Extreme SDN Framework for IoT and Mobile Applications Flexible Privacy at the Edge

Mostafa Uddin
Nokia Bell Labs
Holmdel, NJ, USA
mostafa.uddin@nokia-bell-labs.com

Tamer Nadeem, Santosh Nukavarapu
Virginia Commonwealth University
Richmond, VA, USA
{tnadeem,nukavarapuskk}@vcu.edu

***Abstract*—With the current significant penetration of mobile devices (i.e. smartphones and tablets) and the tremendous increase in the number of the corresponding mobile applications, they have become an indispensable part of our lives. Nowadays, there is a significant growth in the number of sensitive applications such as personal health applications, personal financial applications, home monitoring applications, etc. In addition, with the significant growth of Internet-of-Things (IoT) devices, smartphones and the corresponding applications are widely considered as the Internet gateways for these devices. Mobile devices mostly use wireless LANs (WLANs) (i.e., WiFi networks) as the prominent network interface to the Internet. However, due to the broadcast nature of WiFi links, wireless traffics are exposed to any eavesdropping adversary within the WLAN. Despite WiFi encryption, studies show that application usage information could be inferred from the encrypted wireless traffic. The leakage of this sensitive information is very serious issue that will significantly impact users' privacy and security. In addressing this privacy concern, we design and develop a lightweight programmable privacy framework, called *PrivacyGuard*. *PrivacyGuard* is inspired by the vision of pushing the Software Defined Network (SDN)-like paradigm all the way to wireless network edge, is designed to support of adopting privacy preserving policies to protect the wireless communication of the sensitive applications. In this paper, we demonstrate and evaluate a prototype of *PrivacyGuard* framework on Android devices showing the flexibility and efficiency of the framework.**

***Index Terms*—mobile device, IoT, software defined network, privacy, edge, traffic shaping**

I. Introduction

We are approaching a fundamental shift in the computational era as the number of smart device users (e.g., smartphone and tablet users) is expected to exceed 6 billion (more than 50% of the global population) by 2020 [19]. This is supplemented by IoT devices which are transforming the way of living for smart environment users by enabling them with various services. Ericsson predicts the number of connected IoT devices to reach 18 billion by 2022 [13]. Smartphones (and mobile devices in general) are widely considered as Internet gateways to IoT devices through using the corresponding IoT applications on smartphones. These IoT devices, as well as smartphones, sense and monitor several sensitive user information including sleeping patterns [1], exercise routines [12], child behaviors [15], medical information [23], sexual activity [43], and home appliances status [22].

This material is based upon work partially supported by the US National Science Foundation under Grants No. CNS-1764185 & CNS-1836870.

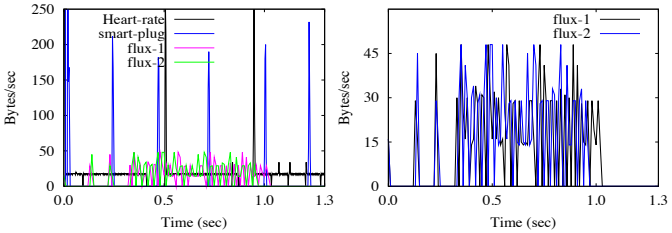

Fig. 1. a) Traffic patterns of four different IoT devices: Heart-rate monitoring, Elegato-plug, and two Flux-lightbulb devices operating at different times. b) Zooming into the traffic of the two Flux-lightbulb devices shows high similarity.

Mobile devices mostly use wireless LANs (WLANs) (i.e., WiFi networks) as the prominent network interface to the Internet. A recent study found that WiFi networks are expected to carry almost 60% of smartphones and tablets data traffic by 2019 [20]. The broadcasting nature of WiFi exposes IoT and smart devices traffic to eavesdropping by adversaries and, consequently, various attacks. Several previous studies already showed that, even with WiFi encryption (e.g., WPA2), statistical analysis of side-channel information of WiFi traffic such as packet sizes, data rate, ratio of incoming to outgoing packets, inter-packet time, etc. could infer several user-related information such as user identity [29], user's online activities [46], and identification of applications used by the user [40]. The traffic analysis of major commercial IoT devices like Nest Camera, Amazon Echo (Personal Voice Assistant), Belkin Smart Plug is found vulnerable to activity inference such as user presence, device interaction and appliance usage [7]. As an example, Figure 1a captures the WiFi traffic of four mobile applications corresponding to four IoT devices, while Figure 1b zooms on the traffic of the same-type devices. As shown, while the traffic patterns of the two Flux bulbs look very similar, different IoT devices have different traffic patterns that could be uniquely distinguished and easily correlated to its corresponding IoT device and even to a specific activity/status of the IoT device.

In addressing the above security concern of side-channel analysis, several techniques like perfect secrecy theory [32], mix based systems and anonymous systems [16] are proposed to hide traffic signatures and characteristics in order to make them less identifiable. The most popular techniques are based

on traffic shaping like traffic padding [8], [11], [26], [30], [36], faking superfluous packet and chopping packets into fixed size segments [8], and traffic morphing [8], [42]. The performance of these traffic shaping techniques in terms of efficiency and overhead varies based on their configuration parameters. For example, the efficiency of the traffic padding approach in obfuscating the traffic signature, as shown later in experiments, increases with the percentage of the padded traffic packets. However, this higher efficiency comes with higher overhead in terms of network bandwidth and power consumption since more bits are transmitted. For example, while a padding configuration providing high obfuscation efficiency with high overhead is suitable for networks with low traffic loads, it significantly degrades the performance of a highly saturated networks and, consequently, switching to a lower overhead configuration would be more desirable.

Motivated by the above observations, in this paper, we design, develop and evaluate a flexible and programmable privacy preserving framework, *PrivacyGuard* that is inspired by our vision of pushing the Software Defined Network (SDN)-like paradigm all the way to the wireless network edge [24], [39]. This vision is realized by extending and deploying SDN components (e.g., Open vSwitch [4]) on mobile devices and WiFi Access Points (in home or campus environments) or Proxy server (in open and public WiFi Hotspots). We refer to these SDN components on mobile end devices and access points as *extreme SDN*[1] to differentiate them from the traditional SDN used in the network core. In our approach, the proposed extreme SDN works independently and without any collaboration or support from the network core SDN. The basic idea of *PrivacyGuard* framework is to create one or more vertical network slicing between mobile devices and WiFi APs/Proxy server corresponding to one or more rules (policies), which applies the optimum per-flow/per-application privacy preserving scheme based on application requirements, user objectives, device characteristics, and network conditions in real-time fashion.

We summarize the contributions of this paper as follows:

- We design and develop *PrivacyGuard*, a privacy preserving framework to obfuscate the activities of sensitive IoT and mobile applications from adversarial attack over encrypted (or unencrypted) WiFi network. The framework supports important features such as flexibility, transparency, real-time adaptability, and context-awareness.
- We realize and implement a prototype of *PrivacyGuard* on Android mobile devices that enable us to apply per-application *traffic shaping* and IPsec tunneling schemes.
- Finally, we evaluate and analyze the performance of *PrivacyGuard* using different applications on mobile devices to evaluate its efficiency, energy consumption, network overhead, and CPU usage.

[1]Recently, "extreme" has been adopted in networking and mobile computing community to refer to network edge devices such as wireless access points or user devices [9] in which we adopt this definition in this paper.

II. Threat Model and Applications/Flows Identification

In this paper, we consider the privacy threat model where the adversary passively eavesdrops the encrypted/unencrypted wireless traffic between the mobile devices and the WiFi Access Points (APs). In IoT context, the mobile devices act as gateways for the IoT devices that utilizes non- WiFi technologies, such as Bluetooth Low Energy (BLE) or Z-wave, to communicate with their corresponding applications on the mobile devices. By eavesdropping, the adversary can extract and analyze the WiFi side-channel information such as packet sizes and inter-arrival packet times to identify the running mobile applications and the corresponding usage activities [40].

In this work, we refer to these applications as the *sensitive applications* in which the goal of the adversary is to identify the usage of these sensitive applications (e.g., mHealth apps, IoT apps, etc.) to infer user's information. The usage activities of different mobile applications could be utilized for inferring users' identities [35], [44], knowing their health conditions, tracking their daily activities, etc. For example, by eavesdropping the WiFi traffic of house members, the adversary will be able to correlate the detected application activities to different members of the house (e.g., husband, wife, kid), monitor their different activities (e.g., using coffee machine, watching TV, etc.), and track the occupancy periods of each member. This example demonstrates that tracking mobile application usages of individuals is not just a privacy issue, but also could be a severe threat to the individual safety.

Previous studies show that any identification scheme based on the side-channel information (e.g., packet sizes, number of packets, inter-arrival packet times) is more reliable and accurate in identifying the running applications than any of the traffic packet values (e.g., IP Src address, port number, protocol type) [37], [38], [40]. Therefore, in our attack model (i.e., side-channel attack), we assume that the adversary doesn't need to access any of the packet fields and performs the attack solely based on the traffic side-information, which applies to both the encrypted and the unencrypted WiFi traffic.

While the side-channel attack seems to be challenging in the presence of the overlapped encrypted traffic from multiple applications running concurrently, previous works proposed various approaches to efficiently classify and separate the encrypted overlapping traffic into individual applications and then apply the side-channel attack on each class to fingerprint and identify individual applications [37], [38]. Moreover, recent works show that IoT devices and corresponding applications generate short bursts of traffic frequently [6], [34], which make it vulnerable to side-channel attack since it is not hard to separate these bursts even when multiple IoT devices/applications are running concurrently.

Different activities of an application will generate different flows with different traffic characteristics that would require different obfuscation handling. For example, a *Nest Camera* will have two different flows corresponding to the live camera

feed and the motion detection activities respectively [7]. Therefore, to launch an efficient side-channel attack, the adversary needs to build a signature identification model for each mobile application and its different corresponding flows. Typically, the signatures of the different flows of an application will be built offline by extracting the side-channel information under both encrypted and unencrypted WiFi traffic. Zhang et al. [46] show how to leverage various machine learning (ML) techniques, such as Support Vector Machine (SVM), Neural Network, and Hidden Markov Model, to build application's signatures by using the statistical characteristics of the side-channel information. In this paper, we assume no bound on the adversary's ability to build such ML identification model off-line.

In this paper, based on our previous work [39], we offline build a C5.0 decision tree and a k-NN (k=3) classifiers based on the statistical features of the side-channel information for identifying the active applications and their corresponding flows in real-time. Two different sets of features are used for these two classifiers. The first set of features is based on the sizes of the first N packets (i.e., N=7) and is used by the C5.0 decision tree classifier to identify and recognize the different applications. On the other hand, the K-NN classifier is used to classify the WiFi traffic into its corresponding applications/flows by using the second set of features that includes means, medians, minimums, maximums, and variances of both packet sizes and inter-packet times (IPT) calculated every two seconds window, as well as, the Discrete Wavelet Transform coefficients calculated over the packet size and IPT sequences. Both classifiers show more than 90% accuracy in identifying the applications and their corresponding flows [39]. In this work, we assume the adversary uses these classifiers to identify the active applications/flows. In addition, we use these classifiers in our experiments to evaluate the efficiency of the used obfuscation schemes.

III. PrivacyGuard Framework

A. PrivacyGuard Objectives

In considering the above threat model, we design *PrivacyGuard* with the following objectives.

Flexible per-application per-flow privacy preserving schemes: Given different applications have different sensitivities and requirements, *PrivacyGuard* should have the flexibility of applying different privacy preserving schemes to different applications. Moreover, different applications and even different flows of the same application would have different traffic characteristics and, consequently, would require different schemes to obfuscate the application. For example, Dropbox generates two flows for uploading/downloading a file, wherein one direction data packets are at its maximum possible size, while the other direction contains just identical TCP ACK packets. Therefore, the TCP ACK flow should use a scheme that pads these TCP frames to look like the data packets flow, which might not need any padding scheme. Therefore, *PrivacyGuard* should be designed, through introducing new action commands in Open vSwitch (OVS), as we will describe later, to support applying per-application per-flow configurable schemes.

Programmable privacy preserving policies: Given the performance of privacy preserving schemes (e.g., traffic shaping schemes) depends on their configuration, *PrivacyGuard* should support programmable APIs to define and configure different schemes dynamically. In addition, it also should support to define set of rules (policies) that map individual applications/flows to their optimum schemes based on on the application, user, device, and network conditions and characteristics in real-time fashion.

Context aware privacy preserving policies: Different application requirements, user objectives, device characteristics, and network conditions, which we refer to them as *contexts*, require different performance levels of the applied privacy preserving schemes. Therefore, *PrivacyGuard* should support to integrate the context into the defined policies in order to select in real-time the optimum scheme for individual applications/flows that adapt to the given context.

Policies are transparent to applications: Unlike previous systems that often require redesigning both the client side and the server side of the application *PrivacyGuard* should seamlessly support any application without requiring any modification on either client or server-side of the application.

B. Privacy Preserving Schemes

PrivacyGuard adopts and utilizes a number of the popular *traffic shaping* schemes, due to its popularity and simplicity, to obfuscate applications traffic signatures from any adversary.

One of the adopted traffic shaping schemes is ***packet-padding*** that applies a padding bytes to a percentage p of the application traffic packets. Although the selection of these padded packets could follow different distributions, in this paper we select these packets uniformly in which each packet of the traffic will be padded with a certain probability p. The size of the padding bytes will follow a random distribution such as Poisson distribution or Gaussian distribution. Note that the actual padding size will depend on the configuration parameters of the distribution. For example, if we set the standard deviation σ and the mean μ parameters of the Gaussian distribution to a very small value and a large value respectively, the outcome padding sizes will follow a large uniform padding distribution. To guarantee that none of the padded packets exceed the maximum transmission unit (MTU), we truncate the outcome padding sizes as needed.

Another popular traffic shaping technique is ***packet-delaying*** that shapes the inter-packet transmission times (IPTs). In doing this, we increase the queueing time of each packet, before sending it down to the WiFi driver, with a random delay selected from a uniform distribution. Similar to *packet-padding* scheme, *packet-delaying* scheme could follow any other distribution.

Recent studies show that over half of the connections made by mobile applications are insecure since they don't use any of the network or application level encryption [14]. Consequently, open or unencrypted WiFi hotspot connections expose several

data packet fields such as "IP" header in which the adversary could easily identify the applications/flows from these packets even when we are using any of the privacy preserving schemes. In mitigating this vulnerability, *PrivacyGuard* applies IPSec tunneling scheme between the *PrivacyGuard*'s ends as described in Section III-D, on top of any used traffic shaping scheme, to prevent eavesdropping any of the packet fields. The choice of using tunneling is entirely configurable and would be selectively applied for selected applications based on the network condition (i.e., WiFi is either open or unencrypted).

Although the current implementation of *PrivacyGuard* adopts the traffic shaping and IPSec tunneling techniques discussed above as a proof of concept of privacy preserving schemes, *PrivacyGuard* is a flexible framework that easily could be extended to support several other privacy preserving schemes such as injecting fake superfluous packets, chopping packets into fixed-size segments, traffic morphing, etc. Moreover, *PrivacyGuard* enables, through defining flow policies as described later, to configure the applied privacy preserving to adapt its performance to the current context.

C. Context Information

Different privacy preserving schemes with different configurations would have different performance in terms of obfuscation efficiency, network bandwidth overhead, power consumption, and latency overhead. Therefore, a well-suited scheme that should be applied is the one that its performance suites multiple contexts, which we categorize as follows:

Application Context: Applications and corresponding flows could be categorized into different sensitive classes (e.g., non-sensitive, moderate sensitive, high sensitive). Different privacy schemes should be selected for different sensitive classes. For example, high sensitive applications that reveal, for example, sexual and medical activities should use high efficient obfuscation scheme despite the accompanied high overhead. On the other hand, non-sensitive or moderate sensitive applications should not apply any scheme or use low efficient scheme in order to reduce or eliminate any overhead. Similarly, applications could be classified into real-time and non-real-time applications in which real-time application should not use any significant delay-based privacy scheme.

User Context: User location and time are examples of user context in which different policies could be defined based on user context. For instance, a user's home could be classified as a secure environment in which a low efficient privacy scheme will be applied to any high sensitive application. On the other hand, a high efficient scheme will be used on both the moderate and the high sensitive applications when the user is sitting in his regular coffee-shop, which is defined as an insecure environment. Similarly, a user could define sensitive and non-sensitive time periods in order to hide certain time-based activity. For example, an early morning time could reveal sensitive routines such as medical activity (e.g., daily morning drug dosage) while a nighttime could reveal sleeping or sexual routines. Hence, a high efficient scheme will be selected for these morning and evening sensitive time periods.

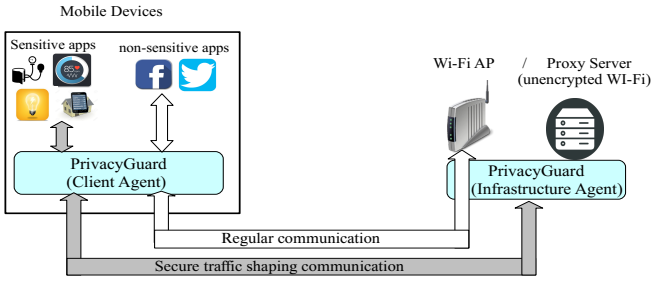

Fig. 2. Use-case scenarios of *PrivacyGuard*.

Device Context: Examples of device context are battery level and computing power. For example, while the battery level of a device is high, it would be more suitable to apply a high efficient scheme on both its moderate and high sensitive applications. However, when the battery level of a device drops below a certain threshold, it should switch to a low power consumption and less efficient scheme for its high sensitive applications only in order to preserve its battery level. Similarly, a powerful mobile device would choose a compute-intensive policy to guarantee strong privacy for all its applications.

Network Context: Network conditions will significantly impact the scheme selection. A public unencrypted WiFi hotspot such as a coffee shop or a train station would require to use IPSec scheme on top of any other scheme on all moderate and high sensitive applications. Similarly, if a network experiences a high-load traffic, privacy preserving schemes with low network bandwidth overhead would be preferable in order to avoid any degradation in network performance.

D. PrivacyGuard Basic Operation

Figure 2 shows a typical use case of *PrivacyGuard*, which consists of two edge agents i) Client Agent, and ii) Infrastructure Agent that run at two different ends as shown. The client agent always runs on the user's mobile devices with system-level permission. However, the place of running the infrastructure agent depends on the configuration ability of the WiFi APs. In environments, where it is a common practice to manage and configure the WiFi APs (i.e., home, campus, office, etc.), infrastructure agent runs on the APs. On the other hand, where it is less common to manage and configure the WiFi APs (i.e., public hotspots, coffee shop, airport, etc.), infrastructure agent runs as a proxy server in the cloud. Note that running these agents on edge devices is similar to the Bring-Your-Own-Device (BYOD) model that is widely widely accepted by the community. Moreover, the community has accepted several works recently to adopt SDN on WiFi APs [28] and mobile devices [10], [24].

In *PrivacyGuard*, both of these agents agree on the *traffic shaping* policies per-application per-flow as well as the symmetric keys for IPsec tunneling between the two agents. While the *client agent* applies *traffic shaping* policies on the uplink network flows generated from the mobile devices, the *infrastructure agent* applies *traffic shaping* policies on the downlink network flows to the mobile devices. Note

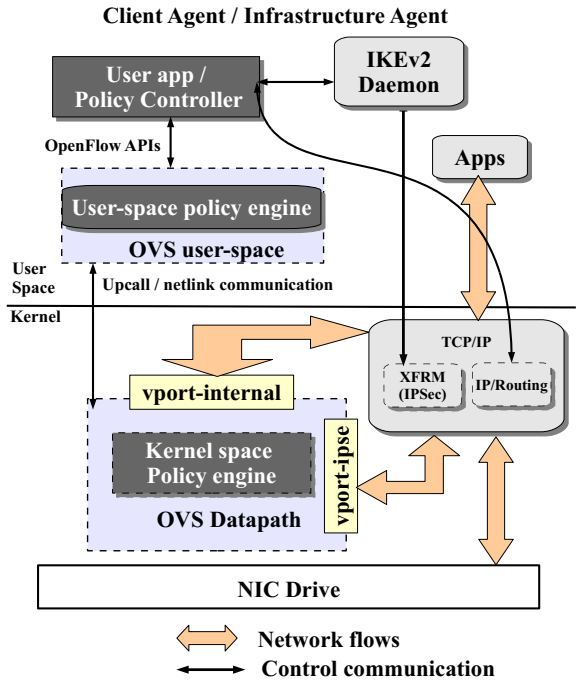

Fig. 3. Architecture overview of PrivacyGuard for mobile devices.

that, *PrivacyGuard* only applies policies on the selected set of sensitive mobile applications provided by the user. In *PrivacyGuard*, both agents utilize Open vSwitch (OVS) to set and enforce *traffic shaping* policies correctly on the network flows from/to sensitive mobile applications as we describe in the next section. It is important to highlight that *PrivacyGuard* is entirely transparent to applications and networks in which it can be seamlessly deployed into any network configuration without requiring any support from application providers.

IV. *PrivacyGuard* System Architecture

Figure 3 shows the overall architecture of *PrivacyGuard* for its client and infrastructure agents. Both agents have some common modules in design such as user policy engine and kernel-space policy engine modules as well as other modules that are specific to each agent type. In the following subsections we discuss each of these modules.

A. User-app Module

This module is a part of the client agent only that supports several essential functions. One of its main functions is to provide a user interface in which users could categorize the installed applications into different sensitive classes. In addition, users utilize this interface to define the privacy preserving schemes for individual applications/flows or applications of similar sensitivity under different contexts.

Another function of this module is to track any new flow and map it to its corresponding application. A 'flow' is defined as the packets with the same 5-tuple fields in the IP header; *Src IP*, *Src Port*, *Dst IP*, *Dst Port*, and *Protocol type*, which we refer to as 5-tuple flow-ID. Since this module runs on the end device, it builds the flow-to-application mappings by monitoring the *netstat* logs [2], which provides the mapping between the active network sockets (TCP, UDP, SSL-encrypted) to their corresponding applications. Once the module detect a new flow, it passes its mapping information as well as any collected side-channel statistics of the flow to the *policy controller* module (discussed next).

This module is also responsible to interact with the IPSec tunneling module (discussed later) for configuring the tunneling settings when the IPSec policy is applied to establish an IPSec tunnel with the infrastructure agent. Finally, once an active application terminates, this module is responsible to release all the allocated resources in this application in the client agent, to communicate this to the infrastructure agent to release the corresponding resources too.

B. Policy Controller Module

The module acts on both the client and the infrastructure agents as the software defined network local controller. On the client agent, this module uses the user-defined privacy schemes and the flow-to-application mapping information (passed from the *user-app* module) for creating and maintaining the *flow-policy* table entries containing the defined privacy preserving policies, which is utilized by the *Policy Engine* modules. Once a new entry is created, this module on the client agent shares this entry with the corresponding peer module on the infrastructure agent that in turn stores and maintains these entries in its own copy of the *flow-policy* table. In addition, this module on the client agent utilizes the device sensors (e.g., GPS, battery) and network interface information (e.g., RSSI, throughput) to periodically estimate the current contexts of the user (e.g., location), the device (e.g., battery level), and the network (e.g., load). Similarly, these contexts are periodically shared with the peer module on the infrastructure agent. In our current implementation, the contexts are estimated and shared every five minutes.

Each entry of the *flow-policy* table is a single policy that consists of the application/flow identification data (i.e., 5-tuple flow -ID), the different contexts (e.g., user location, time, battery level, network load etc.), the applied privacy preserving scheme (e.g., *packet-padding*, *packet-delaying*, IPSec tunneling, etc.), and the corresponding configuration parameters (e.g., padding sampling rate, padding size distribution etc.). Figure 4 shows examples of the table policies entries. The details of how to collect the user defined privacy schemes in the *user-app* module and how to create the entries of the *flow-policy* table in this module are out of the scope of this paper and part of our future work as discussed in Section VII.

This module uses the OpenFlow protocol [27] to create and modify (i.e., program) the policies in the *flow-policy* table maintained in Open vSwithch (OVS) of the *User-space Policy Engine* module. To support this programming capability in *PrivacyGuard*, we extend the OpenFlow protocol and the OVS user-space module (discussed next) and OVS kernel module (i.e, `OVS datapath`) to create and modify the policies in order to accommodate new applications/flows. Runtime creation and

```
Policy #1
  ID: srcIP='A', srcPort='i', dstIP='B', dstPort='j'
  CONTEXT: Location='Home' AND Time=[9PM-12AM, 6AM-9AM]
  ACTION: Padding='Normal:μ=1500,σ=10, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #2
  ID: srcIP='A', srcPort='k', dstIP='B', dstPort='l'
  CONTEXT: Location='Home'
  ACTION: Padding='Normal:μ=400,σ=100, p=0.6'
Policy #3
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=High AND Location=HotSpot
  ACTION: Padding='Normal:μ=1500,σ=10, p=1.0'
          Delay='Uniform:min=0,max=20ms', IPSec
Policy #4
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=High OR WiFi Load=Low
  ACTION: Padding='Normal:μ=1500,σ=10, p=1.0'
          Delay='Uniform:min=0,max=20ms'
Policy #5
  ID: srcIP='A', srcPort='m', dstIP='D', dstPort='n'
  CONTEXT: Battery=Low OR WiFi Load=High
  ACTION: Padding='Normal:μ=1500,σ=10, p=0.6'
          Delay='Uniform:min=0,max=20ms'
```

Fig. 4. Example of a *flow-policy* table for the *traffic shaping* schemes.

modification of these policies allow *PrivacyGuard* to have flexible and programmable privacy preserving policies.

C. User-space Policy Engine Module

In both *client* and *infrastructure* agents, *user-space policy engine* module resides in the OVS user-space (i.e., `vswitchd`). The responsibility of the this module is to maintain and utilize the entries of *flow-policy* table. Furthermore, this module extends OpenFlow APIs in OVS in order to be utilized by the *policy controller* module to setup the table entries (i.e., policies).

The module also runs a user-space thread to process the "upcall" actions from the OVS kernel-space (i.e., `datapath`) in the *Kernal-space Policy Engine* module. Once receiving an "upcall" action for a packet of a new flow, the thread extracts the 5-tuple flow-ID from the packet header, gets the current contexts from the *Policy Controller* module, and then searches the *flow-policy* table entries to find the first best matching policy entry. After finding the matching entry, the thread passes the corresponding privacy preserving scheme and its configuration parameters to the *kernel-space policy engine* module that acts accordingly by applying the selected policy on the rest of the flow packets. Note that, this module keeps track of all the upcall packets and check periodically for any context change. Once a context change is detected, the module repeats the searching process for all the previous upcall packets given the new contexts.

Using the table example shown in Figure 4, *user-space policy engine* module first examines the egress packet header to match the entries. If both the application/flow identifier from the packet header (i.e., 5-tuple) and the current context values matches *ID* and *CONTEXT* fields of a specific policy entry, the traffic shaping scheme with the corresponding parameters in the *ACTION* field of that policy will be applied on this application/flow packets. For example, *Policy #3* in the table states that if the application/flow with 5-tuple identifier is *(A, m, D, n, *)*, and both the user location is *HotSpot* and the device battery level is *High*, then the actions to apply are: i) use *Gaussian* distribution function with parameters μ, σ, and p to generate the size of the padding bytes, ii) use *Uniform* distribution function with range parameters min and max to extend the IPTs with random delays, and iii) use IPSec. Due to the space limitation, we omit the details of the policy language.

D. kernel-space Policy Engine Module

In both *client* and *infrastructure* agents, each flow packet passes through the OVS kernel-space (i.e., `OVS datapath`) of this module that applies the corresponding flow actions on the packet. Initially, once the datapath detects a packet of a new flow, it sends the packet to the *user-space policy engine* module through an *upcall* operation that in turn uses the *flow-policy* table to select the privacy preserving scheme and the corresponding parameters. Then, this information is fed back to the *kernel-space policy engine* module to configure the OVS datapath actions be to applied on the flow.

In `OVS datapath`, we introduce a number of new actions to enforce the privacy preserving policies on each packet of the corresponding flows. For example, we define three new actions: "adaptive sampling", "padding", and "reverse padding" for the *packet-padding* policies. Both the "adaptive sampling" and "padding" actions are applied on the egress packets that will be transmitted by one of the agents, while the "reverse padding" action is applied on the received packets on the other agent. The "padding" action is responsible for generating the padding bytes for each packet based on the information of the *packet-padding* policy received from *user-space policy engine* module and then pads the packet with these bytes.

Note that, in *packet-padding*, not every packet will be selected for the "padding" action. The "adaptive sampling" action uses the sampling probability parameter p to decide for each packet whether to be padded. Moreover, the action also allows to modifying p over the lifetime of the flow. Studies show that the sizes of the initial packets of an application/flow show more noticeable signature than the later packets [39]. Therefore, in "adaptive sampling" action, using a higher value of p for the initial packets of a flow[2] will increase the efficiency of the scheme in obfuscating the signature of the application/flow.

In realizing *packet-delaying*, similar to our previous work [24], we have implemented a new *qdisc* scheduler for *Linux Traffic Control (tc)* [3] in this module. We define one new action; "delay" for the *packet-delaying* policies. This action is responsible, by controlling the*qdisc* scheduler, to increase the queueing time of each packet of the targeted application/flow with a random delay generated by the configured uniform distribution.

In our implementation, we utilize the unused reserved bits 6-7 of the "ToS" field in the IP header to mark the padded

[2]We use packet counter statistics to change the sampling probability.

packet selected by "adaptive sampling" action. In addition, we use the "Options" field, which is currently reserved for future use, to carry the parameters of the corresponding privacy preserving scheme (e.g., padding size). On the receiving agent of *PrivacyGuard*, its `datapath` module checks every receiving ingress packet IP header (i.e. the "ToS" and "Options" fields) to identify whether the packet is padded, which in turn applies the "reverse padding" action that uses the information in the "Options" field to recover the original packet.

E. IPSec Tunneling Module

This module is a part of the `OVS datapath` in both client and infrastructure agents and is responsible to define and configure two new `vports`: internal `vport` (`vport-internal`) and IPsec `vport` (`vport-ipsec`) as shown in Figure 3 to apply IPsec tunneling protocol on top of any other privacy preserving scheme. Although `OVS datapath` could support already existing tunneling ports such as *vport-gre* and *vport-vxlan* that are used for IPsec tunneling protocol, these tunneling protocols have associated overhead in terms of the corresponding GRE protocol or VLAN protocol headers in addition to the IPsec protocol header. Therefore, to eliminate these additional overheads, we introduce a new class of `vport`, called `vport-ipsec` that provides IPSec tunneling protocol without any of the overhead in case of using GRE or VLAN tunneling. *OVS datapath* routes an IP packet through `vport-ipsec` after the corresponding privacy preserving scheme (e.g., traffic shaping scheme) is applied.

PrivacyGuard binds both `vport-internal` and `vport-ipsec` with IP addresses. The IP address of `vport-internal` is used as the default source IP address for all the running applications in the mobile device. Thus, *PrivacyGuard* makes sure that all application's network flows pass through the `OVS datapath` using the `vport-internal` port. Note that, PrivacyGuard uses the IP address of the `vport-internal` port as the source address to set up the flow policies in the *user-space policy engine* module.

Once an egress packet is passed through the `OVS datapath`, it gets forwarded to `vport-ipsec`. Then, before sending it to the `vport-ipsec` port, `OVS datapath` changes the source IP address of the egress packet to the IP address of the `vport-ipsec` port, which is the IP address of the physical network interface. Once the egress packet passes through *vport-ipsec*, it gets forwarded to the `XFRM` framework that is basically responsible for handling the IPSec protocol. Note that the `XFRM` framework only applies IPSec protocol on the flows using IPSec tunneling by utilizing Security Association Database (SAD) and Security Policy Database (SPD) to maintain the IPSec configuration parameters for these flows. Note that, PrivacyGuard *policy controller* uses the IP address of `vport-ipsec` as the source IP address for identifying all the flows using IPSec tunneling to the IKEv2 daemon, and eventually in the `XFRM` framework. Thus, we make sure that the `XFRM` framework applies IPsec protocol on the egress packet after it leaves the `vport-ipsec` port.

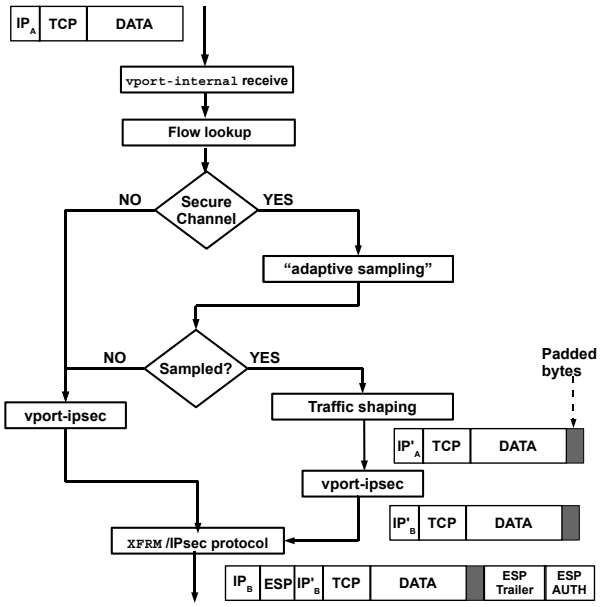

Fig. 5. The flowchart of handling an egress packet in PrivacyGuard OVS datapath.

Figure 5 shows the flowchart of how an egress packet is handled inside the OVS `datapath` in the kernel space of *PrivacyGuard*. Note that, *PrivacyGuard* kernel component needs to change the IP header twice when handling egress packets. Initially, before entering OVS `datapath`, every egress packet have IP header with the source IP address A (i.e., IP_A), which is the IP address of `vport-internal` port. When an egress packet is selected for privacy preserving scheme (e.g., traffic shaping scheme) inside the OVS `datapath`, the "padding" action adds the "IP option" field in the packet header (i.e., IP'_A). Then, before every egress packet get forwarded to the `vport-ipsec` port, OVS changes the source IP address to B (i.e., IP'_B), which is the IP address of`vport-ipsec` port. Finally, the *XFRM* framework applies IPSec tunneling on the egress packet before sending it to the WiFi network interface.

V. Performance Evaluation

A. Experiments Setup

In our experiments we use a Nexus 4 smartphone with Android 4.4 running *PrivacyGuard* client agent as a user device, and an Ubuntu 16.04 laptop with Intel Core i5-2520M @2.5GHz CPU running *PrivacyGuard* infrastructure agent as a Wi-Fi AP. We install 8 commercially available IoT device-based applications on the Nexus device which acts as the gateway. These applications span different domains including home appliance, medical and fitness. We use three different traffic shaping schemes based on *packet-padding* and *packet-delaying*. The first is Norm_Pad, which is a *packet-padding* scheme where the padding bytes size follows a Gaussian distribution with μ and σ parameters are set to 400 and 100 bytes respectively. The second is Norm_Pad_Delay that applies both a *packet-padding* scheme (same as Norm_Pad),

and a *packet-delaying* scheme. In order to calculate the delay, we calculate both the average minimum IPT (min) and the average maximum IPT (max) of all the applications. Then, we use that min-max range to generate uniformly distributed random delays to extend the IPTs of the targeted application/flow. In our implementation, we set min and max to 0ms and 20ms respectively. The last scheme is Max_Pad_Delay that pads packets with the maximum possible bytes and also use the *packet-delaying* scheme. We mimic the maximum padding bytes by using a Gaussian distribution and μ and σ parameters are set to 1500 and 10 bytes respectively. In our evaluation, we run 100 classification experiments of each of the traffic shaping schemes for each application.

We evaluate our experiments with metrics based on both efficiency and overhead. Efficiency is measured using the *accuracy* and *precision* metrics. The accuracy of a traffic shaping scheme of a specific application is the percentage of the true positive classifications (i.e., the number of the target application classification experiments that are correctly identified the target application) to the total application classification experiments. On the other hand, the precision is the percentage of the true positive classifications to the summation of the true positive classifications and the false positive classifications (i.e., the number of the other applications classification experiments that are wrongly identified as the target application). An application with a low precision indicates that the used traffic shaping scheme confuses the adversary in which it makes him falsely identify a large portion of the traffic of the other applications as the target application. Note that the efficiency metrics are calculated based on the classification models discussed in Section II.

The overhead is measured in terms of: i) the network bandwidth overhead measured as the percentage of the additional bytes sent over the network, and ii) the energy overhead measured the percentage of additional power consumption. To measure the actual power consumption, we connect the battery of Nexus 4 to the Monsoon Power Monitor device.

B. Traffic Shaping Schemes Performance

In this section, we analyze and evaluate the performance of Norm_Pad, Norm_Pad_Delay, and Max_Pad_Delay traffic shaping schemes as examples of different privacy preserving schemes. Figure 6 shows the accuracy of Norm_Pad scheme with different probabilities (p) for the eight applications. As shown, as p increases, the scheme becomes more efficient in obfuscating the application signature that results in a decreasing identification accuracy. It is interesting to observe that while the scheme has high efficiency for some of the applications such as the *Fitbit* application with large values of p, it fails in obfuscating other applications such as the *Flux-lightbulb* application. We also tried different configurations of the scheme in which it generates similar results.

By analyzing the traffic characteristics of the applications with low efficiency (i.e., Elegato-plug, Avea-lightbulb, Flux-lightbulb, and iLink-lightbulb), we observed that these applications transmit their packets at periodic patterns. Therefore, any privacy scheme that is based only on *packet-padding* will have a low efficiency in obfuscating these applications. However, in Figure 7 that plots the accuracy of Norm_Pad_Delay scheme, all applications show better efficiency (i.e., low accuracy) as p increases. Therefore, *different applications/flows have different traffic characteristics that require different privacy preserving schemes in order to achieve high efficiency.*

Figure 8 shows the accuracy of Max_Pad_Delay scheme in which its efficiency exceeds the other two schemes even at low values of p. This is because some applications such as *Elegato-plug iLink-lightbulb*, and *Flux-lightbulb* transmit many large size packets in which the signature patterns of these traffic are hard to be obfuscated with a padding scheme using few padding bytes. However, when we pad the packets to the maximum possible packet size (i.e., MTU size), it becomes hard to identify the signature of these traffic. In addition, Figure 9 shows the precision of the same scheme. As shown, the precision drops gracefully as p increases that help significantly in obfuscating the applications/flows by confusing the adversary more. For example, while Figure 8 states that an adversary will be 25% of the time is able to correctly identify the *iLink-lightbulb* application when p is 0.8, Figure 9 states that only 60% all the traffic identified as the *iLink-lightbulb* application is correct identifications. Therefore, we could easily conclude that Max_Pad_Delay scheme with p set to 0.8 will be able to obfuscate the *iLink- lightbulb* application approximately 85% of the time.

Unfortunately, the high efficiency of Max_Pad_Delay scheme comes with its associated overhead. Figures 10 and 11 show the energy consumption and network bandwidth overhead respectively for Max_Pad_Delay scheme. The overhead of this scheme exceeds the overhead of the other schemes at all p values (we omit the other figures due to space limitation). This is an example that *privacy preserving schemes with high efficiency will be typically associated with high overhead*. Moreover, from figures 6 and 7, we can see that the two different schemes have similar efficiency (i.e., accuracy is 60%) for the *Fitbit* application for p values of 0.8 and 0.6 respectively. Note that while the first scheme has significant overhead in terms of network bandwidth because of large p, the other scheme achieves an equivalent efficiency but with lower network overhead (lower p) and additional packet delays. *Since different schemes could have equivalent efficiency but with different overheads, policies need to be carefully designed based on the impact of these overheads on the application, user, device, and network.*

C. PrivacyGuard Programmability and Flexibility

In Subsection III-C, we discussed some real-world scenarios and their ideal privacy preserving schemes. We will evaluate the flexibility and programmability of PrivacyGuard in terms of the ability to provide such flexible obfuscation schemes and their efficiency under different contexts. In the following, we will refer to the policies listed in Figure 4.

We first evaluate the programmability and flexibility of PrivacyGuard. High sensitive applications such as *Fitbit* could

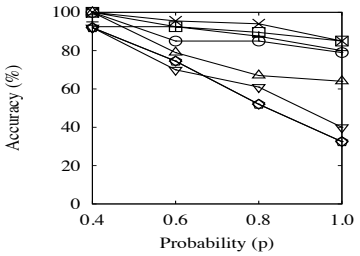

Fig. 6. The accuracy of Norm_Pad scheme for different applications and p values.

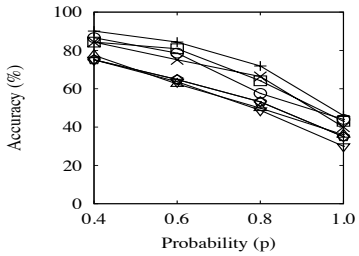

Fig. 7. The accuracy of Norm_Pad_Delay scheme for different applications and p values.

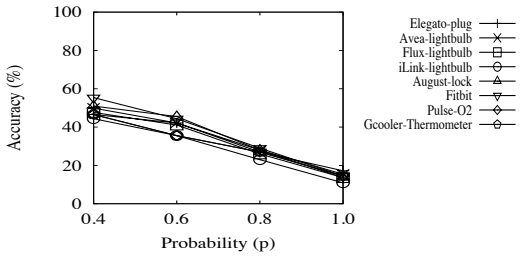

Fig. 8. The accuracy of Max_Pad_Delay scheme for different applications and p values.

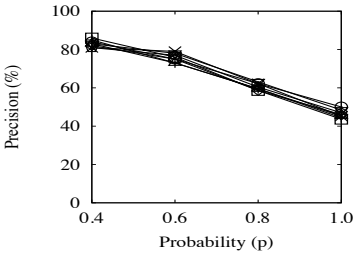

Fig. 9. The precision of Max_Pad_Delay scheme for different applications and p values.

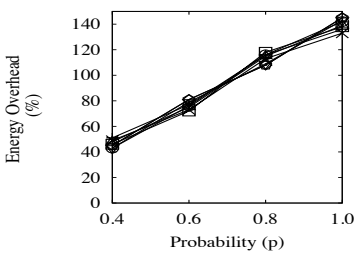

Fig. 10. The power consumption overhead of Max_Pad_Delay scheme.

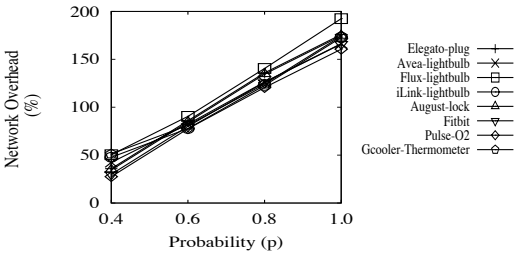

Fig. 11. The network bandwidth overhead of Max_Pad_Delay scheme.

reveal sensitive activities while the user is at home. Therefore, a policy like *Policy #1* that applies high efficient scheme (i.e., Max_Pad_Delay scheme) is used for such sensitive applications/flows. However, Figure 10 shows that such high efficient scheme incurs a 150% increase in energy consumption. Since *PrivacyGuard* has the capability to create and set different context parameters dynamically, *Policy #1* is configured to be only applied during the time periods with the sensitive activities. This flexibility in setting the policy results in a significant energy saving. In addition, *Policy #1* and *Policy #2* show an example of the fine-grained programming ability of PrivacyGuard in configuring different policies for different flows of the same application. Different obfuscation schemes such as like Max_Pad_Delay and Norm_Pad schemes configured in *Policy #1* and *Policy #2* respectively are suitable for different flows with different traffic characteristics (i.e., periodic traffic with large packet sizes versus real time traffic).

Next, we evaluate *PrivacyGuard* ability in adapting the selected policies to the contexts changes with respect to privacy and performance. *Policy #4* is an example of a policy that applies Max_Pad_Delay scheme for high performance efficiency for sensitive applications such as the *Fitbit* application when the network load is unsaturated. Figure 8 shows that the this policy has a very high performance in obfuscating the *Fitbit* application with an accuracy as low as 15% when p is set to 1, which comes with a high network overhead of about 150% additional transmitted bytes as shown in Figure 11. However, when the network condition changes to a saturated network with high load, *PrivacyGuard* switches to apply *Policy #5* for the *Fitbit* application. Figure 11 shows that this switch adapts to the new network condition by significantly dropping the network overhead from 150% to 80% at the cost of reducing the obfuscation scheme efficiency by increasing the accuracy from 15% (high efficiency) to 40% (moderate efficiency) as shown in Figure 8. Similarly, a change in the device context such as its battery level that changes from high to low will trigger *PrivacyGuard* to apply a similar switch from a high efficient scheme (*Policy #4*) to a low energy overhead scheme (*Policy #5*) in order to preserve the remaining battery level. Moreover, a change in the user context by moving into an insecure location from a secure location, PrivacyGuard seamlessly will enforce the IPSec tunneling scheme by switching to *policy #3* as long as the battery level of the device is high.

D. PrivacyGuard Overhead

We evaluate the impact of *PrivacyGuard* modules on the mobile device performance in terms of both processing delay and CPU usage. In measuring any additional delays in processing the application packets, we measure the number of transmitted packets per second when *PrivacyGuard* is not active and when *PrivacyGuard* is enabled. Note that to eliminate any bias in the measurements, we set the min-max range of the *packet-delaying* shaping scheme to zero. Figure 12 shows the cumulative distribution (CDF) of the processing delay overhead on the mobile device. The figure indicates that the *PrivacyGuard* modules has a negligible processing delay overhead (less than 1% for 80% of the time).

We also measure the CPU usage overhead for both the user-space and kernel-space modules of *PrivacyGuard*, which is shown in Figure 13. From the figure, we notice that the kernel-space and user-space modules increase the CPU usage with less than 1.5% and 2% respectively for 80% of the time.

VI. Related Work

In order to protect the application's network data, there have been many proposed solutions for managing network-wide

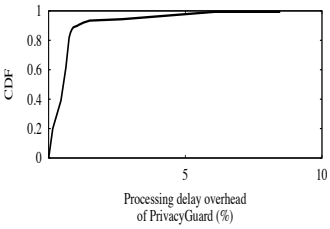

Fig. 12. The CDF of the processing delay overhead of PrivacyGuard.

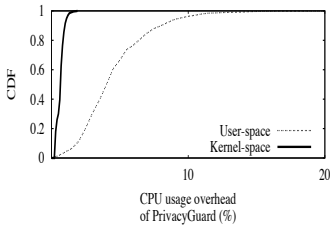

Fig. 13. The CDF of both the kernel-space and user-space CPU usage overhead of PrivacyGuard.

mobile devices from network infrastructure [41]. However, such remote network management solutions are not well-suited for dynamic network devices like mobile devices. Therefore, researchers are focusing recently on client-side network security solutions [18], [21]. Among these works, very few ones have fine-grained and programmable network security policies targeting the applications. For instance, one of these solutions provides application specific and device-context-aware network access policies [18] . In another work, the authors have used network virtualization technique, similar to *PrivacyGuard*, to isolate the network traffic between sensitive (i.e., medical applications) and non-sensitive applications [21]. However, unlike *PrivacyGuard*, none of the client-side security solutions have focused on the network security concern of the side-channel attack for sensitive mobile applications.

Previously, numerous works have addressed the eavesdropping attack [40], [46] based on side-channel information. However, very few works have actually proposed and validated the use of *traffic shaping* techniques to address eavesdropping attack [8], [11]. However, none of these works has considered IoT devices and mobile device applications. Recently, researchers were able to show how a side channel attack could identify IoT devices, and then proposed a traffic shaping technique based on rate-reshaping [7]. This technique uses Independent Link Padding based approach that transmits fixed size packets at a constant rate. Given the programmability of *PrivacyGuard*, it is straightforward to adopt this technique as well as any other *traffic shaping* technique into *PrivacyGuard*.

In another work, authors in [45] apply traffic demultiplexing at the MAC layer to protect the Wi-Fi traffic, which requires expensive MAC layer management between mobile devices and access points (APs). In addition, this technique requires modifying the wireless device driver for supporting the multiple virtual interfaces and distributing the traffic over these interfaces. *PrivacyGuard* doesn't require any driver level modification and it is not limited to any specific Wi-Fi AP configuration.

VII. Conclusion and Future Work

We presented *PrivacyGuard*, a flexible and programmable privacy preserving framework to obfuscate the activities of sensitive IoT and mobile applications from adversarial attack. We developed and evaluated *PrivacyGuard*'s flexibility, efficiency, and overhead using a testbed of Android mobile devices running different sensitive applications.

Currently, we are exploring several approaches that can recommend optimal privacy schemes and automatically build policy engine rules with minimum user effort. One approach is to design a crowdsourcing-based recommendation system that can assist users to configure optimal policies. Based on the recent work on permission control of mobile applications [25], [31] and access control for online social network applications [33], a similar crowdsourcing strategy could be utilized to recommend the optimal privacy preserving policy for a given context. We are also considering a reinforcement learning [17] based approach in discovering the optimal policy configurations that maximize certain rewards.

Although traffic shaping is widely accepted by the security and privacy community as a solution to the side channel attacks, we are currently studying other types of IoT attacks [5], [7], and the corresponding obfuscation schemes. We are planning to integrate these new schemes into *PrivacyGuard* and then evaluate and compare them with the traffic shaping schemes under different contexts.

Finally, we are considering to develop a set of PrivacyGuard APIs to be utilized by application developers. It is expected that these APIs would give more control to the developer in selecting the best combination of a functionality and obfuscation scheme. For instance, during a low battery level, instead of switching to a lower efficiency scheme, the developer could configure the app to drop less useful functional flows such as sending advertising data. We believe having these APIs would help the developers in optimizing the functionalities and behavior of their applications satisfying both high privacy and performance to the users.

Acknowledgment

The authors would like to thank our shepherd Prof. Max Muehlhaeuser and the anonymous reviewers whose comments have greatly improved this manuscript.

References

[1] Beddit Sleep Monitor. http://www.beddit.com/.
[2] Linux Network Management with "netstat". http://www.linuxandubuntu.com/home/linux-network-management-with-netstat.
[3] Linux Traffic Control. https://linux.die.net/man/8/tc.
[4] Open vSwitch. http://openvswitch.org/.
[5] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and A. S. Uluagac. Peek-a-boo: I see your smart home activities, even encrypted! *arXiv preprint arXiv:1808.02741*, 2018.
[6] Y. Amar, H. Haddadi, R. Mortier, A. Brown, J. Colley, and A. Crabtree. An analysis of home iot network traffic and behaviour. *arXiv preprint arXiv:1803.05368*, 2018.
[7] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044*, 2017.
[8] M. Backes, G. Doychev, and B. Köpf. Preventing side-channel leaks in web traffic: A formal approach. In *20th Annual Network & Distributed System Security Symposium (NDSS)*, February 2013.
[9] S. Banerjee, N. Klingensmith, P. Liu, and A. Sridhar. Edge computing in the extreme for sustainability. In N. Sastry and S. Chakraborty, editors, *Communication Systems and Networks*, pages 93–109, Cham, 2017. Springer International Publishing.
[10] I. Ben Mustafa, T. Nadeem, and E. Halepovic. Flexstream: Towards flexible adaptive video streaming on end devices using extreme sdn. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 555–563, 2018.

[11] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206, May 2010.

[12] T. Davenport and J. Lucker. Running on data: Activity trackers and the internet of things, 2015. https://dupress.deloitte.com/dup-us-en/deloitte-review/issue-16/internet-of-thingswearable-technology.html.

[13] Ericsson. Internet of Things forecast., May, 2018. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast.

[14] D. Ferreira, V. Kostakos, A. R. Beresford, J. Lindqvist, and A. K. Dey. Securacy: an empirical investigation of android applications' network usage, privacy and security. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, page 11. ACM, 2015.

[15] L. Franceschi-Bicchierai. Internet of things teddy bear leaked 2 million parent and kids message recordings, 2017. https://motherboard.vice.com/en_us/article/pgwean/internet-of-things-teddy-bear-leaked-2-million-parent-and-kids-message-recordings.

[16] X. Fu, B. Graham, R. Bettati, and W. Zhao. Active traffic analysis attacks and countermeasures. In *International Conference on Computer Networks and Mobile Computing (ICCNMC)*, pages 31–39, Oct 2003.

[17] V. Heidrich-Meisner, M. Lauer, C. Igel, and M. A. Riedmiller. Reinforcement learning in a nutshell. In *15th European Symposium on Artificial Neural Networks (ESANN)*, pages 277–288, 2007.

[18] S. Hong, R. Baykov, L. Xu, S. Nadimpalli, and G. Gu. Towards sdn-defined programmable byod (bring your own device) security. In *23rd Annual Network & Distributed System Security Symposium (NDSS)*, 2016.

[19] IHS Markit. More than Six Billion Smartphones by 2020, IHS Markit Says., Jan, 2017. http://news.ihsmarkit.com/press-release/technology/more-six-billion-smartphones-2020-ihs-markit-says.

[20] Juniper Research Ltd. Wi-fi calling operators., June, 2015. http://www.juniperresearch.com/documentlibrary/white-papers/wifi-calling-operators?utm_source=gorkanapr&utm_medium=email&utm_campaign=dataoffload15pr2.

[21] M. Kano. SeaCat: an SDN end-to-end containment architecture. Master's thesis, 2015.

[22] S. Kumar. Ubiquitous smart home system using android application. *arXiv preprint arXiv:1402.2114*, 2014.

[23] N. Lars. Connected medical devices, apps: Are they leading the iot revolution or vice versa?, 2014. https://www.wired.com/insights/2014/06/connected-medical-devices-apps-leading-iot-revolution-vice-versa/.

[24] J. Lee, M. Uddin, J. Tourrilhes, S. Sen, S. Banerjee, M. Arndt, K.-H. Kim, and T. Nadeem. mesdn: mobile extension of sdn. In *Proceedings of the fifth international workshop on Mobile cloud computing & services*, pages 7–14. ACM, 2014.

[25] R. Liu, J. Cao, K. Zhang, W. Gao, J. Liang, and L. Yang. When privacy meets usability: Unobtrusive privacy permission recommendation system for mobile apps based on crowdsourcing. *IEEE Transactions on Services Computing*, 11(5):864–878, Sep. 2018.

[26] W. M. Liu, L. Wang, P. Cheng, K. Ren, S. Zhu, and M. Debbabi. Pptp: Privacy-preserving traffic padding in web-based applications. *IEEE Transactions on Dependable and Secure Computing*, 11(6):538–552, Nov 2014.

[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[28] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184, June 2017.

[29] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 99–110, 2007.

[30] H. Rahbari and M. Krunz. Secrecy beyond encryption: obfuscating transmission signatures in wireless communications. *IEEE Communications Magazine*, 53(12):54–60, Dec 2015.

[31] B. Rashidi, C. Fung, A. Nguyen, T. Vu, and E. Bertino. Android user privacy preserving through crowdsourcing. *IEEE Transactions on Information Forensics and Security*, 13(3):773–787, March 2018.

[32] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, Oct 1949.

[33] M. Shehab, A. Squicciarini, G.-J. Ahn, and I. Kokkinou. Access control for online social networks third party applications. *Computers & Security*, 31(8):897 – 911, 2012.

[34] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564. IEEE, 2017.

[35] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: smartphone fingerprinting via application behaviour. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, pages 7–12. ACM, 2013.

[36] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 19–30, May 2002.

[37] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 439–454, March 2016.

[38] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, Jan 2018.

[39] M. Uddin and T. Nadeem. Trafficvision: A case for pushing software defined networks to wireless edges. In *IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 37–46. IEEE, 2016.

[40] Q. Wang, A. Yahyavi, B. Kemme, and W. He. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 433–441, Sep. 2015.

[41] X. Wang, K. Sun, Y. Wang, and J. Jing. Deepdroid: Dynamically enforcing enterprise policy on android devices. In *22nd Annual Network & Distributed System Security Symposium (NDSS)*, 2015.

[42] C. V. Wright, S. E. Coull, and F. Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *In Proceedings of the 16th Network and Distributed Security Symposium*, pages 237–250. IEEE, 2009.

[43] M. Wynn, K. Tillotson, R. Kao, A. Calderon, A. Murillo, J. Camargo, R. Mantilla, B. Rangel, A. A. Cardenas, and S. Rueda. Sexual intimacy in the age of smart devices: Are we practicing safe iot? In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 25–30. ACM, 2017.

[44] B. Yan and G. Chen. Appjoy: Personalized mobile application discovery. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 113–126, 2011.

[45] F. Zhang, W. He, Y. Chen, Z. Li, X. Wang, S. Chen, and X. Liu. Thwarting wi-fi side-channel analysis through traffic demultiplexing. *IEEE Transactions on Wireless Communications*, 13(1):86–98, January 2014.

[46] F. Zhang, W. He, X. Liu, and P. G. Bridges. Inferring users' online activities through traffic analysis. In *Proceedings of the Fourth ACM Conference on Wireless Network Security*, WiSec '11, pages 59–70, 2011.