# secureTCP: Securing the TCP/IP stack using a Trusted Execution Environment

Keita Aihara[1,a)] Pierre-Louis Aublin[1,b)] Kenji Kono[1,c)]

***Abstract:*** Trusted Execution Environments allow an application to run securely even in an untrusted environment such as the Internet. However, existing approaches still rely on an untrusted network stack for their communications. While protocols such as TLS can guarantee the integrity and confidentiality of the application-level data, the metadata of the communication is not protected. Not only this allows an attacker to infer sensitive information, but it can also be used to tamper with communications and degrade performance.
To tackle this problem we describe *secureTCP*, a Secure TCP/IP stack for commodity machines. secureTCP provides integrity and confidentiality guarantees from a malicious software stack. We implement secureTCP and evaluate it with a microbenchmark as well as with the Lighttpd web server. Our evaluation demonstrates that it offers strong security guarantees while reducing the performance by less than 7% for ≥ 1 kB messages.

## 1. Introduction

The security of Internet services is of prime importance. Users give their data to an untrusted third-party from which they expect confidentiality and integrity guarantees. While service providers are not inherently malicious, they are a frequent target for attacks, yet securing their infrastructures is a difficult task. The unfortunate consequence is frequent data leakage or corruption on the Internet [12], [29], [38].

Recent years have seen the development of Trusted Execution Environments (*TEE*), in particular with Intel SGX [17], ARM TrustZone [44] and AMD SME/SEV [21] technologies. Trusted Execution Environments are special secure execution environments isolated from the rest of the system. They allow an application to enforce security guarantees even in the presence of a powerful attacker who can control both the software and hardware stacks.

While many applications using a Trusted Execution Environment have been proposed [5], [6], [7], [26], [27], [28], [33], [36], [42], they all rely on the untrusted host operating system to provide communication capabilities, in particular the TCP stack. Existing protocols such as TLS [31] or IPsec [13] can guarantee the confidentiality of the data. However the metadata is not protected. This is a problem, as an attacker could learn sensitive information from the metadata [45] or tamper with the TCP/IP protocol to degrade quality of service [19], which could have unfortunate impact on the economy and reputation of both the service and service provider.

This paper aims at improving the security of the TCP/IP stack in order to provide stronger guarantees to secure applications. We propose secureTCP, a novel secure TCP/IP stack design that leverages a TEE to conceal both the metadata and data of network packets from a potentially malicious service provider. In order to communicate securely with the network, secureTCP assumes an advanced network card with trusted hardware-based encryption. This can for example be provided by a smartNIC [10], [35]. secureTCP receives encrypted IP packets from the network card and securely reconstructs the TCP stream, before sending it to secure application.

We have implemented a prototype of secureTCP, leveraging Intel SGX, the mTCP user-level TCP stack [18] and the DPDK library [15]. To provide good performance and limit the impact of faults [27], secureTCP (i) minimizes the amount of code and data in the TEE; (ii) minimizes the number of transitions between the untrusted and trusted environments; and (iii) implements batching of network functionalities.

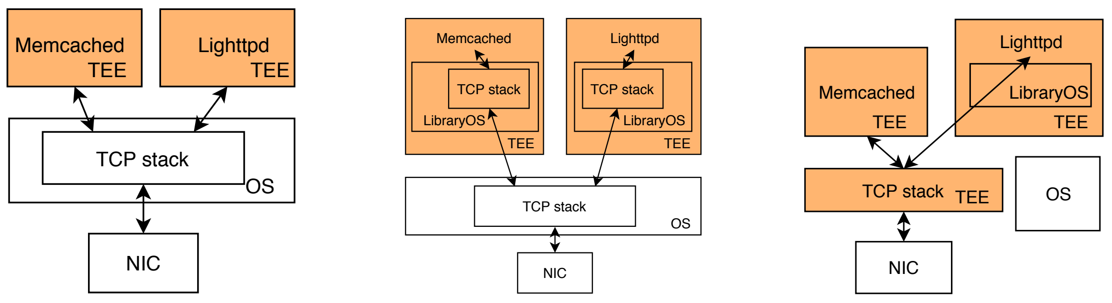
An evaluation, using both a microbenchmark and the Lighttpd web server [23], demonstrates that secureTCP provides a low performance overhead (< 7% for ≥ 1 kB messages) while offering additional security guarantees to Internet services compared to using an untrusted network stack.

The rest of this paper is organized as follows: Section 2 motivates the need for a trusted TCP stack and introduces the threat model. Section 3 presents the design of secureTCP while Section 4 details its implementation. We evaluate secureTCP in Section 5, discuss related work in Section 6 and conclude in Section 7.

## 2. The route to secure the TCP stack

A secure TCP stack needs to meet several requirements:
**R1. Security of communications**: integrity and confidentiality of both data and metadata exchanged between applications and

[1] Keio University, Japan
[a)] k.k.a@sslab.ics.keio.ac.jp
[b)] pl@sslab.ics.keio.ac.jp
[c)] kono@sslab.ics.keio.ac.jp



(a) Trusted app. w/ untrusted stack. (b) LibraryOS secure stack. (c) Trusted stack.

Fig. 1: Approaches for the deployment of trusted applications.

network must be ensured.

**R2. Isolation**: the TCP stack and application must be isolated from each others. It ensures that a bug in the application does not affect the TCP stack and vice-versa [8].

**R3. Performance**: security and performance are at opposite ends of the spectrum: adding security often decreases the performance of systems. This is why minimizing the performance overhead is of prime importance.

## 2.1 TCP stack threats

While confidentiality and integrity guarantees can be provided by encrypting the TCP payload, this is not sufficient. First, the untrusted network stack is involved in handling packets between the network card and the trusted application. Even if the application uses TLS [31] or IPSec [13], the metadata related to the communication (TCP headers and TCP protocol management packets) is not protected. A malicious service provider could extract sensitive information from the metadata or modify the packets for its own benefits.

Second, a malicious service provider could tamper with the network stack, e.g. slowing down connections by controlling the TCP congestion control mechanism [19]. As an example a service provider hosting the video streaming applications of two competing companies could slow down the traffic of one of them in favor of the second one.

## 2.2 Deployment of trusted applications

Several trusted applications for untrusted environments have been proposed in recent years [5], [6], [7], [26], [27], [28], [33], [36], [42]. Their approaches regarding network communications can be separated in two different categories (see Figures 1a and 1b): (i) applications that directly make use of the kernel network stack; and (ii) applications that implement their own stack inside the TEE via a LibraryOS.

### 2.2.1 Unsecure network stack

Trusted applications wanting to access the network can leverage the untrusted host operating system network stack. For example, on Linux, trusted applications can use the BSD socket API [2]. This is the approach chosen by several systems that modify the design of existing applications to enhance their security [6], [26], [27], [33], [36], [37].

Using this approach the trusted applications can ensure the integrity and confidentiality of the data. Nevertheless the communications are processed by an untrusted component (~~R1~~). The trusted applications access the host OS network stack, isolated from the application (**R2**), without involving intermediate components: every call to a network function is translated to a call to the untrusted network stack. This provides a low performance overhead (**R3**).

### 2.2.2 LibraryOS secure network stack

Systems such as Haven [7], Graphene-SGX [42] or SGX-LKL [28] provide applications with essential functionalities by implementing an entire libraryOS [34] inside the TEE.

The libraryOS provides its own network stack. Nevertheless, it eventually needs to interact with the untrusted stack of the host OS (~~R1~~). Duplicating the network stack can lead to performance issues (~~R3~~). For example the network stack inside the application conceives of the scheduling delay as network delay, and starts unnecessary congestion control. Finally, there is no isolation between the network stack and the application (~~R2~~): they both execute in the same address space. A bug in one of these components could corrupt the other one and eventually leak sensitive information.

## 2.3 Threat model

Service providers are not inherently malicious. Nevertheless they can be subject to bugs, misconfigurations, negligence or other human errors that can lead to secrets being exposed. Furthermore it is in the interest of the service provider to protect its reputation and business model, which can potentially lead to unfair and selfish behaviours.

From a technical point of view we assume that the software stack, including the operating system and drivers, is compromised. The attacker has access to the hardware stack, including the memory and PCI buses. The CPU package provides a Trusted Execution Environment (TEE). In addition we assume the existence of an advanced NIC equipped with trusted hardware-based encryption [9], [10], [35].

We consider that the trusted components are implemented in accordance to their specification and do not leak sensitive information via side channels. Side-channel attacks are orthogonal to the problem we address. Existing technique such as [14] can be employed. Similarly, availability is an orthogonal problem: at any point the system could stop executing trusted processes. This can be addressed with other means such as service replication [25], [30].

We assume the network infrastructure is secure. Protecting phys-

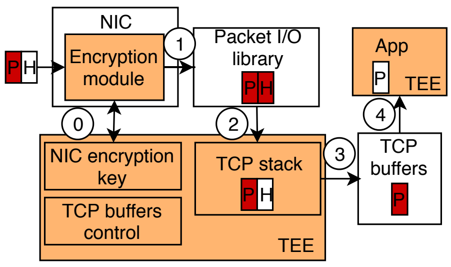


Fig. 2: secureTCP architecture. P is the payload, H is the packet metadata and red means encrypted.

ical access to data centers is a major concern to cloud providers. From a logical point of view, components such as routers, etc., can be implemented inside a TEE.

## 3. secureTCP

We describe secureTCP[*1], a user-space secure TCP/IP stack that ensures the confidentiality of network communications on an untrusted computer system. secureTCP replaces the untrusted network stack of the untrusted host with a trusted network stack, as depicted in Figure 1c. secureTCP can be either used by standalone or libraryOS-based secure applications.

After an overview of secureTCP (Section 3.1), we detail the secure packet processing (Section 3.2) and attestation (Section 3.3) mechanisms.

### 3.1 Overview

secureTCP meets the requirements presented in Section 2:

**R1. Security of communications**: secureTCP ensures the integrity and confidentiality of both the data and metadata by leveraging a TEE.

**R2. Isolation**: the TCP stack and application are executed in two different protection domains.

**R3. Performance**: secureTCP minimize the amount of code and data inside the TEE to provide good performance.

Figure 2 presents the architecture of secureTCP. Both the application and secureTCP run in two separate TEEs to ensure isolation. That is to say, there is one instance of secureTCP per secure application. Note that an orthogonal approach regarding isolation could be to enforce multiple domains inside a single TEE via compiler-based technique [39], [40]. The application and secureTCP execute in a single process. Their interaction consists of normal function calls via a well-defined API that is similar to the BSD sockets API (see Section 4).

secureTCP relies on a fast user-space packet I/O library that does not involve the operating system or network drivers for network communications. The role of the packet I/O library is to forward incoming and outgoing IP packets between the NIC and the TCP stack, without inspection.

In order to minimize the amount of code that has to be trusted, the packet I/O library executes outside of the TEE. This however does not affect the security of secureTCP. The IP packets, including the metadata, are encrypted by the secure encryption capabilities of the network card. The packet I/O library only observes encrypted packets. A malicious attacker cannot observe the plain-text data or corrupt the IP packets without being detected. For a similar reason the TCP buffers are allocated outside of the TEE. These buffers appear only as encrypted data to an attacker, while the control structures, controlling the offsets at which TCP segments start and end, is securely managed inside the TEE.

### 3.2 Secure TCP stack

As depicted in Figure 2, secureTCP is composed of several components. The network card (NIC) interacts with the user-space packet I/O library to send and receive encrypted IP packets (1). The encryption is performed on the entire IP packet, including both its data and metadata.

Then the secure TCP stack retrieves the packets in batches, copies them into TEE memory (2), decrypts them (checking for their integrity) and reconstructs the TCP stream.

The TCP stream is copied into the TCP buffers (3), allocated outside the TEE in untrusted memory. Nevertheless, given that the TCP stream is encrypted by the application (e.g., using the TLS protocol) and the control structures of the buffers are store securely inside the TEE, the TCP buffers do not leak any sensitive information.

Finally, when the application issues a call to read data, it is securely copied by secureTCP from the TCP buffers to the application buffers (4). Sending data across the network is done in a similar way.

### 3.3 Attestation and provisioning

The goal of the attestation mechanism is two folds: (i) ensure that the TEE is correctly initialized with secureTCP, executing the correct code; and (ii) ensure that secureTCP can be securely provisioned with secrets.

The attestation mechanism involves a trusted third party. The TEE creates a cryptographically signed report of its memory content (including code and data). This report is used to prove that secureTCP has been loaded correctly. Then the report is sent to the trusted third-party for verifications. Once the attestation has succeeded, the key used by the NIC to encrypt and decrypt IP packets is exchanged with secureTCP (0).

## 4. Implementation

We ported the mTCP [18] user-level high performance TCP/IP stack to Intel SGX and use DPDK [15] for the user-level packet I/O library. mTCP differs from the BSD socket API in several ways: (i) it implements a per-core accept queue design, meaning that multiple cores can concurrently listen on the same socket; (ii) it does not share sockets nor data structures across multiple cores, which improves concurrency and scalability; (iii) it associates one mTCP thread, communicating with DPDK, with each application thread; and (iv) it avoids expensive system calls to the kernel by running entirely in user-space. mTCP with DPDK relies on polling, consistently checking for new packets to be sent/received, potentially wasting CPU cycles.

mTCP uses several timers, for example to check for timeouts. Access to a fast and trusted time in Intel SGX is only possible starting with SGX v2 [1]. Our implementation of secureTCP cur-

[*1] The source code of secureTCP is available at `https://github.com/sslab-keio/secureTCP`.

rently relies on the untrusted time given by the untrusted operating system.

### 4.1 Intel SGX

Intel SGX [17] is a new set of instructions present on Intel processors since 2015. It provides confidentiality and integrity guarantees even in the presence of a powerful attacker that controls both the hardware and software stacks (with the exception of the CPU package).

Intel SGX implements Trusted Execution Environments called *enclaves*, which are a special secure execution mode. The memory of applications running inside an enclave is stored inside a secure area called the *Enclave Page Cache* (EPC). On current hardware the EPC size is limited to 128 MB, shared between all the enclaves of the system. As the EPC also contains SGX metadata, only around 90 MB is available to applications. Using more than 90 MB starts an expensive paging mechanism [5]. To provide confidentiality and integrity property, the enclave memory is transparently encrypted and its integrity is verified in hardware. Furthermore, while an enclave can access both trusted and untrusted memory, the application can only access untrusted memory.

Enclaves are accessed via a well-defined interface, composed of *enclave calls* (*ecalls*) and *outside calls* (*ocalls*). Executing an *ecall* enters the enclave, changing the execution mode from untrusted to trusted. Similarly, executing an *ocall* changes the execution mode from trusted to untrusted. In both cases Intel SGX adds additional checks to prevent attacks and leakage of secrets. The direct consequence is that enclave transitions are costly: the authors of sgx-perf [43] observed that an enclave transition is $\approx 13,100$ cycles.

Intel SGX provides an attestation mechanism that can be used by applications to prove to a third party the authenticity of their enclave and the system on which they run [3]. This mechanism gives to clients the guarantee that they are communicating with a secure service. secureTCP reuses the Intel SGX attestation mechanism with no modifications.

### 4.2 secureTCP interface

The interface of secureTCP is composed of 24 *ecalls* and 40 *ocalls*. All the *ecalls* but 2 define the mTCP API; the remaining 2 *ecalls* are used as wrappers for the creation of new mTCP threads. The 40 *ocalls* are used for accessing DPDK (15) and the standard library (25). We added 561 lines of code to mTCP and created an enclave wrapper (to handle *ecalls* and *ocalls*) composed of 2, 200 lines of boiler-plate code.

### 4.3 Performance optimisation

The performance of Intel SGX applications is limited by two factors: (i) the amount of memory used by the application; and (ii) enclave transitions. secureTCP fits in the EPC as it uses only 16 MB of enclave memory. Thus we need to minimise the number of enclave transitions.

We used the sgx-perf profiler [43] to pinpoint performance bottlenecks. Our optimisations fall into 4 categories: (i) merge *ecalls* and *ocalls* to reduce the number of enclave transitions; (ii) move DPDK code inside the enclave to avoid *ocalls*; (iii) batch *ecalls*; and (iv) execute the secureTCP thread and application thread on different cores, minimizing the number of thread context switches.

## 5. Evaluation

Our evaluation first shows that secureTCP provides strong security guarantees (Section 5.2). Then, using both a microbenchmark (Section 5.3) and the Lighttpd web server(Section 5.4), it shows that secureTCP offers an acceptable performance overhead.

### 5.1 Experimental set-up

All the experiments are run on an SGX-capable machine composed of a 6-cores Intel Core i5-8500 at 3GHz (no hyperthreading) equipped with 16GB of RAM and running the Intel SGX SDK v2.5. The client machine is a 6-cores Intel Xeon X5650 at 2.67GHz with 8GB of RAM. They both use an Intel 82576 Gigabit Ethernet Controller that supports DPDK and run Ubuntu 18.04.02 LTS with the Linux kernel 4.15.

### 5.2 Security discussion

In this section we list attacks on secureTCP. For each attack we show how secureTCP prevents it, guaranteeing confidentiality and integrity properties.

**Bypass secureTCP.** A malicious service provider could decide to use its own TCP stack instead of secureTCP. To defend against this attack, as explained in Section 3.1, the network card and secureTCP only exchange encrypted IP packets. A malicious service provider is not able to create valid encrypted IP packets that will be correctly decoded by the network card.

**Modification of TCP stream.** A malicious service provider could modify the content of the TCP buffers stored in untrusted memory. Given that the content is encrypted and integrity-protected, this attack is detected by secureTCP and the application.

**Enclave interface attacks.** An attacker could try to access the enclave secrets by manipulating the arguments and returned values of *ecalls* and *ocalls* [11]. To reduce the attack surface we harden the enclave interface with additional checks on the values.

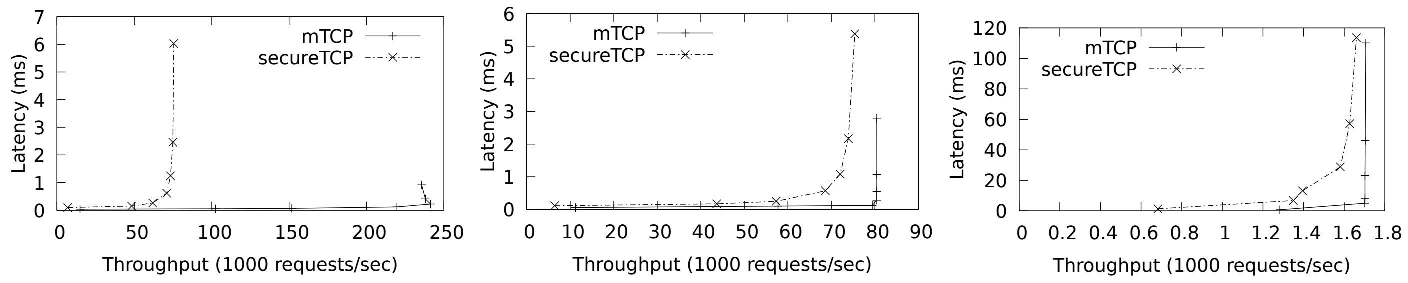
### 5.3 Microbenchmark perfomance

The microbenchmark consists of a simple client-server application where the client and server exchange fix-sized packets. Both the client and server applications are limited to one core, sufficient to saturate the network link.

Figure 3 presents the evolution of the throughput and latency of mTCP and secureTCP for requests of 1 byte and replies between 1 byte and 64 kB. When considering replies of 1 kB and more, secureTCP has a small performance overhead: the throughput drops by 6% with 1 kB replies (from 80,400 requests/sec with mTCP to 75,300 requests/sec with secureTCP) and 3% with 64 kB replies (from 1730 requests/sec with mTCP to 1681 requests/sec with secureTCP). As the replies size increases, the cost of Intel SGX is amortized, thus reducing the perfomance overhead.

With requests of 1 byte, the performance overhead of secureTCP is as high as 69%: from 241,300 requests/sec with mTCP to 75,500 requests/sec with secureTCP.

Two of the optimisations presented in Section 4.3 noticeably improve the performance: (i) batching of *ecalls* increases the through-



(a) 1 byte requests and replies.
(b) 1 byte requests, 1 kB replies.
(c) 1 byte requests, 64 kB replies.

Fig. 3: Throughput vs latency of mTCP and secureTCP.

put by 97%, from 17,000 requests/sec to 33,300 requests/sec; (ii) executing the application and secureTCP threads on different cores further increases the throughput by an additional 126%, up to 75,300 requests/sec. These two optimisations respectively reduce the number of enclave transitions and thread context switch.

The current performance overhead of secureTCP is due to contention on the lock used to synchronize the application and secureTCP threads. Unfortunately solving this issue is not trivial and requires additional engineering.

### 5.4 Lighttpd web server performance

We measure the maximal performance, in terms of throughput and latency, of secureTCP with the Lighttpd web server [23] using the ApacheBench benchmark [4]. The server is configured with 6 cores. The webpage size is 1 MB. This is in line with the average Internet webpage size [22].

In such settings secureTCP exposes a 7% overhead: the throughput decreases from 101  requests/sec to 94  requests/sec while the latency increases from 59 ms to 64 ms. This degradation is primarily due to enclave transitions overhead.

## 6. Related work

To the best of our knowledge we are the first to improve the security guarantees of the TCP stack by porting it to Intel SGX. Nevertheless we are not the first authors to consider the execution of network functionalities inside an SGX enclave. Shieldbox [41] executes the Click modular router [24] inside SCONE [5] and uses DPDK for fast user-level packet processing. In a similar way, Endbox [16] executes a VPN and the Click modular router inside an SGX enclave to provide scalable middlebox functionalities at the client. These systems are complementary to secureTCP.

The LibSEAL library [6] terminates TLS connections inside an SGX enclave and securely logs the interactions between clients and Internet services. This log can then be used to detect whether the service executes correctly according to its specification. secureTCP goes a step further by securely executing the TCP stack inside an enclave.

Several systems that improve the performance and security of the network stack have been proposed in the past. For example, IX [8], Arrakis [32] and Shinjuku [20] use virtualisation techniques to separate the network processing from the rest of the kernel. These systems can provide isolation and better performance and scalability than traditional user-level stacks such as mTCP [18]. Contrarily to these systems, secureTCP protects the TCP stack even in the presence of a powerful attacker who controls both the software (including the operating system) and the hardware (with the exception of the CPU package).

## 7. Conclusions

This paper presents secureTCP, a secure user-level TCP/IP stack that provides confidentiality and integrity guarantees by leveraging a Trusted Execution Environment. We implemented a prototype of secureTCP that uses Intel SGX and demonstrated, using both a microbenchmark and the Lighttpd web server, that secureTCP incurs a a low performance overhead ($< 7\%$ for $\geq 1$ kB messages).

As of future work, a secure TCP stack is a first step towards forensics analysis, auditing and monitoring. These mechanisms are of prime importance to augment the security guarantees offered by cloud providers.

**Acknowledgments** This work was supported by JST CREST Grant Number JPMJCR19F3, Japan.

### References

[1] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3D: System Programming Guide, Part 4* (2016).
[2] *FreeBSD Developers' Handbook* (2018).
[3] Anati et al., I.: Innovative technology for CPU based attestation and sealing, *HASP '13 workshop*, Vol. 13, ACM New York, NY, USA (2013).
[4] Apache Software Foundation: ApacheBench, `http://httpd.apache.org/docs/2.2/en/programs/ab.html` (2019).
[5] Arnautov, S., Trach, B., Gregor, F., Knauth, T. et al.: SCONE: Secure linux containers with Intel SGX, *OSDI '16* (2016).
[6] Aublin et al., P.-L.: LibSEAL: Revealing Service Integrity Violations Using Trusted Execution, Porto, Portugal, ACM (2018).
[7] Baumann, A., Peinado, M. and Hunt, G.: Shielding Applications from an Untrusted Cloud with Haven, OSDI (2014).
[8] Belay et al., A.: IX: A Protected Dataplane Operating System for High Throughput and Low Latency, *OSDI 14*, pp. 49–65 (2014).
[9] Bossuet, L., Aubert, A. et al.: The Security of ARM TrustZone in a FPGA-based SoC, *IEEE Transactions on Computers* (2019).
[10] Caulfield et al., A.: Beyond SmartNICs: Towards a fully programmable cloud, *IEEE HPSR*, Vol. 18 (2018).
[11] Checkoway, S. and Shacham, H.: Iago attacks: Why the system call api is a bad untrusted rpc interface, *ASPLOS*, Vol. 13, pp. 253–264 (2013).
[12] Cox, JosephTroy Hunt: Another Day, Another Hack: Tens of Millions of Neopets Accounts, `https://motherboard.vice.com/en_us/article/neopets-hack-another-day-another-hack-tens-of-millions-of-neopets-accounts` (2016).
[13] Davis, C. R.: *IPSec: Securing VPNs*, McGraw-Hill Professional (2001).
[14] et al., O.: Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks, *USENIX ATC '18* (2018).
[15] Foundation, T. L.: DPDK project, `https://www.dpdk.org/` (2019).
[16] IEEE: *EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution*, Luxembourg, Luxembourg (2018).

[17] Intel: Software Guard Extensions Programming Reference, Ref. 329298-002US, https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf (2014).
[18] Jeong et al., E.: mTCP: a Highly Scalable User-level TCP Stack for Multicore Systems, *NSDI '14*, pp. 489–502 (2014).
[19] Jero et al., S.: Automated attack discovery in TCP congestion control using a model-guided approach, *Proc. of NSDI '18*, pp. 1–15 (2018).
[20] Kaffes et al., K.: Shinjuku: Preemptive Scheduling for $\mu$second-scale Tail Latency, *NSDI 19*, pp. 345–360 (2019).
[21] Kaplan, D., Powell, J. and Woller, T.: AMD memory encryption, *White paper* (2016).
[22] KeyCDN: The Growth of Web Page Size, https://www.keycdn.com/support/the-growth-of-web-page-size (2018).
[23] Kneschke, J.: Lighttpd, https://www.lighttpd.net/ (2003).
[24] Kohler et al., E.: The Click modular router, *ACM Transactions on Computer Systems (TOCS)*, Vol. 18, No. 3, pp. 263–297 (2000).
[25] Lamport, L.: Paxos made simple, *ACM Sigact News*, Vol. 32, No. 4, pp. 18–25 (2001).
[26] Li et al., B.: Troxy: Transparent access to byzantine fault-tolerant systems, *DSN 2018*, IEEE, pp. 59–70 (2018).
[27] Lind et al., J.: Glamdring: Automatic Application Partitioning for Intel SGX, Santa Clara, CA, USA, USENIX (2017).
[28] LSDS team: SGX-LKL, https://github.com/lsds/sgx-lkl (2019).
[29] NIST: CVE-2018-15664, https://nvd.nist.gov/vuln/detail/CVE-2018-15664 (2019).
[30] Ongaro et al., D.: In Search of an Understandable Consensus Algorithm., *USENIX ATC '14*, USENIX Association, pp. 305–319 (2014).
[31] OpenSSL Foundation: OpenSSL, https://www.openssl.org/ (2016).
[32] Peter et al., S.: Arrakis: The operating system is the control plane, *ACM Transactions on Computer Systems (TOCS)*, Vol. 33, No. 4, p. 11 (2016).
[33] Pires et al., R.: Secure Content-Based Routing Using Intel Software Guard Extensions, Middleware (2016).
[34] Porter et al., D. E.: Rethinking the library OS from the top down, *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 1, ACM (2011).
[35] Sabin et al., G.: Security offload using the smartnic, a programmable 10 gbps ethernet nic, *NAECON '15*, IEEE, pp. 273–276 (2015).
[36] Sartakov, Vasily et al.: EActors: Fast and flexible trusted computing using SGX, *Middleware '18*, ACM, pp. 187–200 (2018).
[37] Schuster et al., F.: VC3: Trustworthy Data Analytics in the Cloud Using SGX, SP (2015).
[38] Sharwood, S.: GitLab.com Melts Down After Wrong Directory Deleted, Backups Fail, https://www.theregister.co.uk/2017/02/01/gitlab_data_loss/ (2017).
[39] Shen et al., Y.: To Isolate, or to Share?: That is a Question for Intel SGX, *Proceedings of the 9th Asia-Pacific Workshop on Systems* (2018).
[40] Shinde, S., Le Tien, D., Tople, S. and Saxena, P.: Panoply: Low-TCB Linux Applications With SGX Enclaves, *NDSS* (2017).
[41] Trach et al., B.: Shieldbox: Secure middleboxes using shielded execution, *Proceedings of the Symposium on SDN Research*, ACM, p. 2 (2018).
[42] Tsai, C.-C., Porter, D. E. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX (2017).
[43] Weichbrodt et al., N.: sgx-perf: A Performance Analysis Tool for Intel SGX Enclaves, *ACM Middleware '18*, pp. 201–213 (2018).
[44] Winter, J.: Trusted computing building blocks for embedded linux-based ARM trustzone platforms, *ACM workshop on Scalable trusted computing*, ACM, pp. 21–30 (2008).
[45] Zhang et al., F.: Inferring users' online activities through traffic analysis, *ACM conference on Wireless network security*, ACM, pp. 59–70 (2011).