.xilinx.com/products/intellectual-property/1-174jgch.html

mits the packet to `s2` and when `s2` receives the packet from `s1`, it can either forward the packet to an intermediate node or use the decryption module to send the decrypted packet to the destination host. In case of failures during the key-exchange process, two situations may occur: (i) the register is empty, and a new key must be generated before the traffic data exchanging; and a key is already inserted in the register, and this key will be used until replaced by a new one.

For `dh-aes-p4` setup on the P4 node, we use a simplified Python script that adds the corresponding entries into the table of the P4 processing pipeline.

2) AES encryption: The AES encryption starts with the insertion of the secret key generated by the DH algorithm. In our implementation, the DH algorithm generates a 256-bits secret key that can be used for AES 128/192/256. The transformations carried out by each AES round were combined in a set of lookup tables [18]. For this, we use eight (8) pre-computed lookup tables consisting of 256 4-byte word entries requiring only 8Kbyte of total space and two (2) S-box tables (one for encryption and another one for decryption).

When using lookup tables, the round transformation can be expressed as:

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j-c_1}] \oplus T_2[a_{2,j-c_2}] \oplus T_3[a_{3,j-c_3}] \oplus k_j \quad (1)$$

Where,

- e_j is one column of the round e output in terms of bytes of the round input a.
- $a_{i,j}$ concerns the byte of a in a row i and column j.
- a_j means the column j of the state a.
- k_j is the round key for encryption.
- T_i refers to the encryption lookup tables.

Implementing AES using lookup tables enables fast prototyping of the algorithm in P4 nodes since we can reduce all the round operations to a single transformation. However, when using lookup tables, the round keys for encryption and decryption are different. For computing the key schedule for decryption, we use the same S-box table and lookup tables used for decryption and the previously computed round keys for encryption. The first and last round keys for decryption are the same as the round keys used for encryption. Thus, we perform the decryption key schedule from the second round to the last but one. The key schedule derivation for decryption can be described as follows:

$$wk_i = Td_0[S[w_0]] \oplus Td_1[S[w_1]] \oplus Td_2[S[w_2]] \oplus Td_3[S[w_3]] \quad (2)$$

$$ik_j = wk_0 \frown wk_1 \frown wk_2 \frown wk_3 \quad (3)$$

Where,

- wk_i is a 32-bit word of the encryption round key k_j.
- ik_j is the round key for decryption.
- Td_i is the lookup table for decryption.
- S is the S-box for encryption.

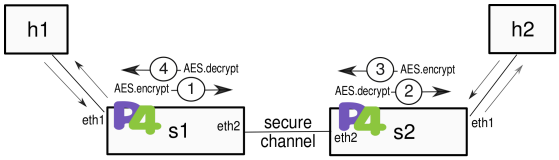

Figure 2: Data transmission.

- w_i is the 8-bit rotated word slice from the respective 32-bit word of the encryption round key.

3) Data transmission: The last but not least phase is about the secure communication between `h1` and `h2`. Considering the network topology illustrated in Figure 2 we can see that there is a secure channel between `s1` and `s2`. When `s1` receives the packet sent by `h1`, it forwards the packet to `s2` (1) and then the packet is delivered to `h2` (2). Eventually, `h2` will send the packet back to `h1` whether a response is required. For the sake of simplicity we create a custom header where `h1` sends a packet to `h2` in a one-way fashion.

Figure 3 summarizes all the steps we have seen so far through a detailed diagram of secure communication. Since our proposal also covers scenarios where SDN controllers can perform the key agreement, the diagram can be extended to represent new entities.

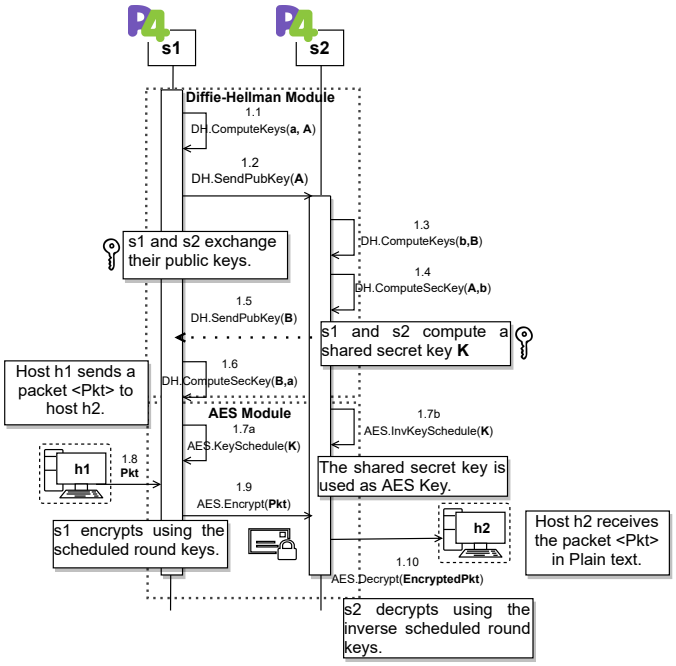

Figure 3: A detailed diagram of secure communication.

V. Towards Validation of the DH-AES-P4 Proposal

This section assesses, analyzes, and discusses the results obtained from `dh-aes-p4`. A set of lessons learned along with the development of this work is also presented. It is worth mentioning that this work is fully reproducible. All the instructions on how to reproduce the results as well as artifacts

(e.g., source code, dataset) can be found at the source code repository.[3]

A. Evaluation setup

The prototype has been evaluated in an emulated testbed with a fork of Mininet [19] and the `BMv2` P4 software switch (a P4-enabled data-plane node representation) with the same network topology that Figure 2 sketches. Evaluations were carried out on an Intel Xeon Silver 4114 CPU 2.20GHz (16 vCPUs), 32GB RAM, and Ubuntu Server (18.04) with Linux kernel 5.8. Next, we validate the algorithm implemented in `dh-aes-p4` and we evaluate the algorithm effectiveness in two metrics: (*i*) secret-key renewal RTT; and (*ii*) encryption time.

B. Algorithm validation

To validate `dh-aes-p4`, a custom packet was sent from `h1` to `h2` and the data traffic in `s1` and `s2` was captured. Figure 4 illustrates the payload extracted from the Ethernet interfaces. It is possible to observe that `h1` transmitted a packet with the "plain text" message in the payload. Additionally, it is clear that all the communication between `s1` and `s2` is completely safe, once the message has been encrypted to a protected message corresponding to the result of the AES algorithm.

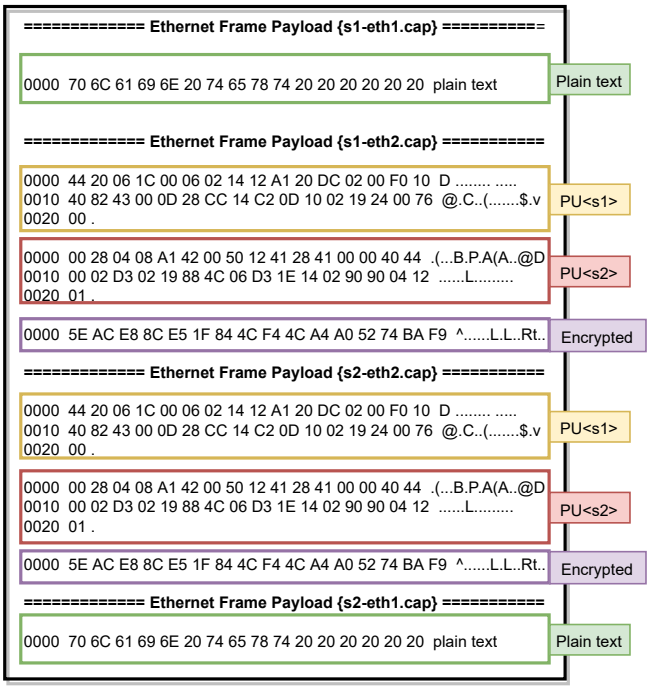

Figure 4: Payloads extracted from switches.

C. Secret-key renewal RTT

The secret-key renewal Round Trip Time (RTT) consists of verifying the time taken for switches to exchange new public keys and consequently generate new private and secret keys,

[3]https://github.com/reginalab/dh-aes-p4

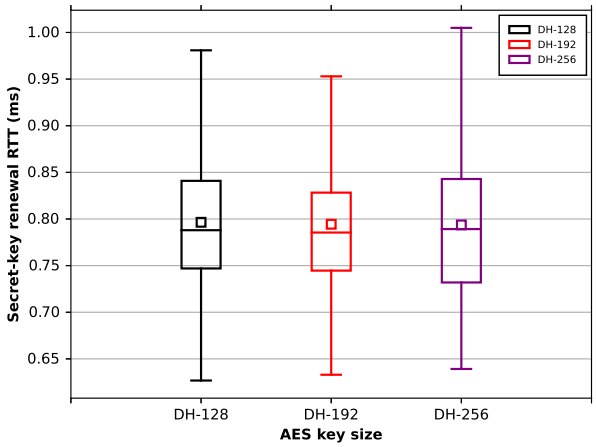

Figure 5: Secret-key renewal RTT

as a way to estimate how long does the DH algorithm takes to be fully accomplished. Figure 5 shown that AES-128/192/256 sizes produce similar results when a switch generates secret keys at random. This is expected for the reason that our implementation considers a 256 bits secret key size regardless of the AES-key size. Our experiments also show that pushing a static private key to switches produces a negligible reduction of the secret-key renewal RTT. In our implementation, the secret key generation at random is performed by the P4 *random()* function supported by the P4 v1model core library.

D. Encryption time

The encryption time stands for the time that an encryption algorithm takes to produce a ciphertext from a plain-text along with the total time taken to produce the plain-text from the cipher one, respectively. Figure 6 illustrates the encryption average time a switch takes to receive a data size of 30 bytes (14 Ethernet + 16 payload) and sends it out to the output port in two scenarios: (*i*) **controller-based**, where the controller is only responsible for generating the private keys; and (*ii*) **embedded** in-band key-exchange method, where the p4 switch is responsible for all the cryptographic system. As expected, results show (a) that the controller-based scenario is more costly than the embedded scenario, and (b) the time increases exponentially with the number of AES rounds. These results are due to the processing being realized out-of-the switch and the number of rounds computed by the different AES-key sizes, respectively.

E. Lessons learned

Accelerating computations by leveraging programmable switches is becoming a trend in data centers and backbone networks because of the benefits of increased flexibility and the advantages of customizing the data plane. However, this work has shown us that improvements are still needed in the programmability of P4 switches. According to the findings obtained during the realization of this work, the technical challenges are highlighted below.

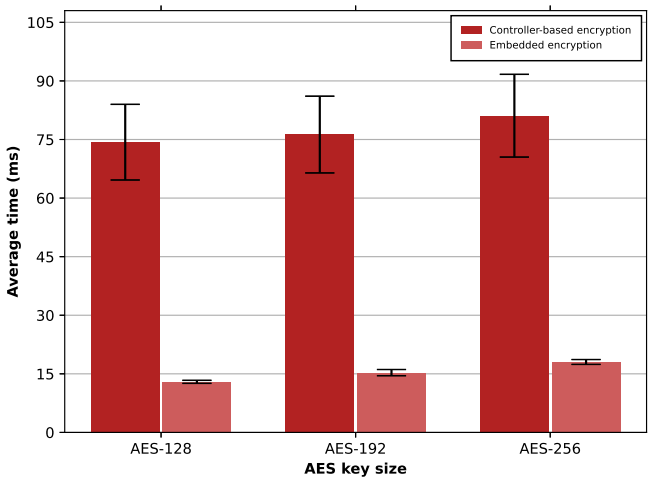

Figure 6: Encryption time

- **Random secret key**: it was noticed that the P4 *random()* function supported by the P4 core library does not produce a random value of 32 bytes since it is limited to 8 bytes. As a result, they had to concatenate 4 random blocks of 8 bytes to reach 256 bits for the secret key size.
- **Packet size**: `varbit` can be used within a packet header definition to indicate a field that has a length that can be different from one packet to another. This would be extremely important to enable the cryptographic module to modify packets with different sizes. However, according to the p4spec, it is not allowed to modify the `varbit` header nor cast it to the bit type [20].
- **Compilation time**: If all the necessary table entries were added (i.e., S-box and Lookup) as constants in the P4 file, the compilation time exceeds 30 hours. For this reason, a simple Python script was used to add all the corresponding P4 table entries (used as constants by the pipeline functions). Thus, the total time taken for compiling and adding table entries does not exceed 3 minutes.

Proposing improvements to the P4 programming language is not trivial, mainly because the P4 reference implementation is a complex system. Fortunately, proposals such as [21] can bring new possibilities, and in the future, we will possibly be discussing solutions to the challenges we encounter when developing this work.

VI. Conclusions and Future Work

This paper proposes `dh-aes-p4`, the first DH implementation tailored to a P4-based programmable data plane. The `dh-aes-p4` enables pairs of P4 nodes, each enclosing on-site features for dynamic keys generation and local enforcement, to establish secure channels among each other autonomously and high-agile harnessing to this end in-band DH key exchange with AES encryption strategy. We prototype and validate AES-128/192/256 encryption for the `BMv2` software switch in terms of secret-key renewal RTT and encryption time. Results suggest advantages when the cryptographic system is entirely implemented in a programmable data plane. We also report the lessons learned and experiences obtained from the features currently supported by the P4 programming language.

As future work, the findings from the validation outcomes indicate the need for optimizing and refining the `dh-aes-p4` prototype to draw new metrics for performance assessment. Regarding network security, we still envision adding authentication mechanisms in order to prevent man-in-the-middle attacks.

References

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[2] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.
[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, p. 87–95, July 2014.
[4] N.-F. Standard, "Announcing the advanced encryption standard (aes)," *Federal Information Processing Standards Publication*, vol. 197, no. 1-51, pp. 3–3, 2001.
[5] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum security analysis of aes," *IACR Transactions on Symmetric Cryptology*, no. 2, pp. 55–93, 2019.
[6] NIST, "Report on post-quantum cryptography." https://nvlpubs.nist.gov/nistpubs/ir/2016/nist.ir.8105.pdf, April 2016. (Accessed on 04/25/2021).
[7] W. Stallings, *Cryptography and Network Security: Principles and Practice*. USA: Prentice Hall Press, 6th ed., 2013.
[8] Jeon, Seok Hee and Gil, Sang-Keun, "Optical Secret Key Sharing Method Based on Diffie-Hellman Key Exchange Algorithm," *Journal of the Optical Society of Korea*, vol. 18, pp. 477–484, Oct. 2014.
[9] E. F. Kfoury, J. Crichigno, and E. Bou-Harb, "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends," 2021.
[10] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with p4: Fundamentals, advances, and applied research," *arXiv preprint arXiv:2101.10632*, 2021.
[11] L. Wang, H. Kim, P. Mittal, and J. Rexford, "Programmable in-network obfuscation of dns traffic," in *NDSS: DNS Privacy Workshop*, 2021.
[12] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-ipsec: Site-to-site and host-to-site vpn with ipsec in p4-based sdn," *IEEE Access*, vol. 8, pp. 139567–139586, 2020.
[13] X. Chen, "Implementing aes encryption on programmable switches via scrambled lookup tables," in *Proceedings of the Workshop on Secure Programmable Network Infrastructure*, pp. 8–14, 2020.
[14] 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. 58845–58858, 2020.
[15] Z. Sangma, "Hardware implementation of elliptic curve diffie-hellman key agreement scheme in gf (p)," 2008.
[16] D. Ionita and E. Simion, "Fpga offloading for diffie-hellman key exchange using elliptic curves," 01 2021.
[17] N. I. of Standards and Technology, "Recommendation for key establishment using symmetric block ciphers," Tech. Rep. National Institute of Standards and Technology, NIST Special Publication 800-71, U.S. Department of Commerce, 2018.
[18] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," 1999.
[19] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg, "Mininet-wifi: Emulating software-defined wireless networks," in *11th CNSM*, pp. 384–389, IEEE, 2015.
[20] T. P. L. Consortium, "P416 language specification." https://p4.org/p4-spec/docs/P4-16-v1.2.1.html, June 2020. (Accessed on 04/25/2021).
[21] R. Doenges, M. T. Arashloo, S. Bautista, A. Chang, N. Ni, S. Parkinson, R. Peterson, A. Solko-Breslin, A. Xu, and N. Foster, "Petr4: formal foundations for p4 data planes," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–32, 2021.