Private Processing of Outsourced Network Functions: Feasibility and Constructions[*]

Luca Melis
University College London, UK
luca.melis.14@ucl.ac.uk

Hassan Jameel Asghar
Data61, CSIRO, Australia
hassan.asghar@data61.csiro.au

Emiliano De Cristofaro
University College London, UK
e.decristofaro@ucl.ac.uk

Mohamed Ali Kaafar
Data61, CSIRO, Australia
dali.kaafar@data61.csiro.au

ABSTRACT

Aiming to reduce the cost and complexity of maintaining networking infrastructures, organizations are increasingly outsourcing their network functions (e.g., firewalls, traffic shapers and intrusion detection systems) to the cloud, and a number of industrial players have started to offer network function virtualization (NFV)-based solutions. Alas, outsourcing network functions in its current setting implies that sensitive network policies, such as firewall rules, are revealed to the cloud provider. In this paper, we investigate the use of cryptographic primitives for processing outsourced network functions, so that the provider does not learn any sensitive information. More specifically, we present a cryptographic treatment of privacy-preserving outsourcing of network functions, introducing security definitions as well as an abstract model of generic network functions, and then propose a few instantiations using partial homomorphic encryption and public-key encryption with keyword search. We include a proof-of-concept implementation of our constructions and show that network functions can be privately processed by an untrusted cloud provider in a few milliseconds.

Keywords

NFV privacy; homomorphic encryption; searchable encryption

1. INTRODUCTION

Network functions, such as firewalls and load balancers, are increasingly moving to "the cloud" by means of software processes outsourced on commodity servers. Using virtualization, network functions can be emulated in software in a cost-effective manner, and outsourced to the cloud reaping the benefits of reduced management and infrastructure costs, pay-per-use, etc. [15]. Specifically, network function virtualization (NFV) is currently being proposed by several major industrial operators like Cisco, Alcatel-Lucent, and Arista, as a service to multiple clients [14]. In such a multi-tenant setting, network functions are run on virtual machines (VMs) belonging to different clients hosted on the same hardware (server). Naturally, this raises a number of security concerns for clients, including confidentiality and integrity. While such issues are common to IT infrastructure outsourcing in general [19], more specific to NFV is the sensitivity of an organization's proprietary network policies, which instruct how network functions are to be performed. These are potentially vulnerable to compromise from competing organizations as well as the cloud service provider itself. For instance, firewall rules do not only reveal IP addresses of hosts and network topology but also defense strategies and sensitivity of different services and resources, which in the traditional setting are only known to a few network administrators [8, 17].

This motivates the need to protect the privacy of network policies against an untrusted cloud provider, as well as other tenants and third parties. We call this the *private NFV problem*, which, as we discuss in Section 2, has been largely overlooked by prior work on NFV security. We construct an abstract model of network functions, which seeks to generalize most network functions used in practice, as well as relevant adversarial models (Sections 3 and 4). Based on this abstraction, we propose two different solutions: one based on partial homomorphic encryption and the other based on public-key encryption with keyword search (PEKS) (Section 5), secure in two different adversarial models, which we define as *strong* and *weak*. Our solution against the weak adversary is also the first to include stateful network functions, e.g., a stateful firewall that keeps track of open TCP/IP connections. Finally, we present a proof-of-concept implementation of our schemes and evaluate their performance overhead using an outsourced firewall as a use-case (Section 6). Using a typical 5-tuple based firewall rule, we show that a packet can be processed within 109 ms and 180 ms, respectively, using our solutions secure against the weak and the strong adversary, and demonstrate that our schemes scale quite well, as processing times reach 250 ms and 1,208 ms, respectively, using 10 rules. Bearing in mind that our proof-of-concept implementation is not optimised for efficiency (e.g., lack of multi-threading), our results indicate that private NFV is feasible using existing cryptographic primitives.

2. RELATED WORK

Khakpour and Liu [8] introduce a data structure called Bloom Filter Firewall Decision Diagram (BFFDD) in order to anonymize firewall policies built from Firewall Decision Diagrams (FDD) [7]. However, as acknowledged by the authors, Bloom filters [2] naturally introduce false positives. Thus, occasionally, packets that do not match any policy are (mistakenly) dropped by the firewall.

[*]The full version of this paper is available at IACR's Cryptology ePrint Archive: https://eprint.iacr.org/2015/949

SDN-NFVSec'16, March 11, 2016, New Orleans, LA, USA.
© 2016 ACM. ISBN 978-1-4503-4078-6/16/03...$15.00
DOI: http://dx.doi.org/10.1145/2876019.2876021

Furthermore, security/privacy of their solution is argued against a black-box assumption of Bloom filters, which does not analyze the security properties of Bloom filters themselves (such as one-wayness).

Shi, Zhang, and Zhong [17] use multilinear maps from Coron, Lepoint and Tibouchi (CLT), which are based on *graded encoding systems* [6], to encode each bit of a firewall rule as a pair of level-1 encodings and a level-$(n+1)$ encoding for the whole rule, where n is the length of a possible packet. Following the security properties of the multilinear map, it is not possible to obtain level-i or lower encodings given a level-$(i+1)$ encoding for each i. Upon receiving a packet, the encodings corresponding to the bits of the packet are multiplied and the result is then matched with the level-$(n + 1)$ encoding for the whole policy through a procedure called `isZero`. Unfortunately, the CLT construction has been recently shown to be insecure, due to an attack on the `isZero` routine [5]; a key ingredient to check if a packet matches a policy.

Although both these constructions focus specifically on outsourcing firewalls, they exclude details of how state tables can be maintained in their framework by a stateful firewall. Furthermore, due to being specific to firewalls, their solutions are only relevant to policies that result in a binary decision (allow or deny), excluding network functions that modify packet contents or perform more complex actions. Compared to these two solutions, our solutions for private NFV cover a much broader range of network functions, including firewalls, and also consider state tables.

Private NFV also resembles real-time processing over encrypted packets. The work in [16] discusses deep packet inspection over encrypted data, however, it requires the sender (third party) to be a participant in the protocol, which makes it impossible to use this solution on existing infrastructures (a requirement that we describe as compatibility in Section 3.2).

3. PRELIMINARIES

3.1 System Model

Examples of Network Functions. Examples of the type of network functions and associated policies considered in this paper are firewalls, load balancers, IDS and carrier-grade NAT. For an illustration of a policy, see the example in Section 4.

Cloud and Client Middleboxes. We consider a scenario where an organization, the *client*, outsources its network functions to the *cloud*, as illustrated in Figure 1. The outsourced network functions run within virtual machines (VMs) on the cloud servers. We call this the *NFV setting*, as opposed to the *traditional* setting in which dedicated network middleboxes perform network functions within the client's private network. The outsourced VMs are collectively called the *cloud middlebox (MB)*. Not all network functionalities need to be outsourced to the cloud, and as such the client still requires its own middlebox to carry out the remaining network functions or to communicate with the cloud MB. We call this the *client MB*. The network policies which describe how the network functions are to be processed are installed in the cloud MB by the client. The cloud MB receives inbound traffic destined for the client, processes the network functions assigned to it, and forwards the result to the client MB. Outbound traffic is the one originating from within the client's private network.

Trust Assumptions. We assume the cloud MB to be honest-but-curious, i.e., it performs network functions dutifully yet wishes to infer the policies. For some proposed solutions, we will assume the cloud MB to have a *semi-trusted* component, which we call the *entry MB*. The entry MB receives the packet and performs some

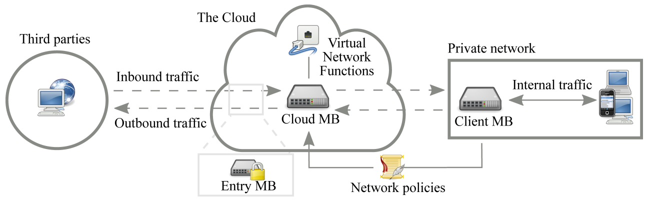


Figure 1: Network Function Virtualization.

preliminary processing before handing the results over to the cloud MB. Inclusion of an entry MB remarkably improves performance. We stress that the entry MB does not share any private keys with the client MB, and all the processing is done using public-key operations.

3.2 Desired Properties

Privacy. The client expects its network policies to remain hidden not only from third parties, but also from other tenants and the cloud.

Performance. The client expects the outsourced network functions to maintain the quality of service of the traditional setting, i.e., real-time processing and minimal client-side intervention.

Compatibility. Third parties should be able to send/receive traffic to/from the client as if the network functions are implemented in the traditional setting, without the need for additional setup (e.g., implementation of customized network and cryptographic protocols) to communicate with the client.

Our goal is to explore the balance between security and performance, while satisfying the compatibility constraint.

3.3 Limitations and Scope

Before introducing our solutions, we discuss a few limitations of our model and make some important remarks.

Traffic Analysis. An adversary may intercept and analyze traffic between the cloud MB and a third party and try to infer network policies based on the pattern of inbound and outbound packets. However, note that this can also be done in the traditional setting.

Virtual Machine Isolation. One way to achieve private NFV is through VM isolation, e.g., isolation of memory and disk storage, together with the assumption that the hypervisor belongs to a trusted base [9, 19]. A crucial aspect for secure isolation is to ensure that the hypervisor, i.e., trusted computing base, is small in terms of lines of codes (LoC) [18, 19]. Unfortunately, commodity hypervisors are not optimized in terms of lines of codes [19]. This approach is also vulnerable to cross-VM side-channel attacks where a malicious VM is co-located at the physical host of the target VM and exploits various side channels (e.g., cache) to obtain information such as cryptographic keys [13, 20].

Coverage of Network Functions. Although our definition of network functions is broad enough to cover many network functions, we cannot claim that it covers all network functions in practice. Our definition can be incrementally modified to cover all network functions. A case in hand is traffic shaping, where delivery of certain packets is delayed to satisfy performance guarantees, which we do not currently consider (at the cloud MB).

Inbound vs Outbound Traffic. In this work, we focus on *inbound* traffic, i.e., traffic coming from third parties toward the client. Although our private NFV solutions are applicable to outbound traffic as well, this would require redirecting traffic from the cloud MB (after private processing of network functions) to the client MB, which in turn forwards it to the third party receiver.

4. MATHEMATICAL FORMULATION

Let n be a positive integer and $\mathbf{x}$ and $\mathbf{y}$ be n-element vectors: then $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes their dot product. The Hadamard product or the entry-wise product of the vectors $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \circ \mathbf{y}$, i.e., the n-element vector whose i-th element is $x_i y_i$. The vector $\mathbf{e}_i$ denotes the n-element vector with all 0s except a 1 in the i-th position. The encryption function E on a vector $\mathbf{x}$ is defined as the vector $E(\mathbf{x}) = \begin{pmatrix} E(x_1) & E(x_2) & \cdots & E(x_n) \end{pmatrix}$.

4.1 Network Functions

Let $n \geq 1$ and $q \geq 2$ be positive integers. We define a packet $\mathbf{x}$ as a vector in $\mathbb{Z}_q^n$, where n represents the number of fields of the packet (source IP address, protocol type, etc.) and q is an upper bound on the length of packet fields. A network function ψ from $\mathbb{Z}_q^n$ onto $\mathbb{Z}_q^n$ is the pair (m, a) defined as

$$\psi(\mathbf{x}) = m(\mathbf{x})a(\mathbf{x}) + (1 - m(\mathbf{x}))\mathbf{x}, \tag{1}$$

where $m : \mathbb{Z}_q^n \rightarrow \{0, 1\}$ is called the matching function, and $a : \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q^n$ is the action function. The intuitive meaning of the above is that when a network function receives a packet $\mathbf{x}$, the matching function decides whether the current policy applies to this packet. If yes, the relevant action is performed by the action function. If the result of the match is zero, the packet is left unchanged. Composition of network functions is defined as $\psi^i(\mathbf{x}) = \psi_i(\cdots \psi_2(\psi_1(\mathbf{x})) \cdots)$ for $i \geq 1$.

The definition of ψ as a match-action pair is motivated by the OpenFlow communications protocol between the control and forwarding planes in Software Defined Networks (SDN) [11]. Note that different fields of a packet are not necessarily of the same length, e.g., the version field of an IPv4 packet is 4 bits long while the source IP field is 32 bits long. Therefore, we consider a value q that is large enough to incorporate the largest header field. The packet payload, which can be much larger, is divided into chunks of length $\log_2 q$ bits.

Virtual Fields. Besides the standard fields, we assume the presence of additional *virtual* fields. These originate from the implementation of our private NFV instantiations and are inserted in the payload of the packet. For instance, a *tag* field with the value `drop` assigned by cloud MB in the case of firewalls.

Example. We assume a simple network address translation (NAT) policy as a running example. For instance, upon receiving a packet $\mathbf{x}$ with destination IP in the range `128.*.*.*`, the NAT changes the destination IP and port to `196.*.*.*` and `22`, respectively. Without loss of generality, we assume that the destination IP and destination port belong to the first two elements of $\mathbf{x}$, i.e., x_1 and x_2. Thus, the matching function is:

$$m(\mathbf{x}) = \begin{cases} 1 & \text{if } x_1 \in [\texttt{128.0.0.0}, \texttt{128.255.255.255}] \\ 0 & \text{otherwise} \end{cases},$$

and the action $a(\mathbf{x})$ is adding `68.0.0.0` to x_1, $-x_2 + 22$ to x_2, and 0 to x_i, for $i \in \{3, \ldots, n\}$. Note that the IP addresses are mapped in $\mathbb{Z}_q$.

4.2 Stateful Network Functions

Some network functions such as (stateful) firewalls maintain dynamically generated states. When a packet arrives, it is first checked against the state table to see if any entry in the state table matches the fields of the packet. If a matching entry is found, the prescribed action is performed on the packet and it does not need to be further processed by other (static) policies. An example is the state of TCP connection maintained by a firewall, as depicted in Table 1. We note that in our model, state tables can be abstracted as dynamic match-action pairs, where the state and time-out columns in the state table can be thought of as *virtual* fields of the IP packet and the action as the addition of the `tag` field with value "allow". However, one key difference is that once a match has been found further processing is discontinued,[1] otherwise there would be no performance gain from maintaining states.

ID	src IP	src port	dst IP	dst port	prot	state	timeout
1	192.168.1.1	120	192.168.1.2	121	6	new	59
2	192.168.1.129	45	192.168.1.140	8080	6	est	3600

Table 1: An example of a firewall state table.

4.3 Private NFV

Our goal is to provide privacy of an outsourced network function ψ given a set of packets $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t$. From an adversarial perspective, the network function ψ can be learned either directly through the description of (m, a) or indirectly by deducing from the outputs $\psi(\mathbf{x}_1), \psi(\mathbf{x}_2), \ldots, \psi(\mathbf{x}_t)$. In order to achieve privacy, we therefore need a scheme that protects both the network function ψ and its output. We call this PNFV (Private NFV). Let $\mathbf{x}$ be a packet as defined before and ψ be a network function such that $\psi(\mathbf{x}) = \mathbf{x}'$.

DEFINITION 1 (PNFV). *A public-key PNFV scheme is a tuple* $(\mathsf{kg}, \mathsf{enc}, \mathsf{dec}, \mathsf{tr}, \mathsf{proc})$ *of probabilistic polynomial time algorithms defined as follows:*

- Key generation: *The algorithm* $s, p \leftarrow \mathsf{kg}(1^k)$ *returns the secret key s and public key p, where k is the security parameter.*
- Packet encryption: *The algorithm* $E(\mathbf{x}) \leftarrow \mathsf{enc}(p, \mathbf{x})$ *takes as input the public key p and the packet $\mathbf{x}$ and outputs the encrypted version $E(\mathbf{x})$. Note that this is element-wise encryption, which results in n ciphertexts.*
- Network function transformation: *The algorithm* $\phi \leftarrow \mathsf{tr}(\psi)$ *takes as input the network function ψ and outputs a* transformed *network function ϕ.*
- Packet processing: *The algorithm* $E(\mathbf{x}') \leftarrow \mathsf{proc}(\phi, E(\mathbf{x}))$ *takes as input the transformed network function ϕ and the encrypted packet $E(\mathbf{x})$ and outputs the encryption of $\mathbf{x}'$.*
- Packet decryption: *The algorithm* $\mathbf{x}' \leftarrow \mathsf{dec}(s, E(\mathbf{x}'))$ *takes as input the secret key s and the encryption of $\mathbf{x}'$ and outputs $\mathbf{x}'$. We may write $D(E(\mathbf{x}))$ to represent* $\mathsf{dec}(s, E(\mathbf{x}))$.

Concisely, we can define the output of PNFV given $\mathbf{x}$ and ψ as PNFV$(\mathbf{x}, \psi)$. Key generation, network function transformation, and packet decryption algorithms are computed by the client MB, while the remaining two algorithms are processed by the cloud MB. We have the following definition for correctness.

DEFINITION 2 (CORRECTNESS). *A* public-key PNFV *scheme is correct if for all* $\mathbf{x} \in \mathbb{Z}_q^n$ *it holds that*

$$\Pr[\mathit{PNFV}(\mathbf{x}, \psi) \neq \psi(\mathbf{x})] \leq \mathsf{negl}(k),$$

where $s, p \leftarrow \mathsf{kg}(1^k)$, negl *is a negligible function and k is the security parameter.*

PNFV Security. As mentioned before, we consider an honest-but-curious adversary, i.e., a passive adversary that correctly computes PNFV but would like to infer ψ. More precisely, we conduct the

[1]There are network functions for which this is not true, e.g., traffic monitoring in which aggregate statistics of packets, such as number of packets received, are maintained.

following experiment involving an adversary $\mathcal{A}$ to model PNFV security. First, $\mathcal{A}$ is given the public key p, the description of algorithms $(\mathsf{kg}, \mathsf{enc}, \mathsf{dec}, \mathsf{tr}, \mathsf{proc})$ and the transformed network function ϕ. While $\mathcal{A}$ is in the *test* state, it can sample any packet $\mathbf{x}$ and obtain its output $E(\mathbf{x}')$ such that $\psi(\mathbf{x}) = \mathbf{x}'$ through the packet processing algorithm. Finally, in the *guess* state $\mathcal{A}$ outputs its guess of the network function ψ as ψ'. If $\psi' = \psi$, $\mathcal{A}$ wins.

The above experiment abstracts what we call the strong adversary, denoted $\mathcal{A}_{\text{strong}}$, to distinguish it from a weaker adversary, denoted $\mathcal{A}_{\text{weak}}$. The weak adversary differs from the strong one in that it is only given *oracle (black box) access to part* of the packet processing algorithm proc, and is not shown the incoming packet $\mathbf{x}$. Instead a packet is chosen randomly from a publicly known distribution $\mathcal{D}$, whenever $\mathcal{A}_{\text{weak}}$ requests for outputs of the above functions on a fresh input $\mathbf{x}$. In practice, this model is realized by introducing an entry MB, which is assumed to be running within a black box. The entry MB receives the packet and performs part of the packet processing algorithm proc, which is hidden from $\mathcal{A}_{\text{weak}}$.

DEFINITION 3. *A* public-key PNFV *scheme is (τ, ϵ)-private if for any adversary $\mathcal{A}$ that runs in time $\tau = \mathsf{poly}(k)$, it holds that*

$$\Pr[\mathcal{A}^{PNFV} = \psi] \leq \epsilon = \epsilon(k),$$

where $\mathbf{x}' = \psi(\mathbf{x})$, $\mathcal{A}$ can be either $\mathcal{A}_{strong}$ or $\mathcal{A}_{weak}$ and k is the security parameter.

5. PNFV INSTANTIATIONS

We describe solutions for a generic network function ψ, which given a packet $\mathbf{x}$ implements the policy:

$$\texttt{if } x_i == y \texttt{ then } x_j \leftarrow z, \tag{P1}$$

where $i, j \in [n]$. We call this the *equality matching* policy, a special case of the more general *range matching* policy defined as:

$$\texttt{if } x_i \in [a, b] \texttt{ then } x_j \leftarrow z. \tag{P2}$$

We begin by giving a brief revision of the different cryptographic primitives used in our schemes.

5.1 Cryptographic Primitives

The BGN Cryptosystem. The Boneh, Goh and Nissim (BGN) cryptosystem is a partial homomorphic encryption scheme which on top of being additively homomorphic, also allows for *one* multiplication of ciphertexts. We denote the encryption algorithm of BGN by E.

PEKS. Public-key Encryption with Keyword Search (PEKS) [3], produces the searchable encryption $\mathcal{E}$ of a keyword w using public key p as $\mathcal{E}(w)$ and the trapdoor $T(w)$ of w using private key s. It includes the public key algorithm $\mathsf{test}(\mathcal{E}(w), T(w'))$ which outputs 1 if $w' = w$ and 0 otherwise.

Pseudorandom Permutation. We also assume the existence of a secure pseudorandom permutation σ, mapping from $[n]$ to itself. In practice, this can be implemented using a block cipher, [10] such as AES.

5.2 Privacy against the Strong Adversary

It is not hard to see that using fully homomorphic encryption (FHE), private NFV is realisable. However, even though much progress in improving the efficiency of FHE has been made [12], we do not have a truly efficient FHE instantiation providing acceptable performance in the context of NFV. However, efficient partial homomorphic encryption schemes, like BGN [4], could be used, as discussed next. We start with the function ψ described by policy P1, and describe the matching function as:

$$m(\mathbf{x}) = 1 - \langle \mathbf{x}, \mathbf{e}_i \rangle + y.$$

If we denote $m(\mathbf{x}) = c$, then $c = 1$ if $y = x_i$, and $c \neq 1$ if $x_i \neq y$. In other words, the matching function will output 1 only if the packet matches the policy and give a value other than 1 otherwise. The action function is:

$$a(\mathbf{x}) = \mathbf{x} - \mathbf{x} \circ \mathbf{e}_j + z\mathbf{e}_j,$$

which replaces x_j with z, as required. Now, we need an encryption algorithm E that can homomorphically compute both m and a. More specifically, E should give the encryption of m as:

$$E(m(\mathbf{x})) = E(1) - \langle E(\mathbf{x}), E(\mathbf{e}_i) \rangle + E(y) \tag{2}$$

and the encryption of a as:

$$E(a(\mathbf{x})) = E(\mathbf{x}) - E(\mathbf{x}) \circ E(\mathbf{e}_j) + E(z\mathbf{e}_j). \tag{3}$$

The BGN cryptosystem allows to homomorphically compute one multiplication and any number of additions. Therefore, we can use it to construct a PNFV scheme secure against the strong adversary. The scheme is presented in Figure 2. We omit the description of the key generation algorithm (which should be obvious from the underlying cryptosystem), and further include the packet encryption routine within the packet processing algorithm.

Range matching. In the full version of the paper, we show that we can also use the BGN cryptosystem for range matching, i.e., the network function ψ defined by policy P2.

Correctness. This follows from the fact that BGN can successfully decrypt homomorphic encryptions of unlimited additions and one multiplication (per ciphertext).

Privacy. Intuitively, the scheme is private as the adversary only sees randomized encryptions of matching and action functions and, therefore, cannot infer whether the matching function resulted in a 1 or some other value. More formally, we prove, in the full version of the paper that if BGN is semantically secure, our PNFV scheme is private against $\mathcal{A}_{\text{strong}}$.

Discussion. Ideally, the client MB would receive the encryption of the whole network function, i.e., $E(\psi(\mathbf{x}))$ and simply decrypt it to get the final packet. In our protocol, it actually has to perform two decryption operations instead of one (one to check the output of the matching function and another to decrypt the result), and, for each packet, three encryptions need to be sent. This is due to the fact that the output of the matching function is a variable (i.e., not a constant value) when there is no match. This also means that we cannot perform iterations of N network functions.

5.3 Privacy against the Weak Adversary

We now present a more efficient solution that is secure against the weak adversary, based on public-key encryption with keyword search (PEKS) [3], a probabilistic encryption scheme (E, D) and a pseudorandom permutation σ. Figure 3 presents our solution, in the context of policy P1. Observe that I denotes the n-element index vector whose i-th element is i itself, and $\mathbf{x}||I$ the n-element vector whose i-th element is $x_i||i$. In this model, the weak adversary $\mathcal{A}_{\text{weak}}$ does not have access to the entry MB packet processing. Thus, we have a somewhat weaker notion of security in this scheme compared to the BGN-based scheme which is secure against the strong adversary. The advantage over the BGN-based scheme is that we only send one encrypted packet, and the client MB only needs to decrypt the packet. Note that steps 1, 2 and 3 in Figure 3

Network function transformation. The client MB computes the tuple $(E(1), E(\mathbf{e}_i), E(y), E(\mathbf{e}_j), E(z\mathbf{e}_j))$ and sends it to the cloud MB.

Packet processing. Upon receiving a packet $\mathbf{x}$ the cloud MB:

1. Encrypts the packet as $E(\mathbf{x})$.
2. Computes $E(a(\mathbf{x}))$ according to Eq. 3 as $E(a(\mathbf{x})) = E(\mathbf{x}) - E(\mathbf{x}) \circ E(\mathbf{e}_j) + E(z\mathbf{e}_j)$ and $E(m(\mathbf{x})) = E(c)$ according to Eq. 2 as $E(c) = E(1) - \langle E(\mathbf{x}), E(\mathbf{e}_i)\rangle + E(y)$
3. Sends $E(\mathbf{x})$, $E(a(\mathbf{x}))$ and $E(c)$ to the client MB.

Packet decryption. Upon receiving $E(\mathbf{x})$, $E(a(\mathbf{x}))$ and $E(c)$ the client MB:

1. Decrypts $E(c)$ to obtain c.
2. If $c = 1$, decrypts $E(a(\mathbf{x}))$ to obtain the transformed packet.
3. Else if $c \neq 1$, decrypts $E(\mathbf{x})$ to obtain the unchanged packet.

Figure 2: PNFV scheme based on the BGN cryptosystem.

Network function transformation. Using PEKS, the client MB computes the trapdoors $T(y||i)$ and $T(j)$. Using E, the client MB creates the encryption $E(z||j)$. The client MB sends $T(y||i)$, $T(j)$ and $E(z||j)$ to the cloud MB.

Packet processing. This is divided into entry MB and cloud MB.

Entry MB: Upon receiving a packet $\mathbf{x}$:

1. Encrypts $\mathbf{x}||I$ using E and shuffles the result as $\sigma(E(\mathbf{x}||I))$.
2. Encrypts $\mathbf{x}||I$ using PEKS and shuffles the result as $\sigma(\mathcal{E}(\mathbf{x}||I))$.
3. Encrypts I using PEKS and shuffles it as $\sigma(\mathcal{E}(I))$.
4. Deletes the original packet $\mathbf{x}$.

Cloud MB: Upon receiving $\sigma(E(\mathbf{x}||I))$, $\sigma(\mathcal{E}(\mathbf{x}||I))$ and $\sigma(\mathcal{E}(I))$:

1. Checks if there exists an $l \in [n]$ such that $\texttt{test}(\mathcal{E}(x_l||l), T(y||i)) = 1$.
    - 1.1. If yes, finds an $l' \in [n]$ such that $\texttt{test}(\mathcal{E}(l'), T(j)) = 1$ (which should exist).
    - 1.2. Replaces $E(x_{l'}||l')$ with $E(z||j)$ in $\sigma(E(\mathbf{x}||I))$ and sends it to the client MB.
2. Else, sends $\sigma(E(\mathbf{x}||I))$ to the client MB.

Packet decryption. The client MB upon receiving $\sigma(E(\mathbf{x}||I))$, decrypts to obtain $\sigma(\mathbf{x}||I)$ and then reconstructs $\mathbf{x}$ according to I.

Figure 3: Scheme based on PEKS, private against the weak adversary.

performed by the entry MB use the same permutation which is set once per new packet arrival.

Correctness. The client MB decrypts $E(\mathbf{x}'||I)$, permuted by σ, to obtain $\mathbf{x}'||I$ and reconstructs $\mathbf{x}'$ according to I. If the original packet matches policy P1, then $x'_j = z$. Likewise, if the packet does not match the policy, the decrypted packet $\mathbf{x}'$ is the original packet $\mathbf{x}$. Therefore, our PNFV scheme is correct.

Privacy. Intuitively, since $\mathcal{A}_{\text{weak}}$ does not know which packet index yields a match and which index the action applies to (due to random shuffle by σ), and since the matching value y and the action value z are encrypted, it cannot infer the policy. In the full version of the paper we prove this formally.

Discussion. The obvious limitation of this scheme is that it is only private against a weaker adversary. In particular, the cloud MB does not retain the packet $\mathbf{x}$ to match its randomly permuted encryptions, neither does it attempt to find j in $T(j)$ by checking all possible encryptions under $\mathcal{E}$ of all possible elements in $[n]$. If the cloud MB tries to do either of these (unwarranted) actions, it will at best learn the index j (and not index i, y or z). To find (i, y), the cloud MB needs to do a brute force search whose complexity is $O(2^{qn})$. On the other hand, we only need to send a number of encryptions per packet independent of the number of network functions N. A further drawback of the scheme is that it is not applicable to the range policy, i.e., policy P2.

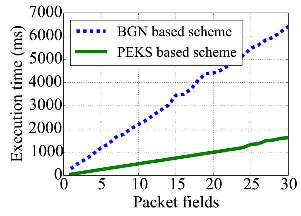
5.4 Handling State Tables

The private state table solution is built from the PEKS based PNFV scheme discussed above. Note that homomorphic encryption based solutions are not applicable to state tables, as the cloud MB should discontinue processing once a match is found in the state table. If processing needs to be continued for the packet, and the current state table only maintains statistics (such as counters), then this can be implemented in the same way as a normal network function. In case no entry in the state table is found, the cloud MB continues processing the static network policies via the underlying PNFV scheme. Due to space constraints, we do not detail the solution here. The reader can refer to the full version of the paper for details. The privacy argument of the proposed state table solution is similar to the one for the PEKS based PNFV scheme.

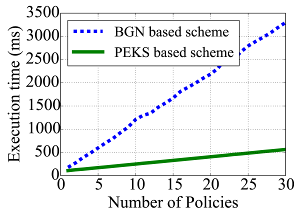
6. POC IMPLEMENTATION

In the following, we provide a proof-of-concept (PoC) of the feasibility of our PNFV schemes. Due to space constraints, we only show aggregate performance comparison of the two proposed PNFV schemes. More detailed analysis appears in the full version of the paper. We implemented PEKS- and BGN-based schemes in C using the `RELIC` cryptographic library [1], and used policy P1 to represent a generic network function. Figures 4(a) and 4(b) plot the aggregate execution times for the two schemes (adding up times for packet encryption, processing and decryption) against, respectively, increasing number of fields (with 10 policies) and increasing number of policies (with 5 fields used for private processing). The PEKS based scheme clearly outperforms the BGN-based scheme. For instance, for a network function with 10 policies, private processing of 5 packet fields takes 250 ms in the PEKS based scheme and 1,208 ms in the BGN-based scheme.

Translated into packets per second (pps), the above two numbers translate to a modest 4 pps and 0.82 pps, respectively. However, we remark that our implementation merely stands as a proof-of-concept. In particular, we did not go for further efficiency, e.g.,



(a) With 10 policies



(b) With 5 packet fields

Figure 4: Aggregate execution times (packet encryption, processing and decryption) for the two schemes.

using more powerful machines or multi-threading. For instance, the time taken by the entry MB, the cloud MB and the client MB for a packet with a single encrypted field and a network function with a single policy was 13.32 ms, 5.41 ms and 2.69 ms, respectively, yielding a total of 21.42 ms. Using multi-threading we can process a larger number of packet fields (in the case of the entry and client MB) and the policies (in the case of the cloud MB) in parallel, thus significantly increasing the number of packets processed per second. With a slightly more powerful machine that can process say 50 threads concurrently, we can achieve a rate of more than 2,300 pps (using 21.42 ms as the baseline).

Nevertheless, even without optimizations, our performance is comparable to that of the schemes proposed in [17]. The three different *modes* in [17] yield 60 ms, 1,000 ms and 3,000 ms for private processing of a 5-tuple with 10 firewall rules. The Bloom filter based scheme from [8] does much better, achieving 0.1 ms for a 10 rule firewall.[2] However, as described in Section 2, both these works are narrower in scope and their security, at best, is questionable.

7. CONCLUSION

This paper addressed the problem of private processing of outsourced network functions, where network function policies need to be kept private from the cloud, other tenants and third parties. We presented a cryptographic treatment of the problem, introducing security definitions as well as an abstract model of generic network functions, and proposed a few instantiations using homomorphic encryption and public-key encryption with keyword search. The performance of our proposed solutions is reasonable considering that we rely on public key operations and provide provable security in the presence of an honest-but-curious cloud, while guaranteeing that third party users, who are sending/receiving traffic, are oblivious to network function outsourcing.

Acknowledgements. Luca Melis and Emiliano De Cristofaro are supported by a Xerox's University Affairs Committee award on "Secure Collaborative Analytics." and "H2020-MSCA-ITN-2015" Project Privacy&Us (ref. 675730).

8. REFERENCES

[1] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7), 1970.

[2]These approximate numbers are deduced from ACL index 16 from Figure 8 in [8].

[3] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Eurocrypt '04*, 2004.

[4] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *TCC '05*, 2005.

[5] J. H. Cheon, K. Han, C. Lee, H. Ryu, and D. Stehle. Cryptanalysis of the Multilinear Map over the Integers. In *Eurocrypt '15*, 2015.

[6] J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical Multilinear Maps over the Integers. In *CRYPTO '13*, 2013.

[7] M. G. Gouda and A. X. Liu. Structured Firewall Design. *Comput. Netw.*, 2007.

[8] A. R. Khakpour and A. X. Liu. First Step Toward Cloud-Based Firewalling. In *SRDS '12*, 2012.

[9] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. seL4: Formal Verification of an OS Kernel. In *SOSP '09*, 2009.

[10] M. Luby and C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM J. Comput.*, 1988.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[12] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *CCSW '11*, 2011.

[13] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *CCS '09*, 2009.

[14] K. Searl. Top 26 Companies in the Global NFV Market. http://www.technavio.com/blog/top-26-companies-in-the-global-nfv-market, 2014.

[15] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *SIGCOMM '12*, 2012.

[16] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. In *SIGCOMM '15*, 2015.

[17] J. Shi, Y. Zhang, and S. Zhong. Privacy-preserving Network Functionality Outsourcing. http://arxiv.org/abs/1502.00389, 2015.

[18] Z. Wang and X. Jiang. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. In *IEEE S&P '10*, 2010.

[19] F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-tenant Cloud with Nested Virtualization. In *SOSP '11*, 2011.

[20] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM Side Channels and Their Use to Extract Private Keys. In *CCS*, 2012.