# Secure and Efficient RDMA NIC Cryptography Offloading for Memory Disaggregation

WONSUP YOON*, The University of Texas at Austin, United States of America
SUE MOON, KAIST, Republic of Korea

Confidentiality and integrity are essential for secure communication. In RDMA-based systems, however, they are often neglected because security mechanisms impose significant performance overheads. In this work, we design a secure RDMA data path that ensures confidentiality and integrity by protecting data in transit and at the remote side using encryption and checksums. To reduce security overhead, our system offloads cryptographic operations to the RDMA NIC (RNIC). However, hardware limitations and relatively slow cryptography performance in RNICs make a secure and efficient design challenging. We address these challenges with three key techniques: dynamic key reconfiguration, configuration-data batching, and context pooling. We also conduct a case study with RDMA memory disaggregation systems to evaluate when offloading is beneficial. We find that naive offloading degrades performance and that overlapping CPU's computation with RNIC's cryptography is essential to realizing its benefits. Our evaluation shows that overlapping achieves up to 9.63× lower P99.9 latency than a CPU-based secure RDMA data path on RocksDB.

CCS Concepts: • **Networks** → **In-network processing**.

Additional Key Words and Phrases: RDMA, Cryptography, Memory disaggregation

**ACM Reference Format:**
Wonsup Yoon and Sue Moon. 2025. Secure and Efficient RDMA NIC Cryptography Offloading for Memory Disaggregation. *Proc. ACM Netw.* 3, CoNEXT4, Article 44 (December 2025), 13 pages. https://doi.org/10.1145/3768991

## 1 Introduction

While RDMA is widely used in modern networking systems, its security aspects have received little attention. As discussed in Rothenberger *et al.* [25], current RDMA systems offer only minimal security mechanisms in order to preserve high performance. For example, to shorten the data path to remote memory, most RDMA-based systems transmit data in plain text, which leads to potential information leaks in the network or at remote nodes. Encryption is the fundamental defense against information leakage, but due to its substantial performance overhead, RDMA-based systems have been largely avoiding it.

Meanwhile, hardware vendors have introduced new cryptography acceleration features on their products. NIC hardware vendors have released cryptography-capable RDMA NICs (RNICs) that perform cryptography in the network [2, 3, 8]. These RNICs accelerate core cryptographic tasks in network workloads, such as IPsec, TLS, and encrypted NVMe-over-Fabrics.

In this paper, we design a secure and efficient RDMA data path that leverages the state-of-the-art cryptography features in RNICs. For confidentiality, the data path employs AES-XTS to encrypt payloads before RDMA WRITEs to remote memory and to decrypt payloads upon RDMA READs.

*The work was done when Wonsup Yoon was a graduate student at KAIST.

Authors' Contact Information: Wonsup Yoon, The University of Texas at Austin, Austin, United States of America, wsyoon@utexas.edu; Sue Moon, KAIST, Daejeon, Republic of Korea, sbmoon@kaist.edu.

ACM 2834-5509/2025/12-ART44
https://doi.org/10.1145/3768991

We choose AES-XTS because it supports encryption parallelism and random access, which are essential features for applications that access remote memory in parallel and randomly. For integrity, we use CRC to protect the payload. By appending and verifying payload checksums, the system detects unintended modifications to encrypted data.

However, cryptographic offloading on RNICs is not straightforward and poses new challenges. First, RNIC hardware restrictions impose careful use of offloading. For example, AES-XTS requires per-remote-page tweak keys (supplementary keys to generate nonces) to defend against watermarking attacks that reveal plaintext through recurring ciphertext structures. This, in turn, demands configuring a large number of keys in the RNIC to support large remote memories. RNIC's SRAM size, however, is too small to store all tweak keys. To overcome this hardware restriction, we devise a dynamic cryptography key reconfiguration mechanism that reduces the number of tweak keys by appending a key-configuration request before each RDMA READ or WRITE. This approach reconfigures the existing tweak key on demand, allowing the RNIC to support large remote memories with a key. Also, we further improve the mechanism with configuration-data batching and cryptography context pooling, which reduce PCIe round-trips and tweak-key-creation costs, respectively.

The second challenge is that cryptography offloading is often slower than CPU execution. Modern CPUs include AES-NI/VAES and vector carry-less multiply (`VPCLMULQDQ`) instructions, enabling high-throughput AES and CRC. As a result, CPUs often outperform current RNICs on encryption tasks.

Our evaluation shows that RNIC-based cryptography is slower than CPU-based cryptography, based on a performance breakdown of the secure RDMA data paths. Thus, naively offloading cryptography to the RNIC reduces throughput and, in turn, increases latency by elongating the cryptography critical path.

How can the secure RDMA data path achieve performance gains by offloading? We conduct a case study with RDMA-based memory disaggregation systems, where compute nodes dynamically allocate and access memory from remote nodes via RDMA-based remote paging. We extend two existing memory disaggregation systems, DiLOS [38] and Adios [37], which employ two different paging mechanisms: busy-waiting and yielding, respectively. DiLOS performs busy-waiting: issuing a page fetch request and busy-waiting for its completion. Because of its simplicity and low latency, many memory disaggregation systems adopt busy-waiting [1, 14, 19, 24, 29, 35]. In contrast, Adios employs yielding for paging. Instead of continuously polling for RDMA completions, the CPU performs context-switching and schedules other work while the RNIC processes RDMA requests. This design prevents busy-waiting from blocking other requests, thereby improving throughput and tail latency.

We integrate the secure RDMA data paths into the memory disaggregation systems to evaluate their system-wide performance impacts. The microbenchmark reveals how to gain performance from RNIC offloading. In DiLOS, RNIC-based offloading offers lower throughput and higher P99.9 latency than CPU-based processing. In Adios, on the other hand, RNIC offloading achieves better P99.9 latency and throughput than CPU processing. These results show that naive offloading (*i.e.*, offloading with busy-waiting) degrades performance, and that overlapping is essential for efficient cryptography offloading.

To see application-level impact, we also evaluate cryptography offloading using two real-world applications running on the memory disaggregation systems: RocksDB [17] and FAISS [9]. Adios with RNIC offloading demonstrates 5.66× and 9.63× better P99.9 latency than DiLOS with RNIC offloading at 398 KRPS and Adios with CPU processing at 645 KRPS, respectively. These results are consistent with our microbenchmark findings. For FAISS, the performance improvement from offloading diminishes, as latency is primarily dominated by request processing in the microsecond-scale domain.

We summarize our key contributions as follows:

- We design and implement a secure and efficient RDMA data path that leverages RNIC-based cryptography acceleration, alongside a CPU-based secure RDMA data path for comparison.
- We propose three mechanisms for efficient RNIC cryptographic offloading: dynamic cryptography key reconfiguration, configuration-data batching, and cryptography context pooling.
- We conduct a performance breakdown of the RNIC-based and CPU-based secure RDMA data paths, revealing that naive RNIC offloading of cryptography degrades performance.
- Throughout a case study, we extend and compare two memory disaggregation systems (DiLOS [38] and Adios [37]) to use secure RDMA data paths.
- On evaluation with the memory disaggregation systems, we demonstrate that overlapping CPU's computation and RNIC's cryptography is essential to realizing offloading benefits.

## 2 Background

In this section, we introduce the cryptography used in this paper (AES-XTS and CRC) and offloading opportunities in RNIC.

**AES-XTS for confidentiality.** AES (Advanced Encryption Standard) is a standard symmetric block cipher most widely used across modern computing systems. To meet various requirements for encryption, AES supports several modes of operation, such as Galois/Counter Mode (GCM) for authenticated encryption with additional data (AEAD) and Cipher Block Chaining (CBC) for diffusion. XTS (XEX Tweakable block cipher with ciphertext Stealing) is a block cipher mode originally designed for storage devices [12]. In XTS mode, *tweak* values, which are usually derived from the sector number (LBA) in storage, are used to prevent watermarking attacks. When encrypting a block with a secret key, the tweak based on LBA is XOR-ed with both the input and output of the block cipher, producing different ciphertexts even for identical plaintexts. Moreover, since the XTS mode does not rely on the ciphertext of the previous block during encryption, it enables parallel processing and random access to sectors. These advantages are also beneficial to high-performance and random-access remote memory; thus, we use AES-XTS for secure RDMA in this work.

**CRC for integrity.** The AES-XTS cipher itself does not offer data integrity. Therefore, it needs an extra mechanism to mitigate data corruption or forgery. CRC (Cyclic Redundancy Check) is a viable option that balances security and performance in low-latency systems. By performing and encrypting data with its checksum, unintended modification of the ciphertext can be detected. Yet there are many other options for integrity, such as cryptographic hashes and public key cryptography, but they require higher computation power than CRC. Due to these reasons, many RDMA systems adopt CRC [13]; thus, we also use CRC to guarantee integrity without much overhead.

**RDMA NIC cryptography offloading.** Modern RNICs have included acceleration features that enable hardware-based encryption [2, 8, 15, 30, 31]. These features accelerate many encryption workloads in datacenters, from widely-used network security protocols, such as IPSec and TLS, to encrypted remote block storage (*e.g.*, iSCSI and NVMe-over-Fabrics). To use these features, DPDK [23] and vendor-specific libraries (*e.g.*, `libmlx5` [16] for NVIDIA RNICS) provide APIs for offloading. In this work, we select NVIDIA ConnectX-6 Dx as a representative RNIC that handles cryptography offloading and design RNIC-based secure memory disaggregation.

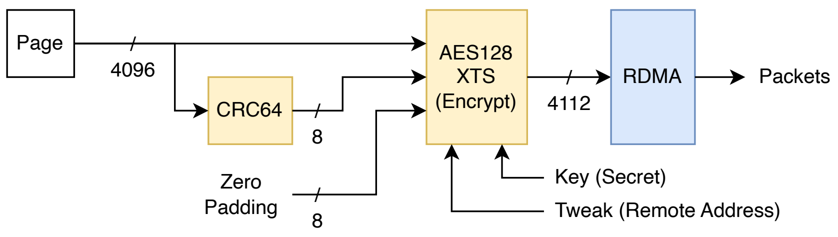

Fig. 1. Secure RDMA WRITE data path for a 4KB page. Yellow boxes are offloaded to RNIC.

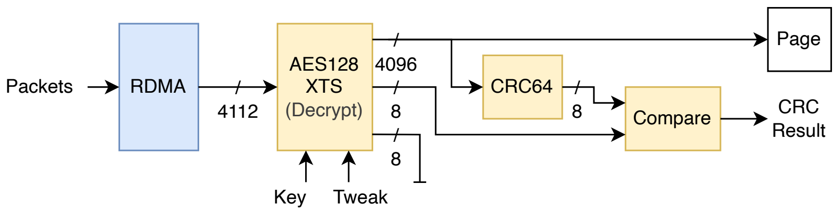

Fig. 2. Secure RDMA READ data path for a 4KB page. Yellow boxes are offloaded to RNIC.

## 3 Secure and Efficient RDMA Data Path

In this section, we present a secure RDMA data path that offloads cryptographic tasks to RNIC. The RDMA WRITE path encrypts and transmits data upon RDMA WRITE, and the RDMA READ path receives and decrypts data upon RDMA READ. During encryption and decryption, CRC checksums are used to offer integrity. We first illustrate the secure data path in an RNIC and highlight its hardware restrictions. To overcome these restrictions, we introduce three techniques: dynamic cryptography key reconfiguration, configuration-data batching, and cryptography context pooling. For CRC, we incorporate zero padding for data transmission and a host-side checksum handler.

### 3.1 Overview of RNIC Cryptography Offloading

Modern RNIC models support cryptography offloading for AES-XTS and CRC. By configuring the RNIC as represented in Figures 1 and 2, RDMA offers both confidentiality and integrity. In the WRITE data path (Figure 1), the RNIC first computes a CRC64 checksum over the 4 KB input page, then encrypts the data along with the checksum. Since the RNIC's AES-128-XTS hardware requires the input size to be a multiple of 16 bytes, we append 8 bytes of zero padding, resulting in a total input size of 4112 bytes. After encryption, the RNIC packetizes the ciphertext using the RDMA subsystem. The READ path (Figure 2) is similar to the WRITE path in reverse order: the RDMA subsystem first depacketizes the received data, then the AES-128-XTS engine decrypts it, and finally, the CRC64 subsystem recomputes the checksum. Additionally, the RNIC compares the CRC64 result with the checksum embedded in the decrypted data and reports the integrity verification result to the host.

To configure AES-XTS, RNICs require two cryptography keys (secret key and tweak). The secret key is a global key shared across all pages, and the tweak is a per-remote-page key to prevent watermarking attacks. However, storing all tweaks in RNIC is unrealistic due to its limited SRAM capacity. For example, supporting 128 GB of remote memory requires loading 32 million tweaks in the RNIC, far exceeding the several megabytes of the RNIC's limited on-chip SRAM.

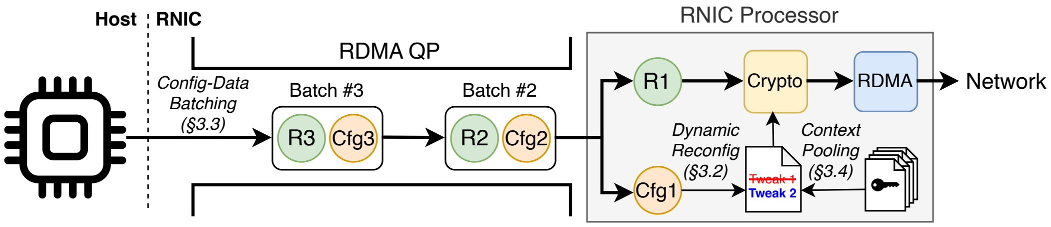

Fig. 3. Overview of key techniques used in data path.

## 3.2 Dynamic Cryptography Key Reconfiguration

To address the SRAM capacity limitation of RNIC's cryptography offloading, we devise a dynamic cryptography key reconfiguration technique. Instead of pre-configuring tweaks for all remote pages, our secure data path reconfigures and reuses an existing cryptography in RNIC by *posting a reconfiguration work-queue element (WQE)* that updates the cryptography context before the data WQE. Notably, the data path submits reconfiguration requests directly through the RDMA QP (Queue Pair). In other words, the reconfiguration process operates entirely in user space, without intervention from the kernel or VMM, avoiding costly mode switches and context switches. As a result, the cryptography key reconfiguration incurs minimal overhead on the host and has a negligible impact on I/O latency.

To further improve dynamic cryptography key reconfiguration, we introduce two additional techniques: configuration-data batching and cryptography context pooling. Figure 3 illustrates our design. In RNIC, the configuration (Cfg1) request changes the tweak used for the cryptography engine, and then the RDMA request is handled by the cryptography and RDMA modules.

## 3.3 Configuration-Data Batching

To configure cryptography keys in RNIC, the host must communicate with the RNIC over the PCIe channel. In a naive design, this additional communication introduces a PCIe round-trip delay for every RDMA request, approximately 900 nanoseconds [20], which accounts for 27% of 3.3 $\mu s$ RDMA READ latency. This overhead undermines the performance benefits of cryptography offloading and must be carefully mitigated.

To hide the latency introduced by configurations, we employ configuration-data batching. When the host submits configuration and data (READ/WRITE) requests to a QP, it chains a configuration request immediately before the corresponding data request. The host then rings the QP's doorbell register to submit both requests as a single batch. The RNIC processes the requests in order: first applying the configuration, then executing the data operation using the newly configured cryptography key. In Figure 3, the host submits Cfg2/R2 and Cfg3/R3 as batches #2 and #3, respectively.

Additionally, the batching mechanism leverages unsignaled requests to reduce the overhead associated with completion handling. Since RDMA with encryption involves two RDMA operations, two completions arise in the host. These redundant completions introduce additional PCIe transactions and increase the host's burden in handling completions. To mitigate this inefficiency, we mark configuration requests as unsignaled, suppressing their completions on the host. As the configuration operation is chained with the subsequent data request, the host infers its completion by polling the completion of the chained data request.

### 3.4 Cryptography Context Pooling

To perform cryptography, RNIC requires a cryptography context containing secret and tweak keys. During an encrypted RDMA request, the cryptography context must not be shared across concurrent requests to prevent races. Creating a new context for every request incurs significant overhead. Instead, we incorporate cryptography context pooling into our design to manage keys in RNIC. At system start-up, the system initializes pools of pre-created cryptography contexts. When issuing an encrypted RDMA request, the system allocates a context from the pool for exclusive use by that request. Each QP is provisioned with its own context pool, sized to match the QP's maximum outstanding request capacity, ensuring that all in-flight requests have access to a dedicated context.

## 4 Implementation

We implement our secure RDMA data path using NVIDIA's vendor-specific cryptography APIs [16], configuring the RNIC to perform cryptographic operations on RDMA payloads. NVIDIA RNICs offer multiple configuration options for when to encrypt and the order of encryption and checksum operations [11]. To configure RNIC as represented in Figures 1 and 2, we turn on `encrypt_on_tx` and `SIGNATURE_BEFORE_CRYPTO_ON_TX`[1] (configuration C in [11]).

NVIDIA RNICs manage cryptography contexts through MKey data structures, which contain configurations for memory layout, encryption, and checksum. The memory layout specifies which host memory regions the RNIC operates on, such as which pages to encrypt and where to place decrypted data. For encryption, the MKey has a secret key and a tweak parameter. We generate a random secret key shared across all MKeys and derive the tweak from the remote address of each request. For checksums, NVIDIA RNIC supports several CRC variants (CRC32, CRC32C, CRC64, and T10DIF) [10]. We select CRC64, which provides the largest checksum and thus the lowest collision probability among the variants.

On ConnectX-6 Dx, an NVIDIA's RNIC model, each MKey covers only a single encryption block (up to 4160 bytes) [21, 27]. Newer ConnectX-7 RNICs add support for multi-block encryption, but they only support incrementing the tweak key by one after encrypting each block. This auto-increment feature works only for sequential remote memory accesses, where the accessing remote address (used as the tweak key) increases linearly. For random access, however, the RNIC still requires a separate MKey with the correct tweak for the target remote address, necessitating dynamic reconfiguration.

### 4.1 Secure RDMA Data Path with CPU Instructions

For comparison, we build baseline systems using CPU cryptography instructions. We use Intel CPU's AES-NI and VAES (vector-extended version of AES-NI) instructions, the de facto standards for CPU-based encryption. For CRC64 calculation, we use the new VPCLMULQDQ instruction in Ice Lake architecture, which boosts checksum performance by SIMD processing of carry-less multiplication. In this design, the CPU handles the cryptographic logic (yellow boxes in Figures 1 and 2), while the RNIC data path (blue boxes) remains unchanged.

To implement CPU-based cryptography, we use libgcrypt [22] for AES-XTS and crc64-cxx [36] for CRC64 checksum. The libgcrypt library leverages AES-NI and, when available, VAES for acceleration. Our evaluation includes results from both AES-NI and VAES-enabled configurations (§5). Due to a lack of AVX-512 support in our case study systems described in §5.2, the VAES-based implementation relies on AVX2 only. Using wider vector instructions (AVX-512 [5] or AVX-10 [4]) would yield higher encryption throughput, and we leave this optimization to future work. We use libgcrypt 1.11, the most recent version available at the time of evaluation. We also have considered

[1]In NVIDIA RNICs, *signature* refers to the checksum.

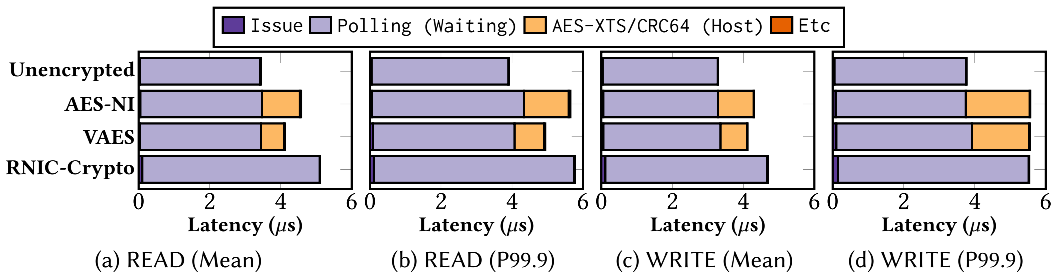

Fig. 4. Performance breakdown of AES-XTS-encrypted 4KB READ and WRITE. In RNIC-crypto, encryption and decryption are also handled in RNIC; thus `AES-XTS/CRC64` is omitted.

OpenSSL 3.5.0 for cryptography implementation, but we choose libgcrypt since OpenSSL supports only AVX-512 for VAES-based AES-XTS encryption, falling back to AES-NI in our systems. The crc64-cxx library supports AVX2-based SIMD processing of CRC64 computation, reporting up to 22× higher throughput than scalar implementations [36].

## 5 Evaluation

In this section, we evaluate the performance impact of our secure RDMA data path. Our evaluation has three goals: (1) to assess the performance of secure RDMA data paths using different cryptographic implementations (RNIC offloading vs. CPU-based computation), (2) to identify the performance impact on RDMA-based systems, and (3) to find an efficient mechanism for utilizing the secure RDMA data path. For (1), we compare RNIC-based encryption against CPU-computed baselines (AES-NI and VAES). For (2) and (3), we conduct a case study with memory disaggregation systems built on DiLOS [38] and Adios [37], which have two different remote paging mechanisms: busy-waiting and yielding, respectively.

**Testbed.** We configure a testbed comprising a compute node and a remote memory node. The compute node connects to the remote memory node via a 100 GbE Ethernet link. The computing and remote memory nodes have an Intel Xeon Gold 6330 CPU @ 2.00GHz, 256GB DDR4 3200 MHz memory, and an NVIDIA ConnectX-6 Dx 100GbE card (MCX623106AC-CDAT) that supports cryptography offloading. Both nodes run on Ubuntu 20.04 with Linux kernel 5.15 and NVIDIA OFED 5.8. For fair comparisons, all the systems under testing use 2 MB huge pages for remote memory nodes and 4 KB pages for compute nodes. In the case study, we use an additional load generator node emulating many clients. The load generator node runs the same open-loop load generator implementation in Yoon *et al.* [37], employing a Poisson process to generate loads. To avoid runtime and software overhead, the load generator sends pre-created requests and measures their latency through hardware timestamps in the NIC. This node uses Ubuntu 20.04 with Linux kernel 5.4 on a server with Intel Xeon Gold 6226R @ 2.90GHz, 384GB DDR4 2933MHz memory, and an NVIDIA ConnectX-6 Dx 100GbE card (MCX623106AN-CDAT).

### 5.1 Performance Breakdown

We first analyze the performance impacts of cryptography in RDMA by comparing four RDMA data paths: unencrypted (`Unencrypted`), encryption with AES-NI CPU instructions (`AES-NI`), encryption with VAES CPU instructions (`VAES`), and the secure RDMA data path with RNIC offloading (`RNIC-crypto`), as described in §3. We measure mean and tail latency of 4 KB RDMA READ and WRITE operations, which are typical operations in RDMA applications such as memory disaggregation and remote storage.

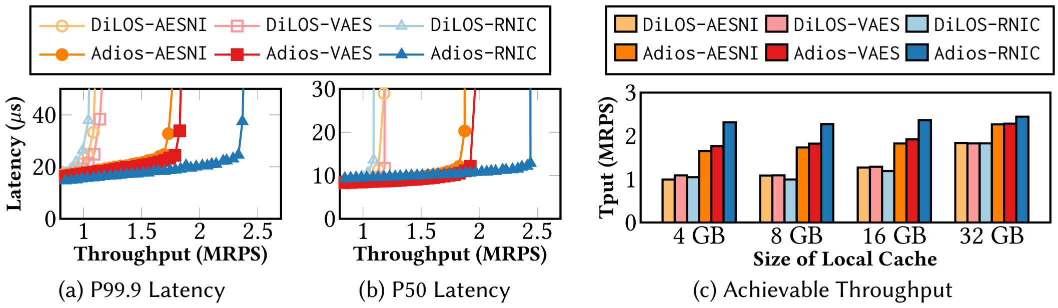

Fig. 5. Random array indirection microbenchmark results. The total array size in memory node is 40 GB. For latency experiments, the local cache size is 8 GB. Achievable throughput is the throughput where P99.9 latency is lower than 100 $\mu$s.

Figure 4 shows the performance breakdown across implementations. Overall, CPU-based encryption methods (AES-NI and VAES) exhibit slightly better latency than RNIC offloading. Especially, the VAES implementation achieves 23% and 13% better mean latency than RNIC for READ and WRITE, respectively. This result shows that cryptography processing is slower in RNICs than in CPUs. In turn, naively offloading cryptography to RNIC will result in slower performance.

## 5.2 Case Study with Memory Disaggregation Systems

In summary, the performance of cryptography offloading is slower than CPU processing. We conduct a case study with memory disaggregation systems to figure out the system-wide performance impact of NIC offloading and how to gain performance from offloading.

**Memory disaggregation systems.** One of the systems using RDMA is memory disaggregation. Memory disaggregation is a new approach that separates memory resources from compute nodes and pools them into memory nodes. Compute nodes dynamically allocate and utilize memory in memory nodes through a fast network, such as RDMA. To offer transparent abstraction, remote paging, which evicts unused pages to memory nodes and fetches them on use, is a widely-used design [1, 14, 19, 24, 29, 35, 37, 38].

The paging-based memory disaggregation systems typically employ one of two strategies: busy-waiting and yielding. In busy-waiting, the host continuously polls for RDMA completion after issuing a request. Due to its simplicity and solid performance, many systems adopt this approach [1, 14, 19, 24, 29, 35, 38]. In contrast, the yielding strategy allows the host to relinquish the CPU and schedule other work while the RNIC processes RDMA requests. Adios demonstrates that when paired with an efficient scheduler and thread management, yielding achieves lower tail latency and higher throughput compared to busy-waiting [37].

For this case study, we choose DiLOS [38] and Adios [37], which represent busy-waiting and yielding, respectively. We extend both systems to use our secure RDMA data path. In Adios, CPU computation overlaps with RNIC cryptographic processing. During page fetching, the host issues RDMA fetch and decryption requests to the RNIC, then yields to execute other tasks while the data is being fetched and decrypted.

**Legends.** We evaluate combinations of AES-NI, VAES, and RNIC-based encryption with DiLOS and Adios, resulting in six configurations: `DiLOS-AESNI`, `DiLOS-VAES`, `DiLOS-RNIC`, `Adios-AESNI`, `Adios-VAES`, and `Adios-RNIC`.

**Raw Performance.** To measure the system-wide performance impact of our secure RDMA data path on both systems, we evaluate their raw performances using a random array indirection

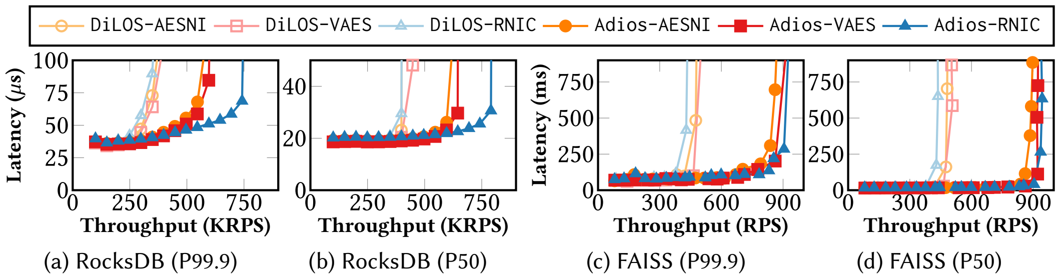

Fig. 6. P99.9 and median latency for RocksDB (99% GET and 1% of SCAN; latency from GETs only) and FAISS. The total sizes of memory used in the memory node are about 40 GB and 48 GB; the local cache sizes for compute nodes are 8 GB and 10 GB, respectively.

microbenchmark. The memory node has a 40 GB integer array, and the compute node has 8 GB of cache for the array. When the compute node accesses an entry of the array, if it is not cached locally, a page fault occurs, and a page containing the entry is fetched through the secure RDMA data path. The load generator creates and sends concurrent requests containing a random array index, and the compute node serves the requests through indirection of the 40 GB integer array.

We measure the P99.9 latency, median latency, and achievable throughput across varying local cache sizes. Figures 5(a) and 5(b) show the measured latency of the evaluated systems. On DiLOS, the VAES-based system (`DiLOS-VAES`) slightly outperforms the RNIC-based one (`DiLOS-RNIC`). In contrast, on Adios, the RNIC-based system (`Adios-RNIC`) significantly outperforms both the AES-NI-based (`Adios-AESNI`) and VAES-based (`Adios-VAES`) systems. On comparison of achievable throughput (Figure 5(c)), results show a similar conclusion: RNIC offloading is effective only in Adios; on DiLOS, it results in a slight throughput degradation. These results show that systems that overlap host computation and cryptography gain performance from RNIC offloading by freeing up host CPU cycles. In particular, as shown in Figure 4(a), the latency gap between unencrypted READs and RNIC-crypto-encrypted READs is around 1 $\mu$s. Adios uses this 1 $\mu$s to serve other requests, whereas DiLOS wastes the same duration in busy-waiting.

To evaluate the system-wide performance impacts on real applications, we also conduct studies using two popular applications: a key-value store and a vector search engine.

**Key-value store.** For the key-value store, we select RocksDB [17] as a representative system. A RocksDB server runs on the compute node connected to the memory node and is populated with approximately 40 GB of key-value data. Similar to the previous experiment, the memory node has the whole 40 GB of data, and the compute node caches 8 GB of the data locally. We use randomly generated key-value pairs, with keys of 50 bytes and values of 1024 bytes. At start-up, the server populates these key-value pairs, and the load generator issues requests by selecting keys from the pre-populated set. On this configuration, we measure P99.9 and median latency of a workload that consists of 99% of GET and 1% of SCAN(100) requests. Figures 6(a) and 6(b) show the latency results of GET requests. The overall trends are consistent with those observed in the microbenchmark: cryptography offloading is effective only on Adios. These results also support that the secure RDMA data path necessitates system designs that overlap computation and cryptography to gain performance.

**In-memory vector database.** We also conduct an experiment with the FAISS vector database [9], which represents complex and millisecond-scale systems. To load and query the systems, we generate a 48 GB workload by sampling the BIGANN dataset [18], and the systems have a 10 GB local cache (approximately 10% of the sampled dataset size). As shown in Figures 6(c) and 6(d), the performance differences between `Adios-VAES` and `Adios-RNIC` are minimal. This is because the benefit of overlapping computation and encryption is relatively small in millisecond-scale workloads, where request processing time dominates overall latency. As a result, the number of requests concurrently handled by each worker, which is the primary limiting factor for achievable throughput, remains consistent across encryption methods, leading to only marginal performance differences.

## 6 Discussion and Limitations

**Support for other RNIC vendors.** We have built our system using vendor-specific cryptography features of NVIDIA's RNICs, which are one of the only off-the-shelf RNICs that provide AES-XTS-encrypted RDMA. However, if the other RNICs have similar APIs and performance behavior for cryptography offloading, our design will also be applicable to them.

**Other offloading opportunities.** Intel has introduced QAT (QuickAssist Technology) [6] and TME-MK (Total Memory Encryption-Multi-Key) [7] to accelerate encryption workloads. QAT accelerators are devices that handle many cryptography and compression workloads. They were initially available as separate PCI-E cards or chipsets, but they have recently been integrated into the CPUs. However, the off-chip QAT (PCIe cards or chipsets) introduces high round-trip latency due to the overhead of the PCIe or DMI buses. To mitigate this latency, techniques such as yielding would be necessary. Even with CPU-integrated QAT, which has shorter latency, overlapping computation with encryption would be essential to fully realize the performance benefits of QAT-based acceleration. TME-MK is an encryption feature integrated into the MMUs in modern Intel Xeon processors. Its primary goal is to mitigate cold boot attacks by encrypting data before going to main memory, but researchers have proposed other use cases, such as memory tagging [28] and fine-grained in-process isolation [32]. Using TME-MK, an alternative secure RDMA design is possible. If the CPU and RNIC map the same physical page but encrypt it with different keys, data written by one side remains hidden from the other. However, current Intel processors do not ensure cache coherency under such configurations, necessitating a software-based coherence protocol. This limitation can incur significant performance overhead from software intervention and frequent cache invalidations. We leave the exploration of secure and efficient RDMA data paths leveraging QAT or TME-MK to future work.

**Stronger Integrity Guarantee.** CRC is an integrity mechanism that protects data against accidental data loss. In general, cryptographic hash functions such as SHA-256 guarantee stronger integrity to prevent tampering attacks. We prevent such attacks by encrypting the checksum along with the data; any attempt to make a checksum collision results in wrong decryption and checksum failure.

## 7 Related Work

**Secure RDMA.** There have been several works on security in RDMA systems. sRDMA demonstrates RDMA encryption offloading on ARM-based programmable network adapters [31]. ReDMArk demonstrates vulnerabilities and their exploits in RDMA-based systems [25]. 1RMA is Google's RDMA implementation suited for multi-tenant environments. 1RMA NICs have AES-GCM encryption blocks that encrypt RDMA packets at line rate [30]. RoCE BALBOA is an ongoing project that incorporates security features such as encryption on FPGA NICs [15]. In this work, we have designed a secure RDMA data path based on a commodity RNIC. However, the lesson learned from this work (*e.g.*, computation-encryption overlapping) would also be applicable to the RNIC implementations with similar designs.

**Memory disaggregation systems.** Many researchers have proposed memory disaggregation systems. Paging-based memory disaggregation systems employ virtual memory and paging to transparently support existing unmodified applications [1, 14, 19, 24, 29, 35, 37, 38], and library-based systems leverage application semantics to further improve their performance [26, 33, 34, 39]. This paper focuses on the security of paging-based memory disaggregation systems, yet our contribution is orthogonal to the design choices. The library-based systems can also improve their security using our findings.

## 8 Conclusion

In this paper, we design and implement a secure RDMA data path that efficiently offloads cryptographic tasks to RNICs. The data path features efficient offloading techniques, such as dynamic cryptography key reconfiguration, configuration-data batching, and cryptography context pooling. In evaluation, we reveal that naive offloading results in performance degradation. Throughout a case study with state-of-the-art RDMA memory disaggregation systems, we conclude that overlapping computation and cryptography is essential for a secure and efficient RDMA data path.

## Acknowledgments

We thank our shepherd and anonymous reviewers for helpful comments. This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT). (No.RS-2022-NR070293)

## References

[1] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can Far Memory Improve Job Throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 16 pages. doi:10.1145/3342195.3387522

[2] Broadcom. 2024. BCM57608 Ethernet NIC Adapters. https://docs.broadcom.com/doc/957608-PB1.

[3] Chelsio. 2025. S72200. https://www.chelsio.com/wp-content/uploads/2025/01/T7/Product-Brief/S72200-pb.pdf.

[4] Intel Corporation. 2025. Intel® Advanced Vector Extensions 10.1 (Intel® AVX10.1) Architecture Specification). https://www.intel.com/content/www/us/en/content-details/848455/intel-advanced-vector-extensions-10-1-intel-avx10-1-architecture-specification.html.

[5] Intel Corporation. [n. d.]. Intel® Advanced Vector Extensions 512 (Intel® AVX-512). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/advanced-vector-extensions-512.html.

[6] Intel Corporation. [n. d.]. Intel® QuickAssist Technology (Intel® QAT). https://www.intel.com/content/www/us/en/architecture-and-technology/intel-quick-assist-technology-overview.html.

[7] Intel Corporation. [n. d.]. Runtime Encryption of Memory with Intel® Total Memory Encryption–Multi-Key (Intel® TME-MK). https://www.intel.com/content/www/us/en/developer/articles/news/runtime-encryption-of-memory-with-intel-tme-mk.html.

[8] NVIDIA Corporation. 2021. NVIDIA CONNECTX-6 DX Ethernet SmartNIC. https://www.nvidia.com/content/dam/en-zz/Solutions/networking/ethernet-adapters/connectX-6-dx-datasheet.pdf.

[9] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[10] Oren Duer and Sergey Gorenko. 2023. mlx5dv_wr_set_mkey_sig_block - Configure a MKEY for block signature (data integrity) operation. https://manpages.debian.org/bookworm/libibverbs-dev/mlx5dv_wr_set_mkey_sig_block.3.en.html.

[11] Oren Duer, Avihai Horon, and Maher Sanalla. 2023. mlx5dv_wr_set_mkey_crypto - Configure a MKey for crypto operation. https://manpages.debian.org/bookworm/libibverbs-dev/mlx5dv_wr_set_mkey_crypto.3.en.html.

[12] Morris Dworkin. 2010. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=904691

[13] NVM Express. 2021. NVM Express over Fabrics Revision 1.1a. https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1a-2021.07.12-Ratified.pdf.

[14] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI*

*17)*. USENIX Association, Boston, MA, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[15] Maximilian J. Heer, Benjamin Ramhorst, Jonas Dann, and Gustavo Alonso. 2025. RoCE BALBOA: Towards FPGA-enhanced RDMA. Poster presented at the European Conference on Computer Systems (EuroSys)'25'. https://your-poster-link-if-any.com Poster presentation.

[16] Avihai Horon. 2023. mlx5dv_crypto_login - Creates a crypto login session. https://manpages.debian.org/bookworm/libibverbs-dev/mlx5dv_crypto_login.3.en.html.

[17] Meta Platforms Inc. 2022. RocksDB. https://rocksdb.org.

[18] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 861–864. doi:10.1109/ICASSP.2011.5946540

[19] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 843–857. https://www.usenix.org/conference/atc20/presentation/al-maruf

[20] Rolf Neugebauer, Gianni Antichi, José Fernando Zazo, Yury Audzevich, Sergio López-Buedo, and Andrew W. Moore. 2018. Understanding PCIe performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 327–341. doi:10.1145/3230543.3230560

[21] NVIDIA. 2023. Linux Kernel Upstream Release Notes v6.1. https://docs.nvidia.com/networking/display/kernelupstreamv61/changes+and+new+features.

[22] The GnuPG Projec. [n. d.]. Libgcrypt. https://gnupg.org/software/libgcrypt/index.html.

[23] DPDK Project. [n. d.]. NVIDIA MLX5 Crypto Driver. https://doc.dpdk.org/guides/cryptodevs/mlx5.html.

[24] Yifan Qiao, Chenxi Wang, Zhenyuan Ruan, Adam Belay, Qingda Lu, Yiying Zhang, Miryung Kim, and Guoqing Harry Xu. 2023. Hermit: Low-Latency, High-Throughput, and Transparent Remote Memory via Feedback-Directed Asynchrony. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 181–198. https://www.usenix.org/conference/nsdi23/presentation/qiao

[25] Benjamin Rothenberger, Konstantin Taranov, Adrian Perrig, and Torsten Hoefler. 2021. ReDMArk: Bypassing RDMA Security Mechanisms. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 4277–4292. https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger

[26] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 315–332. https://www.usenix.org/conference/osdi20/presentation/ruan

[27] Maher Sanalla. 2022. mlx5: Add new AES-XTS single block capability. https://github.com/linux-rdma/rdma-core/commit/bca03d4adc1f40baea7aaa2cd9d03834fb41fdce.

[28] David Schrammel, Martin Unterguggenberger, Lukas Lamster, Salmin Sultana, Karanvir Grewal, Michael LeMay, David M. Durham, and Stefan Mangard. 2024. Memory Tagging using Cryptographic Integrity on Commodity x86 CPUs. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. 311–326. doi:10.1109/EuroSP60621.2024.00024

[29] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 69–87. https://www.usenix.org/conference/osdi18/presentation/shan

[30] Arjun Singhvi, Aditya Akella, Dan Gibson, Thomas F. Wenisch, Monica Wong-Chan, Sean Clark, Milo M. K. Martin, Moray McLaren, Prashant Chandra, Rob Cauble, Hassan M. G. Wassel, Behnam Montazeri, Simon L. Sabato, Joel Scherpelz, and Amin Vahdat. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 708–721. doi:10.1145/3387514.3405897

[31] Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler. 2020. sRDMA – Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 691–704. https://www.usenix.org/conference/atc20/presentation/taranov

[32] Martin Unterguggenberger, Lukas Lamster, David Schrammel, Martin Schwarzl, and Stefan Mangard. 2025. TME-Box: Scalable In-Process Isolation through Intel TME-MK Memory Encryption. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/tme-box-scalable-in-process-isolation-through-intel-tme-mk-memory-encryption/

[33] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2020. Semeru: A Memory-Disaggregated Managed Runtime. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 261–280. https://

www.usenix.org/conference/osdi20/presentation/wang

[34] Chenxi Wang, Haoran Ma, Shi Liu, Yifan Qiao, Jonathan Eyolfson, Christian Navasca, Shan Lu, and Guoqing Harry Xu. 2022. MemLiner: Lining up Tracing and Application for a Far-Memory-Friendly Runtime. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 35–53. https://www.usenix.org/conference/osdi22/presentation/wang

[35] Chenxi Wang, Yifan Qiao, Haoran Ma, Shi Liu, Wenguang Chen, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2023. Canvas: Isolated and Adaptive Swapping for Multi-Applications on Remote Memory. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 161–179. https://www.usenix.org/conference/nsdi23/presentation/wang-chenxi

[36] Schrodinger ZHU Yifan. 2021. crc64-cxx. https://github.com/SchrodingerZhu/crc64-cxx.

[37] Wonsup Yoon, Jisu Ok, Sue Moon, and Youngjin Kwon. 2025. Adios to Busy-Waiting for Microsecond-scale Memory Disaggregation. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3689031.3717475

[38] Wonsup Yoon, Jisu Ok, Jinyoung Oh, Sue Moon, and Youngjin Kwon. 2023. DiLOS: Do Not Trade Compatibility for Performance in Memory Disaggregation. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 266–282. doi:10.1145/3552326.3567488

[39] Yang Zhou, Hassan M. G. Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E. Culler, Henry M. Levy, and Amin Vahdat. 2022. Carbink: Fault-Tolerant Far Memory. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 55–71. https://www.usenix.org/conference/osdi22/presentation/zhou-yang

Received June 2025; accepted September 2025