sRDMA — Efficient NIC-based Authentication and Encryption for Remote Direct Memory Access

Konstantin Taranov, Benjamin Rothenberger, Adrian Perrig, and Torsten Hoefler
Department of Computer Science, ETH Zurich

Abstract

State-of-the-art remote direct memory access (RDMA) technologies have shown to be vulnerable against attacks by in-network adversaries, as they provide only a weak form of protection by including access tokens in each message. A network eavesdropper can easily obtain sensitive information and modify bypassing packets, affecting not only secrecy but also integrity. Tampering with packets can have drastic consequences. For example, when memory pages with code are changed remotely, altering packet contents enables remote code injection. We propose sRDMA, a protocol that provides efficient authentication and encryption for RDMA to prevent information leakage and message tampering. sRDMA uses symmetric cryptography and employs network interface cards to perform cryptographic operations. Additionally, we provide an implementation for sRDMA using programmable network adapters.

1 Introduction

Despite numerous state-of-the-art systems [8, 11, 30] leveraging remote direct memory access (RDMA) primitives to achieve high performance guarantees and resource utilization, current RDMA technologies lack any form of cryptographic authentication or encryption. Instead RDMA mechanisms provide a weak form of protection by including access tokens in each message. Given that RDMA networks are mainly used in data-center environments and at large-scale deployments, detecting bugged wires is seemingly impossible. But not only in-network adversaries are an issue, also malicious end hosts can affect the security of an RDMA network. If an adversary is able to obtain control over a machine in an RDMA network (e.g., by escaping its virtual machine or hypervisor confinement in a cloud service [42]), it can fabricate and inject arbitrary packets. If the adversary can guess or obtain the protection domain and memory protection tokens (which are transmitted in plaintext), it can read and write memory locations that have been exposed using RDMA on *any machine in the network*, leading to a powerful attack vector for lateral movement in a data center network.

Given these threats, the security of current RDMA data center networks highly depends on isolation. However, even isolation cannot defend against in-network attackers. Thus, RDMA networks require cryptographic authentication and encryption. Unfortunately, application-level encryption (e.g., *TLS* [34]) is not possible, since RDMA read and write can operate as purely one-sided communication routines. Furthermore, such an approach requires employing a temporal buffer for incoming encrypted messages. The message would then be decrypted by the CPU and copied to the desired location, which would cause high overhead—negating RDMA's advantages. Additionally, cryptographic protection using *IPSec* [10] does not support RDMA traffic as the protocol is unaware of the underlying RDMA headers and achieves no source authentication (see Section 7).

In our work, we introduce a secure RDMA (sRDMA) design using a secure reliable connection (SRC) queue pair (QP) that uses symmetric cryptography for source and data authentication and employs Network Interface Cards (NICs) to perform cryptographic operations. Symmetric cryptography reduces the computational overhead compared to asymmetric cryptography by 3–5 orders of magnitude. Thus, it is suitable for high-performance and low-latency applications based on RDMA, e.g., [8, 17, 39]. Since symmetric cryptography introduces per-connection memory overhead and memory on NICs is constrained, we augment our proposed mechanisms using protection domain level keys and efficient dynamic key derivation, which eliminates the need for storing QP-level keys and drastically reduces the memory overhead on RDMA-capable NICs (RNICs). Additionally, we propose extended memory protection mechanisms that enable delegation of memory access to other trusted peers without requiring additional communication with the accessed host (e.g., an access control proxy for databases [29]).

In summary, we make the following contributions:

- we design a SRC QP that effectively prevents attacks in an RDMA network, with minimal changes to the current

InfiniBand Architecture (IBA) standard (Section 4.2);
- we improve our design by introducing PD-level keys to reduce the memory overhead on the RNIC (Section 4.5), and augment IBA with extended memory protection that enables delegation of memory accesses to third parties without the need of direct communication to the target entity (Section 4.6);
- we provide an implementation of our design using modern programmable network adapters equipped with ARM multi-core processors [7, 40] (Section 5);
- we extensively evaluate our design using artificial and real-world traces. Additionally, we modified the RDMA-based key value store, HERD [17], to make use of sRDMA (Section 6).

2 Remote Direct Memory Access

RDMA is a mechanism allowing one machine to directly access data in remote machines across the network.

Memory accesses are performed using dedicated hardware without any CPU intervention or context switches, which decreases CPU usage on both the initiator and the target. When an application performs an RDMA read or write request, the application data is delivered directly to the network, reducing latency and enabling fast message transfer. RDMA also exhibits the concept of one-sided operations when the CPU at a target node is not notified of incoming RDMA requests.

Several network architectures support RDMA: InfiniBand (IB) [4] , RDMA over Converged Ethernet (RoCE) [5], and internet Wide Area RDMA Protocol (iWARP) [33]. InfiniBand is a network architecture fully designed to enable reliable RDMA with its own hardware and protocol specification. RoCE is an extension to Ethernet to enable RDMA over an Ethernet network. Finally, iWARP is a protocol that allows using RDMA over TCP/IP. In this work, we focus on the InfiniBand architecture (IBA) and RoCE as they are the most widely used interconnect for RDMA, but the proposed ideas can be easily extended to other RDMA architectures.

2.1 InfiniBand Transport

Several transport types are supported by the IBA to communicate between endpoints: reliable connection (RC), unreliable connection, unreliable datagram, extended reliable connection, and raw packet. In this paper, we only consider the RC transport type, since it is the only type that supports both RDMA read and write requests.

The RC transport type establishes a *queue pair (QP)* between the two communicating parties. QPs are bi-directional message transport engines used to send and receive data in InfiniBand. Endpoints of a single RC QP can only communicate with each other but not with any other QP in the same or any other target adapter. Each QP endpoint has a queue pair number (QPN) assigned by the RNIC which uniquely identifies the QP within the RNIC.

The RC transport uses several techniques to ensure reliability. The target must respond to each request packet with a positive acknowledge packet or a negative acknowledge packet. The acknowledgement-based protocol permits the requester to verify that all packets are delivered to the target. To ensure the integrity of a packet, each packet contains two checksums that are verified by the target node. Finally, the RNIC counts received and sent packets using a packet sequence number (PSN), which is included in each packet. Thus, endpoints of a QP must know the PSN of each other to enforce in-order delivery and detect duplicate and lost packets.

2.2 IBA Memory Protection

The IBA protection mechanisms provide protection from unauthorized access to the local memory by network controllers. The local memory can also be protected against prohibited memory accesses. Three mechanisms exist to enforce memory access restrictions: Memory Regions, Protection Domains (PD), and Memory Windows.

Memory Regions. To get access to host memory, the RNIC must first allocate a corresponding memory region. This process involves copying page table entries of the corresponding memory to the memory management unit of the RNIC. When a memory region is created, the RNIC generates keys for local and remote accesses, namely *l_key* and *r_key*. The memory region can be accessed by any local QP which has the *l_key* as long as they are in the same PD, and by their remote QP endpoints which have the *r_key*. The endpoints must prove the possession of this key by including it in every RDMA request, such as RDMA Write and Read. *r_key* is not used in any form of cryptographic computation, but rather is used as access tokens that are transmitted in plaintext.

Protection Domain. PDs provide protection from unauthorized or inadvertent use of memory regions. PDs group IB resources such as QP connections and memory regions that can work together: QP connections created in one PD cannot access memory regions allocated in another PD. In other words, a memory region can be accessed by any QP from its PD. All QPs and memory regions must have a PD and can be a member of one PD only.

Memory Windows. Memory windows extend protection of memory regions by allowing remote QPs to have different access rights within a memory region and grant access to only a slice of the memory region.

3 Problem Definition

This section describes the adversary model we consider, outlines different types of attacks, and the security properties we strive to achieve.

3.1 Desired Security Properties

The current IBA protection mechanisms do not suffice to ensure secure communication between endpoints, allowing

adversaries numerous attacks. Thus, the primary goal of our work is to *secure RDMA protocols against attacks* by providing source and data authentication along with data secrecy and data freshness. Source authentication denotes the verification of the source address of a host that sends a packet and is designed to determine whether a packet originated from the claimed source. Data authentication ensures that the packet content has not been modified. Data secrecy ensures that the data remains hidden from a network eavesdropper. Data freshness ensures that data has not been recorded and replayed by a network attacker. Additionally, our proposal should require *minimal changes to the protocol*, and introduce only a *minor performance overhead*. This does not only include latency and processing overhead for RDMA requests but also memory state overhead on the RNIC.

3.2 Adversary Model

In our adversary model we consider end hosts that are equipped with RNICs and interact with each other through RDMA, and an adversary with the following parameters.

Location. We assume that the adversary can reside at *arbitrary locations* within the network. Thus, we consider both network-based adversaries (e.g., rogue cloud provider, rogue administrator, malicious bump-in-the-wire device) and adversaries located at end hosts (e.g., compromise of an end host). This includes compromise of the machines of communicating parties. However, we assume that RNICs are *trusted* by their host. This could be achieved using remote attestation, whereby a trusted party checks the internal state of a potentially compromised network device. We further assume that the internal bus is trusted, such that the CPU can securely communicate with the RNIC.

Capabilities. A network-based adversary can passively eavesdrop on messages, but also actively tamper with the communication. Since RDMA communication is performed in plaintext, an adversary that is located on the path between communicating parties can obtain any information in all IB and Ethernet headers. Furthermore, he can also alter any of these values, as this only requires recalculation of packet checksums, whose algorithms and seeds are known and specified by the IBA.

Given these capabilities, the adversary can also fabricate packets and send them towards a destination of its choice using spoofed QP numbers, *r_key*s, and PSNs (e.g., to modify a memory region without authorization to influence the behavior of applications running on the remote host).

Cryptography. The adversary has no efficient way of breaking cryptographic primitives. For pseudorandom function families, this means that no efficient algorithm can distinguish between an output of a function chosen randomly from the pseudorandom function family (PRF) and a random value.

Table 1: Notation used in this paper.

$\|$	Bitstring concatenation
$PRF_K(\cdot)$	Pseudorandom function using key K
$MAC_K(\cdot)$	Message authentication code using key K
A, B	Endpoints uniquely identified by the combination of the adapter port address (APA) and Queue Pair Numbers (QPN)
$K_{A,B}$	Symmetric key shared between node A and B
$nonce_{A\to B}$	cryptographic nonce used for communication in the direction from node A to node B
K_{PD}, K_{MR}, K_{SR}	Symmetric key used for protection domain, memory region, or sub memory region

4 Secure RDMA System Design

We propose a new transport type for reliable communication based on the IBA. We introduce a secure reliable connection (SRC) QP that uses symmetric cryptography for source and data authentication, and thus provides guarantees for the origin of a packet, data authenticity and payload secrecy.

To require minimal changes to the current IBA specification, our proposed design of the SRC QP consists of two main changes: 1) we add symmetric key initialization for QPs, and 2) we propose a new packet header called secure transport header (STH) which contains a message authentication code (MAC) providing integrity of the packet content. The STH must be included in all requests and response packets corresponding to RDMA reads and writes.

Besides basic QP-channel protection, we also propose *PD level protection* eliminating the need for storing cryptographic keys for each QP, which drastically reduces the memory overhead on RNICs. Additionally, it enables *extended memory protection* that provides memory access control based on encryption and the ability of delegating memory access to other trusted entities without additional communication to the accessed host. All QPs and memory regions created in a secure PD will be inherently secured by it.

Table 1 lists the security-related notation used in this paper.

4.1 Assumptions

Trust in RNIC. We assume that the RNIC is trusted by its host. It can not only perform authentication of outgoing packets, but is also trusted to perform en-/decryption of the packet payload. We further assume that the internal bus is trusted, such that the CPU can securely communicate with the RNIC.

QP-level Key Establishment. Our system enables the establishment of a QP-level symmetric key. To guarantee interoperability, our design is agnostic of this underlying mechanism. IBA could use for instance a (D)TLS [34] or QUIC [15] handshake as a mechanism to obtain a QP-level symmetric key.

Key Validity. As the validity period of a QP-level symmetric key is bound to the lifetime of a QP, key rollover can be performed by closing and reopening a QP between the communicating entities. Thus, key lifetime can be managed on the application level.

Size (bits)	0	96	128	160	224	256	384	512
Value	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7

Table 2: Possible sizes of STH, depending on the 3 bit value indicated in the Base Transport Header of an IB packet.

4.2 Secure Reliable Connection Queue Pair

We propose new transport type—Secure Reliable Connection (SRC) QP—that uses symmetric cryptography for source and data authentication. The introduction of SRC requires minimal changes to the current specification. Specifically, the QP initialization requires specifying a protection algorithm and a symmetric key. This allows us to bootstrap secrecy and authentication for QP-based communication.

Secure Transport Header. Secure Transport Header (STH) consists of MAC to provide header and packet authentication. The STH must be included in all request and response packets of sRDMA. Depending on the authentication mode installed to the secure QP, the MAC either authenticates only the packet header or the entire packet. To specify the length of the STH, we use 3 (out of 7) reserved invariant bits in the Base Transport Header. Based on the 3 bit value (see Table 2), the size of the MAC changes: minimum 96 bits, and maximum 512 bits. If the reserved 3 bits are all zero, then the STH is not present in the packet, thereby enabling support of both classical and secure QP connections.

Reusing PSN as a Per-Packet Nonce. sRDMA prevents replay attacks by including a unique nonce in the MAC computation of each packet (Section 4.3). Nonces are used as initialization vectors for ciphers to ensure that every packet is unique. They must *only be used once*, but their choice can be *predictable* and they can be *transmitted in clear* [19, 37]. In case a nonce is reused, the cryptographic properties of a cipher are affected (e.g., "sudden death" property of Poly1305 [6]).

A naive solution is to transmit a nonce as cleartext with each packet (e.g., as in TLS up to version 1.2 [36]). However, this would incur an additional transmission overhead of at least 64 bits, and additional 64 bits memory overhead on RNICs memory to store the nonce.

To avoid the overhead of transmitting the nonce, our protocol takes advantage of the sequential nature of IB packets. It uses the sequence number as nonces as they are tracked by end points and can never be reused. The approach resembles how TLS 1.3 [35] exploits the packet number as a nonce; however, the size of the PSN in the IB packet is only 24 bits, which would cause a reuse of a nonce after 80 ms assuming that an RNIC is able to send 200 million packets per second [28].

sRDMA extends the local PSN counters for inbound and outbound packets on the RNIC to 64 bits each, and reuse them as a per-packet nonce, thereby introducing only 40 bits overhead for each nonce. However, the size of the PSN transmitted on the wire remains unchanged (24 bits) and contains the least significant bits of the 64 bit counter. sRDMA is able to infer the 64 bit nonce used to secure the packet using only the 24 bit PSN specified in the header. Under the same assumption on the packet rate, the reuse of nonce occurs after 3,000 years.

To ensure that the nonce never gets reused by both endpoints, we use the most significant bit to identify the direction of communication between the entities A and B using their endpoint identifiers: the combination of adapter port address and Queue Pair Number (QPN).

Table 3: Overheads of sRDMA for N RC QP connections with AES-128 cipher in 4 different protection modes. Here, *pd-prot* and *ext-mem* stand for PD-level protection and extended memory protection, and are described in Section 4.5 and 4.6.

AES-128 protection	Key overhead	Nonce	Header
basic	16B * N	10B * N	16B
pd-prot	16B	10B * N	16B
ext-mem	16B * N + 16B	10B * N	16B
pd-prot + *ext-mem*	16B + 16B	10B * N	16B

4.3 Header Authentication

To perform header authentication, sRDMA uses the established symmetric keys and calculates a MAC for each packet:

$$mac_{hdr} = MAC_{K_{A,B}}(nonce_{A\to B} \parallel RH \parallel BTH)$$

Here, RH denotes the routing header, which defines the adapter port address, and BTH the base transport header, which includes destination QPN. Note that these headers uniquely identify the sender and receiver RNIC, and limit the input size of the MAC computation (only the packet header instead of the entire packet with arbitrary payload length). Thus, assuming an block-cipher-based MAC is used, a fixed number invocations of the block-cipher are required to calculate a MAC.

The RNIC of the receiving node will recompute the MAC for each packet and compare it to the MAC appended in the STH. Fields that are modified during the packet's transmission are replaced with ones during the MAC computation (same as for invariant checksum).

Header authentication prevents not only source-address-spoofing attacks, but also unauthorized access to memory regions by augmenting the existing IBA memory-protection mechanisms (i.e., *r_key* and memory windows).

4.4 Packet Authentication and Encryption

For packet authentication and payload encryption, we assume that the RNIC is trusted. Thus, the host is allowed to offload all cryptographic operations to the RNIC. We use authenticated encryption with associated data (AEAD), to simultaneously obtain secrecy and authenticity for the payload. The authentication tag is transmitted using the MAC field in the STH.

4.5 PD-level Protection

Introducing QP-level keys requires storing a 16 byte key per QP (see Table 3). As an RNIC might have a large number of QPs simultaneously, this can lead to a significant memory overhead on the RNIC. Memory on RNICs is a constrained resource, and a large part is consumed by IB connection contexts and page-table entries for registered memory. Multiple works report significant performance degradation of RDMA operations when the amount of memory registered or the number of QPs is increased [11, 18]. This is due to the RNIC running out of memory for storing page-table entries and starting to fetch them from system memory across the PCI bus. For instance, Dragojevic et al. [11] observe ~4x throughput drop in their evaluation when 4,096 memory pages are registered within the RNIC compared to a single-page experiment. Thus, we aim to mitigate the memory overheads introduced by QP-level keys.

To reduce the memory overhead and eliminate the need of storing a symmetric key per QP, we introduce PD-level protection. In this mechanism, we assign a symmetric key K_{PD} to each protection domain *PD* and use this key to derive QP-level keys using efficient key derivation [14]. PD-level keys are exchanged using the same mechanism as QP-level keys (see Section 4.1). The derivation process works as follows:

$$K_{A,B} = PRF_{K_{PD}}(APA_A \parallel QPN_A \parallel APA_A \parallel QPN_B)$$

PRF denotes a pseudorandom function with a PD-level key K_{PD} and a pair of unique end point identifiers (i.e., adapter port address (APA) and queue pair number (QPN)) as input. When an RDMA request targets a QP that is located within a protection domain *PD*, the RNIC uses the corresponding symmetric key K_{PD} to derive the QP-level key on-the-fly. The QP-level key is then used to perform authentication and encryption. Thus, instead of storing a symmetric key per QP, the RNIC is only required to store a key per PD. To minimize the processing overhead, the RNIC can cache the derived QP-level keys (e.g., after the first packet of a message arrives). K_{PD} is initialized upon creation of the PD and thus the lifetime of K_{PD} is bound to the lifetime of the PD. In order to perform a key rollover, a new protection domain must be created.

4.6 Extended Memory Protection

Using encryption of memory regions enables an even stronger mechanism for access control, as only entities in possession of the required key are able to read the content of a memory region. For this purpose, we use PD-level memory protection and derive memory level keys from K_{PD} for memory regions that are created within the protection domain. The derivation process works as follows:

$$K_{MR} = PRF_{K_{PD}}(START_{MR} \parallel END_{MR} \parallel r\_key_{MR})$$

Alternatively, the K_{MR} can be provided by the application to protect memory from unauthorized accesses.

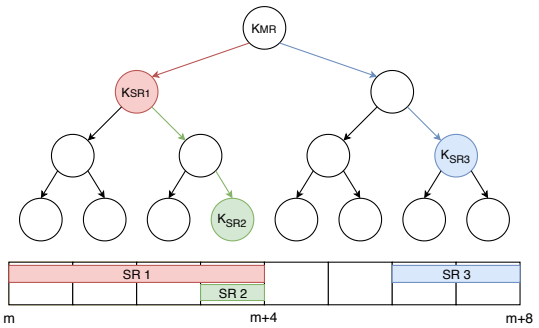


Figure 1: Access Sub-Delegation with one-way tree.

When remote parties want to access a subregion (*SR*) of the region *MR*, they need to prove the possession of the K_{MR} by computing a key to the SR:

$$K_{SR} = PRF_{K_{MR}}(START_{SR} \parallel END_{SR}) \quad (1)$$

Nonce for Key Derivation. To avoid replay attacks, our system must use a separate nonce for each memory region. However, it is not possible to use a memory access counter as nonce, as multiple QPs can access the same memory region. Therefore, this would require the RNIC to include a random nonce in each packet, which must be unique among all nonces used to access the memory region. Given that multiple parties have access to the region, this property is hard to achieve. Additionally, we want to avoid transferring a separate MAC for memory access in the packet header. Thus, we suggest to reuse the MAC of the header by overwriting it as follows:

$$mac_{hdr} = MAC_{K_{A,B}}(K_{SR} || mac_{hdr})$$

Such design allows sRDMA to reuse the per-packet nonce used in computation of mac_{hdr} and ensure the possession of K_{MR} to access memory. This construction is secure since the key is unknown to an adversary.

4.7 Sub-Delegation of Access to Memory

To allow sub-delegation of access to memory regions, we further extend the proposed extended memory protection with a binary one-way function tree [26]. The one-way function tree is built top-down where the memory region key K_{MR} is represented by the root of the binary tree and all child nodes are generated by applying the PRF:

$$K_{MR_{child}} = PRF_{K_{MR_{parent}}}(START_{MR_{child}} \parallel END_{MR_{child}})$$

Each memory block represents a leaf of the binary one-way tree (see Figure 1). Thus, the height of the tree depends on the size of the region and on the memory block size. Delegating access to a subregion works by sending the key of an intermediate node to a remote party. Given the key for a memory region, the subregion offset and subregion size uniquely identify which inner node is required for delegation.

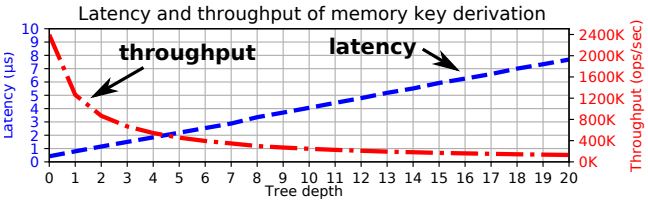


Figure 2: Performance of K_{MR} derivation for different tree depths. The derivation is performed sequentially, thus latency grows linearly and the throughput decreases exponentially.

An example of the delegation process is illustrated in Figure 1. To obtain a key for subregion 3, the entity must derive an intermediate key first, which can then be used to derive K_{SR_3}. To delegate access for subregion 2 to another host, a host in possession of K_{SR_1} derives K_{SR_2} and then shares this key with the delegated host. This allows access to subregion 2, but not to any other part of the memory region.

Only logarithmically many derivations are needed to obtain a key for memory access. For example, a key for accessing a 1 MiB subregion of a 16 MiB memory region can be obtained in at most 4 steps, which takes 2 μs on our RNIC (see Figure 2). Additionally, to prevent long key derivation chains, sRDMA allows users to limit the number of steps in the key derivation process. The sub-delegation is optional and can be disabled by restricting the tree depth to zero. In this case, the key derivation works as in Equation 1.

Advantages. This approach offers multiple advantages: 1) It trivially enables sub-delegation, as the delegated party can also calculate inner nodes of the subtree that is rooted at the subregion key. 2) The block size and thus the tree size is variable and can be adjusted for each memory region, e.g., depending on window size, depth of sub-delegation, or computational power of the corresponding nodes. This also allows limiting the depth of the tree to restrict the computational overhead. 3) The RNIC is required to compute only a single branch of the tree to verify. 4) The larger the delegate memory space, the smaller is the computational overhead on the RNIC. 5) The packet size remains constant as accessing a memory subregion requires only the start and size of the region, the offset and the size of subregion, and a MAC computed using the appropriate key.

Restrictions. We restrict delegation to powers of two to ensure that a single key is sufficient to delegate access to any sub-region. Alternatively, a solution based on segment trees would overcome this limitation, but require the exchange of multiple keys.

5 Implementation

Towards our goal of supporting secure QP connections, this section describes how we implement the sRDMA protocol using modern programmable network adapters equipped with ARM multi-core processors [7, 40]. sRDMA core primitives are implemented in 3,500 lines of C++ code and rely on various libraries: libibverbs, an implementation of the RDMA verbs; librdmacm, an implementation of the RDMA connection manager; Openssl 1.1.1a, a general-purpose cryptography library; and libev, a high-performance event loop. Our implementation supports more than 20 different cryptographic algorithms, such as the AES cipher and SHA hash families, to enable authentication and data secrecy for secure QPs. The implementation, all tests, and benchmark scripts are available in the open-source release.*

5.1 Notation and Experimental Setup

In the rest of the paper, we refer to a programmable network adapter as a *SmartNIC*. Our SmartNIC is capable of running a full Linux stack, supports RDMA over RoCEv2 and has crypto acceleration enabled. When RDMA requests are initiated on the SmartNIC and target the local host we refer to them as DMA requests as they only pass across the PCIe bus.

Our implementation is bi-directional, i.e., sRDMA writes and reads can be sent in both directions passing through both SmartNICs of the initiator and the target. Therefore, we distinguish between three roles as depicted in Figure 4: *Initiator*, *SmartNIC*, and *Target*, and the initiator always communicates with the target via two SmartNICs. Such a design allows full offloading of cryptographic computations from the initiator and the target to their respective SmartNICs.

5.2 Implementation of the Secure QP

We provide a library that models a secure QP connection between an initiator and a target as three standard RC QPs: one DMA connection between the SmartNIC and the host on *each* endpoint, and one connection between the SmartNICs.

Connection Establishment. Our secure QP library encapsulates connection establishment, which is performed in three stages as for a classical RC QP. When an application wants to establish a secure QP, it first creates a local QP in the INIT state. In this state, the connection between the host and the SmartNIC is created, and all necessary symmetric keys are copied to the SmartNIC. Then the QP must be transitioned to the RTR state by passing information about the target such as the QPN, the LID, and the PSN. To perform this transition we establish an RC QP connection between the two SmartNICs and create a special connection context on each SmartNIC. Finally, to send messages we transit the secure QP to the RTS state by passing the local send PSN. The application workers on the SmartNIC are responsible for packet counting, key derivation, and cryptographic algorithms.

Memory Registration. When a memory region should be secured with extended memory protection, the library intercepts a memory registration request and sends memory region information to a thread on the local SmartNIC.

*https://spcl.inf.ethz.ch/Research/Security/sRDMA/

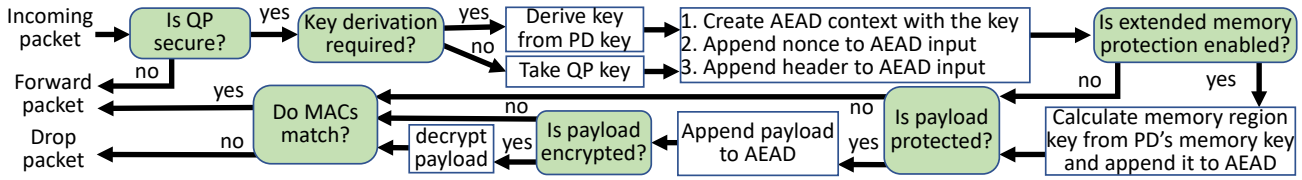


Figure 3: RNIC packet processing on receive.

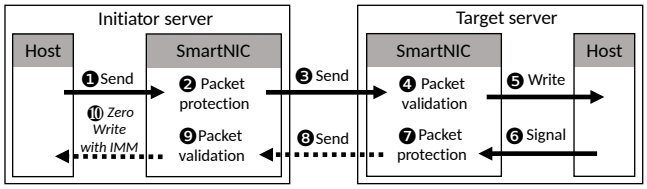
Secure QP communication. The initiator uses IB Send to deliver packets for both sRDMA reads and writes to its SmartNIC. The SmartNIC uses IB Receive to receive incoming packets from DMA connections and from remote SmartNICs. The SmartNIC secures all incoming packets from a DMA connection according to the cryptographic mechanism agreed on with the target. To secure a packet, the SmartNIC appends the IB transport and RDMA headers along with the generated MAC to the packet header. In our implementation, we use IB scatter/gather entries to attach an additional header before the main payload provided by the initiator. Scatter/gather entries allow building up an outgoing message from multiple buffers. After that, the packets are forwarded to the SmartNIC of the target QP. The target's SmartNIC verifies the security header as depicted in Figure 3, and decides on initiating an RDMA Read or RDMA write depending on the type of the request towards the target's host. The replies from the target are secured by its SmartNIC and forwarded back to the initiator.

sRDMA request completion. If the initiator expects an acknowledgment for a signaled request, the SmartNIC is responsible for acknowledging the initiator about the completion of the request. We use IB requests with immediate data to generate completion events on the host. The secure QP library is able to intercept completion events to distinguish between classical IB completions and sRDMA completions. The intercepted sRDMA completions are modified to inform the initiator about the sRDMA completion instead of the classical IB completion.

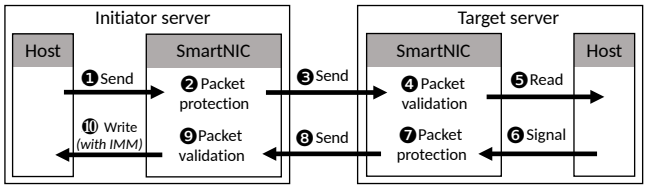
Packet security. The whole process of packet verification and key generation is shown in Figure 3. The SmartNIC performs header authentication, packet authentication, or payload encryption depending on which security protocol has been set up for the QP and which packet is processed. The SmartNIC will derive the QP's key if the QP is initialized in a secure PD, and also verify extended memory protection if the registered memory region has extended memory protection set up. On receiving, the SmartNIC also checks whether the QP is indeed a secure QP, as our implementation also supports classical insecure RC QPs. For insecure RC QPs, packets do not carry a MAC and are always trusted by SmartNICs.

5.3 sRDMA requests

Figure 4(a) depicts the implementation of an sRDMA write. The initiator ❶ sends a packet to the local SmartNIC containing the payload and the remote memory address. The local SmartNIC ❷ appends the IB header and the STH and



(a) sRDMA writes



(b) sRDMA reads

Figure 4: Implementation of RDMA operations.

❸ sends the secured packet to the remote SmartNIC. The remote SmartNIC ❹ processes the header and ❺ initiates a signaled DMA write to the host memory specified in the header. Upon ❻ the completion of the DMA write, the SmartNIC, depending on whether the sRDMA write is signaled, ❼❽ sends an authenticated Ack packet to the initiator's SmartNIC. The SmartNIC of the initiator then ❾ verifies the packet and ❿ performs an empty RDMA write with immediate data to its host, which consumes one posted receive at the host application. Finally, the secure QP interface intercepts such completions and modifies them to notify the application about the secure request completion.

sRDMA also implements secure Send operations which are similar to sRDMA writes, but they always generate the completion on the target and do not require knowing destination buffers. Since a Send request does not contain the header with destination buffer, it does not support memory protection.

sRDMA read has a similar structure as an sRDMA write as depicted in Figure 4(b) but there are some subtle differences. The initiator ❶ sends the message containing remote and local memory addresses and their *r_key*s to the local SmartNIC. The initiator's SmartNIC creates a special local read completion context with the initiator's memory address where the remote data must be copied to. Then the local SmartNIC ❷❸ sends the authenticated read request to the remote SmartNIC, which ❹ verifies the request and ❺ initiates a signaled DMA read from the target host memory to one of the SmartNIC's buffers. When ❻ the completion of a DMA read is generated,

the SmartNIC ❼❽ sends an authenticated read response with read data to the initiator's SmartNIC. The initiator's SmartNIC ❾ verifies the MAC of response packets and decides whether to ❿ write their content to the memory address specified in the matched local read completion context using a DMA write request. The DMA write will be with immediate data if the sRDMA read is signaled.

6 Evaluation

We conduct a series of benchmarks to thoroughly profile our system. To evaluate the overall sRDMA performance and the impact of cryptographic operations, we first evaluate the performance of each cryptographic algorithm. Secondly, we evaluate the latency and bandwidth of sRDMA writes and reads to assess the overheads of secure QPs over insecure QPs. Subsequently, we study the impact of bulk sRDMA operations by measuring the achievable bandwidth for different read/write ratios. Later, we evaluate the performance of the HERD key-value store [17] to examine the impact of sRDMA.

Test settings. The experiments are conducted on two servers directly connected to each other using the RoCEv2 protocol. These servers run Ubuntu 18.04.1 LTS with a 4.15.0-43-generic Linux kernel. Each server is equipped with a Broadcom PS225 25 Gbit/s programmable network controller. Both network adapters have eight-core 64-bit ARM Cortex-A72 3.0 GHz processors and 8 GiB of dual-channel DDR4 DRAM.

6.1 Authentication performance

We first study the performance of the cryptographic engine installed in the SmartNICs. We evaluate 7 different cryptographic algorithms of the openssl 1.1.1a library for message authentication: *aes-128*, *aes-192*, *aes-256*, *chacha20-poly1305*, *sha1-160*, *sha2-256*, *sha2-512*.

Figure 5 depicts the achievable throughput in Gbit/s of those algorithms for different numbers of threads and block sizes. The line rate of the tested RNIC over the RoCEv2 protocol is 20.6 Gbit/s, which is goodput of 25 Gbit/s link. AES algorithms are the fastest for small blocks and achieve 8 Gbit/s for 64 byte blocks using single thread. Thus, our sRDMA library uses the AES128 algorithm as the PRF function needed for key derivation. For larger blocks hash-based methods perform almost as fast as cipher-based algorithms. We observe that chacha20-poly1305 is ~4x slower on average than the AES algorithms. The data also reveals that we cannot achieve the line rate for packet authentication with SHA512.

For varying key sizes of AES algorithms, we have not noticed significant differences in performance and hereafter report results exclusively for the AES128 algorithm. As SHA1-based authentication provides similar performance as SHA256 in all tests, we omit its data in all plots. Additionally, we label chacha20-poly1305 in Figures as *poly1305*.

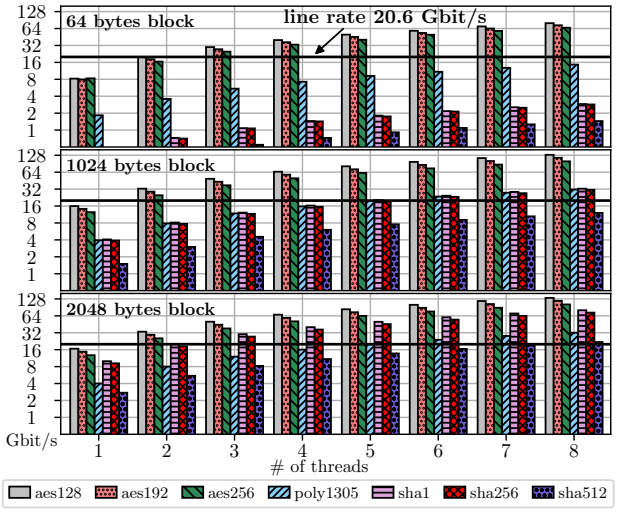


Figure 5: Authentication performance using openssl.

6.2 Evaluation modes

All evaluations have been performed with no security enabled (*NO security*) and in four protection modes:

No security. In *No security* mode RDMA reads and writes are performed as described in Section 5.3 but with skipping packet protection and validation (❷❹❼❾ in Figure 4).

Basic mode. In *basic* mode the key is attached to the secure QP connection directly, so the key is in the RNIC's cache when an incoming packet must be processed.

Pd-prot mode. The secure QP is created without an individual key, but in the secure PD (*pd-prot*) with a key derivation algorithm. We consider the case when the RNIC does not cache derived keys, and therefore, every time a packet arrives, the RNIC must derive the QP key from the PD key. In these experiments we want to show the performance of the system with constant cache misses. Using the cache we expect the same performance as in basic mode without key derivation.

Ext-mem mode. In this mode, the QP is created with an individual key and with extended memory protection (*ext-mem*) enabled. Extended memory protection requires derivation of memory level keys from a PD-level key. In this case, when a packet arrives, the RNIC must generate a key to access memory specified in the RDMA header from the PD-level key and include the generated key in MAC calculation. For this test we also consider that the initiator has the primary memory key which grants access to whole memory region, so the memory key can be derived in one step (depth 0 in Figure 1).

Pd-prot + ext-mem mode. The last mode combines our two protection methods: secure PD and extended memory protection (*pd-prot + ext-mem*). Therefore, the RNIC is responsible for generating both keys when a packet should be processed.

6.3 Latency

To evaluate the overall sRDMA performance and the impact of cryptographic operations, we split latency tests in two cate-

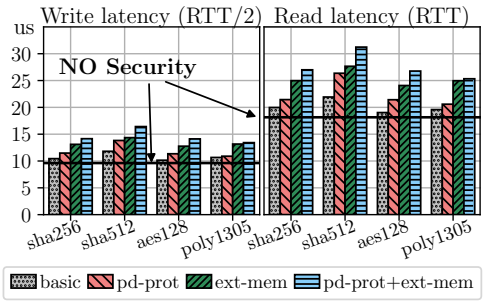


Figure 6: Source authentication latency of reads and writes carrying 32 Bytes payload.

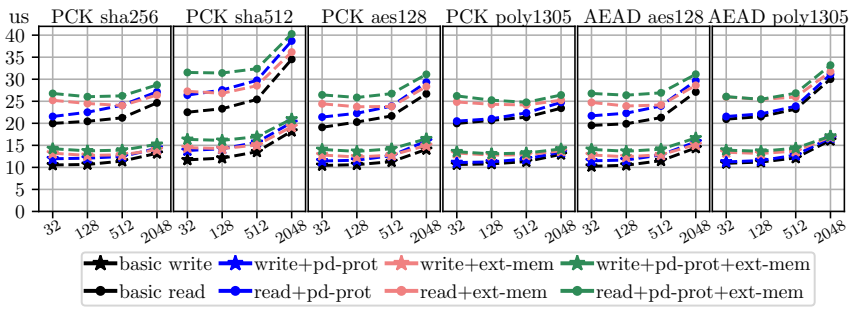


Figure 7: Latency of packet authentication (*PCK*) and encryption (*AEAD*) as a function of payload sizes.

gories: header authentication only and full packet security.

Header authentication. Figure 6 presents the median latency of sRDMA reads and writes in all four protection modes for header authentication. The figure reports the median only as for all measurements deviation from the median is less than 0.4 μs. All measurements are done for packets carrying the payload of 32 bytes. sRDMA write latency is measured for a half round trip, whereas sRDMA reads are for a full round trip. The latency of sRDMA writes without security is 9.55 μs and of sRDMA reads is 18.2 μs which build the baseline for our experiments.

Figure 6 shows that all tested security algorithms in the first mode add about 0.9 μs for sRDMA writes which is approximately 9% more than the insecure version. Another interesting observation is that the QP key derivation is more expensive than memory key derivation. The difference stems from the fact that a key-derivation process involves reinitialization of cryptographic contexts and different algorithms have different reinitialization performance (e.g., AES generates round keys [9]). The same phenomenon occurs for sRDMA reads. As expected, the highest latency is achieved for sRDMA operations with both key derivation and extended memory protection.

Packet security. We evaluate the latency of packet authentication (*PCK*) and packet encryption (*AEAD*) for different payload sizes and in four protection modes. Figure 7 illustrates the median latency of sRDMA reads and writes for SHA256, SHA512, AES128, and POLY1305. In each subplot, the top four lines illustrate sRDMA read round-trip latency, and the bottom four lines half-round-trip latency of sRDMA writes.

Figure 7 highlights that payload authentication is more expensive than header authentication. It takes 15 μs to write and secure 2 KiB payload in the first mode in comparison to header authentication of the same packet with the median of 12 μs. The graph also illustrates that latency increases for both reads and writes with payload size as more data must be authenticated. For AEAD, latency goes up even faster with respect to payload size since more data is en-/decrypted. As anticipated, SHA512 has the highest latency as the most expensive algorithm. We observe that for smaller payload sizes payload authentication and payload encryption achieves approximately the same performance in terms of latency.

6.4 Bandwidth

We measure performance separately for sRDMA reads and writes. As for latency benchmarks, all evaluations are performed in four protection modes. Our implementation is multi-threaded where each thread can process requests from a single secure QP. The number of threads represents the number of connections between endpoints. For n threads, each host establishes n secure connections with its SmartNIC, and SmartNICs establish n connections between each other. Thread workers on a SmartNIC do not share any resources and are pinned to distinct cores. In all evaluations the initiator issues requests continuously to the target, but with a limited number of outstanding requests (96 per connection). Once the initiator receives the signal for an sRDMA request completion it posts new requests to maintain 96 outstanding requests. The payload size is 2,048 bytes and bandwidth is measured in Gbit/s of goodput. We also assume the worst case scenario for the secure PD mode (*pd-prot*): the RNIC derives the QP key from the PD key for each packet. In other words, we consider the case when the RNIC does not cache derived keys. The main reason for that is that *pd-prot* mode with caching has the same performance as *basic* mode.

Figure 8 depicts communication bandwidth for sRDMA writes with different cryptographic algorithms. The black line (*NO*) in the header column stands for sRDMA writes with no security enabled. We observe that the single-threaded test with no security achieves only 8 Gbit/s while the highest RDMA goodput bandwidth achievable on our interconnect is 20.6 Gbit/s. The slowdown is caused by processing and parsing headers of messages by the general purpose ARM CPUs of the SmartNICs. Even if no security is enabled, a thread worker reads and parses headers of incoming packets and, depending on the operation code, initiates RDMA requests according to our implementation described in Section 5.3. In our tests we treat performance of secure operations with no security as the baseline. The highest achievable goodput bandwidth with no security is 20.5 Gbit/s which is line rate.

Figure 8 illustrates that sRDMA writes with header authentication can achieve line rate in all four protection modes

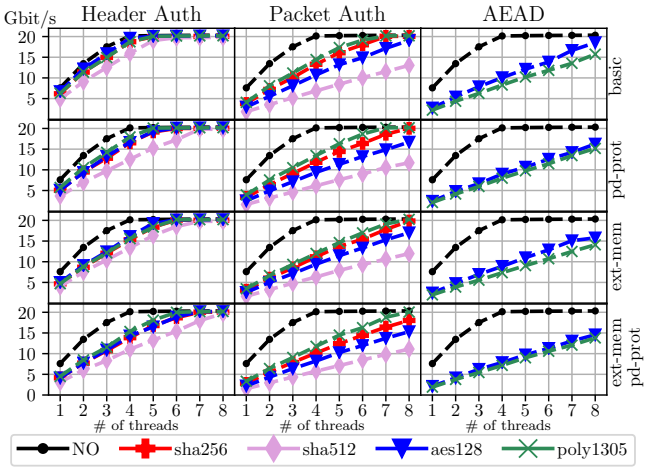


Figure 8: Bandwidth of sRDMA Writes in four different protection modes, and with *NO* security enabled.

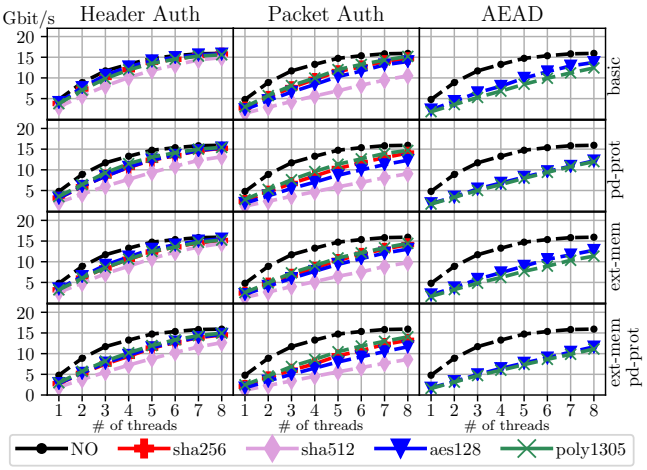


Figure 9: Bandwidth of sRDMA Reads in four different protection modes, and with *NO* security enabled.

if we use all 8 threads. The slowest header authentication is observed for SHA512 due to hashing performance. For full packet authentication SHA512 reaches only a goodput of 13 Gbit/s which is even slower than AEAD algorithms. In the payload encryption mode, our implementation can also achieve line rate for the SHA256 and POLY1305 algorithms. AES128 based authentication achieves 19.6 Gbit/s which is 95% of the line rate. The data also demonstrates that key derivation algorithms slow down sRDMA writes by 2 Gbit/s on average. However, in header authentication mode all algorithms can achieve 20 Gbit/s without performance loss when all 8 threads are used. Another interesting observation is that POLY1305 is faster than AES128 in packet-authentication mode, but slower in packet-encryption one. In AEAD mode, the highest write bandwidth of 19 Gbit/s is observed for the AES128 algorithm.

We have performed a similar benchmark for sRDMA reads in various protection modes. Results of our evaluations are depicted in Figure 9. Again, the black line (*NO*) stands for no security installed and represents the baseline for sRDMA reads. sRDMA reads are more expensive than writes despite the fact that they transfer the same amount of protected bytes as signaled sRDMA writes. Both sRDMA operations require six hops for a full round trip, and they both transfer the same payload size but in different directions. For writes, data is sent from the initiator to the target, and for reads from the target to the initiator. The differences in performance stem from the fact that an sRDMA read is a more complex operation than an sRDMA write and requires to create a special read context and matching it at initiator's SmartNIC (see Section 5.3). In addition, receive buffers on SmartNICs for reads and writes have different lifetimes. For example, a receive buffer can be released on the target SmartNIC once the completion of the RDMA write is received (❻ in Figure 4(a)), however, for reads the buffer on the target SmartNIC can be released once the completion of ❽ is received from Figure 4(b). According to the data, the highest achievable sRDMA read bandwidth is 16.71 GBit/s for 8 threads and about 4.7 Gbit/s for single-threaded test. Overall, our measurements indicate that reads are 16% slower than writes for all tests due to the complexity of sRDMA reads.

CPU Usage in Bandwidth Experiments. In our experiments, sRDMA introduces no overhead on the host CPU usage as packet processing is fully offloaded to the SmartNIC. The host application only needs to submit an RDMA request to the SmartNIC, which performs all cryptographic computations as described in Section 5.3. The SmartNIC, on the other hand, has full CPU usage in almost all experiments, which can be observed in the inability of the majority of security schemes to achieve line rate. The main reason for that is the SmartNIC needs to load the incoming packets from its DRAM to the L1 cache of its CPU cores in order to process the packets depending on the installed security level. Thus, all protection levels which require the CPU to read the whole packet have 800% CPU usage for 8 worker threads, even though in authentication performance experiment (see Figure 5) all authentication algorithms achieves the line rate for 2 KiB blocks. It comes from the fact that the packet authentication and AEAD are memory-bound problems, and, therefore, CPU works at full capacity to copy the data to its caches.

Header authentication requires reading only the header to authenticate the packet. Thus, header authentication algorithms could achieve 100% of line-rate, although, the performance still suffers from cache misses. The lowest CPU usage is observed for AES128 authentication scheme, which is 440% CPU usage for the bandwidth experiment with sRDMA Write requests. The sRDMA reads, on other hand, consume almost 750% on the target SmartNIC.

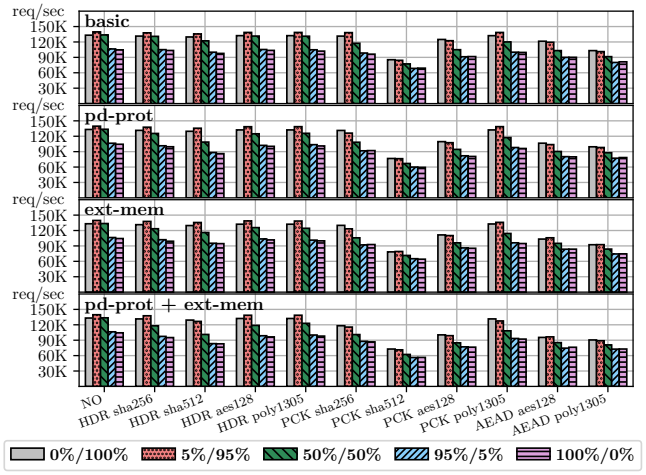


Figure 10: Throughput of mixed read/write benchmark.

6.5 Mixed write/read workload

The results of Figure 8 and Figure 9 are valid for either read-only or write-only workloads, which are uncommon for read-world applications. Therefore, we measure the throughput of sRDMA in a more realistic scenario as used in key-value stores that exploit one-sided RDMA operations. Figure 10 shows the throughput for workloads with different (read/write) ratios, including write only (0%/100%), write mostly (5%/95%), equal-shares (50%/50%), read-mostly (95%/5%) and read-only (100%/0%). The read-heavy workload is representative for applications such as photo tagging. The update-heavy workload is typical for applications such as an advertisement log that records recent user activities. In this benchmark the payload size is 2,048 bytes, and sRDMA is deployed with all 8 workers. We also consider the worst case scenario for the secure PD mode (*pd-prot*), when the RNIC derives the QP key for each packet. The *pd-prot* mode with QP key caching has the same performance as *basic* mode.

Figure 10 illustrates that (5%/95%) workload performs better than (0%/100%) one. The reason for that is better utilization of the bi-directional connection between endpoints since sRDMA writes send data from the initiator to the target, whereas sRDMA reads from the target to the initiator. Therefore, in that case we achieve a better utilization of the connection in the direction of the initiator. In theory, a (50%/50%) ratio would lead to the highest throughput as both links would be loaded evenly; however the lower performance of sRDMA reads overwhelms benefits of the network utilization. For the same reason, the throughput decreases for higher read ratios.

6.6 Key-value store workload

HERD [17] is an RDMA-accelerated key-value store which uses a mix of RDMA write and IB send verbs. HERD uses MICA's [25] algorithm for both GETs and PUTs: each GET requires up to two random memory lookups, and each PUT

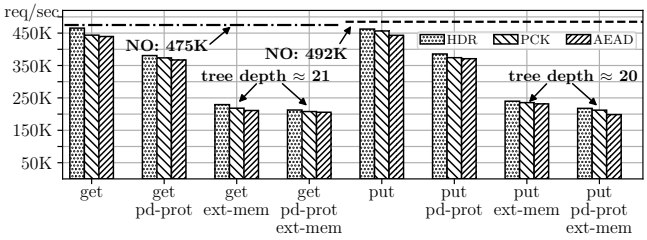


Figure 11: Throughput of the HERD kvs over sRDMA.

requires one. In HERD, clients transmit their request to the server's memory using RDMA writes, and get responses via unreliable datagram QPs. To comply with our sRDMA design, we made some changes to the original HERD implementation. First of all, we replace all unreliable datagram QPs with RC QPs as they are not reliable and not point-to-point and thus incompatible with sRDMA. That is, the server replies to clients via RC QPs, but still uses IB Send verbs. For that, we also implement secure SEND operations which are similar to sRDMA writes, but they always generate the completion on the target and do not require knowing destination buffers. Since an IB Send request does not contain the header with destination buffer, it does not support extended memory protection. The second change is that clients send requests via reliable sRDMA writes instead of unreliable writes.

Key-value-store (KVS) experiments use one server machine and one client machine. The server machine has only one worker process when the client machine has 8 processes. Each client process establishes an sRDMA connection to the server. The key size is 16 bytes and the value size is 32 bytes. Therefore, clients send and receive small messages of less than 40 bytes. The KVS contains 8,388,608 keys and occupies 1 GiB of memory. Figure 11 depicts the throughput for puts and gets in different protections modes based on the AES128 cipher. We also measure HERD's throughput with NO protection which is 475K req/sec for gets and 492K req/sec for puts. Puts are faster than gets because they cause fewer lookups in internal memory structures.

According to the data in Figure 11, basic packet authentication without key-derivation algorithms achieves almost the same throughput as the unprotected version. Interestingly, even the AEAD mode decreases the throughput by 7.3%. In the setting with a secure PD when the key must be generated for each request, we observe a 21% slow down in both puts and gets. It is worth mentioning that we intentionally derive the QP keys for each request in the secure PD mode (*pd-prot*) to see the effect of constant misses in QP keys. In real settings, an RNIC would have a cache with generated keys to reduce computation. In such case, the *pd-prot* mode has the same performance as *basic* mode.

A drastic decrease in performance can be observed for evaluations with enabled extended memory protection. The reason for this is that HERD's clients WRITE their GET

Table 4: Comp. of sRDMA to IPSec and TLS over RoCE.

Protocol	Sec. comm.	IBA supp.	One-sided comm.	Hdr overhead.
RDMA	✗	✓	✓	-
IPSec	✓	✗	✗	50-80 B
(d)TLS	✓	✗	✗	25-40 B
[23, 24]	✗	✗	✓	12-16 B
sRDMA	✓	✓	✓	12-64 B

requests of 17 bytes and PUT requests of 40 bytes to the contiguous memory region of 16 MiB on the server machine. Therefore, it takes on average 20 steps for PUTs and 21 steps for GETs to derive the memory MAC using our binary tree approach, which causes such significant drop in performance. To alleviate the problem, the depth of the tree can be limited to 0, and then the *ext-mem* would achieve the same performance as the *pd-prot* case.

7 Related Work on Securing IBA

RFC 5042 [32] analyzes the security issues around uses of RDMA protocols. It reviews various attacks against resources including spoofing, tampering, information disclosure, and DoS. As a countermeasure the authors suggest to employ IPsec authentication and encryption [10]. However, IPSec currently does not support RDMA traffic, because it is unaware of the encapsulated RDMA headers and thus cannot distinguish QP endpoints. A naive application of IPSec to RoCE packets would not achieve source authentication as all RoCE traffic is destined to the same UDP port (and not the QPN). Thus, the use of IPsec would incur changes in the packet format, whereas sRDMA is supported by native IBA and RoCE. Additionally, the complexity of IPsec and its high processing overheads [31] make it ill-suited for high-performance and low-latency applications and would introduce a header overhead of 50-80 bytes [21]. While the IPsec-enabled Cavium LiquidIO II [2] and Mellanox Innova [3] NICs support RoCE, they do not support IPsec-based protection of RoCE packets.

Lee et al. [23,24] discuss security vulnerabilities in IBA and show that they could be exploited by an adversary with moderate overhead. The authors suggest to replace the Invariant CRC field with a MAC to achieve packet authentication. Unfortunately, this might lead to routers dropping packets with invalid ICRC, making the proposed solution incompatible with legacy routers. Additionally, they discuss how IBA could reduce its key exposure risk by introducing partition- and queue-level key distributions. However, modifying partition-level keys can lead to packets being dropped as they might be used by routers and switches to enforce partitioning. Furthermore, their design uses the 24 bit PSN as a nonce which cause a reuse of a PSN after 80 ms on modern RNICs [28]. Finally, the authors provide no implementation of their system, but rather simulate the performance of symmetric ciphers to show that they are suitable for high performance networking.

RDMA Side-Channel Attack. Kurth et al. [22] have shown that the Intel DDIO [1] and RDMA features facilitate a side-channel attack named NetCAT. Intel DDIO technology allows RDMA reads and writes access not only the pinned memory region but also parts of the last level cache of the CPU. NetCAT remotely measures cache activity caused by a victim SSH connection to perform a keystroke timing analysis. An attacker can make use of the attack to recover words typed by a victim client in the SSH session from another computer.

Tsai et al. [41] implemented a set of RDMA-based remote sidechannel attacks that allow an attacker on one client machine to learn how victims on other client machines access data. They further extend their work by building side-channel attacks on Crail [38].

Using sRDMA a large attack surface could be removed by permitting only trusted entities to initiate RDMA requests.

8 Conclusion

Using NIC-based authentication and encryption enables secure communication for systems that require high performance guarantees such as RDMA mechanisms. sRDMA provides strong authenticity and secrecy, and prevents several forms of DoS attacks. Thus, safety- and security-critical applications that rely on RDMA must use sRDMA to prevent attacks by malicious entities within the same network.

Our software implementation on the SmartNIC causes a high load due to data movement overheads. The datapath could be optimized with a different architecture using specialized programmable packet processing units [13, 20]. Furthermore, sRDMA could also be hardened into fixed logic as the area and power consumption overhead are marginal compared to regular input/output processing [12, 16, 27]. Additionally, sRDMA minimizes memory consumption on the RNIC using PD-level protection.

Acknowledgment

We thank our shepherd, Heming Cui, and the anonymous reviewers for their helpful feedback. We thank Broadcom Inc., especially Fazil Osman, for the donation of two SmartNICs as well as continuous support. We gratefully acknowledge support from ETH Zurich, and from the Zurich Information Security and Privacy Center (ZISC). Furthermore, we thank for Microsoft Swiss Joint Research Centre for support.

References

[1] Intel® Data Direct I/O Technology Overview. https://www.intel.co.jp/content/dam/

www/public/us/en/documents/white-papers/data-direct-i-o-technology-overview-paper.pdf, 2019. [Accessed 15-May-2020].

[2] LiquidIO®II 10/25G Smart NIC Family. https://www.marvell.com/documents/08icqisgkbtn6kstgzh4/, 2019. [Accessed 15-May-2020].

[3] Mellanox Innova-2 Flex Open Programmable SmartNIC. https://www.mellanox.com/related-docs/prod_adapter_cards/PB_Innova-2_Flex.pdf, 2019. [Accessed 15-May-2020].

[4] InfiniBand Trade Association et al. The InfiniBand architecture specification. 2000.

[5] Infiniband Trade Association et al. Supplement to InfiniBand Architecture Specification Volume 1, Release 1.2. annex A16: RDMA over Converged Ethernet (RoCE), 2010.

[6] Daniel J Bernstein. The Poly1305-AES message-authentication code. In *International Workshop on Fast Software Encryption*, pages 32–49. Springer, 2005.

[7] Broadcom. Stingray 2x25Gb High-Performance Data Center Smart NIC. https://www.broadcom.com/products/ethernet-connectivity/smartnic/ps225, 2019. [Accessed 15-May-2020].

[8] Qingchao Cai, Wentian Guo, Hao Zhang, Divyakant Agrawal, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, Yong Meng Teo, and Sheng Wang. Efficient distributed memory management with rdma and caching. *Proc. VLDB Endow.*, 11(11):1604–1617, July 2018.

[9] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[10] Naganand Doraswamy and Dan Harkins. *IPSec: the new security standard for the Internet, intranets, and virtual private networks*. Prentice Hall Professional, 2003.

[11] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. Farm: Fast remote memory. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation*, NSDI, pages 401–414, 2014.

[12] Kris Gaj and Pawel Chodowiec. FPGA and ASIC implementations of AES. In *Cryptographic engineering*, pages 235–294. Springer, 2009.

[13] Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E Grant, and Ron Brightwell. spin: High-performance streaming processing in the network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16, 2017.

[14] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24. ACM, 1989.

[15] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. Internet-Draft draft-ietf-quic-transport-17, Internet Engineering Task Force, December 2018. Work in Progress.

[16] Hyun-Wook Jin, Pavan Balaji, Chuck Yoo, Jin-Young Choi, and Dhabaleswar K Panda. Exploiting nic architectural support for enhancing ip-based protocols on high-performance networks. *Journal of Parallel and Distributed Computing*, 65(11):1348–1365, 2005.

[17] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Using rdma efficiently for key-value services. In *Proceedings of ACM SIGCOMM*, pages 295–306, 2014.

[18] Anuj Kalia, Michael Kaminsky, and David G. Andersen. Design guidelines for high performance rdma systems. In *Proceedings of the USENIX Annual Technical Conference*, ATC, pages 437–450, 2016.

[19] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[20] Antoine Kaufmann, Simon Peter, Naveen Kr Sharma, Thomas Anderson, and Arvind Krishnamurthy. High performance packet processing with flexnic. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 67–81, 2016.

[21] Stephen Kent. IP authentication header. Technical report, 2005.

[22] Michael Kurth, Ben Gras, Dennis Andriesse, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. NetCAT: Practical Cache Attacks from the Network. In *S&P*, May 2020. Intel Bounty Reward.

[23] Manhee Lee and Eun Jung Kim. A comprehensive framework for enhancing security in InfiniBand architecture. *IEEE Transactions on Parallel and Distributed Systems*, 18(10), 2007.

[24] Manhee Lee, Eun Jung Kim, and Mazin Yousif. Security enhancement in InfiniBand architecture. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 10–pp. IEEE, 2005.

[25] Hyeontaek Lim, Dongsu Han, David G. Andersen, and Michael Kaminsky. Mica: A holistic approach to fast in-memory key-value storage. In *Proceedings of USENIX Conference on Networked Systems Design and Implementation*, NSDI, pages 429–444, 2014.

[26] Chu-Hsing Lin. Dynamic key management schemes for access control in a hierarchy. *Computer communications*, 20(15):1381–1385, 1997.

[27] Bin Liu and Bevan M Baas. Parallel AES encryption engines for many-core processor arrays. *IEEE transactions on computers*, 62(3):536–547, 2013.

[28] Mellanox. ConnectX-6 EN Single/Dual-Port Adapter. https://www.mellanox.com/products/infiniband-adapters/connectx-6, 2019. [Accessed 15-May-2020].

[29] B Clifford Neuman. Proxy-based authorization and accounting for distributed systems. In *Proceedings of IEEE International Conference on Distributed Computing Systems-ICDCS*, pages 283–291. IEEE, 1993.

[30] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. The ramcloud storage system. *ACM Trans. Comput. Syst.*, 33(3):7:1–7:55, August 2015.

[31] Jungho Park, Wookeun Jung, Gangwon Jo, Ilkoo Lee, and Jaejin Lee. Pipsea: A practical ipsec gateway on embedded apus. In *Proceedings of ACM Conference on Computer and Communications Security*, CCS, pages 1255–1267, 2016.

[32] J. Pinkerton and E. Deleganes. Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security. RFC 5042, October 2007.

[33] Renato Recio, Bernard Metzler, Paul Culley, Jeff Hilland, and Dave Garcia. A remote direct memory access protocol specification. Technical report, 2007.

[34] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, January 2012.

[35] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[36] Eric Rescorla and Tim Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[37] Phillip Rogaway. Nonce-based symmetric encryption. In *International Workshop on Fast Software Encryption*, pages 348–358. Springer, 2004.

[38] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Radu Stoica, Bernard Metzler, Nikolas Ioannou, and Ioannis Koltsidas. Crail: A high-performance I/O architecture for distributed data processing. *IEEE Data Eng. Bull.*, 40(1):38–49, 2017.

[39] Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. Fast and strongly-consistent per-item resilience in key-value stores. In *Proceedings of EuroSys Conference*, EuroSys, pages 39:1–39:14, 2018.

[40] Mellanox Technologies. Mellanox BlueField SmartNIC. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_BlueField_Smart_NIC.pdf, 2019. [Accessed 15-May-2020].

[41] Shin-Yeh Tsai, Mathias Payer, and Yiying Zhang. Pythia: remote oracles for the masses. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 693–710, 2019.

[42] VMWare. ESXi VM and Hypervisor Escape Advisory. https://blogs.vmware.com/security/2018/11/vmware-and-the-geekpwn2018-event.html, 2019. [Accessed 15-May-2020].