Securing outsourced VNFs: challenges, state-of-the-art and future directions

Enio Marku, *Member, IEEE,* Gergely Biczók, and Colin Boyd

***Abstract*—It is becoming increasingly common for enterprises to outsource network functions to a third party provider such as a public cloud. Besides its well-documented benefits in cost and flexibility, outsourcing also introduces security issues. Peeking into or modifying traffic destined to the cloud are not the only threats we have to deal with; it can also be desirable to protect VNF code, input policies and states from a malicious cloud provider. In recent years several solutions have been proposed towards mitigating the threats of outsourcing VNFs, using either cryptographic or trusted hardware-based mechanisms (the latter typically applying SGX).**

In this paper, we provide an overview of methods for protecting the security of outsourced network functions. We introduce the challenges and emerging requirements, analyze the state-of-the-art, and identify the gaps between the requirements and existing solutions. Furthermore, we outline a potential way to fill these gaps in order to devise a more complete solution.

***Index Terms*—virtualization, security, confidentiality, middlebox, cloud, outsourcing, VNF, SGX, 5G, RAP, LAP**

I. Introduction

MODERN networks utilize a wide range of network functions (NFs) to perform advanced networking tasks beyond merely forwarding packets. NFs can implement a variety of important components, for example a network address translator (NAT), a proxy or a firewall, and are a vital part of modern networks. Traditional NFs are implemented in dedicated hardware (also referred to as middleboxes), operating at line rate speed and obtain some security from their proprietary nature. However, as network functions are becoming more complex and new services emerge, owning (or renting) traditional middleboxes puts considerable financial and operation & management burden on enterprises. Nowadays, hardware middleboxes are being replaced with software implementations of NFs (resulting in Virtual Network Functions, VNFs, i.e., software middleboxes) running on commodity hardware [1].

The execution of VNFs can be naturally outsourced to cloud providers (see Fig. 1). The combination of virtualization and cloud computing brings advantages

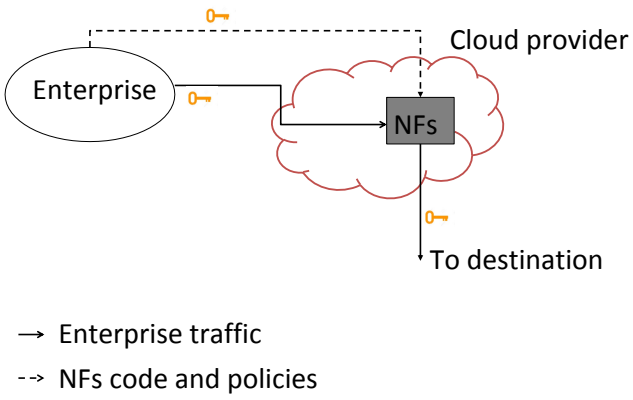

Fig. 1. Outsourcing NFs to a cloud provider domain

in cost, flexibility, scalability, availability, and ease of management. Besides these known benefits, outsourcing of VNFs introduces security issues: malicious providers may peek into the traffic, or even modify or drop packets. Moreover, the policies and the VNF itself are usually provided by the outsourcing enterprise; therefore protecting the code and input policies of VNFs is also desirable. To further complicate things, the outsourcer and the outsourcee could be competitors in certain market segments, making confidentiality protection a sensible requirement even in the case of benevolent but curious stakeholders [2]. Therefore, an important question emerges: *how can we provide security for outsourced VNFs in terms of enterprise traffic, VNF code and input policies, while maintaining a level of performance comparable to traditional hardware middleboxes?*

To address the question above, several solutions have been proposed in recent years. These solutions fall into one of two categories: i) cryptographic approaches [3]–[5] and ii) trusted hardware approaches [6]–[11]. Note that, even though we have carefully examined cryptographic approaches and summarized them in Table 1, we do not focus on them in this paper. This is due to their limited functionality and low performance, making them unsuitable for current deployment in real world enterprise networking scenarios. Furthermore, our proposed solution falls into the hardware approach category.

The main contribution of this paper is threefold: i) we introduce the problem of securely outsourcing network functions to the cloud, outlining its challenges and emerging requirements; ii) we provide an overview of existing trusted hardware based solutions and identify the gaps between requirements and the state-of-the-art; and iii) we propose a novel architecture (currently under implementation) capable of filling these gaps.

The rest of the paper is organised as follows. Section II introduces the necessary background on trusted hardware based VNF outsourcing including architecture, threat model and emerging requirements. Section III details and compares the most promising systems based on Intel SGX. Section IV provides guidelines for developers regarding which existing solution should be used in specific scenarios. Section V briefly introduces SafeLib, a novel design aiming at providing "full-stack" protection and support for all VNF types. Section VI concludes the paper.

II. Trusted hardware approach: system architecture, threat model and emerging requirements

Existing trusted hardware based VNF outsourcing solutions ([6], [7], [9]–[11]) share a common end-to-end and system architecture, and, subsequently, a common threat model stemming from using Intel SGX.

End-to-end architecture. The end-to-end architecture of hardware solutions is depicted in Fig. 2 (see [6], [7] for more details). In these solutions, the VNF is deployed in a cloud acting as Service Provider (SP). First, enterprise traffic is redirected to the cloud for processing at the VNF implementing the service. Next, after the traffic is processed, it is sent back to the enterprise and then to the receiver at an external site. Enterprises usually employ a Gateway (GW) to forward traffic to the SP. This outsourcing setup essentially conforms to the well-known *Bounce* architecture [1].

Note that in Fig. 2 two different security protocols are being used, providing a balance between security and flexibility. The traffic outsourced to a SP for processing introduces a more significant security risk than the traffic between the client and GW. This is because a client usually has a trust relationship with the enterprise, but does not trust a third party to process its traffic; for such a scenario, an IPSec tunnel provides security for all IP traffic, including packet headers (see Section V for more details). On the other hand, SSL/TLS is the superior alternative between client and GW, as it is easier to configure and implement service access restrictions for different clients and specific applications, a key feature for enterprises.

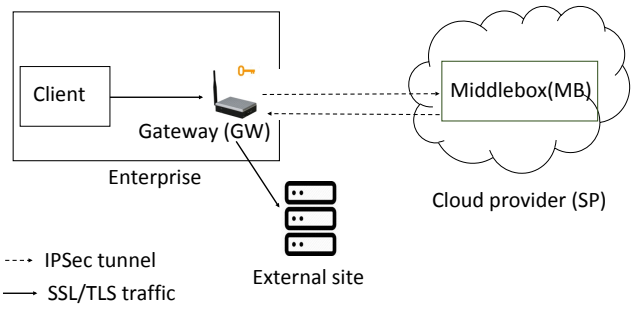

Fig. 2. End-to-end architecture of trusted hardware solutions

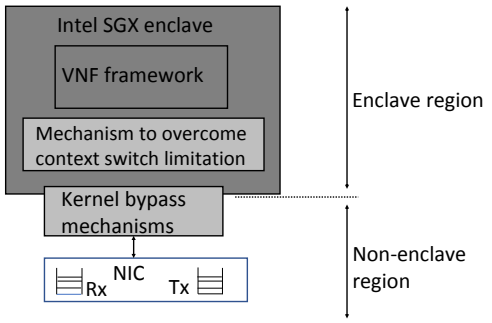

Fig. 3. High-level system architecture of trusted hardware solutions

Some hardware solutions [7], [10], [11] are also capable of providing support for the *Enterprise-to-Enterprise* architecture in which the traffic is sent directly to the GW of the other enterprise (after processing), and then forwarded to the receiver [1]. Such architecture avoids the setup latency of *Bounce*, but it can only be used after both enterprises established a mutual trust. A prime example for such a scenario is the collaboration of two (or more) network/cloud operators jointly delivering a service to a customer via Service Function Chaining (SFC); a key scenario in 5G systems, where the customer might be an industry vertical [2]. Note that our solution [12] aims to support both these architectures.

System architecture. We refer to solutions protecting VNF processing from an adversarial/curious cloud provider based on a trusted execution environment [6]–[11], as *trusted hardware approaches*. All of them follow the same model for VNF outsourcing (see Fig. 1) and have a similar end-to-end architecture (see Fig. 2).

By studying numerous systems, we have devised their common high-level system architecture depicted in Fig. 3. The common denominator in this architecture is Intel's Software Guard Extensions (SGX, [13]) offering hardware-based memory encryption that enables application code and data isolation in memory. Private regions of memory allocated by user-level code are

called *enclaves*; these regions are protected even from processes running at higher privilege levels.

In Fig. 3, boxes of black solid lines represent enabling technologies: dark gray boxes denote the "must have" technologies for all solution alternatives (SGX is commonly used, but the VNF framework can be specific to an alternative), while light gray boxes denote optional mechanisms which can improve VNF performance and are specific to a solution alternative. Every solution partitions its system into enclave and non-enclave regions to find a sweet spot between security and performance. This trade-off stems from two major limitations of Intel SGX.

The first limitation concerns *memory size*. The protected memory region called enclave page cache (EPC) has a limited size of 128 MB (previously 94 MB). Exceeding this memory triggers the procedure of secure paging, and therefore leads to a performance penalty. The goal behind this design choice is to minimize the trusted area in order to reduce the attack surface and let the other many processes utilize the maximum amount of RAM possible. Solutions analysed here overcome this limitation by carefully placing inside the enclave region only modules responsible for handling sensitive information and leaving other parts in a non-enclave region. Note that each solution uses different technologies (i.e., VNF frameworks, networking stacks), and such technologies determine enclave and non-enclave regions.

The second limitation concerns *illegal enclave instructions*. As the most common example, CPU instructions leading to a change in privilege level, e.g., system calls, are not allowed from within the enclave. Therefore, a costly transition between enclave and non-enclave regions is required. The rationale behind this limitation is that the OS is not part of the trusted computing base, hence allowing a jump from trusted enclave code to untrusted OS code would defeat the purpose of SGX. This limitation is handled in different ways by the proposed solutions, nevertheless they can be divided into two categories; i) solutions [6], [8], [9], [11] using SGX *ECALL* (entry point to enclave from non-enclave region), and *OCALL* (allows an enclave to call non-enclave functions and then return to enclave); ii) solutions using an asynchronous interface [7], [10]. Using an asynchronous interface typically offers better performance due to the negative impact of *ECALL* and *OCALL*.

Solutions using *ECALL* and *OCALL* for enclave transitions basically follow a standard SGX developer procedure. An enclave definition language file (EDL) defines *ECALLs* in a trusted section and *OCALLs* in an untrusted section. Solutions using an asynchronous interface to perform enclave transition utilize shared queues which allows enclave threads to send and receive a batch of packets to/from a non-enclave region from within the enclave. Shieldbox [10] uses SCONE to provide such operations, while SafeBricks [7] developed two interfaces (one inside the enclave, one outside the enclave) to provide such operations via the help of two circular queues located at heap memory outside the enclave. Note that the two major limitations of SGX are at odds with each other, making it challenging to overcome both at the same time.

Threat model. All trusted hardware based solutions, including our proposed system, consider a powerful adversary who can compromise the entire software stack of the SP outside the trusted enclave, including privileged software such as kernel and hypervisor. This implies that the adversary can observe the communication on the network and between enclaves (see Fig. 3). All existing solutions (and ours) rely on Intel SGX [13].

Such a threat model raises the concern that all hardware solutions have to trust Intel and make peace with SGX's shortcomings when protecting VNFs. It is well-known that SGX has failed to protect against side-channel attacks (see Foreshadow [14] and references of earlier attacks therein). In order to be secure against those attacks, VNF developers would have to implement cryptographic primitives themselves, increasing the switching costs and potentially sacrificing performance. Intel has promised to partially mitigate known side-channel attacks in new CPUs; but there is no guarantee that future versions of SGX will not suffer from certain weaknesses. However, it is plausible that Intel can make such attacks very difficult to perform in practice.

Emerging requirements of a complete solution. For a trusted hardware based VNF outsourcing solution to be considered complete, it needs to comply with the same functional and security requirements as NFs in dedicated hardware.

1) The solution has to *support both stateless and stateful VNFs*. Stateless VNFs, operating at L2 and L3, process packets one-by-one, but more complex VNFs have to keep flow level states in order to implement advanced functionality. Examples of stateful VNFs are load balancers and proxies which maintain packet pools and connection data in order to provide support for end-to-end communication;
2) The solution has to provide *"full-stack" protection*. Such a solution needs to protect all parts of a packet such as *payload and header*, and also *metadata* including timestamps, packet size and low-level protocol headers, related to the the raw

traffic flowing through VNFs. (Often, adversaries use metadata to mount an attack or infer sensitive properties of the traffic itself.) Moreover, it also needs to *protect VNF policies and code*, e.g., when network operator 1 (enterprise) outsources traffic processing to network operator 2 (SP), while still competing in user-facing services (an emerging use-case in many 5G scenarios [2]). In case of VNFs operating at L4, *protection of VNF states* is also important. Such states can contain sensitive information such as personal data; leaving such information vulnerable is undesirable and may also be illegal.

3) The solution has to *operate at near line rate speed*. It is important for a solution to maintain (almost) the same performance level as NFs in dedicated hardware while performing its task.

To the best of our knowledge such a solution does not yet exist; we describe potential steps to be taken towards such a complete solution in Section V.

III. Trusted hardware approach: existing solutions

Some existing systems are designed for particular types of VNFs; e.g., S-NFV [8] protects only the VNF state for a very limited set of VNFs. SGX-Box [9] protects only deep packet inspection (DPI) VNFs and cannot be used for other types. Moreover, by not implementing a kernel bypass mechanism such as DPDK[1] and an asynchronous interface for transitions between enclave and non-enclave parts, SGX-Box does not achieve line rate speed. Other solutions use the same VNF framework but to different effect; e.g., Trusted Click [6] is built on the same Click framework as ShieldBox [10], but is simpler in its design and technologies used, and is less effective in overcoming the limitations of SGX. Trusted Click uses *OCALL* and *ECALL* SGX instructions heavily for transitioning between trusted and untrusted regions, resulting in a large performance overhead making Trusted Click impractical for many use cases.

We single out three solutions for detailed introduction [7], [10], [11]. Two of these [7], [10] use a *VNF framework inside the enclave* and, to our best knowledge, offer the best performance, security and functionality. The third design [11] is the only solution from Table I not to provide a *VNF framework inside the enclave*; and the only one to provide both full stack protection of packets and support for stateful VNFs. Note that depending on the scenario, one of these three solutions offers the optimal state-of-the-art design alternative. However, the definition of "optimal" depends on the priorities of enterprises outsourcing VNFs. In section IV we consider how to choose a solution for specific needs.

ShieldBox [10] is a solution based on SGX and built on top of *Click*, a framework for building VNFs by exposing to VNF developers a set of elements (abstractions), and can be used to build various types of VNFs. ShieldBox is built on *SCONE*, a framework that protects the running VNFs from the outside world (e.g., untrusted operating system) through *shields*. SCONE uses *user-level threading* and *asynchronous system call* mechanisms from inside the trusted memory (enclave) which allows threads in untrusted memory to execute system calls asynchronously without forcing the enclave threads to exit. Due to the usage of SCONE and DPDK, ShieldBox achieves nearly line rate speed making it practical in real-world scenarios. Owing to the usage of Click, this solution provides support for a wide range of network functions, with the notable exception of stateful VNFs; Click does not have built-in functionalities for flow-based stateful traffic. Our analysis also revealed another disadvantage: placing DPDK inside the enclave. This design choice increases the size of the enclave, violating the trust minimization argument and harming RAM usage flexibility. In addition, ShieldBox does not protect the VNF code, therefore it does not fully satisfy the requirements in Section II.

SafeBricks [7] is built on top of NetBricks, another framework used to build various types of VNFs. Our analysis showed that NetBricks has multiple advantages over Click: i) it is based on Rust, an inherently secure programming language, ii) it enables Service Function Chaining, i.e., realizing complex services via chaining VNFs after each other, and iii) it uses zero-copy semantics leading to superior performance. To our knowledge, SafeBricks is the only solution which protects the code of VNFs, while also offering the best performance and lowest overhead out of all studied solutions; it operates at almost line rate speed owing to the usage of DPDK and an asynchronous mechanism for handling transitions between trusted and untrusted memory regions. By placing DPDK outside the enclave, SafeBricks reduces the EPC size resulting in improved performance and a reduced attack surface. Moreover the chaining of all VNFs in a single enclave improves performance due to the encryption/decryption process being performed only once. Note that SafeBricks does not protect traffic metadata, and does not explicitly declare support for stateful VNFs.

LightBox [11] is a system used to provide support for stateful VNFs while offering full-stack protection for packets. It consists of two modules: *etap* and *state*

[1]DPDK, Data Plane Development Kit, https://www.dpdk.org

TABLE I
COMPARISON OF PROPOSED SOLUTIONS REGARDING SECURITY AND FUNCTIONALITY.

		Protection					Supported Functionality		Supported
	System	Header	Payload	Code	Policies	State	Stateful VNF	Stateless VNF	Operation
Crypto	BlindBox [4]	✗	✓	✗	✓	✗	✗	✓	regular expression
	SplitBox [5]	✓	✓	✗	✓	✗	✗	✓	range matching
	Embark [3]	✓	✓	✗	✓	✗	✗	✓	range matching
Trusted Hardware	S-NFV [8]	✗	✗	✗	✗	✓	✓	✗	generic operation
	Trusted Click [6]	✗	✓	✗	✓	✗	✗	✓	generic operation
	ShieldBox [10]	✓	✓	✗	✓	✗	✗	✓	generic operation
	SGX-Box [9]	✗	✓	✗	✓	✓	✓	✗	generic operation
	SafeBricks [7]	✓	✓	✓	✓	⋆	⋆	✓	generic operation
	LightBox [11]	✓	✓	✗	✓	✓	✓	⋆	generic operation
	SafeLib [12]	✓	✓	✓	✓	✓	✓	✓	generic operation

✓– Feature provided
✗ – Feature not provided
⋆ – Feature not explicitly handled

management. The former is a virtual network interface used to provide in-enclave access of traffic from within enclave, while the latter provides an automatic memory efficient method for managing the huge amount of states tracked by VNFs.

We emphasize that LightBox does not fall into the same category as the solutions mentioned above; rather, it is complementary to them. On one hand, ShieldBox and SafeBricks provide modular implementation of VNFs from within the enclave and, therefore, VNFs are secured *by design*. On the other hand, when VNF developers use LightBox they need to port their VNFs inside the enclave in order to utilise Lightbox's performance and security enhancing features.

The main advantage of LightBox is that it provides full stack protection of packets while maintaining near line rate speed. It also provides efficient state management in the case of stateful VNFs. The *etap* module has also been adapted to work with *mOS*, an advanced networking stack used to develop stateful VNFs. On the other hand, i) LightBox does not offer VNF code protection, ii) it does not provide a VNF framework, meaning that VNF developers need to port VNFs on their own, requiring familiarity with SGX, and iii) the *etap* device may need to be adapted in order to support the necessary system calls.

IV. DISCUSSION

Table I summarizes our analysis and compares all studied solution alternatives focusing on security and supported functionality. It is safe to say that cryptographic solutions are limited in both functionality and performance, thus they cannot provide support for complex network functions in an enterprise, let alone in carrier-grade deployment. Even relaxing these requirements, no cryptographic solution provides protection even close to the desired level. Trusted hardware based alternatives in general show great promise with regard to functionality and performance, and have made considerable progress towards satisfying the requirements of a complete protection solution for outsourced network functions. Depending on the scenario, different state-of-the-art hardware solutions are preferable.

VNF developers protecting stateful VNFs. In such a case, SGX-Box [9] or Lightbox [11] should be chosen. When packet protection is the main concern then Lightbox [11] is the way to go since it provides full stack protection. In cases when VNF developers need a VNF framework to provide a set of abstractions and to avoid porting their developed DPI VNFs to an enclave by themselves they should use SGX-Box [9]. Note that there is no existing solution which provides a general VNF framework to provide APIs for different types of stateful VNFs.

VNF developers protecting stateless VNFs. In such a case, either SafeBricks [7] or ShieldBox [10] should be chosen. Trusted Click [6] does not provide favorable performance when compared to other solutions, nor full stack protection, so we conclude there are better options. When VNF developers need code protection then SafeBricks [7] is the only solution available. If VNF developers prefer to use Click for developing their VNFs, we suggest they use ShieldBox [10]. Both options [7], [10] provide good performance.

V. A FUTURE DIRECTION: SAFELIB

Imagine a scenario in which an enterprise wants to outsource both stateful and stateless NFs operating at near line speed while the confidentiality of traffic data,

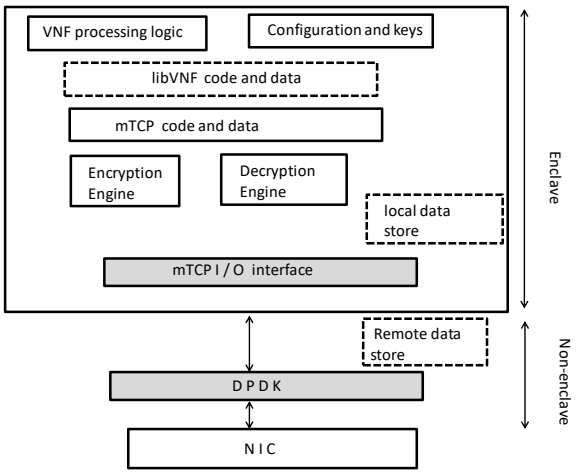

Fig. 4. Detailed SafeLib architecture: light grey boxes denote the *mTCP stack*, dashed boxes denote the *libVNF framework*. Note that enclave here represents the *main enclave* and is used for stateful VNFs. When it comes to stateless VNFs then mTCP and remote data store are not required.

metadata, VNF policies, code and states are all being protected. As of now, the enterprise cannot achieve its goal using a single VNF outsourcing solution, as evidenced by our analysis in Table I. Our proposed solution SafeLib [12] is designed specifically to fulfil these requirements; while it is currently under implementation, we are confident that the meticulous design and the choice of technologies used (detailed below) will ensure the envisioned properties.

Technologies used. SafeLib is built on top of *libVNF* [15], a framework for scalable and high performance VNFs. The main reason for choosing libVNF over other VNF implementation frameworks is its remarkable API; it is easy-to-use, flexible and generic enough to provide support for VNFs operating at L2/L3 and also at L4. Furthermore, libVNF is built on top of *mTCP*, a user level TCP stack; the usage of mTCP, brings two main benefits: i) departure from kernel complexity which allows us to directly benefit from *DPDK* and ii) performing batch I/O processing, partially alleviating the burden of transitioning between enclave and non-enclave regions.

SafeLib deployment. The deployment of SafeLib involves a two-phase procedure and two enclaves. *Pre-phase enclave* is used to access the raw VNF source code and then to compile it. During the pre-phase procedure, the GW runs the remote attestation protocol (RAP) in order to verify the *pre-phase* enclave which in turn returns a public key to the GW. Using that key, the GW encrypts the VNF code and policies, and sends them to the *pre-phase enclave*, which in turn decrypts the code, and then compiles it. After the pre-phase enclave compiled the code, it attests the *main enclave* using the local attestation procedure (LAP) of SGX. If this procedure is successful then the *pre-phase enclave* sends the compiled code to the *main enclave*.

Note that we have used the standard RAP of SGX between the GW and the *pre-phase enclave*. Standard RAP includes 4 messages, and we modify the last message according to our needs. Specifically we include encrypted VNF policies, codes, and cryptographic keys depicted by the box *Configuration and keys* in Fig. 4.

During the second phase, the GW creates a set of IPSec tunnels with the *main enclave* and starts sending packets. The GW does not have to perform RAP with the *main enclave* after performing it with the *pre-phase enclave*. Since the *pre-phase enclave* performed LAP with the *main enclave* then by design the GW trusts the *main enclave*.

Using a two-phase procedure, SafeLib protects the VNF code and policies. Setting up a set of IPSec tunnels between the GW and the *main enclave*, SafeLib protects packet payloads and headers. Note that SafeLib re-uses the idea of these mechanisms from SafeBricks [7].

SafeLib high level architecture. Designing SafeLib has been a non-trivial endeavour, mainly owing to the two major limitations of SGX being at odds with each other. As shown in Fig. 4, we have divided our solution into enclave and non-enclave regions. We place only components responsible for processing sensitive information inside the enclave. At the bare minimum, we should place *libVNF code and data* and *VNF specific processing logic* inside the enclave. Concerning the first limitation, in order to alleviate the overhead of system calls we use DPDK, a packet capture library processing packets in batches. In our design we place DPDK outside the enclave. This approach requires some additional steps because DPDK does gain access to packets inside the enclave once they are decrypted. We overcome this issue by implementing IPSec endpoints within the enclave: *Encryption engine* and *Decryption engine* represent IPSec endpoints in Fig. 4.

To overcome the second limitation of SGX, we use an asynchronous interface following the ideas in [7], [10]. To achieve that we modify DPDK I/O and mTCP I/O in order to communicate with each other. Our initial approach is to place the *mTCP I/O interface* inside the enclave, and *DPDK I/O* outside the enclave (see Fig. 4). Note that we anticipate the need of extensive changes to the I/O interfaces of DPDK, mTCP and libVNF. Alternatively, we may fall back to using Graphene-SGX, which offers an asynchronous interface by design.

Using libVNF as a VNF framework allows us to

provide support for both stateful and stateless VNFs [15]. Note that libVNF provides built-in data structures used to keep the states. To design and implement an efficient state management procedure our aim is to keep only the states of active flows inside the enclave, potentially at the *local data store*, a data structure of libVNF. The rest we encrypt and store at the *remote data store* outside the enclave. We are aware that libVNF's data structures used for storing states may not be efficient enough for our case. To this end, we are currently working on utilizing more succinct data structures. We are also in the process of designing metadata protection methods, exploring the usage of random IP fragmentation and maximum transmission unit (MTU) for each packet.

VI. Conclusion

In this paper we provided an overview of confidential outsourcing of network functions to the cloud, highlighting the challenges and emerging requirements, state-of-the-art solutions of different types and potential future directions. We concluded that existing systems based on trusted execution environments, notably Intel SGX, outperform pure cryptographic solutions regarding protection level, functionalities provided and performance. Furthermore, we identified the gaps between requirements and the state-of-the-art, and briefly introduced SafeLib, a novel design currently under implementation, filling these gaps.

References

[1] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[2] G. Biczók, M. Dramitinos, L. Toka, P. Heegaard, and H. Lønsethagen, "Manufactured by Software: SDN-Enabled Multi-Operator Composite Services with the 5G Exchange," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 80–86, 2017.

[3] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016*, 2016, pp. 255–273.

[4] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *ACM Conference on Special Interest Group on Data Communication, SIGCOMM*, 2015, pp. 213–226.

[5] H. J. Asghar, L. Melis, C. Soldani, E. D. Cristofaro, M. A. Kâafar, and L. Mathy, "Splitbox: Toward efficient private network function virtualization," in *Workshop on Hot topics in Middleboxes and Network Function Virtualization, HotMiddlebox@SIGCOMM*, 2016, pp. 7–13.

[6] M. Coughlin, E. Keller, and E. Wustrow, "Trusted click: Overcoming security issues of NFV in the cloud," in *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, SDN-NFVSec@CODASPY*, 2017, pp. 31–36.

[7] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "SafeBricks: Shielding Network Functions in the Cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation*, 2018, pp. 201–216.

[8] M. Shih, M. Kumar, T. Kim, and A. Gavrilovska, "S-NFV: securing NFV states by using SGX," in *ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, SDN-NFV@CODASPY 2016*, 2016, pp. 45–48.

[9] J. Han, S. M. Kim, J. Ha, and D. Han, "SGX-Box: enabling visibility on encrypted traffic using a secure middlebox module," in *First Asia-Pacific Workshop on Networking, APNet 2017*, 2017, pp. 99–105.

[10] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer, "Shieldbox: Secure middleboxes using shielded execution," in *Symposium on SDN Research, SOSR 2018*, 2018, pp. 2:1–2:14.

[11] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren, "Lightbox: Full-stack protected stateful middlebox at lightning speed," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 2019, pp. 2351–2367.

[12] E. Marku, G. Biczok, and C. Boyd, "Towards protected VNFs for multi-operator service delivery," in *IEEE The 1st International Workshop on Cyber-Security Threats, Trust and Privacy Management in Software-defined and Virtualized Infrastructures, co-located with IEEE NetSoft 2019*, 2019.

[13] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptology ePrint Archive*, vol. 2016, no. 86, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[14] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel {SGX} kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.

[15] P. Naik and M. Vutukuru, "libVNF: A Framework for Building Scalable High Performance Virtual Network Functions," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*. ACM, 2017, p. 12.

Enio Marku (enio.marku@ntnu.no) received his M.Sc. degree in electrical engineering in 2017 from CTU Prague, and is currently a Ph.D. candidate at Norwegian University of Science and Technology in Trondheim. His research interests include network security, 5G networking and network function virtualization (NFV).

Gergely Biczók (biczok@crysys.hu) is an associate professor in the CrySyS Lab at the Budapest University of Technology and Economics (BME). He received the PhD (2010) and MSc (2003) degrees in Computer Science from BME. Previously, he was a postdoctoral fellow at the Norwegian University of Science and Technology, a Fulbright Visiting Researcher to Northwestern University and a research fellow at Ericsson Research. His research focuses on the privacy, security and economics of networked systems.

Colin Boyd (colin.boyd@ntnu.no) is Professor in the Applied Cryptology Laboratory at NTNU, Trondheim, Norway. He previously held posts at the Queensland University of Technology, Australia; University of Manchester, UK; and British Telecom Research Laboratories, UK. He holds a BSc and PhD in Mathematics from University of Warwick, UK. His research interests focus on design and analysis of cryptographic protocols and their applications in areas including electronic voting and payments.