Poster: Toward an Optimal Implementation of ChaCha20-Poly1305 for SmartNICs

Kanta Tamura*, Yuki Koizumi*, Junji Takemasa*, and Toru Hasegawa†
*Graduate School of Information Science and Technology, Osaka University
†Faculty of Materials for Energey, Shimane University

***Abstract*—Secure communication relies heavily on authenticated encryption (AE). This poster implements ChaCha20-Poly1305, a widely-used AE scheme, on a smartNIC, which is particularly beneficial for high-speed networks. The key design rationale is manually optimizing data layout on hierarchical memory devices on the smartNIC. Our implementation achieves 5.67×10^6 and 0.58×10^6 packets/s for 64-byte and 1024-byte packets, respectively. As a use case of our AE implementation, we also implement a subset of TLS.**

***Index Terms*—SmartNIC, ChaCha20-Poly1305**

I. Introduction

Authenticated encryption (AE), also referred to as AE with associated data (AEAD), plays a vital role in secure communication, such as TLS, by providing both confidentiality and integrity simultaneously. Offloading AE operations to network infrastructure offers a promising solution for high-speed secure communication, as these operations typically incur significant computational overhead. Since smartNICs are strategically located on packet processing paths, implementing these operations on smartNICs rather than relying on external accelerators is particularly effective.

Several smartNICs, such as NVIDIA BlueField-3 [1], offer cryptographic engines. However, these engines are typically accessed via the controller of the smartNIC. In contrast, we directly implement an AE algorithm on a smartNIC, bypassing the need for indirect access through the controller. We implement ChaCha20-Poly1305, which combines the ChaCha20 stream cipher [2] with the Poly1305 [3] message authentication code (MAC) algorithm. ChaCha20-Poly1305 is one of the standard AE schemes for IETF protocols [4].

Although several studies have implemented ChaCha20 on a programmable data plane devices [5], [6], Poly1305 has not been implemented. Since Poly1305 requires more complex operations than ChaCha20, it can be challenging to realize Poly1305 on such devices. Additionally, the existing ChaCha implementation for a smartNIC was not optimized with consideration for the characteristics of the assumed smartNIC [5].

This poster makes two key contributions. First, we implemented full-fledged ChaCha20-Poly1305 on a smartNIC. Second, we present an approach to highly optimizing implementation of ChaCha20-Poly1305 for a smartNIC.

This work has been supported by NICT under the contract number 22401.

II. Design Rationales

Achieving optimal computation performance on a smartNIC generally requires thoughtful placement of data on memory devices of the smartNIC. SmartNICs have multiple memory devices with diverse capacities and access latencies. However, data placement on these devices is not automatically optimized by the smartNIC controller or compilers unlike computers. Moreover, to fully utilize low-latency but small memory devices, it is crucial to minimize memory footprint , thereby allowing as much data as possible to be stored on these devices.

In this poster, we assume a network flow processor (NFP) integrated into the Agilio CX smartNIC [7]. A schematic diagram of its memory architecture is depicted in Fig. 1. It has four types of memory devices: general purpose registers (GPRs), local memory (LM), cluster local scratch memory (CLS), and cluster target memory (CTM), of which specifications are as follows: GPRs (1 kB, 1.25 ns), LM (4 kB, 3.75 ns), CLS (64 kB, 25 ns), and CTM (256 kB, 62.5 ns). Since the NFP processors operate with 8 independent threads, each thread has access to only 1/8 of the memory capacity.

III. Implementation

ChaCha20-Poly1305 is an AE scheme that integrates ChaCha20 and Poly1305. Although the key used for Poly1305 is obtained from the keystream produced by ChaCha20, these algorithms can be implemented independently. Hence, we will describe each algorithm individually. Due to space limitations, our discussion will focus only on their outlines in terms of memory usage and data placement policies. The data placement in our implementation is summarized in Fig. 1 and an overview of our implementation is illustrated in Fig. 2.

1) ChaCha20: ChaCha20 is a stream cipher that generates a block of keystream from a given key. It uses this keystream to encrypt and decrypt plaintext and ciphertext by performing an XOR operation between the text and the keystream. A keystream block is generated by iteratively applying the quarter-round operation to a 512-bit state, which consists of a 128-bit constant, a 256-bit key, a 64-bit counter, and a 64-bit nonce. The quarter-round dominates the entire computation of ChaCha20 and consists of three operations: addition, XOR, and rotation. These are typically executed in a single cycle. If the data required for these operations is stored in slower memory than GPRs, it results in processor stalls. Therefore, our implementation places the data related to the quarter-round on GPRs to maintain efficiency.

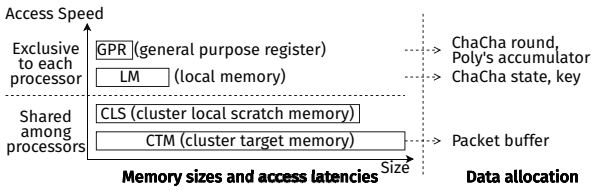

Fig. 1: NFP memory architecture

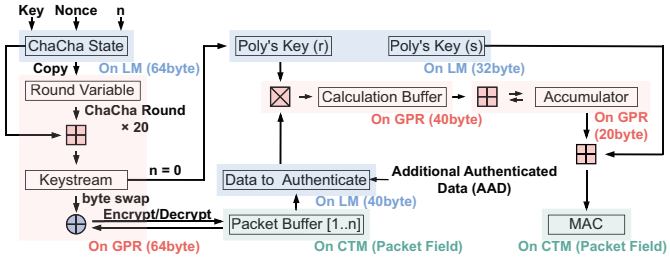

Fig. 2: Overview of our ChaCha20-Poly1305 implementation

Furthermore, ChaCha20 assumes little-endian data ordering, whereas the NFP adopts big-endian, requiring byte order swapping. However, performing byte order swapping at the packet buffer level (i.e., the CTM), as is common in standard ChaCha20 implementations, results in increased latency. To mitigate this issue, our implementation swaps the byte order of keystream blocks rather than packets, enabling encryption and decryption of packets without increasing latency.

2) Poly1305: Poly1305 takes two keys and uses polynomial evaluation along with the Wegman-Carter construction to generate an authentication tag that is robust enough to detect message tampering. The tag generation process involves iteratively performing modular arithmetic in a prime field, where each step multiplies 128-bit blocks of data by a key and adds the result to an accumulator. The second key is added in the final step to produce a 128-bit authentication tag. We leverage an efficient computation technique for modular arithmetic in this prime field, which has been previously developed [8]. Our implementation stores the most frequently accessed accumulator and computation buffers in the GPRs, and the key and preprocessed ciphertext for authentication in the LM. This configuration minimizes access latency, enabling high-speed authentication on the NFP.

Additionally, the memory footprint of Poly1305 exceeds the capacity of the GPRs and LM. Our implementation overcomes this problem by reusing variables that can be independently accessed during computation, allowing all non-packet buffer data to be stored in the faster GPR or LM.

IV. Performance Evaluation

1) Experimental Environment: This section evaluates the performance of our implementation on actual computers equipped with an Agilio CX NFP-4000 series smartNIC. Our implementation is available on GitHub [9]. It has 60 micro engines, which execute our program, and two 40-Gbps Ethernet ports.

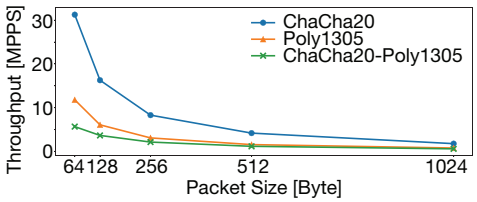

Fig. 3: Throughput of ChaCha20, Poly1305, and ChaCha20-Poly1305

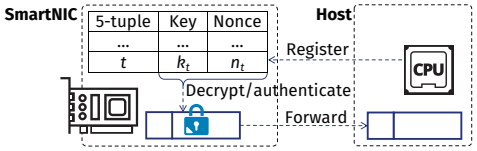

Fig. 4: Use case: TLS implementation

2) Results: Figure 3 indicates the individual throughput of ChaCha20 and Poly1305 as well as the combined throughput of ChaCha20-Poly1305 for different packet sizes. Our ChaCha20-Poly1305 implementation achieves 5.67 MPPS (million packet/s) for 64-byte packets and 0.58 MPPS for 1024-byte packets. This performance is equivalent to 4.78 Gbps for 1024-byte packets.

We compare the performance of our implementation with that of Kottur et al. [5]. For this purpose, we also implement ChaCha10, which applies a smaller number of quarter-round operations than ChaCha20. Our ChaCha10 implementation achieves 21.98 Gbps, which outperforms theirs by a factor of approximately 3.7.

V. Use Case

We implement a subset of TLS 1.3 [10] as a case study for our ChaCha20-Poly1305 implementation. Our TLS implementation stores the key and nonce shared during the handshake protocol in a table, where each entry is identified by a 5-tuple. The host CPU loads this table into a memory device, which is higher-capacity but slower than the CTM, on the smartNIC, and then the smartNIC replicates it in the LM. During the record protocol, our implementation retrieves the key and nonce using the 5-tuple, decrypts and authenticates the payload using the corresponding key and nonce, and forwards the decrypted payload to the host CPU. As this operation is performed on the packet processing path, it does not incur additional latency, thereby maintaining the processing performance evaluated in Section IV.

VI. Conclusion

This poster describes an optimized implementation of ChaCha20-Poly1305 for a smartNIC. Our implementation involves two key ideas: One is carefully designing data placement across the NIC's hierarchical memory architecture and the other is reducing the overall memory footprint. As a proof-of-concept, we apply this approach to a subset of TLS 1.3.

REFERENCES

[1] NVIDIA, "NVIDIA BlueField-3 networking platform." [Online]. Available: https://resources.nvidia.com/en-us-accelerated-networking-resource-library/datasheet-nvidia-bluefield

[2] D. J. Bernstein, "Chacha, a variant of salsa20," 2008. [Online]. Available: https://cr.yp.to/chacha/chacha-20080128.pdf

[3] ——, "The Poly1305-AES message-authentication code," in *Proceedings of Fast Software Encryption*, vol. 3557, 2005, pp. 32–49. [Online]. Available: https://iacr.org/archive/fse2005/35570033/35570033.pdf

[4] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," RFC 8439, Jun. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8439.txt

[5] S. Z. Kottur, K. Kadiyala, P. Tammana, and R. Shah, "Implementing ChaCha based crypto primitives on programmable smartNICs," in *Proceedings of ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures*, 2022, pp. 15–23.

[6] Y. Yoshinaka, J. Takemasa, Y. Koizumi, and T. Hasegawa, "On implementing chacha on a programmable switch," in *Proceedings of International Workshop on P4 in Europe*, 2022, pp. 15–18.

[7] "Netronome Agilio CX SmartNICs." [Online]. Available: https://netronome.com/agilio-smartnics/

[8] "Poly1305-donna." [Online]. Available: https://github.com/floodyberry/poly1305-donna

[9] K. Tamura, Y. Koizumi, J. Takemasa, and T. Hasegawa, "A ChaCha20-Poly1305 implementation for SmartNICs." [Online]. Available: https://github.com/Hasegawa-Laboratory/ChaCha20-Poly1305-SmartNIC

[10] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446.txt