Towards protected VNFs for multi-operator service delivery

Enio Marku, Colin Boyd
Dept. of Information Security and Communication Technology
Norwegian Univ. of Science and Technology (NTNU)
Trondheim, Norway
{enio.marku,colin.boyd}@ntnu.no

Gergely Biczók
CrySyS Lab, Dept. of Networked Systems and Services
Budapest Univ. of Technology and Economics (BME)
biczok@crysys.hu

***Abstract*—Value-added 5G verticals are foreseen to be delivered as a service chain over multiple network operators with extensive outsourcing of Virtual Network Functions (VNFs). In this short paper we introduce the initial design of SafeLib, a software middlebox platform based on Intel SGX, which protects user traffic, VNF code, policy input and state in such scenarios, while also retaining high performance. Augmenting the smart integration of existing hardware and software building blocks with new secure elements, the SafeLib architecture shows considerable promise in a carrier-grade service context.**

***Index Terms*—5G, VNF, middlebox, security, multi-operator, service chain**

I. Introduction

Novel 5G verticals are expected to add real value to services. Most of these verticals will be delivered as a service chain over multiple independent network operators collaborating with each other [1]. Users would see a one-stop shop as they need to contact a single network operator in order to obtain the requested service. This operator would in turn outsource part of its network functions to other operators when it lacks the geographical footprint or available compute resources to deliver the service on its own.

Flexible service delivery is required in order to provide such collaboration between multiple operators [2]. Software middleboxes (Virtual Network Functions, VNFs) are a key enabler in such a scenario, since VNFs can easily be outsourced from one operator (OP1) to other network operators with which OP1 has a trust relationship. However, as illustrated in Figure 1, such an environment comes with a complex security context, where different parties have distinct requirements regarding protection of different data [3].

Protect the user traffic. The user traffic, will be processed by the VNF deployed in OP2's domain. User 1 does not have a trust relationship with operator OP2, therefore the deployed software middlebox should protect user traffic.

Protect VNF code and policy input. The VNF running in OP2's domain receives its policy input (e.g., firewall rules, coding parameters, filter expressions, etc.) from OP1. OP1 does not want reveal these policies to OP2 for a number of reasons, such as competitive advantage or hiding its cyber-defense strategy. For similar reasons OP1 might not want to reveal the VNF's implementation (code) to OP2.

978-1-5386-9376-6/19/$31.00 ©2019 IEEE

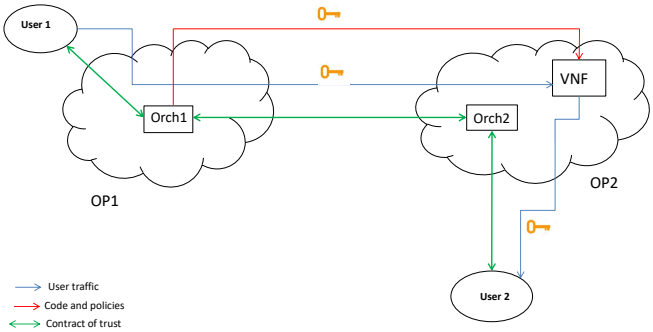

Fig. 1. Multi-operator service function chaining: example scenario

Protect VNF state. Stateful NFs store global states (e.g., user data details), private per-flow states (e.g., used for searching, terminating per-flow packets) and shared multi-flow states (e.g., states shared between VNF components used for packet management). Such states contain personal information (e.g., end-user data such as IP address and profile information) which should remain hidden to OP2.

Even an honest-but-curious OP2 could pose risks for user 1 and OP1. It is therefore desirable to provide a software middlebox which allows OP1 to securely outsource both stateful and stateless VNFs to OP2.

Related work. To the best of our knowledge there is no current software middlebox which provides all confidentiality and functional requirements above. Previous work which addresses similar problems may be divided in two main categories: *cryptographic vs. hardware approaches*. Middleboxes designed using a cryptographic approach [4], [5], [6], [7], [8], [9] support a limited set of functionalities, and have a huge performance overhead, making them impractical for carrier-grade deployment. Proposals in the second category [10], [11], [12], [13], [14] rely on trusted hardware such as Intel SGX, generally exhibiting higher performance. A comparison between current middleboxes and ours is shown in Table I.

Our contribution. In this short paper we present our initial efforts in designing a software middlebox platform (SafeLib) which satisfies the confidentiality and functional requirements

TABLE I
COMPARISON OF PROTECTED MIDDLEBOX PROPOSALS

		Security Protection					Functionality
	Software Middleboxes	Header	Payload	Code	Policies	State	Stateful
Cryptographic	BlindIDS [5]	✘	✓	✘	✓	✘	✘
	Embark [7]	✓	✓	✘	✓	✘	✘
	BlindBox [4]	✘	✓	✘	✓	✘	✘
	SPABox [6]	✘	✓	✘	✓	✘	✘
	SplitBox [8]	✓	✓	✘	✓	✘	✘
Hardware-assisted	SGX-Box [13]	✘	✓	✘	✓	✓	✓
	Trusted Click [11]	✘	✓	✘	✓	✘	✘
	S-NFV [10]	✘	✘	✘	✘	✓	✓
	Shield Box [14]	✓	✓	✘	✓	✘	✘
	SafeBricks [12]	✓	✓	✓	✓	N/A	N/A
	SafeLib	✓	✓	✓	✓	✓	✓

above *and* is able to operate close to line speed. We draw on existing hardware and software building blocks such Intel SGX, mTCP and libVNF, and present the detailed architectural and functional design of SafeLib. The rest of the paper is organised as follows. Section II introduces a brief overview of the technologies used for SafeLib. Section III introduces the basic design of SafeLib. Section IV describes the detailed architecture and the deployment procedure of SafeLib. Section V discusses trade-offs and lays out future work. Finally, Section VI concludes the paper.

II. BACKGROUND

Here we provide a brief overview on the technologies relevant to SafeLib.

Intel SGX. Intel SGX [15] is a set of CPU instructions that provides a trusted memory region named enclave to execute the trusted code, and a remote attestation protocol (RAP) to verify the enclave's code.

Software isolation. It allows a process to request an enclave, which can only be accessed by this process. Since the access is enforced by the processor, even a malicious OS with a root-level exploit can not access the enclave.

Remote Attestation Protocol. To verify that the enclave is running the right code, a remote verifier initiates a challenge-response protocol, using a mechanism to produce an enclave's measurement signed by the processor, and then verified (directly or indirectly) by Intel. If successful, this verification shows to the owner of the middlebox that the enclave in OP2's domain is both correctly created (established in a specific SGX-enabled system) and running the correct code.

mTCP. mTCP [16] is a user-level TCP stack, which is used in SafeLib whenever a VNF terminating the transport layer and operating at the application layer is outsourced to OP2 domain. Using mTCP as the back-end network stack brings our middlebox the significant benefits.

Departure from the kernel's complexity. mTCP allows us to directly benefit from a high-performance packet I/O library such as Intel DPDK [17] and netmap [18].

Perform batch processing. mTCP enables us to perform batch packet I/O, alleviating the performance penalty of enclave transitions at some point. It further improves the performance of SafeLib by collecting flow-level events to and from the application running inside the enclave without the need for system calls; avoiding a large overhead.

libVNF. The libVNF framework [19] can be used to develop high performance and horizontally scalable VNFs. We choose libVNF over other similar frameworks [20]–[22] because i) its API is generic, supporting the development of both L2/L3 VNFs and VNFs that terminate L4 and operate at the application layer; ii) it allows developers to only focus on the implementation of specific VNFs processing logic, rather than considering details such as low-level networking operations. The framework consists of five libraries accessible through dedicated APIs handling i) VNF initialization, ii) inter-VNF communication for service chains, iii) object requests iv) VNF state management, and v) managing and monitoring VNF replicas. Armed with these APIs, libVNF helps middlebox developers to only focus on writing the VNF specific processing logic into callback functions that are called when an event of interest (e.g., new connection, packet arrival, etc.) occurs.

III. SYSTEM DESIGN

We design SafeLib as a middlebox with the following properties.

- **Confidentiality:** protects sensitive information, including payloads and headers, cryptographic keys, VNF code and input policies, and securely executes VNF operations.
- **Generality:** supports both L2/L3 VNFs (e.g, NAT, firewall) as well as VNFs that terminate L4 connections and operate at the application layer (e.g, Intrusion Prevention System)
- **Performance:** operates on encrypted user traffic with an acceptable performance (preferably at line rate speed)
- **Low cost:** deployment should not be costly.

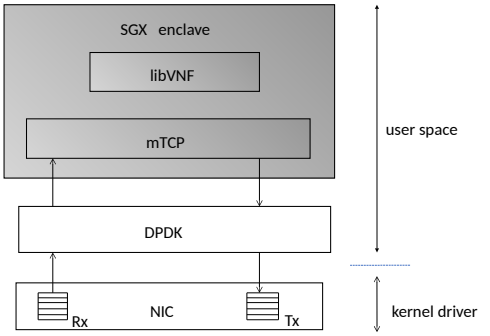

Fig. 2. SafeLib: high-level design

- **Rich API:** provides high-level APIs (e.g., low-level networking, cryptographic, routing operations) that allow developers to concentrate only on specific VNFs operations.

A. Threat model

Our target scenario is the untrusted domain of network operators where the middleboxes used to process sensitive information are deployed. We assume a powerful adversary that can control the entire software stack of the network provider outside the enclave, including kernel and hypervisor, and the entire hardware system except the CPU. For example, we assume the adversary can perform *Direct Memory Access* (DMA) attacks from malicious peripherals, privileged software attacks by exploiting bugs in software components, or DRAM attacks [23] by exploiting vulnerabilities in hardware.

To protect against such a powerful adversary, we rely on Intel SGX. By design, an adversary is unable to learn anything about the protected data and code in the enclave, and the remote attestation protocol is used to create a secure communication channel between the right parties.

We note that the current implementation of SGX does not protect against side channel attacks, such as cache timing attacks [24]–[26], page-fault-based attacks [27], [28], and denial of service attacks. SGX cannot protect against cache timing attacks since the enclave does not have data dependent memory access. A recent attack [29] leverages a bug in Intel SGX processors to leak the enclave secrets from the CPU cache. In our work we do not take any actions to protect against such kind of attacks. Should middlebox developers need to protect against such attacks, they should use cryptographic primitives. Upcoming Intel processors are likely to be designed to resist such attacks.

B. SafeLib: basic design

At a very high level, SafeLib consists of the smart integration of libVNF running inside an SGX enclave. The libVNF framework is built over mTCP which in turn communicates with DPDK for sending and receiving data to and from the network interface card (NIC). Fig. 2 describes the high-level design of SafeLib. SafeLib's architecture considers both characteristic limitations of SGX.

- **Enclave does not support system calls:** Instructions that change the privilege levels (i.e., system calls) cannot be executed inside the enclave. As a result, the application has to exit the enclave and execute such instructions outside the enclave memory region. However, such enclave transitions are expensive. As described in Section II, mTCP alleviates such limitation.
- **Memory size:** The protected memory region named enclave page cache (EPC) has a limited size of 94 MB. Minimizing the code size inside the enclave brings us performance and security benefits (e.g, reduces attack surface). We overcome this limitation by partitioning SafeLib code base into trusted and untrusted parts.

DPDK. mTCP communicates with the NIC via a kernel bypass mechanism such as DPDK. DPDK polls packets from the NIC and then mTCP threads query DPDK via an I/O interface for processing a batch of packets. In our design DPDK runs outside the enclave.

System bootstrap. A SafeLib middlebox is bootstrapped by running the remote attestation protocol. The protocol is initiated by a remote verifier which must be trusted (e.g., a gateway, GW) in the OP1 domain. If the verification is successful, the remote verifier sets up a set of IPSec tunnels to the OP2 domain. The outsourced VNF in the OP2 domain is horizontally scaled into replicas. For each replica we set up a single IPSec tunnel to the GW due to the limitation of mTCP (it supports a single VNF thread per CPU core). We implement a *load balancer* into GW to identify to which replica it should forward the packets, and to load balance flows (the packets from the same flow are sent over the same IPSec tunnel).

Packet flow. The gateway of OP1 (GW1) intercepts TLS traffic initiated by user 1, decrypts it, identifies the right replica to forward it to, and then sends it over the IPSec tunnel. As part of this process packets are entirely encrypted (header plus payload) and a new header is added to each packet. VNF applications running inside SafeLib first call the `startEventLoop()` method of libVNF to initialize per core mTCP thread. (In each core there is an mTCP running, and in each thread there is an `epoll` loop.) When the encrypted packets arrive to DPDK from the NIC, mTCP takes them and add them to the transport buffer, and then the `epoll` loop reads them. (As part of this process a `READ` event is generated.) A VNF application running inside the enclave queries the buffer, reads the packets, decrypts them and processes them. After the packets are processed, they are encrypted and sent back to the NIC via DPDK. After SafeLib processes the traffic, it tunnels the processed traffic to GW2, which then establishes a TLS connection with user 2. Note that this is possible because OP1 and OP2 has established a trust relationship before.

IV. SafeLib: detailed architecture

Next, we present the components and deployment process of SafeLib.

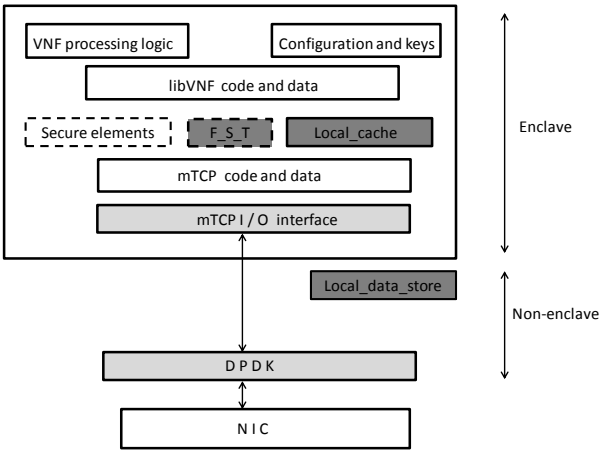

Fig. 3. Detailed SafeLib architecture: dark grey boxes denote *State management*, light grey boxes denote modified components, while dashed boxes denote newly added components

A. Components

Figure 3 shows which SafeLib components are placed inside enclave, and which components are placed outside.

I/O interface. The use of DPDK brings the opportunity of alleviating the overhead of system calls to some extent, because the packets are sent and received in batches. However, placing DPDK outside the enclave opens two ways of interaction with the enclave; via synchronous interface and via shared memory. The first option promises low performance owing to the constant entering/exiting the enclave for every packet operation. In fact, it is crucial to send and receive the batches of packets from inside the enclave. We plan to extend DPDK to support an I/O interface and modify the mTCP I/O interface to properly communicate with each other via shared memory.

State management. We modify the *State Management API* of libVNF in order to run it with desirable performance inside the enclave. To achieve that we partition the states based on active and inactive flows. We keep the active flow states inside the enclave, and encrypt the inactive states and transfer them from the enclave to a *local data store* outside. The current libVNF implementation provides a *local cache* and a *local data store* as data structures. We use *local cache* to keep the active flow states and *local data store* to keep the encrypted inactive states. For that reason we place *local cache* inside the enclave and *local datastore* outside the enclave. A new data structure will be required to provide a fast search of all flow states from inside the enclave (F_S _T).

Secure elements. Placing DPDK outside the enclave without additional modifications to the current mTCP and libVNF APIs would not be good security practice. As it is, DPDK will have direct access to the packets once they are decrypted. To enhance security, we add four functionalities.

The `sendEncryptedPacketsToEnclave()` function will be used to transfer an encrypted packet from DPDK to mTCP.

The `decryptPacket(key, cipher)` will be used to decrypt the packet inside enclave.

The `encryptPacket(key, cipher)` will be used to encrypt the packet after it is processed.

Finally, The `sendPacketToDPDK()` will be used to send the encrypted processed packets back to DPDK.

More specifically, the four functionalities above implement the IPSec endpoints. The IPSec key used to encrypt and decrypt packets is stored inside the enclave. We use industry standard encryption, such AES in GCM mode, for the security algorithms of IPSec.

B. Deployment

To protect the VNF code, our scheme accesses the raw VNF source code and then passes it through a compiler running inside the enclave. The deployment procedure consists of two phases, pre-phase and main-phase. For the pre-phase, we use an additional enclave (pre-phase enclave) with two components; a loader and a compiler.

OP1 initiates RAP to verify that the loader and the compiler are securely established inside the pre-phase enclave. As part of this process, the enclave returns a public key to OP1 who in turn encrypts the VNF code, input policies and system configuration, and transfers them to the *loader* using either a command line tool or a REST API provided by the *loader*. The *loader* decrypts the VNF code inside the enclave and sends it to the *compiler* who compiles the code. The *compiler* then returns a hash measurement of the compiled code to OP1; OP1 uses this to verify the compiled code once it is transferred to the main enclave together with the decryption key (by the *loader*) used for processing. The main enclave then decrypts the code and starts the execution. OP1 attests the main enclave using the measurement obtained by the *compiler*, and then establishes an IPSec tunnel with the main enclave for secure communication.

Note that here we do not consider OS access to the unencrypted binary. One possibility is to use the method proposed in [12]; we are currently investigating new methods.

V. Discussion and Future Work

It is clear that the next step for us is implementing and evaluating the security and performance of SafeLib. Here we also discuss some open questions regarding SafeLib's architecture and capabilities.

Placing DPDK inside the enclave. It is possible to place DPDK inside the enclave but there is a trade-off. In the case when DPDK is inside the enclave we do not need to extend DPDK and modify the mTCP I/O interface to communicate with each other via shared memory as described in Section IV-A. This would reduce the effort of developing new code. Furthermore the usage of shared memory between enclave and non-enclave regions leads to the waste of cores since they only read the user traffic from the concurrent queues. On the other hand, placing DPDK outside the enclave drastically decreases the size of the trusted computing base (TCB). As already mentioned, the smaller the TCB the better the security and

the performance, therefore we choose to place DPDK outside the enclave.

VNF chaining. In a more complex scenario, an additional network operator (OP3) outsources a VNF (VNFB) to OP2's domain. The VNF outsourced by OP1 (VNFA) communicates with VNFB using the *Communication API* of libVNF. Enclaves cannot be shared across different sockets and therefore one SafeLib instance can be run in each CPU socket. Since VNFA and VNFB are coming from different developers, isolation between the two enclaves is required, otherwise confidentiality of user data would be compromised. One possible way to provide isolation is by leveraging the DPDK features. However, we leave this as future work.

VNFs operating at L2/L3. This paper is concerned with the design of a stateful middlebox using mTCP as the backend network stack. However, libVNF also provides support for middleboxes operating at L2/L3. In such a case we can also use mTCP, but DPDK is then directly polled for packets, completely bypassing L4 in mTCP. However, the libVNF API is not expressive enough for packet header manipulation, therefore we plan to make the API more expressive in this direction in the future.

VI. Conclusion

The desire for confidentiality and the lack of trust among stakeholders in 5G multi-operator service chaining scenarios give rise to the need for protected Virtual Network Functions. In this short paper, we presented the initial architecture of SafeLib, a software middlebox platform based on Intel SGX, mTCP and libVNF. SafeLib provides confidentiality to user traffic, VNF code, policy input and state in such scenarios, while retaining both implementation flexibility for a wide range of VNFs and high performance. The implementation of SafeLib is currently underway.

References

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.

[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] G. Biczók, B. Sonkoly, N. Bereczky, and C. Boyd, "Private VNFs for collaborative multi-operator service delivery: An architectural case," in *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS*, 2016, pp. 1249–1252.

[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] S. Canard, A. Diop, N. Kheir, M. Paindavoine, and M. Sabt, "BlindIDS: Market-compliant and privacy-friendly intrusion detection system over encrypted traffic," in *ACM on Asia Conference on Computer and Communications Security, AsiaCCS*, 2017, pp. 561–574.

[6] J. Fan, C. Guan, K. Ren, Y. Cui, and C. Qiao, "Spabox: Safeguarding privacy during deep packet inspection at a middlebox," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3753–3766, 2017.

[7] 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.

[8] 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.

[9] X. Yuan, X. Wang, J. Lin, and C. Wang, "Privacy-preserving deep packet inspection in outsourced middleboxes," in *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016*, 2016, pp. 1–9.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *HASP 2013, Hardware and Architectural Support for Security and Privacy*. ACM, 2013, p. 10.

[16] E. Jeong, S. Woo, M. A. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, "mTCP: a highly scalable user-level TCP stack for multicore systems," in *11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2014*, 2014, pp. 489–502.

[17] "Data plane development kit." [Online]. Available: https://www.dpdk.org/

[18] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *2012 USENIX Annual Technical Conference*, 2012, pp. 101–112.

[19] P. Naik, A. Kanase, T. Patel, and M. Vutukuru, "libVNF: building virtual network functions made easy," in *ACM Symposium on Cloud Computing, SoCC 2018*, 2018, pp. 212–224.

[20] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker, "NetBricks: Taking the V out of NFV," in *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 203–216.

[21] A. Alim, R. G. Clegg, L. Mai, L. Rupprecht, E. Seckler, P. Costa, P. R. Pietzuch, A. L. Wolf, N. Sultana, J. Crowcroft, A. Madhavapeddy, A. W. Moore, R. Mortier, M. Koleni, L. Oviedo, M. Migliavacca, and D. McAuley, "FLICK: developing and running application-specific network services," in *USENIX Annual Technical Conference*, 2016, pp. 1–14.

[22] E. Kohler, "The click modular router," 2000. [Online]. Available: http://hdl.handle.net/1721.1/86585

[23] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ACM/IEEE 41st International Symposium on Computer Architecture*, 2014, pp. 361–372.

[24] S. Banescu, "Cache timing attacks," 2011, http://www.academia.edu/download/31092493/report.pdf.

[25] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *11th USENIX Workshop on Offensive Technologies, WOOT*, 2017.

[26] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How SGX amplifies the power of cache attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2017*, 2017, pp. 69–90.

[27] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *26th USENIX Security Symposium*, 2017, pp. 1041–1056.

[28] 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

[29] J. V. 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*, 2018, pp. 991–1008.