cSFC: Building Credible Service Function Chain on the Cloud

Shengsheng Yao*†‡, Mingwei Xu*†, Qi Li†§, Jiahao Cao*†, Qiyang Song*†‡
* Department of Computer Science and Technology, Tsinghua University
† Beijing National Research Center for Information Science and Technology, Tsinghua University
‡ Graduate School at Shenzhen, Tsinghua Univesity
§ Institute for Network Sciences and Cyberspace, Tsinghua University
{yaoss17, caojh15, songqy17}@mails.tsinghua.edu.cn; xmw@cernet.edu.cn; qi.li@sz.tsinghua.edu.cn

***Abstract*—To reduce the management costs, outsourcing network function (NF) to the cloud becomes prevalent in enterprises. This trend is increasing with the advent of network function virtualization (NFV). However, such outsourcing cannot guarantee the order and security of service function chains (SFCs) as the cloud is susceptible to attacks. In this paper, we introduce credible SFC (cSFC), a practical scheme to build secure service function chains on the untrusted cloud, cooperating with encrypted transport protocols. cSFC simultaneously shields NFs from an untrusted cloud and preserves the order of SFC sequence. Meanwhile, this scheme supports a wide range of NF functionalities and preserves the privacy of session data. We implement the cSFC prototype, and the evaluation result shows that it is practical with acceptable performance.**

I. Introduction

Today Internet session involves a wide range of intermediary network functions (NFs) in enterprise environments to improve network security (e.g., virus scanners and intrusion detection systems) and performance (e.g., compression proxies and web caches). However, implementing NFs with dedicated hardware devices bring significant problems including high costs, inflexible deployment, and complex management to enterprise. To address these problems, the academic community and industry have proposed an alternative approach that using Network Function Virtualization (NFV) to replace traditional NF implementation [1, 2]. By moving NFs out of dedicated physical devices into software applications running on commodity hardware, NFV has prompted a new service model, i.e., third-parties offer network function as a service on cloud [3]. Such a model allows enterprises outsourcing NFs to the cloud as service function chains (SFCs) [4] and obtains the benefits of cloud computing such as decreased costs and easy management.

With the warrant of deploying and managing SFC, cloud providers get full access to customer traffic and have the ability to define packet processing in NF autonomously [5]. This means that cloud providers can alter the order of SFC, tamper with NF processing and obtain enterprises' sensitive information. This is worrisome as the cloud is susceptible to threats such as data breach by cloud employees or hackers [6, 7] or NF configuration failure on cloud [5, 8]. Also, more and more cloud providers are offering edge services collaborating with third-party ISPs and CDN operators to meet low-latency performance requirements [9]. This hybrid computing infrastructure consists of multiple third-party providers which further increases the risks of NFs deployment. These weaknesses underscore the fact that, while the cloud itself might be benign, it is still difficult to promise the security of SFC.

Recent work has proposed some approaches to deal with the security issues in SFC. One approach is to perform the inspection directly on the encrypted payload without decrypting the content at middleboxes, such as BlindBox [10] and Embark [11]. However, this approach only supports simple functions, which significantly limits its extensive usage in NFs performing sophisticated operations on packet payload. Multi-Context TLS (mcTLS) [12] and Middlebox TLS (mbTLS) [13] presented new TLS extension protocol to secure multi-entity communication which require significant changes to TLS protocol, and both are hard to be widely adopted in practice.

In this paper, we strive to address four challenges for enterprises without changing standard encrypted transport protocols when outsourcing NFs to untrusted third-party cloud providers: (1) preserve the privacy of session data; (2) support a wide range of NF functionalities; (3) ensure the order of SFC sequence; (4) shield NF execution from untrusted cloud environments.

To solve the challenges mentioned above, we propose credible SFC (cSFC), a generic scheme for building credible service function chains in untrusted cloud environments. cSFC targets real-world unmet security requirements of NF outsourcing with data transmitting over encrypted transport protocols. Our design leverages an existed trusted controller [14] in the enterprise to distribute secret keys to NFs. Based on this architecture, we mainly propose three secure modules to protect NF outsourcing against attacks, while maintaining the performance and NF functionalities, i.e., *selective key distribution scheme*, *tag verification mechanism* and *secure NF execution*. The *selective key distribution scheme* aims to preserve the privacy of session data and allow NFs to support a wide range of functionalities. It only distributes unique session keys to the NFs that require access to the payloads of encrypted packets. To promise the SFC path integrity, the hop-by-hop *tag verification mechanism* performs per-hop tagging

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

and verification in each NF. Attacks that violate SFC order (e.g., NF skipping or unexpected NF appending) or maliciously modify packets can be effectively detected. Furthermore, we deploy above security designs and NF processing in secure execution environments based on Intel Software Guard Extensions (Intel SGX) [15] to provide *secure NF execution* in an untrusted environment. This promises that even when the Basic Input Output System (BIOS) or Virtual Machine Monitor (VMM) is compromised, the attacker cannot observe the data in unencrypted form or tamper the NF execution.

We implement the cSFC prototype with OpenSSL and Intel SGX SDK for Ubuntu in the testbed and evaluate the performance. cSFC introduces extra operations including the tag verification and the cryptographic operations. We found that the tag verification in cSFC only degrades 1.3% throughput and incurs 1.2% processing delay compared to the native NF. Moreover, compared to the mbTLS scheme which supports a wider range of NFs than other solutions, cSFC can improve the performance by 26% on average. Finally, we demonstrate that implementing NF functionalities inside an SGX enclave slighly increases the processing delay, i.e., approximately 3% processing delay on average.

In summary, we make three contributions in this paper:

- We design a generic scheme for building credible service function chains in untrusted cloud environments without changing standard encrypted transport protocols.
- We conduct a comprehensive security analysis and demonstrate that our scheme can ensure the SFC path integrity, preserve the data privacy and provide secure NF execution.
- We implement the cSFC prototype and evaluate its performance. The results show that cSFC incurs negligible overhead.

II. Problem Statement

A. Usage Scenarios

Our scheme targets the scenario where SFCs is used to process the confidential data of customers in untrusted cloud environments. Apart from the advantages of flexible deployment and low costs, NF outsourcing has also introduced risks such as out-of-order NF traverse, data leakage and packets tampering. These cases can be caused by malicious attacks or accidents such as link failure. Moreover, the underlying infrastructure is managed by cloud providers themselves or, in some cases, by third-party ISPs and CDN operators to meet strict low-latency performance requirements in edge networks. Complex underlying infrastructures increase the NF outsourcing risks because of the multiple managers of NF processing.

B. Threat Model

We assume the cloud can be compromised or controlled by an attacker [13, 16]. The attacker can achieve the root privilege of the cloud provider software, the kernel or hypervisor [17]. We summarize the attacks as follows:

Undermine SFC Order. The SFC sequence defined by customers may be violated due to misoperations by cloud employees or possible attacks from attackers. Such violations have three basic forms: (1) *Disordering SFC Sequence*. When malicious cloud employees incorrectly configure switches or an attacker makes some malicious operations, the traffic may be processed in an incorrect NF sequence order. (2) *NF Skipping*. Insufficient NF resources, an accident like link failure or NF crash may lead lazy cloud employees to skip some NF instance. (3) *Unexpected NF Appending*. An attacker may leverage NF appending to perform some malicious operations on customer traffic such as collecting confidential user information or modifying the traffic data.

Subvert NF Execution. The attacker may try to subvert NF execution such as modifying the filtering or classification rules in NFs or changing any other processing procedure. Note that CPU is trusted in our assumption.

Access or Modify Confidential Data. With the capability of control software stacks, an attacker can access the confidential communication data between *endpoints*. In particular, the attacker can sniff the session data or alter the communication data.

C. Design Goals

Security. We strive to guarantee SFC path integrity, NF process correctness and traffic confidentiality on the cloud against a powerful adversary.

Generality. Our design aims to support a wide range of network functions.

Performance. We try to build a high-performance prototype the system with negligible overhead.

III. Credible SFC Design

A. Overview

Our scheme mainly depends on three mechanisms to build credible service function chains: (1) *Unique Tag Verification.* In cSFC, each NF leverages a unique key to generate a tag appending on the packet to verify the path integrity of SFC. This prevents the attackers from forwarding packets in a wrong SFC sequence. (2) *Selective Key Distribution.* To support a wide range of NF functionality, the cSFC controller allocates unique symmetric keys to NFs that need to perform operations on packet payloads. It enables these NFs to process payloads by decrypting packets with allocated keys. (3) *Seucre NF Execution.* We implement core NF processing in a secure execution environment named Intel SGX [15] enclave, which runs code in possessed partial encryption memory isolated from the rest of the system. It protects NF processing and confidential communication from untrusted infrastructures. In our design, important operations such as tag verification and original NF processing are implemented in the enclave to provide high-security properties. This design also improves the packet I/O efficiency by reducing unnecessary data transfer between the enclave and host part.

As Fig. 1 illustrates, the architecture of cSFC consists of two parts: the controller and NF entities. The centralized and trusted controller manages all NFs, which is similar to

APLOMB [3] and vSFC [14]. It verifies the identities of NFs. Afterward, it distributes unique keys to NFs. Such keys are used to preserve the privacy of traffic and verify the integrity of the SFC path. Anomalies of packet processing in the NF will be reported to the controller. In the NF part, each entity is further divided into the host part and the enclave part. The host part is responsible for communication between NFs and interaction with the enclave. The enclave part executes core functions of NFs. In Fig. 1, the tag verification module firstly verifies the correctness of the tag in a packet to check the integrity of the SFC path. If the NF requires accessing packet payloads, the decrypt module will decrypt payloads before the original NF processing. After processing packet payloads, the encrypt module encrypts payloads and generates a new tag appending on the packet. Note that NFs that only need access packet headers will skip the cryptographic modules to avoid unnecessary costs.

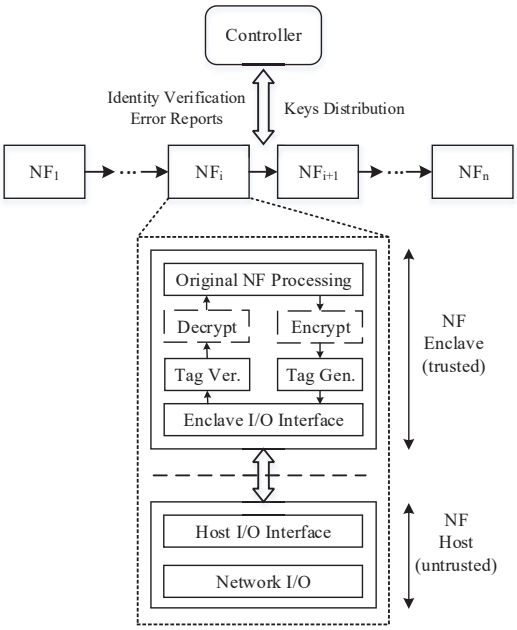

Fig. 1. The architecture of cSFC.

B. cSFC Mechanisms

Unique Tag Verification. The controller generates a key for each hop in SFC. Moreover, the controller distributes these keys to specified NFs through a secure channel [3]. We achieve the secure channel by encrypting the communication between the controller and the verified NF. Each NF can leverage its key to generate a unique $Auth$ based on full packet except the $Auth$ field, which is shown in Equation (1).

$$Auth_i \leftarrow HMAC(DATA, Key_{NF_i - NF_{i+1}}). \quad (1)$$

The new tag will replace the old $Auth$ field in packet. In our design, each NF receives two keys consisting of the key of next hop and the key of prior NF to achieve hop-by-hop SFC path verification. As shown in Fig. 1, whenever an NF receives a packet, it uses the key of prior hop to compute all packet except $Auth$ field and compares the result with original packet $Auth$ field to verify prior NF identity. We can use $Auth$ to verify whether or not a packet is modified by a legal NF. The violations of $Auth$ will be reported as an error to the controller through a secure channel.

Selective Key Distribution. This mechanism allows the controller to generate a symmetric key for each hop between two NFs that require accessing the packet payload. These unique keys are only assigned to NFs that need access packet payloads, which promise each NF granted with the least privilege. For example, an IP firewall that only needs access to packet header will not get symmetric keys in our design and the cryptographic modules will be skipped. While cryptographic modules are indispensable for Deep Packet Inspection (DPI) which need access packet payload. The controller distributes these unique keys to specified NFs through a secure channel. After the key distribution phase, these specified NFs get two keys including their prior hop key and next hop key. These NFs leverage prior hop key to decrypt the packet payload and use the next hop key to encrypt. For instance, in Fig. 2, NF_{i+1} is a NF that only need access packet header will not get a key and skip cryptographic modules. NF_i, NF_{i+2} and NF_{i+3} need access packet payloads. NF_{i+2} gets two keys: $Key_{NF_i - NF_{i+2}}$ and $Key_{NF_{i+2} - NF_{i+3}}$. In NF_{i+2}, decryption module will transform the encrypted payload encrypted by NF_i into plain-text for further NF processing with $Key_{NF_i - NF_{i+2}}$, then NF_{i+2} can perform agile operations on packet payloads. After NF processing, the encryption module will encrypt the payload with $Key_{NF_{i+2} - NF_{i+3}}$ and deliver the packet to NF_{i+3}.

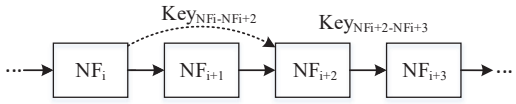

Fig. 2. Selective-hop key verification.

Secure NF Execution. To prevent possible attacks that may subvert NF execution, the outsourcing NFs should run in a secure execution environment and the specific functions of these NFs should be correctly configured. The *Secure NF Execution* mechanism achieves these goals through two steps as follows. Firstly, the remote attestation in Intel SGX is enabled for the trusted *controller* to verify the identity of each NF instance deployment. Thus, the controller can judge the correctness of NF configuration by verifying the attestation report which includes measurement results of the code and data in the enclave. Secondly, the core functions of each NF are implemented in the SGX hardware assistant enclaves. As shown in Fig. 1, the NF application is divided into two parts including the trusted enclave part and the untrusted host part. The enclave part performs the core NF processing including original NF functions, cryptographic module, tag processing, and enclave I/O interface. Enclaves can provide a

secure execution environment for these operations via hardware protection. The host part is responsible for network communication between NF entities and interaction with the enclave part by enclave I/O interfaces. Such design improves the performance of cSFC by reducing unnecessary packet I/O between the enclave part and the host part.

IV. Security Analysis

SFC Order Protection. This follows the fact that cSFC uses a unique tag verification in the trusted enclave part of each NF. We take Fig. 2 as an example to analyze the security properties. The correct packet forwarding order of the SFC sequence is from NF_i to NF_{i+3} in Fig. 2.

- *Disordering SFC Sequence.* Suppose an adversary sniffs a packet from NF_i and tries to forward the packet in a malicious order: $NF_i - NF_{i+2} - NF_{i+1} - NF_{i+3}$. When the attacker forwards the packet from NF_i to NF_{i+2}, the packet will be appended a $Auth$ field which is obtained with $HMAC(DATA, Key_{NF_i - NF_{i+1}})$. However, NF_{i+2} expects the $Auth$ field secured with $HMAC(DATA, Key_{NF_{i+1} - NF_{i+2}})$. Thus, the verification of the $Auth$ fails in NF_{i+2} and the error will be reported to controller.
- *NF Skipping.* Due to an accident like NF crash or malicious operations, the packet may be forwarded in a sequence like: $NF_i - NF_{i+2} - NF_{i+3}$ (thereby skipping NF_{i+1}). However, the tag verification in NF_{i+2} fails, which is similar to the example described in the *Disordering SFC Sequence*. Therefore, the NF skipping attack will be detected in cSFC.
- *Unexpected NF Appending.* As shown in Fig. 3, an adversary appends a malicious NF between NF_{i+1} and NF_{i+2} to collect or modify confidential user data. Without the key owned by NF_{i+1} and NF_{i+2}, this malicious NF can not compute a valid $Auth$ to append on packet headers. Thus, NF_{i+2} can detect the malicious operation by tag verification and report this error to controller.

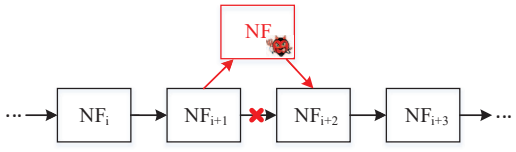

Fig. 3. NF appending attack.

Secure NF Execution. Through NF verification, we can make sure that the code and data are indeed deployed in an enclave. In our design, as shown in Fig. 1, we implement the confidential operations such as the tag verification, decryption and original NF processing in the hardware-based enclave. These codes and data are encrypted in the memory, and only the CPU and the enclave can access. In this way, the enclave provides high-level protection for sensitive codes and data based on hardware, which prevents the attacker from accessing protected data or tampering the execution of NFs.

Confidential Data Security. We will discuss this security property in two aspects as follows:

- *Data Privacy.* Decrypting confidential session data requires access to one of the symmetric keys shown in Fig. 2, such as $Key_{NF_i - NF_{i+2}}$ and $Key_{NF_{i+2} - NF_{i+3}}$. These keys are distributed to the NF instances over a secure channel. Moreover, such keys are processed inside the enclave, which promises that only the enclave and CPU can access the keys. Thus, an attacker cannot access these keys and plain-text traffic even if the operating system is subverted.
- *Data Authentication.* We append a $Auth$ field to each packet, that is a small tag generated by the specific key and packet content. NF instances can detect illegal changes in a packet by matching the $Auth$ field. Only the NFs authorized by the controller can modify these confidential data.

V. Evaluation

A. Experimental Setup

We evaluate the performance of cSFC by comparing our architecture with an insecure baseline and the mbTLS [13]. Here we choose mbTLS as a comparison because it supports a wider range of NFs than other solutions while protecting the security of NF outsourcing.

Testbed. We implement the cSFC prototype with the Intel SGX SDK for Ubuntu (v2.1). The machine is based on an SGX-enabled Intel Core i7 7700 processor with 4 cores running at 3.6GHz. Cipher suites use cryptographic functions in Intel SGX SDK. The testbed has 8 GB memory and runs Ubuntu 16.04 LTS with Linux kernel version 4.13.

NF Entity. To measure the performance of cSFC, we implement several types of NF entity as follows:

- *Simple NF.* The NF that only requires access to packet header with tag operations.
- *Complex NF.* The NF instance that requires access to traffic payload with both tag and cryptographic operations.
- *Basic NF.* The NF only performs original network functions acting as an insecure baseline.
- *MbTLS NF.* We implement the NF described in mbTLS [13] that leverages unique per-hop key to preserve the order of SFC sequence.

Method. In our experiments, we evaluate the performance of cSFC from the following aspects: (1) the overhead of performing tag verification, tag generation, encryption, and decryption; (2) the packet processing latency and throughput in different NF types; (3) the impact of changing the *Simple NF* and *Complex NF* proportion in SFC; (4) the extra overhead of SGX brought to NF processing.

B. Overhead of Extra Operations

Here we investigate the overhead of tag verification, tag generation, decryption, and encryption operations. In our experiments, we create a separate connection for each hop.

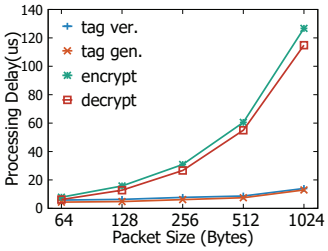

Fig. 4. Extra processing delay.

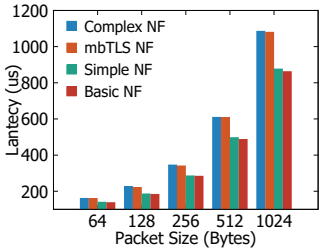

Fig. 5. Latency of various NFs.

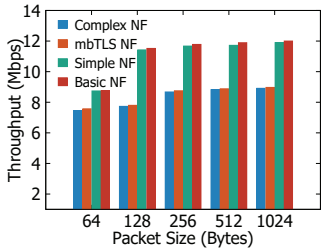

Fig. 6. Throughput of various NFs.

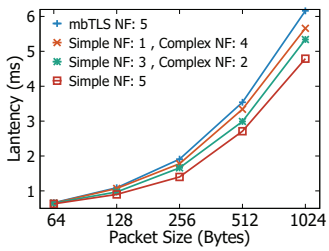

Fig. 7. Latency of cSFC.

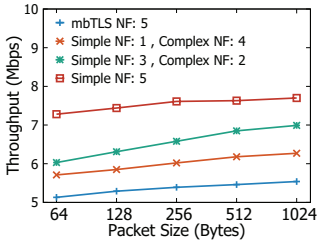

Fig. 8. Throughput of cSFC.

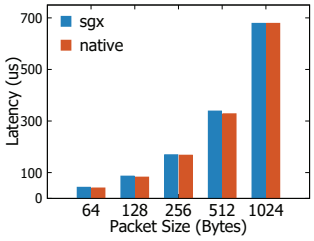

Fig. 9. SGX overhead on NF.

The smallest packet size is 64 bytes, involving 20 bytes IP header, 20 bytes TCP header, 16 bytes tag verification field and 8 bytes payload. To figure out the overhead of these extra operations, we report the processing delay of each operation by varying different packet sizes, i.e., 64 bytes, 256 bytes, 512 bytes, and 1024 bytes.

Cryptographic modules use AES-CBC algorithm with 128 bits block size. Fig. 4 presents the processing delay of tag and cryptographic operations impacted by the packet size. The decryption and encryption modules significantly consume more computational resources than tag operations due to sophisticated cryptographic operations. The processing delay of the encryption climbs from $7.85us$ to $126.71us$ with the growth of the packet size and the decryption takes around $54.84us$ on average. The overhead of tag generation ranges from $4.37us$ to $12.94us$ and the tag verification takes up to $13.97us$ as more operations are performed.

C. NF Entity Performance

We measure the latency and throughput of cSFC consisting of a single NF with different packet sizes to evaluate the impacts introduced by Simple NF and Complex NF. Here, we set the single NF to four types described in Section V-A: simple NF, complex NF, mbTLS NF, and basic NF. The packet size changes from 64 bytes to 1024 bytes like the way in Section V-B. Fig. 5 presents the processing latency of four kinds NF respectively. We can see that the latency is increasing with the packet size growing. Compared to the basic NF, a simple NF introduces negligible latency due to simple operations of tag verification and generation. Since the cost of cryptographic operations is relatively expensive, the mbTLS NF adds approximately 20% latency of the basic NF. The complex NF involves extra tag operations which introduces extra 0.3% latency compared with the mbTLS NF.

The throughput of the four types of NFs is shown in Fig. 6. We can observe that the simple NF only incurs 2.2%, 0.87%, 0.93%, 1.4% and 0.83% throughput degradation with different packet sizes, which is minor compared to the basic NF. The throughput of the complex NF is also close to the mbTLS NF with 0.91% throughput degradation on average. From the experimental results, we can conclude that the operations on tags incur negligible delay and throughput degradation. For NFs that perform sophisticated operations on packet payload, the cryptographic operations are inevitable. In such a case, the complex NF also introduces small latency and throughput degradation compared to the mbTLS NF. In order to guarantee NF supporting all functions while ensuring data privacy, cryptography is essential. Compared to existing solutions, above results demonstrate that the overhead of our scheme is acceptable.

D. The Impact of Different NF Proportions

In real scenarios, a service function chain consists of various NFs performing different operations on packet headers or payload. Thus we evaluate the performance of cSFC composed of simple NFs and complex NFs with different proportions. Previous studies [3] demonstrates that the middlebox deployments in enterprise networks consist of more simple NFs (e.g., IP firewall, WAN optimization or proxy) than complex NFs (e.g., IDS/IPS). The worst case is that all complex NFs compose cSFC, which is almost impossible in real enterprise networks. Thus, we do not measure such case. In our experiments, we set the length of cSFC to 5, the number of simple NFs is 1, 3 or 5, and the rest are complex NFs. Meanwhile, we implement an SFC consisting of 5 mbTLS NF as the baseline. We measure the latency and throughput with different packet sizes. As shown in Fig. 7, compared to the mbTLS scheme, the latency decreases about 6%, 13%, 22% when the number of simple NFs in cSFC changes in 1, 3, 5, respectively. Fig. 8 presents the throughput with different packet sizes. The results show that the throughput of cSFC is around 26% better than the mbTLS scheme.

E. SGX Overhead

Here we evaluate the impacts of enclaves on original NF processing delay. We implement the NF as an IDS. We set the native NF which implements network functions without enclaves as the baseline. Fig. 9 shows the processing delay impacted by SGX with different packet sizes. We can observe that SGX only introduces a small processing delay compared to the native NF. in SGX, the latency increases only around 3% on average and 7.5% in the worst case.

VI. Discussion

Some researches find that SGX is vulnerable to attacks such as page fault attacks [18], page table attacks [19], cache-based attacks [20] and branch-prediction attacks [21]. At the same time, various defenses have been proposed to solve these problems. T-SGX [22] uses Transactional Memory (TSX) to prevent the page table attacks. A software approach [23] is proposed to defeat cache-based attacks. OBFUSCURO [24] provides a solution against all memory-based side-channel attacks. Investigating the impacts of these proposals in the NFV scenario and developing solutions are future work.

VII. Related Work

NF Outsourcing. The earliest study APLOMB [3] proposes a practical service for enterprise outsourcing middleboxes to cloud with the advantage of resource conserving and high performance. Besides, E2 [25] proposes a opensource framework which provide the operator with a single coherent system for deploying NFs and relieving developers from having to develop specific NF solutions under different circumstances. However, these approaches can not consider the security and privacy of NF processing in untrusted cloud environments. Instead, cSFC provides strong, secure properties for NF outsourcing.

Cryptographic Approaches. To ensure desirable privacy properties as well as support multiple functionalities of middleboxes, BlindBox [10] and Embark [11] advocate leveraging new encryption schemes to perform network data processing directly on encrypted traffic. However, BlindBox only supports DPI filtering, such as IDS and parental filtering. Embark tries to extend BlindBox and supports a wider range of NFs. However, only simple operations are supported in this approach, such as "$\in$" and "$=$". Above approaches have a significant limitation on NF functionalities and cannot meet the sophisticated needs of enterprises. In contrast, cSFC supports all operations that the original NF supports.

TLS Extension. With the increasing use of TLS in Internet, Multi-context TLS (mcTLS) [12] and Middlebox TLS (mbTLS) [13] explore other alternatives utilizing TLS extension to provide security properties for NF outsourcing. mcTLS grants the endpoints to define clear access rights (read/write) for each middlebox. mbTLS leverages per-hop key to ensure the order of NF processing and use Intel SGX to protect middleboxes from untrusted hardware. Nevertheless, mbTLS does not take into account the security issues of SGX. Instead, we propose a selective cryptographic computation solution to ensure the least privilege of NFs.

NF Verification. vNFO [5] provides a solution for the customer to verify whether the service functionality, performance, and accounting are performed on the cloud as the customer expected. But vNFO does not concern confidential data leakage. vSFC [14] proposes a generic and agile framework to verify the correctness of SFC in realtime. However, this approach can not provide trusted NF execution and data privacy protection. In our scheme, cSFC realizes this requirement through encryption transmission and hardware enclaves.

VIII. Conclusion

In this paper we propose cSFC, a practical scheme to build secure service function chains on the cloud. cSFC can protect SFC from a powerful adversary and support a wide range of NF functionalities. We conduct a comprehensive security analysis of cSFC and demonstrate that cSFC can provide strong security properties for NF outsourcing. We implement the cSFC prototype, and the evaluation result shows that cSFC introduces a negligible overhead.

Acknowledgment

The research is supported by the National Key R&D Program of China under Grant 2018YFB1800304, the National Natural Science Foundation of China (NSFC) under Grants 61625203, 61832013, 61572278, 61811530336 and U1736209. Mingwei Xu is the corresponding author of the paper.

References

[1] Mijumbi, Rashid, et al., "Network function virtualization: State-of-the-art and research challenges," in *Communications Surveys & Tutorials*. IEEE, 2016.
[2] "Network functions virtualisation," White Paper, EUROPEAN TELECOMMUNICATIONS STANDARDS INSTITUTE, 2012.
[3] Sherry, Justine, et al., "Making middleboxes someone else's problem: network processing as a cloud service," in *SIGCOMM*, 2012.
[4] D. Dolson, et al., "Service function chaining," IETF Draft, 2018.
[5] Fayazbakhsh, Seyed Kaveh, et al., "Verifiable network function outsourcing: requirements, challenges, and roadmap," in *HotMiddlebox*, 2013.
[6] K. POULSEN. (2008, May) Five irs employees charged with snooping on tax returns. [Online]. Available: https://www.wired.com/2008/05/five-irs-employ/
[7] K. ZETTER. (2010, Sep.) Ex-googler allegedly spied on user emails, chats. [Online]. Available: https://www.wired.com/2010/09/google-spy/
[8] Gill, Phillipa, et al., "Understanding network failures in data centers: measurement, analysis, and implications," in *SIGCOMM*, 2011.
[9] Varghese, Blesson, et al., "Challenges and opportunities in edge computing," 2016.
[10] Sherry, Justine, et al., "Blindbox: Deep packet inspection over encrypted traffic," in *SIGCOMM*, 2015.
[11] Lan, Chang, et al., "Embark: Securely outsourcing middleboxes to the cloud." in *NSDI*, 2016.
[12] Naylor, David, et al., "Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS," in *SIGCOMM*, 2015.
[13] Naylor , David, et al., "And then there were more: Secure communication for more than two parties," in *CoNEXT*, 2017.
[14] Zhang, Xiaoli, et al., "Generic and agile service function chain verification on cloud," in *IWQoS*, 2017.
[15] C. Intel. (2018) Intel software guard extensions (intel sgx) homepage. [Online]. Available: https://software.intel.com/en-us/sgx
[16] Baumann, Andrew, et al., "Shielding applications from an untrusted cloud with haven," in *TOCS*, 2015.
[17] Schuster, Felix, et al., "Vc3: Trustworthy data analytics in the cloud using sgx," in *S&P*, 2015.
[18] Xu, Yuanzhong, et al., "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *S&P*, 2015.
[19] Van Bulck, Jo, et al., "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *USENIX Security*, 2017.
[20] Schwarz, Michael, et al., "Malware guard extension: Using sgx to conceal cache attacks," in *DIMVA*, 2017.
[21] Evtyushkin, Dmitry, et al., "Branchscope: A new side-channel attack on directional branch predictor," in *ACM SIGPLAN Notices*, 2018.
[22] Shih, Ming-Wei, et al., "T-sgx: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.
[23] Zhou, Ziqiao, et al., "A software approach to defeating side channels in last-level caches," in *CCS*, 2016.
[24] Ahmad, Adil, et al., "Obfuscuro: A commodity obfuscation engine on intel sgx," in *NDSS*, 2019.
[25] Palkar, Shoumik, et al., "E2: a framework for nfv applications," in *SOSP*, 2015.