Securitization of cloud, edge and IoT communications through hardware accelerations/offloadings

Boris Pismenny, J.J. Vegas Olmos, Yoray Zack and Liran Liss
NVIDIA Corporation
Software Architecture
Yokneam, Israel

Afra Dömeke, Catalina Ioana Stan, Carlos Rubio Garcia, and Idelfonso Tafur Monroy
Department of Electrical Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands

Panagiotis Kokkinos, Aristotelis Kretsis, and Manos Varvarigos
Institute of Communication and Computer Systems National Technical University of Athens, Athens, Greece

***Abstract*—Networking is enabling a continuum comprising cloud, edge and last-mile systems. This continuum allows services and applications to join the fabric at any point and have access to processing power independently of the location of entrance. To enable this paradigm, we need to make sure we have a networking fabric capable of transporting and access network resources seamlessly and communication channels that are securitized, ensuring confidentiality and integrity of data. A challenge in securitization is that it takes computing resources away, and hence it may reduce the overall network performance. Hardware accelerations or offloads enable to reduce the burden on processing resources of securitization processes; in this contribution, we present how IPsec and TLS protocols provide a tool for a high level of securitization of the communication channels and approaches to accelerate/offload those protocols.**

Keywords—securitization, encryption, cloud computing, edge computing, IoT, communications, acceleration

I. INTRODUCTION

Communication networks rely on control processing units (CPUs) as main engine to support the control and management of all the network stack. For example, if we consider the Open System Interconnection model (OSI model), every single operation that any of the layers conduct use CPU cycles. As networks become more complex in terms of layer composition and supported traffic due to diverse applications, the amount of needed CPU cycles grows. Needless to say, we can scale network capacity by increasing the amount of CPU clocks the CPU can conduct per unit of time – however, aspects such as cost, price, power consumption or parallelization trade-offs make the capacity of CPUs plateau in this respect. Hence, it is very relevant to adopt measures to off-load CPU processes and execute them on dedicated system-on-chips (SoC), either in the form of field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs). For example, in artificial intelligence (AI), machine learning (ML) or deep learning (DL), graphic processing units (GPUs) are found to be much more efficient in conducting processing tasks than CPUs.

In the area of networking, one of the most demanding CPU activities is the securitization of communications [1]. When a connection is established between two network elements, data transmitted over that link is normally encrypted, a process through which data is encoded so that it remains hidden from or inaccessible to unauthorized users or third parties. The gold standard of for encoding is the Advanced Encryption Standard (AES), which is a set of specifications for the encryption of electronic data established by the National Institute of Standards and Technology (NIST) in 2001 [2]. AES is a symmetric key cipher, which means the same secret key is used for both encryption and decryption, and both the sender and receiver of the data need a copy of the key. By contrast, asymmetric key systems use a different key for each of the two processes. The advantage of symmetric systems like AES is their speed because a symmetric key algorithm requires less computational power than an asymmetric one. AES can be used in three flavors: 128-bit, 192-bit, and 256-bit. Each type uses 128-bit blocks, with the difference lying in the length of the key. As the longest, the 256-bit key provides the strongest level of encryption (2^{256} combinations). The three AES varieties are also distinguished by the number of rounds of encryption. AES 128 uses 10 rounds, AES 192 uses 12 rounds, and AES 256 uses 14 rounds. AES 256 uses 40% more system resources than AES 192.

In the context of long-distance communication links, perhaps the utilization of AES or other encryption methodologies do not impact the overall performance in relation to relevant metrics such as power consumption, latency or simply scalability of the network. However, when considering last-mile networks supporting 5G or beyond communication channels or Internet-of-Things (IoT) networks with thousands of disperse network components providing streams of data, the securitization of the network by adding encryption becomes a real challenge [3]. In edge computing platforms, which are now placed close to the last-mile in order to provide low-latency AI-driven services, encryption is a must to ensure isolation and protection among the different users, applications or services utilizing the platform. In the cloud infrastructure, encryption becomes even more relevant since the underlying fabric may be very heterogeneous and is certainly rich in point-to-point links. Figure 1 presents the overall network architecture highlighting these three segments and how securitization is present in all of them.

The remainder of this paper is organized as follows: Section II presents the technologies used for confidentiality and integrity of communications. Section III and Section IV present the features of those technologies and their acceleration/offloads approaches. Finally, Section V provides some short conclusions. During the workshop a tech deep dive will be provided, including experimental results.

This work has been partly funded by the SERRANO (ID 101017168), IoTalentum (ID 953442) and BRAINE (ID 876967) projects, funded by the European Commission.

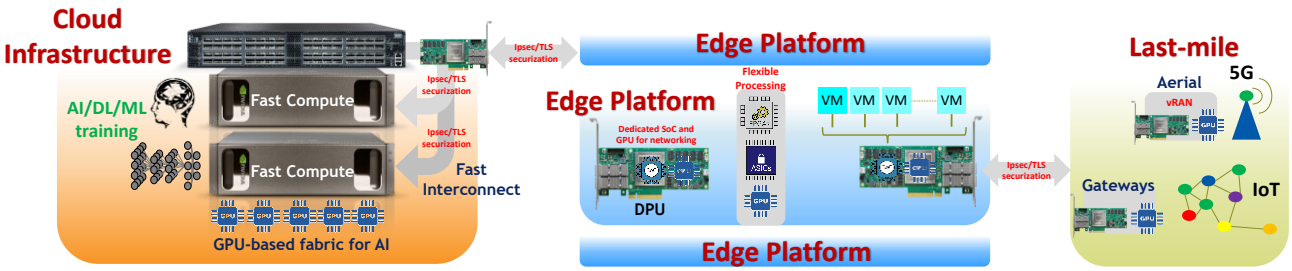


Fig. 1. Cloud infrastructure, edge platforms and last-mile networks co-existing in a seamless continuum fully securizited.

II. Confidentiality and Integrity of Communications

Current networks provide security functionalities at layer 2 OSI and 3 OSI. At layer 2 OSI, internet protocol security (IPsec) serves to authenticated and encrypt packets of data to provide secure encrypted communications between two network elements over an Internet Protocol (IP) network. At layer 3 OSI, transport layer security (TLS) are cryptographic protocols designed to provide communications security over a computer network, with an emphasis on privacy and data integrity between the communicating network elements. Table I shows the summary of technical aspects in relation to confidentiality and integrity of IPsec and TLS.

TABLE I. Summary of IPsec and TLS technical comparison

Feature	IPsec	TLS
Authentication	Yes	Yes
Integrity	Yes	Yes
Confidentiality	Yes	Yes
Configuration	Complex	Straightforward
Interoperability problems	Yes	No
TCP apps support	All	Some
UDP support	Yes	Only Data gram TLS
PKI	No	Yes
Compression	Yes	Only OpenSSL
Client-specific software	Yes	No
Multi-environment support	Sometimes	Yes
Apps filter	No	Yes

IPsec includes protocols for establishing mutual authentication between agents at the beginning of a session and negotiation of cryptographic keys to use during the session. IPsec can protect data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host). IPsec uses cryptographic security services to protect communications over IP networks. It supports network-level peer authentication, data-origin authentication, data integrity, data confidentiality (encryption), and replay protection. On the other hand, TLS are a set of cryptographic protocols designed to provide communications security over network units. TLS in itself runs on top of reliable transport protocols (e.g., TCP), aiming primarily to provide privacy and data integrity.

Hence, IPsec and TLS can work together, and by doing so, we can effectively create point-to-point virtual private network (VPN)-like tunnels between network elements, which is a very powerful paradigm providing confidentiality and integrity of communications even within the shortest or smallest communication links. The following two sections describe approaches to offload the IPsec and TLS protocols reduce the CPU requirements that they impose, hence allowing for a higher network scalability while maintaining all the technical specifications of both technologies.

III. IPsec Offloading

Network elements access the network fabric through network interface cards (NICs), which take care of all network operations. NICs that include system-on-chip processors, a dedicated GPU for network and a switching element are data-processing units (DPUs); DPUs are in fact the smallest data centers. When it comes to IPsec, it is desirable to provide both full data-path encryption offload and standalone encryption-only offload. In TLS, DPUs provide standalone encryption-only offload as not to offload L4 functionality, such as TCP.

In DPUs, the encryption is done in-line, which means that data traversing the network experiences the implemented protocols on-the-move, rather than at rest. 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 encrypting/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 (Table II). 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.

TABLE II. Encryption bandwidth (MB/s) of AES-NI (on-CPU) vs. QAT (off-CPU) accelerators. Results for 16KB blocks with 1 or 128 threads using a single core (2.40GHz IntelXeon E5-2620 v3 CPU)

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

IPsec encryption has two modes of operation: transport (see Figure 2) and tunnel, and they can be partly or fully offloaded.

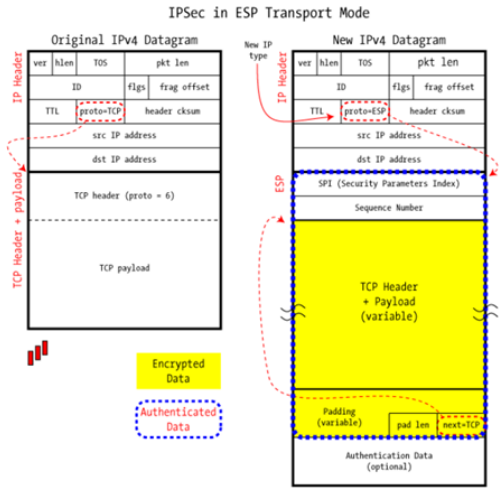


Fig. 2. NIST Round 3 PKE algorithms. ESP: Encapsulating Security Payload.

- Partial IPsec data-path offload: encryption/decryption.
- 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 virtual machines (VMs) in data-centers. The following subsections briefly describe each approach.

A. Partial IPsec encryption offload

In partial IPsec offload, DPU hardware encrypts ESP packet data as it goes through the wire (Figure 3). 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 DPU 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 DPU parsing to skip intermediate ESP headers and update the checksum according to the correct set of transport and network headers. Furthermore, segmentation offload requires DPUs to advance ESP header sequence numbers and initialization vectors (IVs) with each packet, and to encrypt packets accordingly.

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

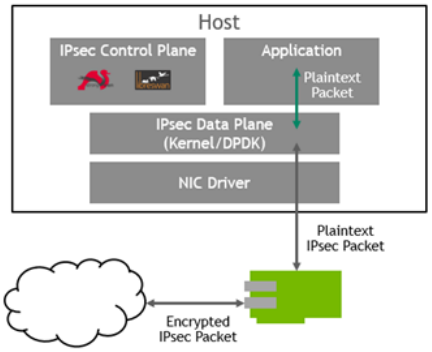


Fig. 3. IPsec partial offload.

B. 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 5.5.2).

On transmit, software sends TCP/UDP packets that are oblivious to IPsec. The DPU 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 DPU driver about new SAs and selectors, and the driver will verify they can be offloaded, and offload/unoffload them accordingly.

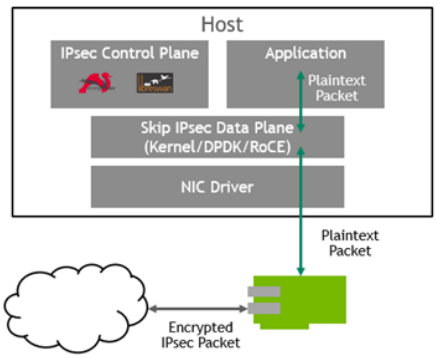


Fig. 4. IPsec full protocol offload. NIC HW perform IPsec encapsulation/decapsulation, encryption/decryption, replay-protection, and ESP sequence number generation

C. *Transparent full IPsec offload*

The killer application of full IPsec offload is to provide IPsec as a service to VMs, such that their single-root input/output virtualization (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 (Table III). IP fragments must be avoided as DPU 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, DPUs use an overlay (virtual extensive local area network - VXLAN, generic routing encapsulation - GRE, etc.). Such overlays offloads are already provided today by DPUs, and they guarantee no fragmentation as packets are encapsulated in IP/UDP with the IP don't fragment bit set.

TABLE III. VM'S TCP PACKET (ORANGE) IS ENCAPSULATED IN VXLAN THAT IS ENCAPSULATED AND ENCRYPTED WITH TRANSPORT MODE ESP (BLUE). THE OUTER HEADERS ARE ADDED BY NIC HARDWARE.

Eth	IP	ESP	UDP	VXLAN	ETH inner	IP inner	TCP inner	App data	ESP-trailer
			Encrypted						
		Authenticated							

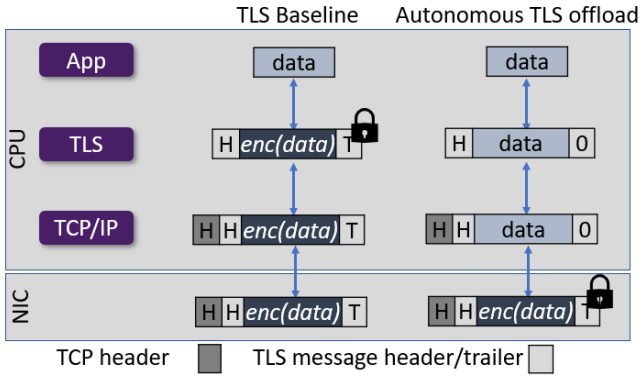
IV. TLS OFFLOADING

TLS is a widely-deployed protocol used for securing TCP connections on the Internet. TLS is also a required feature for HTTP/2, the latest web standard. Kernel implementation of TLS (kTLS) provides new opportunities for offloading the protocol into the hardware. TLS data-path offload allows the DPU to accelerate encryption, decryption and authentication of AES. TLS offload handles data as it goes through the device without storing any data, but only updating context. If the packet cannot be encrypted/decrypted by the device, then a software fallback handles the packet. There are two main goals when offloading TLS encryption, decryption, and authentication:

- TCP transparency
- Handling loss and reordering

DPUs aim for transmission control protocol (TCP) transparency to achieve interoperability with existing network stacks and avoid the pitfalls of existing TCP Offload Engines (TOEs) that depend on offloading all ≤Layer 4 functionality and struggle to keep up with constantly changing TCP/IP features such as congestion control. This approach is called autonomous TLS offload for it is independent of other layer offloading.

Figure 5 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 DPU HW. The TCP/IP stack operation is unmodified and TLS records are segmented as before. Finally, as DPU 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 DPU 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.



.

Fig. 5. Autonomous TLS offload.

V. CONCLUSIONS

The paper presents a brief overview of IPsec and TLS hardware offloading as a mean to securitize cloud, edge and IoT communications while maintaining network performance. During the workshop presentations a deep dive into the technology and its impact on systems will be presented; in particular, experimental results on the different aspects described in the paper will be presented and discussed.

ACKNOWLEDGMENT

The authors would like to express their gratitude to the European Commission for partly funding this research (SERRANO 101017168, IoTalentum 953442 and BRAINE 876967). J.J. Vegas Olmos would like to thank Miguel Sanchez from Mitsubishi Electric for fruitful discussions on encryption and privacy. B. Pismenny would like to thank Adam Morrison from Tel Aviv University and Dan Tsafrir from Technion and VMware Research for their fruitful discussions on the topics of IPsec, TLS and autonomous offloads.

REFERENCES

[1] S. Soliman et al., "Efficient implementationof the AES algorithm for security applications," IEEE International System-on-Chip Conference, April, 2017.

[2] National Institute of Standards and Technology, https://www.nist.gov/

[3] M. Zhong et al., "5G and IoT: towards a new era of communications and measurements," IEEE Instrumntation and Measurements Magazine, vol. 22, Issue 6, December, 2019.

[4] B. Pismenny, H. Eran, A. Yehezkel, L. Lirss, A. Morrison, and D. Tsafrir, "Autonomous NIC offloads," In proceedings of the 26th ACM International Conference on Architectureal Support for Programming Languages and Operating Systems, Detroit, MI, USA, April, 2021.