TRANSPARENT APPLICATION DEPLOYMENT IN A SECURE, ACCELERATED AND COGNITIVE CLOUD CONTINUUM

Grant Agreement no. 101017168

Deliverable D3.1
Accelerated encrypted storage architecture

Programme:	H2020-ICT-2020-2
Project number:	101017168
Project acronym:	SERRANO
Start/End date:	01/01/2021 – 31/12/2023

Deliverable type:	Report
Related WP:	WP3
Responsible Editor:	MLNX
Due date:	31/03/2022
Actual submission date:	01/04/2022

Dissemination level:	Public
Revision:	0.3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017168

Revision History

Date	Editor	Status	Version	Changes
01.11.21	MLNX	Draft	0.1	Initial ToC
02.02.22	MLNX	Draft	0.1	Populating HW acceleration
02.10.22	CC	Draft	0.1	Described Secure Storage use case
03.11.22	MLNX	Draft	0.2	Generated introduction sections and conclusion sections. Overall narrative and typo corrections.
03.31.22	CC	Draft	0.2	Made changes to UC1 description based on review by HLRS.
04.01.22	MLNX	Final	0.3	Integration of feedback from internal peer reviewers.

Author List

Organization	Author
ICCS	Panagiotis Kokkinos, Aristotelis Kretsis
MLNX	Yoray Zack, Juan Jose Vegas Olmos, Sandra Starck
CC	Marton Sipos
USTUTT/HLRS	Javad Fadaie Ghotbi
AUTH	
INTRA	
INB	Ferad Zyulkyarov
INNOV	
IDEKO	Aitor Fernandez, Javier Martin
UVT	
NBFC	

Internal Reviewers

HLRS, ICCS

Abstract: This deliverable (D3.1) provides an initial reference architecture of the encryption accelerators. In particular, the design choices and implementation of the accelerators in the contexts of TLS and NVMe offloads, which are relevant for the SERRANO platform.

Keywords: SERRANO architecture, SERRANO platform, transparent deployment, hardware acceleration, secure storage, cognitive orchestration, service assurance

Disclaimer: *The information, documentation and figures available in this deliverable are written by the SERRANO Consortium partners under EC co-financing (project H2020-ICT-101017168) and do not necessarily reflect the view of the European Commission. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.*

Table of Contents

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
ASIC	Application-Specific Integrated Circuit
BSI	Belief Desire Intention
CFD	Computational Fluid Dynamics
CI/CD	Continuous integration/Continuous Deployment
CNC	Computer Numerical Control
CORS	Cross-Origin Resource Sharing
DOCA	Data center On a Chip Architecture
DPU	Data Processing Unit
DSE	Design Space Exploration
EDE	Event Detection Engine
EU	European Union
FaaS	Function as a Service
FPGA	Field Programmable Gate Array
GCP	Google Cloud Platform
GPU	Graphics Processing Unit
HLRS	High Performance Computing Center Stuttgart
HLS	High-Level Synthesis
HPC	High Performance Computing
HW	Hardware
IO	Input/Output
MAAS	Metal as a Service
ML	Machine Learning
MOM	Message-Oriented Middleware
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
OCI	Open Container Initiative
OSS	Object Storage Server
OST	Object Storage Target
PCIe	Peripheral Component Interconnect Express
PXE	Preboot Execution Environment

QoS	Quality-of-Service
RDMA	Remote Direct Memory Access
REST	Representational State Transfer
RLNC	Random Linear Network Coding
RoCE	RDMA over Converged Ethernet
ROT	Resource Optimization Toolkit
RPC	Remote Procedure Call
RTL	Register-Transfer Level
SAR	Service Assurance and Remediation
SDK	Software Development Kit
SFTP	Secure File Transfer Protocol
SLA	Service Level Agreement
SoC	System on a Chip
SW	Software
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTM	Time to Market
UC	Use Case
URL	Uniform Resource Locator
VM	Virtual Machine
VMM	virtual Machine Monitor
VVUQ	Verification, Validation and Uncertainty Quantification
WP	Work Package
WSGI	Web Server Gateway Interface

1 Executive Summary

SERRANO envisages the development and deployment of disaggregated federated cloud infrastructures that operate, process and store in the edge, enabling accelerated edge nodes as integral parts of the computation and storage chain. The SERRANO ecosystem expansion includes HPC infrastructures that can be utilized for exceptionally computationally intensive simulations and data analysis, bridging the gap between these currently largely separated computing paradigms.

Deliverable D3.1 “SERRANO Accelerated encrypted storage architecture” provides an initial reference architecture of the encryption accelerators. In particular, the design choices and implementation of the accelerators in the contexts of TLS and NVMe offloads, which are relevant for the SERRANO platform.

The information provided in the present deliverable is expected to comprise a guideline framework for the rest of the project’s Work Packages, and is relevant to considered part of the other work packages 3 deliverables (D3.2 and D3.3) that complement the current document. The SERRANO accelerated platform will be finalized in M30 in D3.4 “Final release of SERRANO Secure Infrastructure Layer”.

2 Introduction

SERRANO targets the efficient and transparent integration of heterogeneous resources, providing an infrastructure that goes beyond the scope of the "typical" cloud and realizes a true computing continuum. The project relies on the SERRANO platform, which can operate a federated underlying fabric consisting of cloud, edge and HPC resources (Figure 1). The communications between those elements owe to be secure, which means that the information being exchanged is somehow isolated against third parties (including the system itself that handles it).

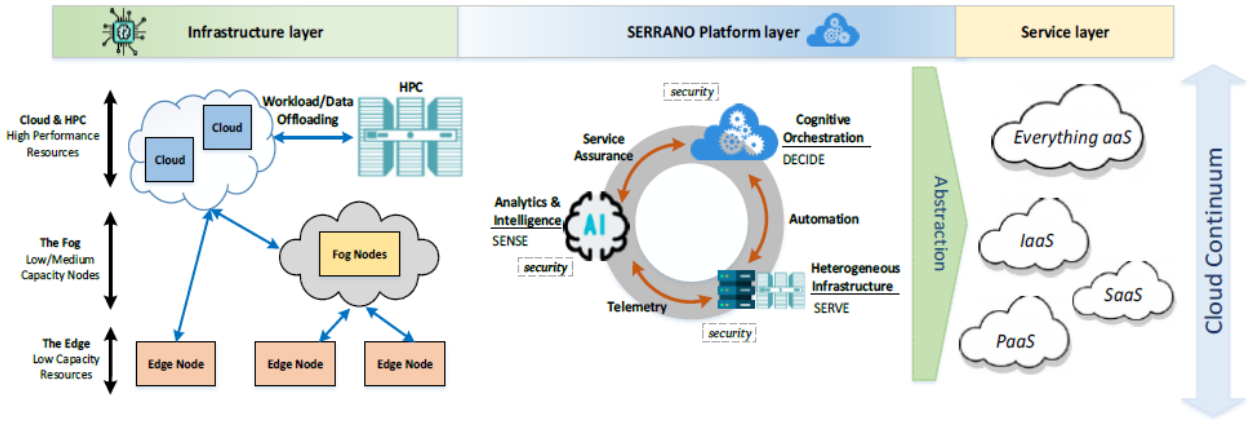

Figure 1 : The SERRANO platform, utilizing edge, cloud and HPC resources and empowering the Everything as a Service (EaaS) notion towards the cloud continuum

2.1 Purpose of this document

The present deliverable (D3.1) presents the current outcome of the ongoing Task 3.1 – "Accelerated encrypted storage architecture" of Work Package 3 – Hardware and Software Platforms for Enhanced Security. T3.1 is associated with the overall architecture design of the acceleration encryption blocks that are later integrated into the use cases as a SERRANO component and part of the SERRANO infrastructure. The encryption building block is operational on a network interface card, which is an integral part of the SERRANO infrastructure.

The objective of D3.1 is to lay down the architecture of the encryption accelerator and conduct an initial emulated benchmark, in liaison with Task 3.2 and Task 3.3, in order to integrate the acceleration engine with the use case demonstrators at a later project phase.

D3.1 presents the requirement of encryption for SERRANO use cases and the architectural considerations to implement such encryptions; also, D3.1 presents how these accelerations are integrated into a TLS and a NVMe context (transport layer and storage, respectively).

D3.1 main initial assumptions are drawn from the deliverables in Work Package 2 that defined the SERRANO architecture, and hence, it may be relevant to touch base with those deliverables to understand the scope of the current document. In addition, this Deliverable

works in conjunction with D3.2 and D3.3, which deal with the secure cloud storage system built using D3.1 and the trust and isolated execution framework that is supported by TLS. This body of work is later integrated into the use cases (the last point in this document in fact describes at high level this technology transfer) for testbed demonstrations.

Hence, D3.1-D3.2-D3.3 are part of the technology development that culminates in D3.4 – Final release of SERRANO secure infrastructure layer, which aims as a SERRANO platform with enhanced security and ready to be tested by the use cases.

2.2 Document structure

The present deliverable is split into seven major chapters:

- Executive Summary
- Introduction
- SERRANO need of encryption
- TLS protocol
- NVMeoTCP
- Acceleration solutions in use cases
- Conclusions

2.3 Audience

This document is publicly available and should be of use to anyone interested in the initial description of the SERRANO components, the specification of the overall SERRANO architecture and the preliminary interfaces. Moreover, this document can be also useful to the general public for obtaining a better understanding of the framework and scope of the SERRANO project.

3 Why do we need encryption in SERRANO

This section describes how the use cases in SERRANO rely on strong encryption and the relevance of accelerating such encryption processes at network level. The use case functionalities review the three SERRANO use cases: secure storage, investment management and Factory 4.0.

3.1 Use case functionalities

SERRANO's overall ambition is to introduce a novel ecosystem of cloud-based technologies, spanning from specialised hardware resources up to software toolsets. This will enable application-specific service instantiation and optimal customizations based on the workloads to be processed, in a holistic manner, thus supporting highly demanding, dynamic and security-critical applications. This subsection describes the use case functionalities in relation to encryption.

3.1.1. Use Case 1: Secure Storage

Security is one of the key aspects of the SERRANO platform showcased by Use Case 1 (UC1). Customer data is kept confidential both in transit and at rest. This guarantee is ensured by a combination of encryption, erasure coding and a set of design principles that minimises the attack surface.

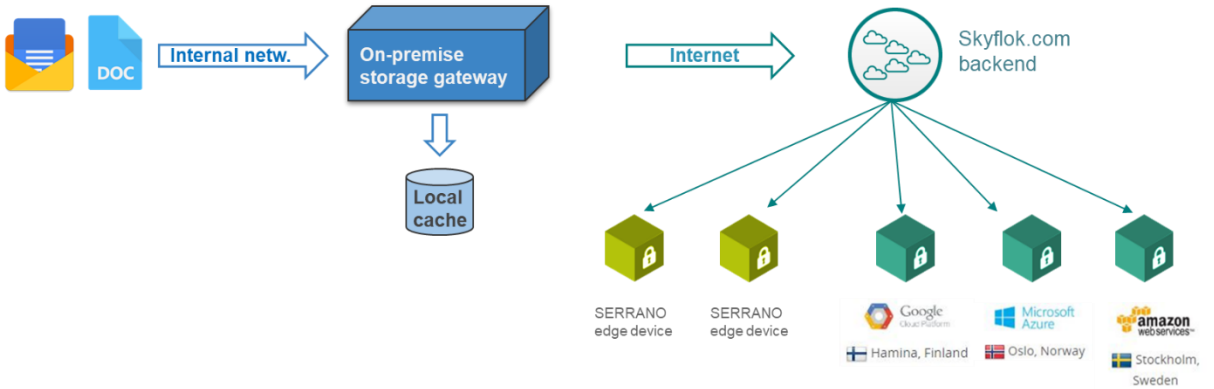

Figure 2 : The components of the SERRANO-enhanced Storage Service

The core functionality of this use case is provided by the SERRANO-enhanced Storage Service, shown on Figure 2, a component of the SERRANO platform that is built to offer S3-compatible file storage. It comprises the Skyflok.com backend, a component developed by Chocolate Cloud (CC) prior to the beginning of the project that is the backbone of CC's main commercial product. SkyFlok is a secure file storage and sharing solution. It lets users select exactly where on the globe, with which cloud providers their data should be stored.

The SERRANO-enhanced Storage Service extends SkyFlok, taking it from a cloud-only solution to one that stretches to the edge. It does this by offering edge storage locations on SERRANO

edge devices and an On-premise storage gateway specifically tailored to enterprise users. As a part of the SERRANO platform, the service takes advantage of platform features and services such as orchestration to deliver automated storage policy creation/assignment and hardware acceleration possibilities. For more details, please refer to Deliverable 2.3 for the general description of the use case and Deliverable 3.2 for the technical details of the SERRANO-enhanced Storage Service.

All communication between the different components of the use case is performed using HTTPS. The encryption performed by the TLS protocol means that a conventional man-in-the-middle attack does not allow access to any user data or metadata.

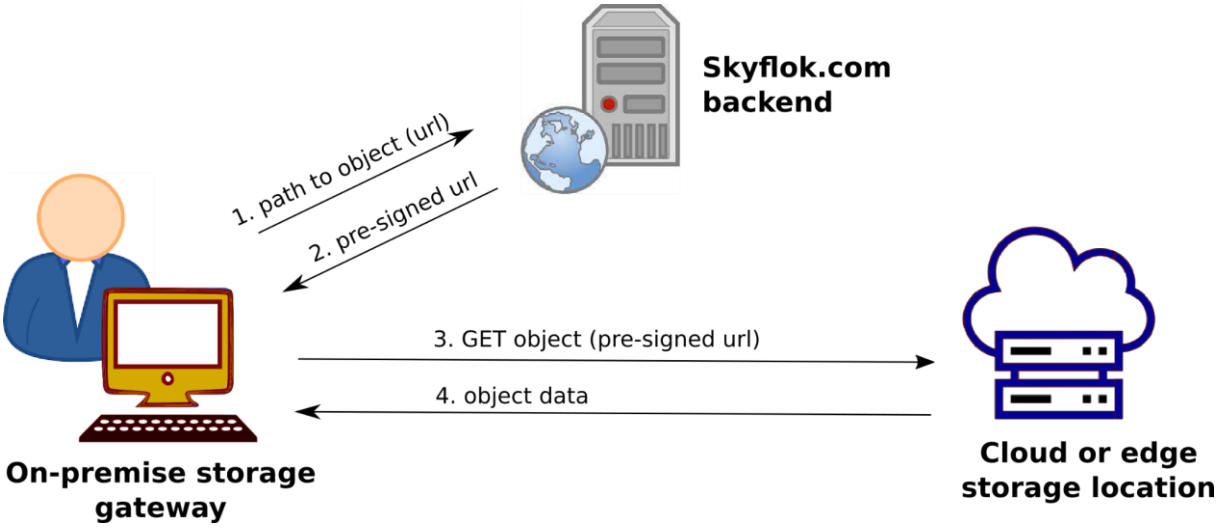

Figure 3 : Using pre-signed URLs to download a file

A key design consideration that has benefits in both scaling, as well as security, is that the owners of the data upload/download fragments of their files directly to/from storage locations. As such, the Skyflok.com backend does not handle data directly. This is achieved by generating pre-signed URLs (temporary URLs in OpenStack Swift parlance) on the backend to connect users directly with the data stored in the storage locations. The in-transit security is further enhanced by the measures taken to ensure security at rest thanks to a very simple design principle that Skyflok follows. Encryption and decryption are always performed on the user's computer. This is followed in SERRANO as well as data is encrypted/decrypted on the on-premise storage gateway. As such, non-encrypted data never leaves the company's infrastructure.

Security at rest is achieved by using AES-GCM encryption with 256-bit keys. It is a widely used scheme that is relied upon by several communication protocols. It is also complemented by a novel erasure coding technique, Random Linear Network Coding (RLNC). RLNC creates ***n*** linear combinations of the data using randomly selected coefficients from the ***k*** original fragments of the file. By distributing these combinations to different storage locations, an attacker is forced to try to gain entry to multiple such locations. At least ***k*** combinations must be retrieved to solve the linear system of equations. After this, an attacker must also break the AES encryption, an impractical task.

3.1.2. Use case 2: Investment management

InbestMe provides investment management services. It processes personal and financial data. This data is very sensitive and because of that security is at the core of its services. InbestMe uses standard security measures for preventing malicious un-authorized access, detection of breach as well as leak of information. To implement these measures, InbestMe uses various well-known techniques such as firewalls, VPNs, encryption of data at rest, network encryption, monitoring of suspicious activity and others. With the current use of the InbestMe system, these techniques do not cause performance problems and bottlenecks. Although the investment management use case will not be able to demonstrate encryption acceleration at its best, encryption acceleration and its easy implementation through the SERRANO remain as an important feature that the use case will benefit.

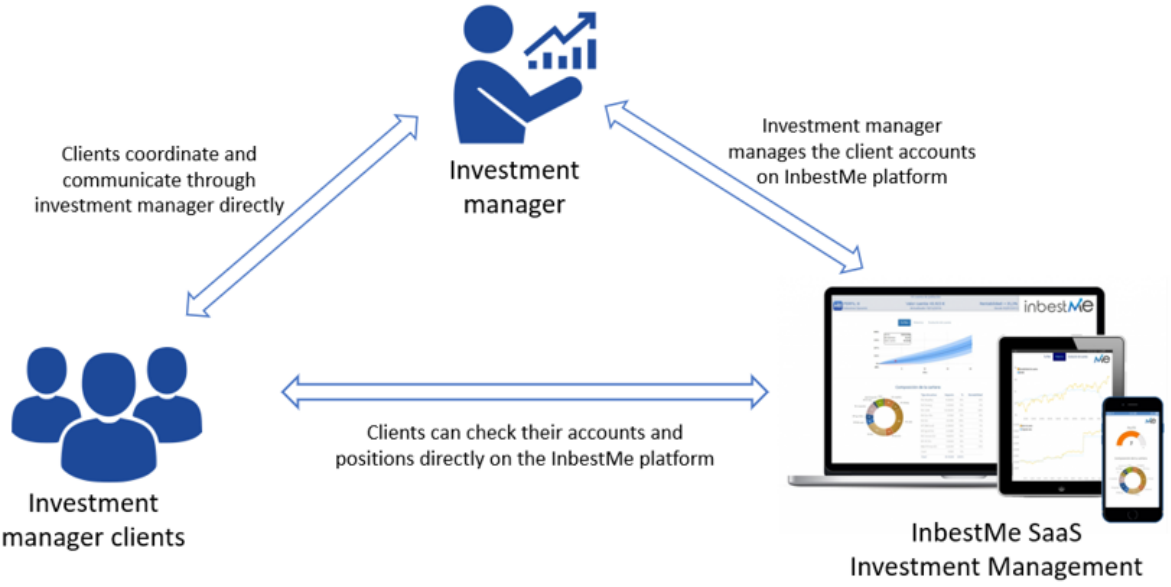

Figure 4 : Investment management SaaS, provided to third party investment managers and respectively to their clients. Due to the very sensitive personal and financial information which is processed, it is indispensable that such information remains always secure and private including during transmission, storage as well as processing.

Specifically, the provision of investment management use case as a SaaS, will benefit most from SERRANO's security features including the secure cloud storage. InbestMe develops a SaaS platform for investment management which can be used by 3rd parties such as banks, brokers and other investment managers (see Figure 3). In such a setup, sensitive information such as personal data and financial information about clients will need to be processed by the SaaS. It is indispensable that such information is secure and remains private for InbestMe as well as other users of the SaaS service. Secure storage services offered by CC will facilitate the implementation of a secure environment for processing sensitive information on publicly available investment management SaaS. Additionally, encryption acceleration would

significantly improve the data access and hide the latency incurred during encryption and decryption.

3.1.3. Use case 3: Factory 4.0

IoT data collected from the machine-tools is considered as core knowledge of the production processes. Although this data per-se is complex to trace to the source machine and its underlying process, clients are very thorough with the security measurements around their data. Moreover, when the data leaves the client's facilities, the security measurements tend to increase.

IDEKO empowers the machines with a SmartBox. The SmartBox is an edge device with cloud connectivity that enables data gathering, an application execution environment and data persistence mechanisms at the edge. Our infrastructure is secured and certified the following way:

- TLS 1.2 everywhere.
- Certified communication channels: Common Criteria, ISO/IEC 15408, ISO/IEC 18045.
- Certified data center: ISO 9001 e ISO 27001.

Local data at rest at client facilities is not usually required to be encrypted, while, when a machine is connected to the cloud, both data transit and data at rest are always encrypted. The certificates used to encrypt communications between the machines and the cloud are validated using "certificate pinning", which guarantees that the communication channel can never be compromised by an intermediate hacker. When SSL inspection is enabled into the client’s infrastructure, connectivity gets suddenly stopped to avoid potential leaks.

Taking into account the above, SERRANO must try to adopt the certifications applied in our infrastructure to maintain the security that the adoption of these standards offers.

3.2 Why do we need to accelerate encryption

In network communications, the term security encompasses the goals of ensuring confidentiality, authentication, integrity, and non-repudiation when referring to data transmission.

Confidentiality is defined as hiding the substance of a message so that only the intended recipient reads it. It is ensured by encryption mechanisms, which provide the means to prevent un-authorized receivers of data from being able to read and use it.

Authentication ensures the identity of a sender. It is ensured by authentication mechanisms, which provide the means for a data receiver to verify that a message was sent by the node that claims to have sent it.

Integrity verifies that the message has not been altered. It is ensured by related authentication mechanisms, which also provide the means for a data receiver to verify that data has been received exactly as it was sent, and has not been modified in transit.

Non-Repudiation guarantees that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message. It is ensured by the use of digital signatures and having an arbitrator (third party) verify the identification information of the sender before transmitting it to the receiver. e was sent by the node that claims to have sent it.

The entrance point of these security aspects starts with data encryption, which serves as a building block to build upon secure network layers (IP), transport layer (TCP), and application layers (HTTP/FTP/SMTF). Hence, ideally, SERRANO's approach is to have by default encryption in all communications. However, encryption is very CPU intensive, and hence, resources quickly run out within a system due to the encryption process, hindering the overall scalability of the system and limiting its capacity. In particular, if encryption processes are conducted at the CPU, the CPU can not be in fact utilized for critical operations that are non-networking related but rather related to the applications (in our case, SERRANO use cases).

Hence, in SERRANO we developed autonomous offloads for encryption and decryption; these offloads allow us to reduce the CPU overhead. The following subsections describe these developments.

4 TLS protocol

4.1 In-line acceleration for data-in-motion protection

In SERRANO, data generated by CC API will traverse the network to their fragmented destination storage points; hence, we need to implement inline encryption offload with NICs, so the encryption is done at the network layer and not by the local CPU generating the traffic. In this section, we will propose and describe solutions targeting both IPsec and TLS. In IPsec, we provide both full data-path encryption offload and standalone encryption-only offload. In TLS, we provide standalone encryption-only offload as not to offload L4 functionality, such as TCP. In all solutions, we focus on the AES-GCM cipher, which is a modern cipher that is very efficiently implemented in both CPUs and silicon (See Deliverable D3.2).

Securing network data is increasingly important as user data confidentiality and integrity is required to preserve privacy and protect against malicious actors. However, the cost of protecting data may overwhelm service providers that need to balance resources for core AI tasks against network protection functions. By offloading the data-path inline, we break this trade-off, eliminating data protection overheads, and freeing processing units to focus on core AI tasks.

Our inline encryption approach is ideal for network processing as the overhead to perform offload is minimal compared to any available acceleration alternative: on-CPU or off-CPU.

On-CPU acceleration using dedicated Instruction Set Architecture (ISA) extensions, such as Intel AES-NI, are widely used to improve AES cipher performance showing ~7x improvement. But, even the highly optimized AES-NI cannot avoid the overheads imposed by encryption and still accounts for significant portions of CPU cycles to compute. For instance, measuring the cycles spent on TLS encryption using AES-NI for an IPerf TLS benchmark shows that more than 50% of cycles are spent encrtypting/decrypting data.

Off-CPU acceleration using dedicated PCIe cards, such as Intel Quickassist Technology (QAT), are common for many compute heavy operations such as RSA, SHA, AES-CBC, and AES-GCM. But, we find that off-CPU accelerators are less efficient compared to on-CPU accelerators (Figure 5). Additionally, off-CPU accelerators require significant parallelism to operate effectively and to obtain it programmers often need to re-engineer their code for more parallelism[1].

[1]X. Hu et al., QTLS: high-performance TLS asynchronus framework with Intel QuickAssist technology," PPoPP'19, February 2019, https://dl.acm.org/doi/10.1145/3293883.3295705

Cipher	QAT 1 thread	QAT 128 threads	AES-NI 1 thread
AES128-CBC-SHA1	249	3144	695
AES128-GCM	249	3109	3150

Figure 5 : Comparison of on-CPU (AES-NI) and off-CPU (QAT) acceleration using a single core with various ciphers and parallelism.

4.1.1. IPsec offload

IPsec is a popular encryption protocol for IP packets defined across numerous RFCs (4301, 4303, 4106). There are two modes for IPsec encryption: transport (see Figure 6) and tunnel.

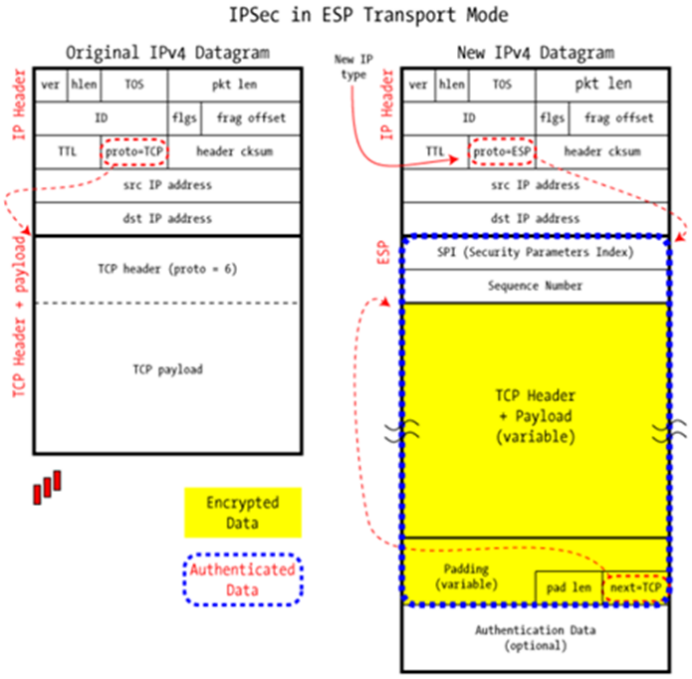

Figure 6 : NIST Round 3 PKE Algorithms.

The NICs used in SERRANO provide two flavors for inline IPsec offload to address different needs:

1. Partial IPsec data-path offload: encryption/decryption.
2. Full IPsec data-path offload: ESP encapsulation/decapsulation, encryption/decryption, replay-protection on receive, sequence number generation on transmit.

Partial offload provides good performance while maintaining maximum flexibility. Software encapsulates/decapsulates packets, while also handling IP fragmentation and other exceptional cases in a timely manner.

Full offload provides maximum performance as it offloads more functionality, however it is also limited to the features available in hardware. The most important limitations are replay window size and limited support for IP fragmentation using software fallback for reassembly which may result in packet loss due concurrent arrival of fragmented and non-fragmented packets that go through separate paths. Therefore, the applicability of full offload is mainly for tunnels that can guarantee no fragmentation, such as traffic between VMs in data-centers.

4.1.2. Partial IPsec offload encryption offload

In partial IPsec offload, NIC hardware encrypts ESP packet data as it goes through the wire (Figure 7). Packets are sent from software as plaintext ESP packets that contain the ESP header but not the ESP trailer. In turn, the NIC encrypts/packet data, replaces plaintext with ciphertext, and adds the ESP packet trailer with its authentication tag field. On receive, the process is symmetrical. Packet are received decrypted with an indication of authentication tag check result. These are passed to software which decapsulates ESP headers and passes inner packet data to higher layers while skipping decryption if hardware already performed it.

Encryption offload enables NIC hardware to observe packet fields in plaintext and operate on them, providing additional performance critical offloads such as checksum and segmentation. Composing partial IPsec encryption offload with segmentation and checksum offload requires NIC parsing to skip intermediate ESP headers and update the checksum according to the correct set of transport and network headers. Furthermore, segmentation offload requires NICs to advance ESP header sequence numbers and IVs with each packet, and to encrypt packets accordingly.

Key management is mostly unaffected by IPsec offloading. The IPsec keying daemon notifies the NIC driver about new SAs, and the driver will verify Security Associations (SAs), and offload/unoffload them accordingly.

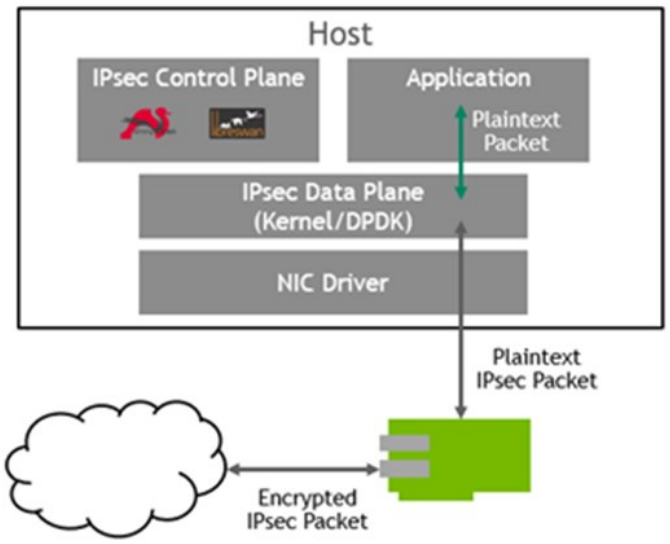

Figure 7 : IPsec partial offload.

4.1.2. Full IPsec offload

In full IPsec offload, the NIC performs all IPsec protocol operations: (de)encapsulation, replay protection, sequence number generation, (de)encryption, and notifying software about the need to change keys when some user defined limits are reached (Figure 8).

On transmit, software sends TCP/UDP packets that are oblivious to IPsec. NIC HW identifies packets that require IPsec using the offloaded selectors and applies IPsec transport/tunnel mode to these packets using offloaded SAs.

Similarly, on receive, ESP packets are decrypted, replay checked, decapsulated, and passed to their target. The inner TCP/UDP packets are received as plaintext as if no IPsec encapsulation took place.

Key management is mostly unaffected by full IPsec offloading. The IPsec keying daemon notifies the NIC driver about new SAs and selectors, and the driver will verify they can be offloaded, and offload/unoffload them accordingly.

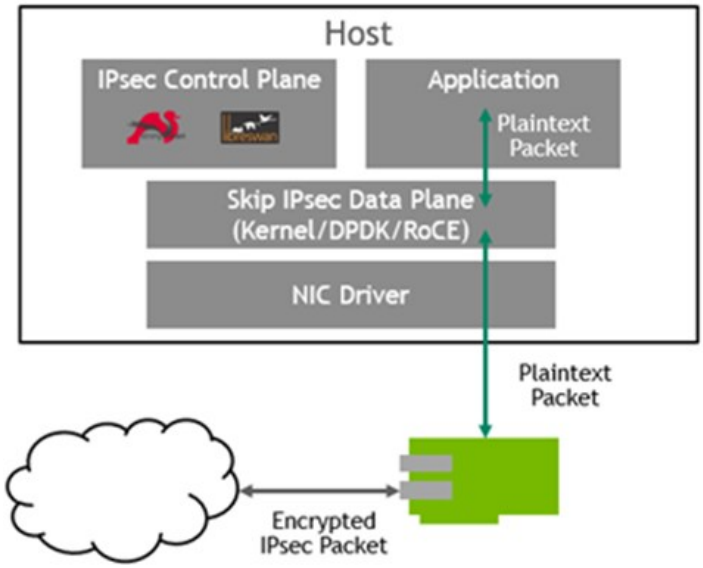

Figure 8 : IPsec full protocol offload. NIC HW perform IPsec encapsulation/decapsulation, encryption/decryption, replay-protection, and ESP sequence number generation.

4.1.3. Transparent full IPsec offload

The killer application of full IPsec offload is to provide IPsec as a service to VMs, such that their SRIOV traffic gets automatically encrypted when it goes to the wire. This service is provided with zero CPU overhead, and it can scale well with the number of VMs.

Full offloading of IPsec without supporting IP fragmentation requires the use of an overlay (Figure 9). IP fragments must be avoided as NIC HW cannot (de)encrypt them, but this may be impossible when unaware VMs send/receive packets without knowledge of full IPsec offloading. To overcome this issue, we use an overlay (VXLAN, GRE, etc.). Such overlays offloads are already provided today by NICs, and they guarantee no fragmentation as packets are encapsulated in IP/UDP.

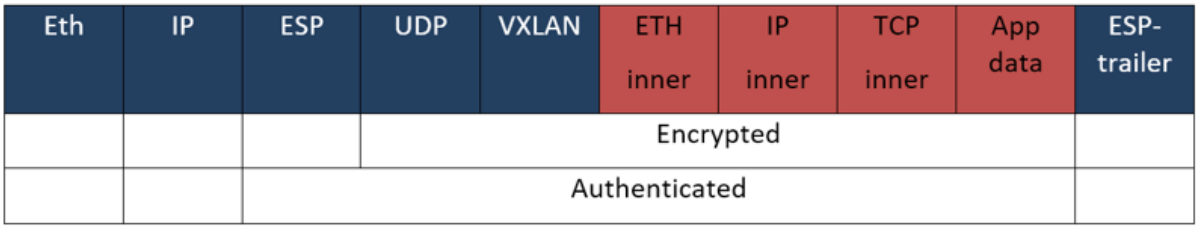

Figure 9 : VM's TCP packet (red) is encapsulated in VXLAN that is encapsulated and encrypted with transport mode ESP (blue). The outer headers are added by NIC hardware.

4.1.4. TLS offload

We have several goals in offloading TLS encryption, decryption, and authentication:

- TCP transparency
- Handling loss and reordering

We aim for TCP transparency to achieve interoperability with existing network stacks and avoid the pitfalls of existing TCP Offload Engines (TOEs) that depend on offloading all functionality and struggle to keep up with constantly changing TCP/IP features such as congestion control. We call our approach autonomous TLS offload for it is independent of other layer offloading.

Figure 10 presents the software architecture at a high-level and contrasts our autonomous TLS offload with the state-of-the-art TLS baseline. On transmit, applications use the TLS baseline to encapsulate data in records and encrypt/decrypt data on the CPU as part of TLS library operations, and then pass data down to the TCP/IP stack which segments it according to the network Maximum Transmission Unit (MTU) and sends it to the wire. In contrast, in autonomous TLS offload, data is passed through the TLS library unmodified. The TLS library only encapsulates data with its record header and trailer; filling the header while leaving the trailer to be filled by NIC HW. The TCP/IP stack operation is unmodified and TLS records are segmented as before. Finally, as NIC HW sends data to the wire, it replaces plaintext with ciphertext and fills the authentication tag at the trailer. As a result, packets on the wire look the same as if no offload took place. The receive path is symmetric; the NIC decrypts packets as they are received, providing plaintext data in TCP segments that are passed to the host. In addition, the receive path provides an indication of authentication success via a single bit of information. The TLS layer verifies that this bit is set for all offloaded packets to ensure that offload was successful, and that decrypted data is authenticated.

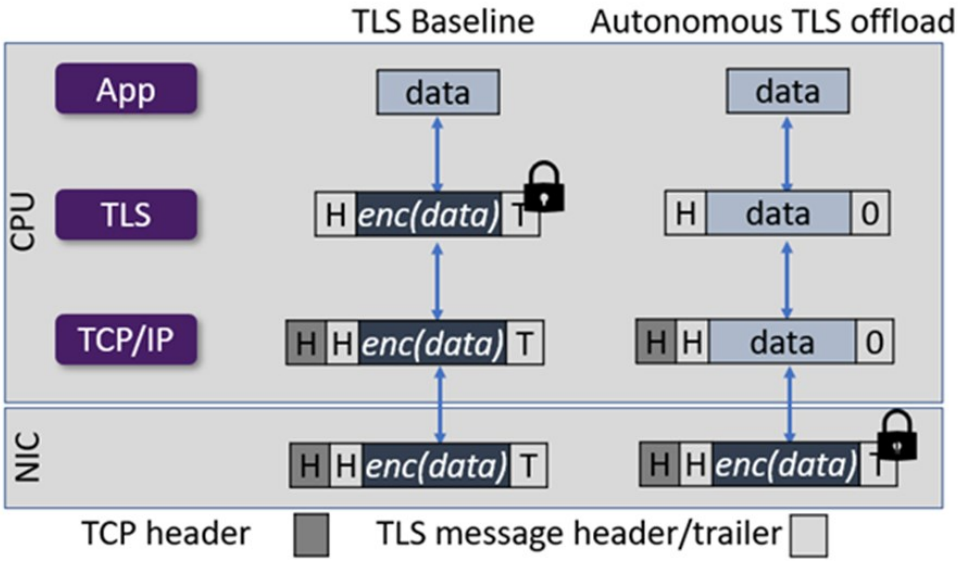

Figure 10 : Autonomous TLS offload compared to baseline.

4.1.4. TLS offload

We measured TLS's offloadable overheads: encryption, decryption, and authentication, which we collectively denote as "crypto" operations. For this purpose, we use iperf [1] (open source), which measures the maximal TCP bandwidth between two machines, and which we modified to support OpenSSL. We use 256 KiB messages at the sender and ensure that the server's core always operates at 100% CPU. Each message consists of a sequence of TLS records, which can be as big as 16 KiB.

Figure 11 shows the results. Bigger TLS records reduce the weight of network stack processing relative to the crypto operations, making the potential offload benefit more pronounced at the right. Typically, network stacks operate more efficiently when sending than when receiving, because batching is easier; the receive side has to work harder. Consequently, the potential benefit of offloading is higher for transmitting (≤74%) than for receiving (≤60%). With real TLS offloading, we find that iperf's single-core throughput improves by 3.3x and 2.2x upon transmit and receive, respectively, relative to the non-offloaded baseline. When saturating the NIC with multiple iperf instances, CPU utilization respectively improves by up to 2.4x and 1.7x. This improvement in the NIC performance will be of great benefit in SERRANO, as more TLS links can be established simultaneously.

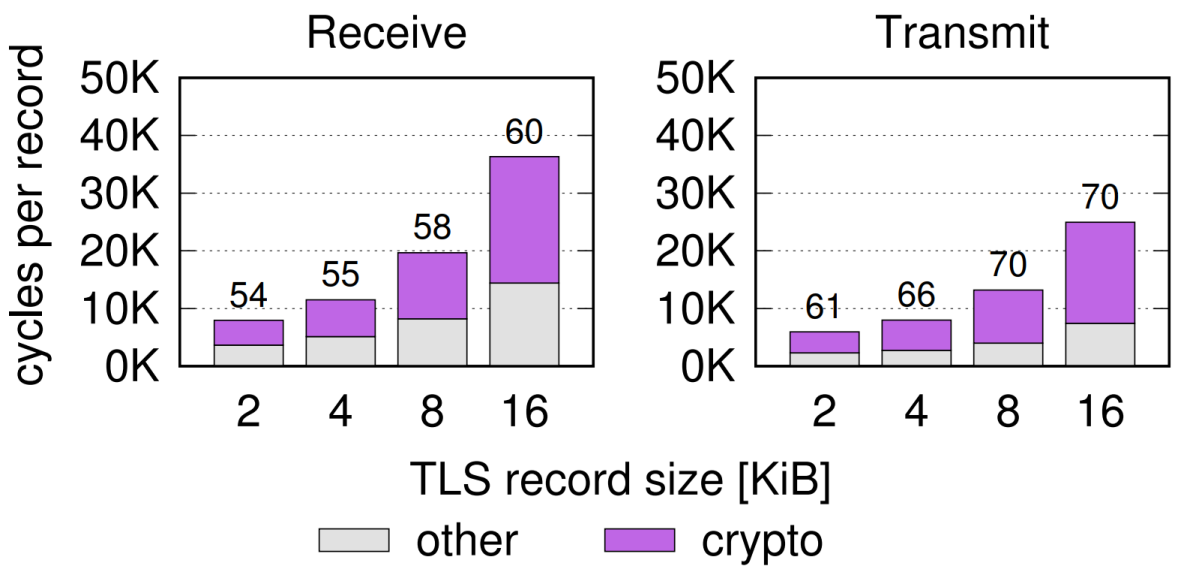

Figure 11 : Kernel-TLS/iperf per-record cycles when using AES-GCM crypto operations (encryption, decryption, and authentication using AES-NI); standard deviation is between 0%–8.2%.

5. NVMeoTCP protocol

5.1 NVMe fundamentals

Access to memory blocks for storage and retrieval operations are key for AI processes which need to constantly look up and storage intermediate processing points. It is hence also a main bottleneck for AI scalability, since these operations are very demanding in terms of clock cycles (i.e. it takes take to store and retrieve). The following figure highlights the latency for write/retrieve for different processing, memory and storage technologies.

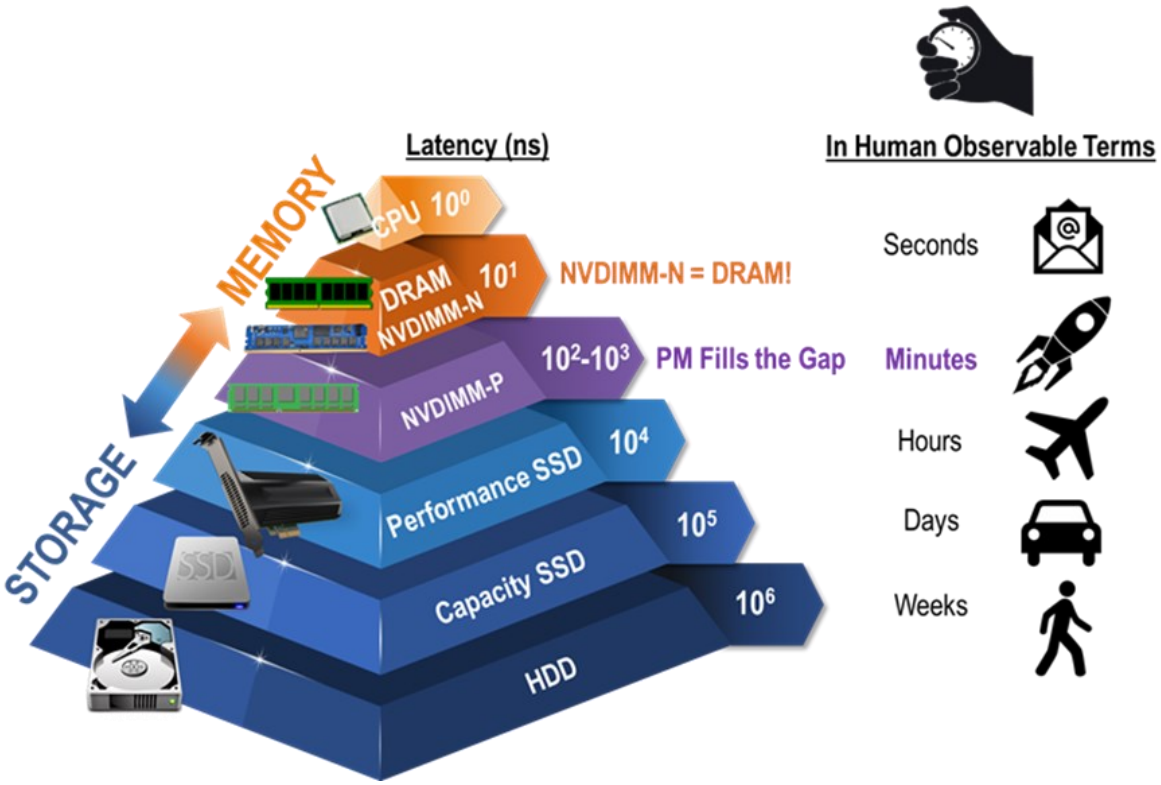

Figure 12 : Latency pyramid for different memory technologies.

NVM Express (NVMe) or Non-Volatile Memory Host Controller Interface Specification (NVMHCIS) is an open logical-device interface specification for accessing non-volatile storage media attached via PCI Express (PCIe) bus. The acronym NVM stands for non-volatile memory, which is often NAND flash memory that comes in several physical form factors, including solid-state drives (SSDs), PCI Express (PCIe) add-in cards, M.2 cards, and other forms. NVM Express, as a logical-device interface, has been designed to capitalize on the low latency and internal parallelism of solid-state storage devices. Architecturally, the logic for NVMe is physically stored within and executed by the NVMe controller chip that is physically co-located with the storage media, commonly these days an SSD. By its design, NVM Express allows host hardware and software to fully exploit the levels of parallelism possible in modern SSDs. As a result, NVM Express reduces I/O overhead and brings various performance improvements relative to

previous logical-device interfaces, including multiple long command queues, and reduced latency. The previous interface protocols were developed for use with far slower hard disk drives (HDD) where a very lengthy delay (relative to CPU operations) exists between a request and data transfer, where data speeds are much slower than RAM speeds, and where disk rotation and seek time give rise to further optimization requirements.

In the context of SERRANO, the cloud storage use case heavily benefits of NVMe accelerations, as effectively, storage and retrieve is in itself the main functionality of the use case. Hence, it is very relevant to ensure that storage cloud contain such accelerations, as it reduces the overall latency and improves the user-experience due to a real-time feeling.

In order to reduce the overall memory utilization, hardware offloading of the memory operations is desirable. The following figure highlights the high-level topology architecture of such attempt on a smart network interface card. Details on the network interface card employed in SERRANO can be found in Deliverable 3.2.

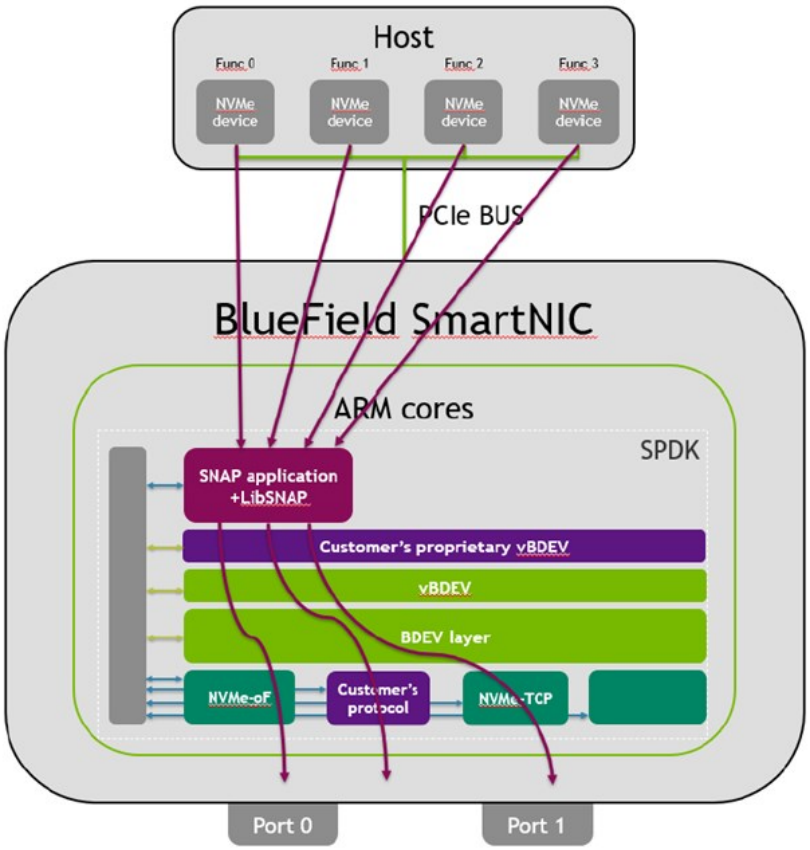

Figure 13 : SNAP application in a smart NIC handling memory utilization.

In this context, NVMe SNAP (Software-defined Network Accelerated Processing) enables hardware virtualization of NVMe storage. NVMe SNAP brings virtualized storage to bare-metal clouds and makes composable storage simple. It enables the efficient disaggregation of compute and storage to allow fully-optimized resource utilization. NVMe SNAP logically presents networked storage, such as NVMe over Fabrics (NVMe-oF), as a local NVMe drive. This allows the host OS/Hypervisor to use a standard NVMe-driver instead of a remote networking storage protocol. The host benefits from the performance and simplicity of local NVMe storage, unaware that remote Ethernet or InfiniBand connected storage is being utilized and virtualized by NVMe SNAP. Furthermore SNAP may apply sophisticated logic and data protection mechanisms (mirroring, compression, data-de-duplication, thin-provisioning, encryption, etc.) to the network storage that it virtualizes as local NVMe.

5.2 NVMe-oF target offload

NVMe-oF target offload is a hardware feature that terminates the target-side NVMe-oF protocol and apply the received IO requests on a set of NVMe drives accessible from the PCIe bus using peer-to-peer PCI communication. This HW feature replaces the software flow implemented in many NVMe-oF target implementations: receiving the NVMe-oF IO request from the network controller, parsing it, generating a block request and posting it on the relevant NVMe device.

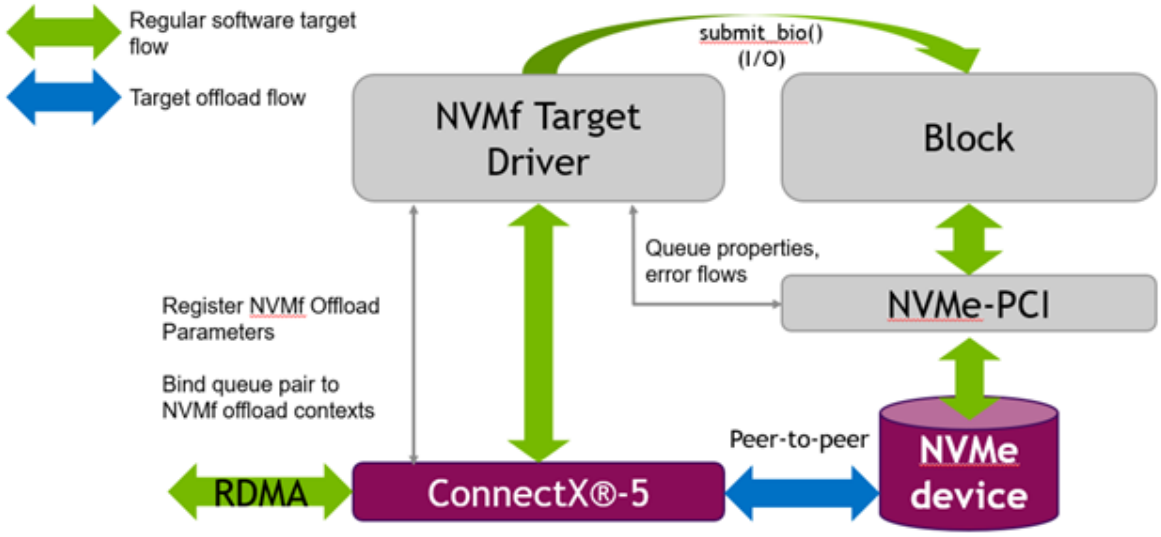

Figure 14 : SNAP application in a smart NIC handling memory utilization.

Using this feature, the host software is completely not involved in the data path of serving those NVme-oF IO requests, and CPU cycles can be used for other activities. The system will allow to perform three distinct operations:

- **Integrity offload**: In high-end storage it is required to keep integrity field with each block of data. This integrity field holds some kind of checksum calculated from the block

of data, and possibly other metadata on the block, so that when the block is later retrieved from the storage it can be verified that nothing changed it it. This offload adds, verifies, removes and transform such integrity fields, on the path of sending and receiving data to/from NVIDIA-Mellanox NICs.

· **AES-XTS storage crypto**: This crypto offload engine allows encrypting and decrypting data for storage purposes, on the path of sending and receiving data to/from NVIDIA-Mellanox NICs (See Deliverable 3.2). It supports practically unlimited number of data encryption keys (DEKs), which makes it good fit for multi-tenancy use cases: each tenant and each volume exposed to a tenant may have a dedicated DEK for encrypting the data. Task 2.3 presents part of the AES offload, particularly related for secure communications.

· **Compression and decompression**: These engines allow offloading some compression and decompression algorithms to the hardware. They operate in a memory-to-memory manner.

5.3 Preliminary performance of NVMe accelerations

We emulated NVMe-TCP offloading under the following conditions: (1) setting the value of all stored data to be a repetitive sequence of an 8-byte “magic” word (0xCC...CC); (2) modifying NVMe-TCP receive side to refrain from copying incoming “magic capsules” (that start with the magic word) to their target buffers, and also; (3) skipping CRC computation and verification for magic capsules. These conditions generate a close-to-reality scenario, which can be later be benchmarked during the testbed work.

The subsequent uses nginx http web server [2] (web server that can be used as a reverse proxy or load balancer), configured to serve files from an ext4 filesystem mounted on our NVMe-TCP block device. We pre-populate ext4 with “magic files”, which exclusively contain magic word sequences – in SERRANO use case these would be the end users employing the storage system. We set the size of magic files to be an integral multiple of 4 KiB, and we configure nginx clients to only request these files. We also set ext4 read-ahead to the file size, such that there are no block requests that exceed this size. Neither the kernel of the server machine, nor nginx and its clients care about the content of the files that they send/receive. Ext4 does not collocate metadata within the 4KiB blocks of magic files, so it is indifferent to whether their content is copied to the server’s page cache; nginx, which sends this page cache content to its clients, is likewise indifferent to the content; and the clients do not actually use the content either.

Figure 15 shows the improvements experienced by nginx with the NVMe-TCP offload. The figure shows different amount of CPU cores. We can observe that baseline operation (no-offloads) in purple is lower than off-loading ON operation (green), reaching a 44% improve in the linerate capacity for long file sizes (the longer the file size the higher the improvement).

We can also observe (case c) that the amount of busy cores is reduced when using the offloading, which allows for a higher utilization of the system, and hence, better scalability.

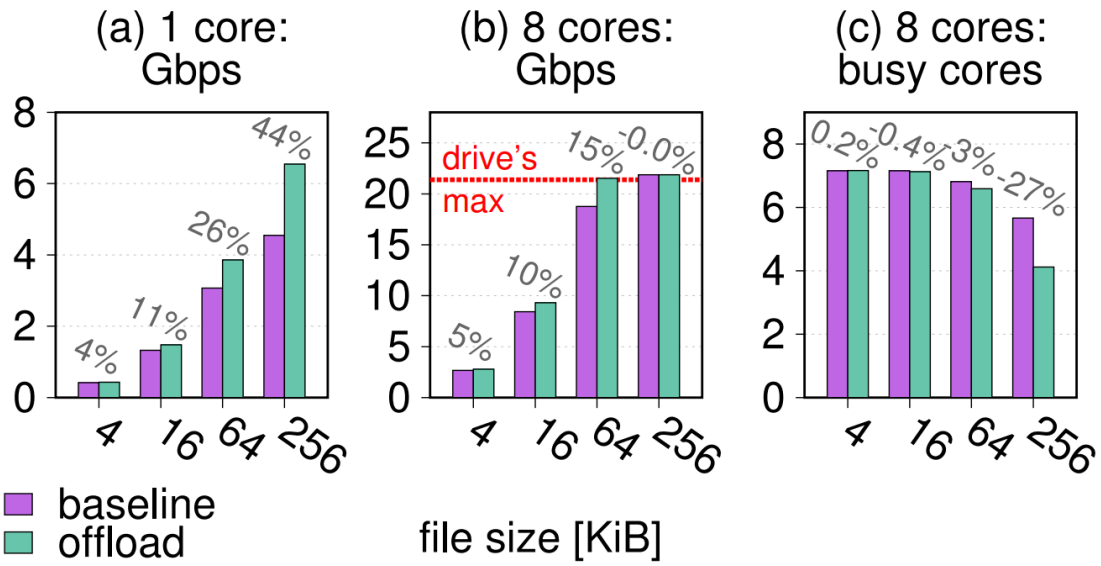

Figure 15 : Nginx improvements with the NVMe-TCP offload. None of the files reside in the server's page cache (configuration $C1$), so the throughput is bounded by the drive's maximal bandwidth. Bar labels show offload improvement over the baseline.

Figure 16 shows the results of combining TLS offloads with NVMe offloads, this is to say, securing the tunnelling along with offloads for storage simultaneously. The results are equally relevant: full offloading allows for a 2.8x increase in the linerate capacity of a single call when using long files. Also a 41% reduction on number of busy cores is achieved when using long files, which means the capacity of the system is almost doubled.

Needless to say, real life traffic is normally a combination (depending on the traffic profile) of file sizes, and hence the real-performance is probably balanced out between the cases. This leads to the following hypothesis: is the SERRANO system better of by filling up file sizes to operate under long file sizes (256 KiB) always? Can we have a machine learning protocol that taking into account telemetry data from the NIC can feed the SERRANO storage system what is the optimal file size for a given moment?

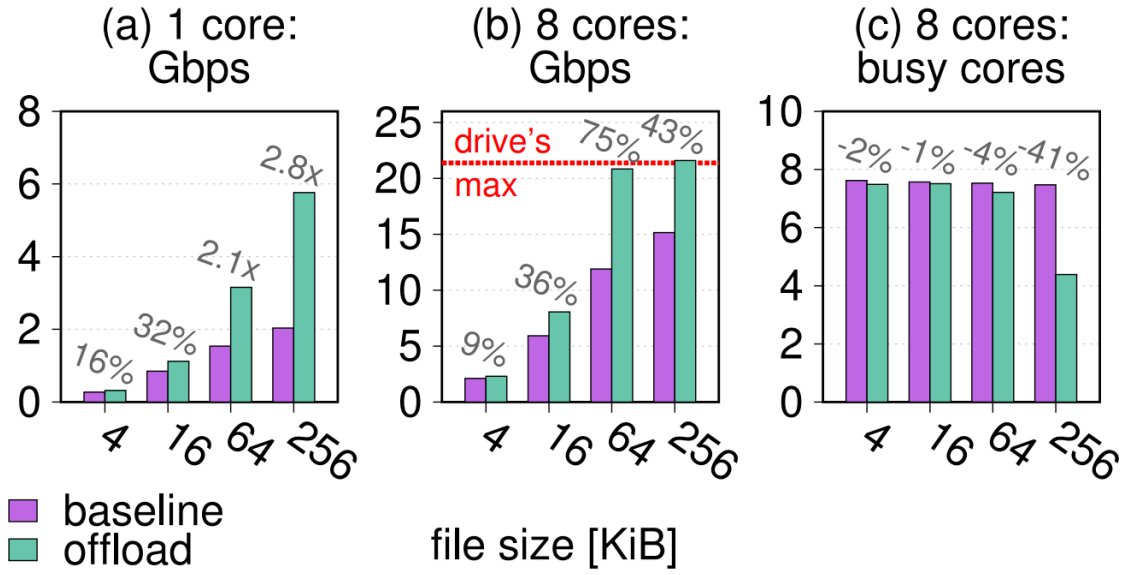

Figure 16 : Nginx improvements obtained with the NVMeTLS combined offload.

6 Acceleration solutions in use cases - technology transfer

The developments described in this Deliverable 3.1 are further integrated into the use cases, which are described in Deliverable 3.2 and Deliverable 3.3. For example, the secure storage use case is implementing encryption accelerators to improve the user experience in terms of latency of users while increasing the security levels of the stored data. One of the key focuses of "Use Case 1: Secure Storage" is to showcase the results of Task 3.1 (described in this deliverable) and Task 3.2 (described in Deliverable 3.2). The use case will include a demonstrator that highlights the importance of offloading some of the computations performed by the TLS protocol. In particular, the components of the use case will be able to perform TLS encryption and decryption using the CPU. This will act as a baseline. Furthermore, if Nvidia's custom hardware is available, these computations can be offloaded to this specialised resource. The integration will happen seamlessly thanks to a software library developed by Mellanox.

The use case will showcase the benefits of this approach by measuring decreases in CPU usage. It will also measure the effect on other storage service-related QoS metrics such as read and write latencies or total achievable throughput. Beyond this task, the use case will also feature the acceleration of kernels related to file encryption, erasure coding and compression and its components will integrate with the SERRANO orchestrator as well. Both of these aspects will be highlighted using the demonstrator.

There are a few key points to consider regarding the integration of various components that are related to accelerated storage:

- **Early identification of incompatibilities**: this can be achieved by providing clear documentation of integration requirements and interfaces to be used.
- **Use of containers for virtualization:** containers can abstract away from specific system requirements and provide a clear way to repeat the steps of a deployment and identify system requirements that could otherwise be missed.
- **Sufficient integration testing**: integration testing is invaluable in identifying possible issues and missed specification items. It can also be used as reference to ensure compatibility between components even before the actual integration takes place. Integration tests can be implemented and executed without requiring other components to be present. This approach will allow the tests to be executed at any occasion without requiring a deployment environment with sufficient resources to support more than one component. Ways that are usually employed to allow the integration testing of a component in isolation include the creation of mock external components with similar interfaces to the real ones as well as configuration options to disable types of functionality that depend on other components and external services.
- **sufficient documentation of functionality and interfaces**: the provided functionality should be documented in a clear way. Interfaces should preferably follow a well-

established standard that corresponds to specifications for documentation and testing, such as Open API v3.0.

These issues are further identified in Deliverable 3.2 and Deliverable 3.3.

7 Conclusions

Deliverable 3.1 reports on the work performance in WP3 during the first year of SERRANO project.

The deliverable presents the SERRANO motivation to operate with encrypted data and describes its needs for each of the use cases. Then, a justification for encryption acceleration is provided in the context of the use cases.

The deliverable then presents the overall approach for the encryption acceleration in two particular cases, for TLS and for NVMe, to establish secure data-paths and to store data, respectively. The implementations are benchmarked through emulation of performance. These encryption solutions are compatible and drown from the specifications of the SERRANO platform architecture (presented in D2.5 “Final version of SERRANO architecture”).

This document will be further used as a guideline for the technical activities in adjacent workpackages and the technology transfer towards the use case implementations. In particular, it is relevant to highlight that this deliverable is part of a three-sided body of work that is being put forward together with Deliverable 3.2 and Deliverable 3.3, and hence, it is recommended to read them together.

The developments in encryption acceleration will be evaluated in full during the testing of the use case demonstrators.

References

[1] Iperf.fr , Open Source active measurement tool for IP networks.

[2] https://docs.nginx.com/, Web server (reverse proxy, load balance).