Packet-Level Signatures for Smart Home Devices

Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, Brian Demsky
University of California, Irvine
{rtrimana, jvarmark, athina, bdemsky}@uci.edu

***Abstract*—Smart home devices are vulnerable to passive inference attacks based on network traffic, even in the presence of encryption. In this paper, we present PINGPONG, a tool that can automatically extract packet-level signatures for device events (*e.g.,* light bulb turning ON/OFF) from network traffic. We evaluated PINGPONG on popular smart home devices ranging from smart plugs and thermostats to cameras, voice-activated devices, and smart TVs. We were able to: (1) automatically extract previously unknown signatures that consist of simple sequences of packet lengths and directions; (2) use those signatures to detect the devices or specific events with an average recall of more than 97%; (3) show that the signatures are unique among hundreds of millions of packets of real world network traffic; (4) show that our methodology is also applicable to publicly available datasets; and (5) demonstrate its robustness in different settings: events triggered by local and remote smartphones, as well as by home-automation systems.**

I. INTRODUCTION

Modern smart home devices are seeing widespread adoption. They typically connect to the Internet via the home Wi-Fi router and can be controlled using a smartphone or voice assistant. Although most modern smart home devices encrypt their network traffic, recent work has demonstrated that the smart home is susceptible to passive inference attacks [3], [10]–[13], [19], [29], [44], [45]. An eavesdropper may use characteristics of the network traffic generated by smart home devices to infer the device type and activity, and eventually user behavior. However, existing passive inference techniques have limitations. Most can only identify the device type and whether there is device activity (an event), but not the exact type of event or command [10]–[13], [29], [44], [45]. Others only apply to a limited number of devices from a specific vendor [19], or need more information from other protocols (*e.g.,* Zigbee/Z-Wave) [3], [54] and the application source code [54]. Inference based on traffic volume analysis can be prevented by traffic shaping [3], [10]. Finally, many of these attacks assume that IP traffic is sniffed upstream from the home router, while the scenario where a local attacker sniffs encrypted Wi-Fi traffic has received less attention [10], [23].

In this paper, we experiment with a diverse range of smart home devices, namely 19 popular Wi-Fi and Zigbee devices (12 of which are the most popular smart home devices on Amazon) from 16 popular vendors, including smart plugs, light bulbs, thermostats, home security systems, etc. During our analysis of the network traffic that these devices generate, we observed that events on smart home devices typically result in communication between the device, the smartphone, and the cloud servers that contains pairs of packets with predictable lengths. A packet pair typically consists of a *request* packet from a device/phone ("PING"), and a *reply* packet back to the device/phone ("PONG"). In most cases, the packet lengths are distinct for different device types and events, thus, can be used to infer the device and the specific type of event that occurred. Building on this observation, we were able to identify new *packet-level signatures* (or *signatures* for short) that consist only of the lengths and directions of a few packets in the smart home device traffic. In this paper, we show that these signatures: (1) can be extracted in an automated and systematic way without prior knowledge of the device's behavior; (2) can be used to infer fine-grained information, *e.g.,* event types; (3) correspond to a variety of different events (*e.g.,* "toggle ON/OFF" and "Intensity"/"Color"); and (4) have a number of advantages compared to prior (*e.g.,* statistical, volume-based) approaches. More specifically, this paper makes the following contributions.

New Packet-Level Signatures. We discover new IoT device signatures that are simple and intuitive: they consist of short sequences of (typically 2-6) packets of specific lengths, exchanged between the device, the smartphone, and the cloud. The signatures are effective:

1) They detect event occurrences with an average recall of more than 97%, surpassing the state-of-the-art techniques (see Sections II and V-B).
2) They are unique: we observe a low false positive rate (FPR), namely 1 false positive per 40 million packets in network traces with hundreds of millions of packets (see Section V-C).
3) They characterize a wide range of devices: (i) we extract signatures for 18 out of the 19 devices we experimented with, including the most popular home security devices such as the Ring Alarm Home Security System and Arlo Q Camera (see Section V-A); (ii) we extract signatures for 21 additional devices from a public dataset [39], including more complex devices, *e.g.,* voice-command devices, smart TVs, and even a fridge (see Section V-F).
4) They are robust across a diverse range of settings: (i) we extract signatures both from testbed experiments and publicly available datasets; and (ii) we trigger events in different ways, *i.e.,* using both a local and a remote smartphone, and using a home automation system.
5) They can be extracted from both unencrypted and encrypted traffic.
6) They allow quick detection of events as they rely only on a few packet lengths and directions, and do not require any statistical computation.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24097
www.ndss-symposium.org

Automated Extraction of Packet-Level Signatures. We present PINGPONG, a methodology and software tool that: (1) automates the extraction of packet-level signatures without prior knowledge about the device, and (2) detects signatures in network traces and real network traffic. For signature extraction, PINGPONG first generates training data by repeatedly triggering the event, for which a signature is desired, while capturing network traffic. Next, PINGPONG extracts request-reply packet pairs per flow ("PING-PONG"), clusters these pairs, and post-processes them to concatenate pairs into longer sequences where possible. Finally, PINGPONG selects sequences with frequencies close to the number of triggered events as the final signatures. The signature detection part of PINGPONG leverages the simplicity of packet-level signatures and is implemented using simple state machines. PINGPONG's implementation and datasets are made available at [49].

The remainder of this paper is structured as follows. Section II outlines related work and puts PINGPONG in perspective. Section III presents the threat model (including two distinct adversaries: a WAN sniffer and a Wi-Fi sniffer), our experimental setup, and an illustrative example of packet-level signatures in smart plugs. Section IV presents the design of the PINGPONG system, including extraction and detection of signatures. Section V presents the evaluation of PINGPONG, using our own testbed experiments, as well as several external—publicly available—datasets. Section VI presents an in-depth discussion on possible defenses against packet-level signatures. Section VII concludes and outlines directions for future work. Further details on discussion and evaluation results are provided in the technical report [48].

II. RELATED WORK

Table I summarizes the properties of PINGPONG and compares it to the other IoT traffic analysis approaches.

Network Signatures for IoT devices. A growing body of work uses network traffic (metadata) analysis to characterize the type and activity of IoT devices. A series of papers by Apthorpe *et al*. [10]–[13] use traffic volume/shape-based signatures to infer IoT device activity, but cannot always determine the exact type of the event. Furthermore, the signatures corresponding to different traffic shapes are intuitive, but not automatically extracted. The authors propose stochastic traffic padding (STP) to mitigate volume-based inference attacks.

HomeSnitch [33] by OConnor *et al*. identifies IoT activity using a key observation that is similar to ours, *i.e.,* the client (the IoT device) and the server take turns in a request-reply communication style. HomeSnitch and PINGPONG both exclude IP addresses, port numbers, and DNS information from their event inference methodologies, but differ in terms of the granularity of the features they use: HomeSnitch uses statistics derived from the entire client-server dialog, whereas PINGPONG considers the direction and length of each individual packet. Interestingly, the most important feature used in HomeSnitch is the average number of bytes sent from the IoT device to the server per turn. This result aligns with the main observation of this paper, *i.e.,* packet lengths of individual requests (and replies) uniquely identify device events.

A recent paper by Ren *et al*. [39] presents a large-scale measurement study of IoT devices and reveals how these

	Approaches for IoT Network Traffic Signatures						
	Vol. +DNS based [10]–[13]	Nest device [19]	Machine Learning [33]	Machine Learning [3]	Machine Learning [44] [45]	ZigBee/ Z-Wave device [54]	PING PONG
(1) Signature can detect							
Device type	✓	✓	✓	✓	✓	✓	✓
Event type	×	✓	✓	✓	×	✓	✓
(2) Applicability to devices							
>15 Models	×	×	✓	✓	✓	×	✓
(3) Observation points/threat models							
LAN	×	✓	✓	✓	✓	N/A	✓
WAN	✓	×	×	×	×	N/A	✓
Wi-Fi	✓	×	×	✓	×	N/A	✓
(4) Signature characteristics							
Feat.	Traffic vol., DNS	TCP conn. size, proto.	13, ADU	(795) 197	12	Packet length & dir.	Packet length & dir.
Inter-pretable	✓	×	✓	×	×	✓	✓
Auto. Extract.	×	×	✓	✓	✓	✓	✓
(5) Resilient against defenses							
VPN	×	×	×	×	×	N/A	✓
Traffic shaping	×	×	×	×	×	N/A	✓

TABLE I. PINGPONG'S PROPERTIES VS. ALTERNATIVE APPROACHES (✓= YES; × = NO).

devices operate differently in the US and the UK with respect to Internet endpoints contacted, exposure of private information, etc. We use that dataset to evaluate our methodology in Section V-F. The paper also presents a classifier that can infer event types spanning many device categories; this, however, is not the focus of the paper. Other well-known measurement studies and publicly available IoT network traffic datasets include YourThings [5], [6] and [45], which we use in our evaluation in Section V-C.

Other papers consider specific types of devices or protocols. Copos *et al*. [19] analyze network traffic of the Nest Thermostat and Nest Protect (only) and show that the thermostat's transitions between the *Home* and *Auto Away* modes can be inferred. Other work [3], [54] focus on Zigbee/Z-Wave devices and leverage specialized Zigbee/Z-Wave sniffers.

Most event inference techniques rely on machine learning [29], [44], [45] or statistical analysis of traffic time series [3], [19], [33], [39]. Limitations of these approaches include: the inability to differentiate event *types* [29], [44], [45] (*e.g.,* distinguishing ON from OFF), and lack of resistance to traffic shaping techniques [3], [19], [33], [39] such as [10]. On the other hand, our work identifies simple packet exchange(s) between the device/smartphone and the cloud that uniquely identify event types. At the same time, PINGPONG's classification performance (recall of more than 97%) is better than most statistical approaches: [3] reported 90% accuracy, [19] reported 88% and 67% accuracy, and [39] reported some F1 scores as low as 0.75. Unsupervised learning techniques may be hard to interpret, especially for large feature sets (*e.g.,* 197 features in [3]). PINGPONG also uses clustering to identify reoccurring packet pairs, but provides an intuitive interpretation of those pairs: they correspond to a request and the subsequent reply.

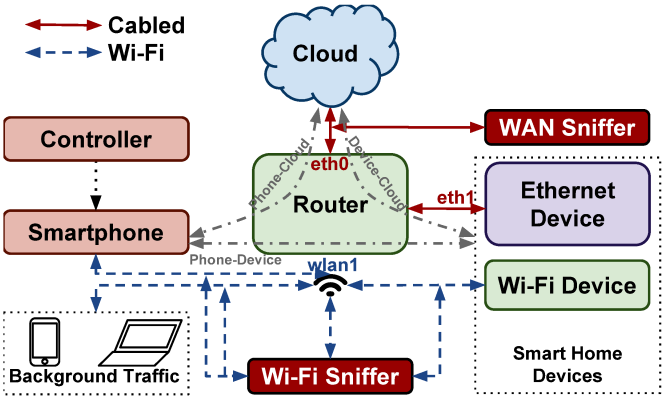

Fig. 1. Our experimental setup for studying smart home devices. "Wi-Fi Device" is any smart home device connected to the router via Wi-Fi (*e.g.,* Amazon and WeMo plugs). "Ethernet Device" is any smart home device connected to the router via Ethernet (*e.g.,* SmartThings hub that relays the communication of Zigbee devices). Smart home device events may result in communication between Phone-Cloud, Device-Cloud, or Phone-Device. There may also be background traffic from additional computing devices in the home.

Network Traffic Analysis beyond IoT. There is a large body of work in the network measurement community that uses traffic analysis to classify applications and identify anomalies [26], [27], [32], attacks [20], or malware [8], [37]. There has also been a significant amount of work on fingerprinting techniques in the presence of encryption for web browsing [14], [17], [18], [21], [24], [25], [28], [30], [35], [36], [51], and variable bit-rate encodings for communication [52], [53] and movies [42]. For these examples, the underlying protocols are well understood, while PINGPONG can work with (and is agnostic to) any arbitrary, even proprietary, application-layer protocol.

Defenses. Related to profiling and fingerprinting is also the body of work on defenses that obfuscate traffic signatures. Examples include [31], [36] that use packet padding and traffic injection techniques to prevent website fingerprinting. In Table I, we mention two general defense approaches: (1) *traffic shaping* that refers broadly to changing the shape of traffic over time; and (2) *VPN* that brings multiple benefits such as encryption (that our signatures survive), and multiplexing of several flows. We partly evaluate these defenses (see Appendix C in [48]). A VPN also provides a natural place to implement additional defenses (*e.g.,* packet padding, which is discussed in Section VI).

III. PROBLEM SETUP

In this section, we first present our threat model. Then, we present the smart home environment and the passive inference attacks we consider. We also discuss a key insight we obtained from manually analyzing network traffic from the simplest devices—smart plugs. The packet sequences we observed in smart plugs inspired the PINGPONG methodology for automatically extracting signatures.

A. Threat Model

In this paper, we are concerned with the network traffic of smart home devices leaking private information about smart home devices and users. Although most smart home devices encrypt their communication, information can be leaked by traffic metadata such as the lengths and directions of these encrypted packets.

No.	Device Name	Model Details
1.	Amazon plug	Amazon Smart Plug
2.	WeMo plug	Belkin WeMo Switch
3.	WeMo Insight plug	Belkin WeMo Insight Switch
4.	Sengled light bulb	Sengled Element Classic
5.	Hue light bulb	Philips Hue white
6.	LiFX light bulb	LiFX A19
7.	Nest thermostat	Nest T3007ES
8.	Ecobee thermostat	Ecobee3
9.	Rachio sprinkler	Rachio Smart Sprinkler Controller Generation 2
10.	Arlo camera	Arlo Q
11.	Roomba robot	iRobot Roomba 690
12.	Ring alarm	Ring Alarm Home Security System
13.	TP-Link plug	TP-Link HS-110
14.	D-Link plug	D-Link DSP-W215
15.	D-Link siren	D-Link DCH-S220
16.	TP-Link light bulb	TP-Link LB-130
17.	SmartThings plug	Samsung SmartThings Outlet (2016 model)
18.	Kwikset lock	Kwikset SmartCode 910
19.	Blossom sprinkler	Blossom 7 Smart Watering Controller

TABLE II. THE SET OF SMART HOME DEVICES CONSIDERED IN THIS PAPER. DEVICES HIGHLIGHTED IN GREEN ARE AMONG THE MOST POPULAR ON AMAZON.

We consider two different types of adversaries: a *WAN sniffer* and a *Wi-Fi sniffer*. The adversaries differ in terms of the vantage point where traffic is inspected and, thus, what information is available to the adversary. The WAN sniffer monitors network traffic in the communication between the home router and the ISP network (or beyond) [10]–[13]. This adversary can inspect the IP headers of all packets, but does not know the device MAC addresses to identify which device has sent the traffic. We assume a standard home network that uses NAT: all traffic from the home is multiplexed onto the router's IP address. Examples of such adversaries include intelligence agencies and ISPs. The Wi-Fi sniffer monitors encrypted IEEE 802.11 traffic, and has not been as widely studied [10], [23]. We assume that the Wi-Fi sniffer does not know the WPA2 key, and thus only has access to the information sent in clear text—the MAC addresses, packet lengths, and timing information. As packets are encrypted, the Wi-Fi sniffer does not have access to network and transport layer information.

For both adversaries, we assume that the adversary knows the type of the smart home device that they wish to target and passively monitor. Thus, they can train the system on another device of the same type offline, extract the signature of the device, and perform the detection of the signature on the traffic coming from the smart home they target. We assume that the devices encrypt their communication and thus neither adversary has access to the clear-text communication.

B. Smart Home Environment and Experimental Testbed

Experimental Testbed. Figure 1 depicts our experimental setup, which resembles a typical smart home environment. We experiment with 19 widely-used smart home devices from 16 different vendors (see Table II). We attempted to select a set of devices with a wide range of functionality—from plugs to cameras. They are also widely used: these devices are popular and they come from well-known vendors. The first 12

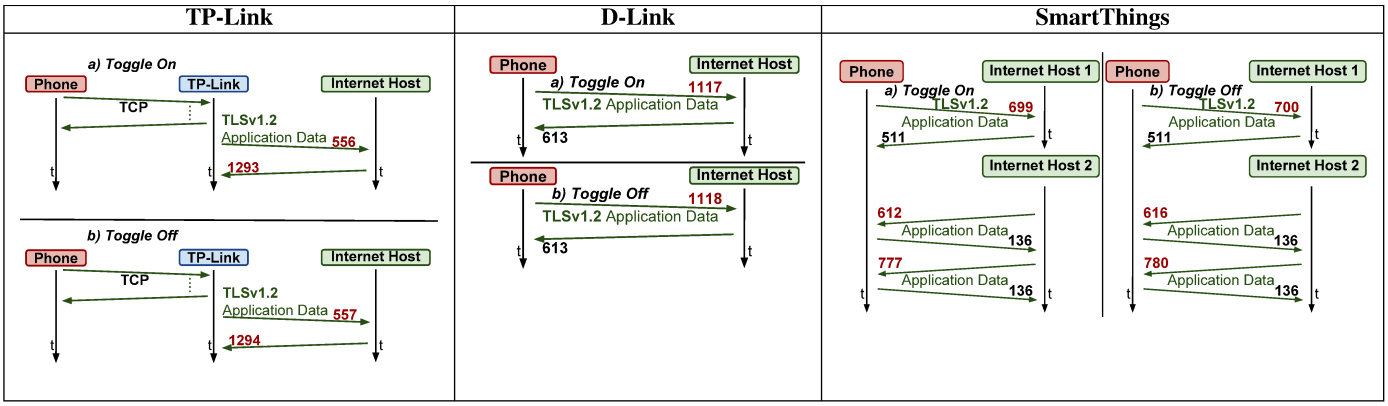

TABLE III. PACKET-LEVEL SIGNATURES OF TP-LINK, D-LINK, AND SMARTTHINGS SMART PLUGS OBSERVABLE BY THE WAN SNIFFER. THE NUMBERS REPRESENT PACKET LENGTHS, WITH RED INDICATING THAT THE LENGTH IS DIFFERENT FOR ON VS. OFF, AND THE ARROWS REPRESENT PACKET DIRECTIONS.

(highlighted in green) are the most popular on Amazon [7]: (1) each received the most reviews for its respective device type and (2) each had at least a 3.5-star rating—they are both popular and of high quality (*e.g.,* the Nest T3007ES and Ecobee3 thermostats are the two most-reviewed with 4-star rating for thermostats). Some devices are connected to the router via Wi-Fi (*e.g.,* the Amazon plug) and others through Ethernet. The latter includes the SmartThings, Sengled, and Hue hubs that relay communication to/from Zigbee/Z-Wave devices: the SmartThings plug, Kwikset doorlock, Sengled light bulb, and Hue light bulb.

Each smart home device in Figure 1 is controlled from the smartphone using its vendor's official Android application. In Figure 1, the smartphone is connected to a local network, which the devices are also connected to. When the smartphone is connected to a remote network, only the Device-Cloud communication is observable in the local network—the smartphone controls a device by communicating with its vendor-specific cloud, and the cloud relays the command to the device. The controller represents the agent that operates the smartphone to control the smart home device of interest. This may be done manually by a human (as in Section III-C) or through software (as in Section IV). Additionally, there are other computing devices (*e.g.,* laptops, tablets, phones) in the house that also generate network traffic, which we refer to as "Background Traffic". The router runs OpenWrt/LEDE [34], a Linux-based OS for network devices, and serves as our vantage point for collecting traffic for experiments. We run `tcpdump` on the router's WAN interface (`eth0`) and local interfaces (`wlan1` and `eth1`) to capture Internet traffic as well as local traffic for all Wi-Fi and Ethernet devices. We use the testbed to generate training data for each device, from which we in turn extract signatures (Section V-A). In Section V-B, the same testbed is used for testing, *i.e.,* to detect the presence of the extracted signatures in traffic generated by all the devices as well as by other computing devices (background traffic).

Communication. Smart home device events may result in communication between three different pairs of devices, as depicted in Figure 1: (1) the smartphone and the smart home device (*Phone-Device*); (2) the smart home device and an Internet host (*Device-Cloud*), and (3) the smartphone and an Internet host (*Phone-Cloud*). The idea behind a passive inference attack is that network traffic on any of these three communication paths may contain unique traffic signatures that can be exploited to infer the occurrence of events.

C. Motivating Case: Smart Plugs

As an illustrative example, let us discuss our manual analysis of 3 smart plugs: the TP-Link plug, the D-Link plug, and the SmartThings plug. Data for the manual analysis was collected using the setup in Figure 1. For each device, we toggled it ON, waited for approximately one minute, and then toggled it OFF. This procedure was repeated for a total of 3 ON and 3 OFF events, separated by one minute in between. Timestamps were manually noted for each event. The PCAP files logged at the router were analyzed using a combination of scripts and manual inspection in Wireshark.

New Observation: Packet Pairs. We identified the traffic flows that occurred immediately after each event and observed that certain pairs of packets with specific lengths and directions followed each ON/OFF event: the same pairs consistently showed up for all events of the same type (*e.g.,* ON), but were slightly different across event types (ON vs. OFF). The pairs were comprised of a *request* packet in one direction, and a *reply* packet in the opposite direction. Intuitively, this makes sense: if the smart home device changes state, this information needs to be sent to (request), and acknowledged by (reply), the cloud server to enable devices that are not connected to the home network to query the smart home device's current state. These exchanges resemble the ball that moves back and forth between players in a game of pingpong, which inspired the name for our software tool.

Table III illustrates the observed packet exchanges. For the TP-Link plug, we observed an exchange of 2 TLS Application Data packets between the plug and an Internet host where the packet lengths were 556 and 1293 when the plug was toggled ON, but 557 and 1294 for OFF. We did not observe any pattern in the D-Link plug's own communication. However, for ON events, the controlling smartphone would always send a request packet of length 1117 to an Internet host and receive a reply packet of length 613. For OFF, these packets were of lengths 1118 and 613, respectively. Similarly for the SmartThings plug, we found consistently occurring packet pairs in the smartphone's communication with two different Internet hosts where the lengths of the request packets were different for ON

and OFF events. Thus, this request-reply pattern can occur in the communication of any of the three pairs: Phone-Device, Device-Cloud, or Phone-Cloud (see Figure 1).

Key Insight. This preliminary analysis indicates that each type of event is uniquely identified by the exchange of pairs (or longer sequences) of packets of specific lengths. To the best of our knowledge, this type of network signature has not been observed before, and we refer to it as a *packet-level signature*.

IV. PingPong Design

The key insight obtained from our manual analysis in Section III-C was that unique sequences of packet lengths (for packet pairs or longer packet sequences) typically follow simple events (*e.g.,* ON vs. OFF) on smart plugs, and can potentially be exploited as signatures to infer these events. This observation motivated us to investigate whether: (1) more smart home devices, and potentially the smartphones that control them as well, exhibit their own unique packet-level sequences following an event, (2) these signatures can be learned and automatically extracted, and (3) they are sufficiently unique to accurately detect events. In this section, we present the design of PingPong—a system that addresses the above questions with a resounding YES!

PingPong automates the collection of training data, extraction of packet-level signatures, and detection of the occurrence of a signature in a network trace. PingPong has two components: (1) training (Section IV-A), and (2) detection (Section IV-B). Figure 2 shows the building blocks and flow of PingPong on the left-hand side, and the TP-Link plug as an example on the right-hand side. We use the latter as a running example throughout this section.

A. Training

The training component is responsible for the extraction of packet-level signatures for a device the attacker wants to profile and attack. It consists of 5 steps (see Figure 2).

Data Collection. The first step towards signature generation is to collect a *training set* for the device. A training set is a network trace (a PCAP file) that contains the network traffic generated by the device and smartphone as a result of events; this trace is accompanied by a text file that contains the set of event timestamps.

PingPong partially automates training set collection by providing a shell script that uses the Android Debug Bridge (`adb`) [9] to issue touch inputs on the smartphone's screen. The script is run on a laptop that acts as the controller in Figure 1. The script is tailored to issue the sequence of touch events corresponding to the events for which a training set is to be generated. For example, if a training set is desired for a smart plug's ON and OFF events, the script issues a touch event at the screen coordinates that correspond to the respective buttons in the user interface of the plug's official Android app. As device vendors may choose arbitrary positions for the buttons in their respective Android applications, and since the feature sets differ from device to device, the script must be manually modified for the given device. The script issues the touch sequence corresponding to each specific event n times,

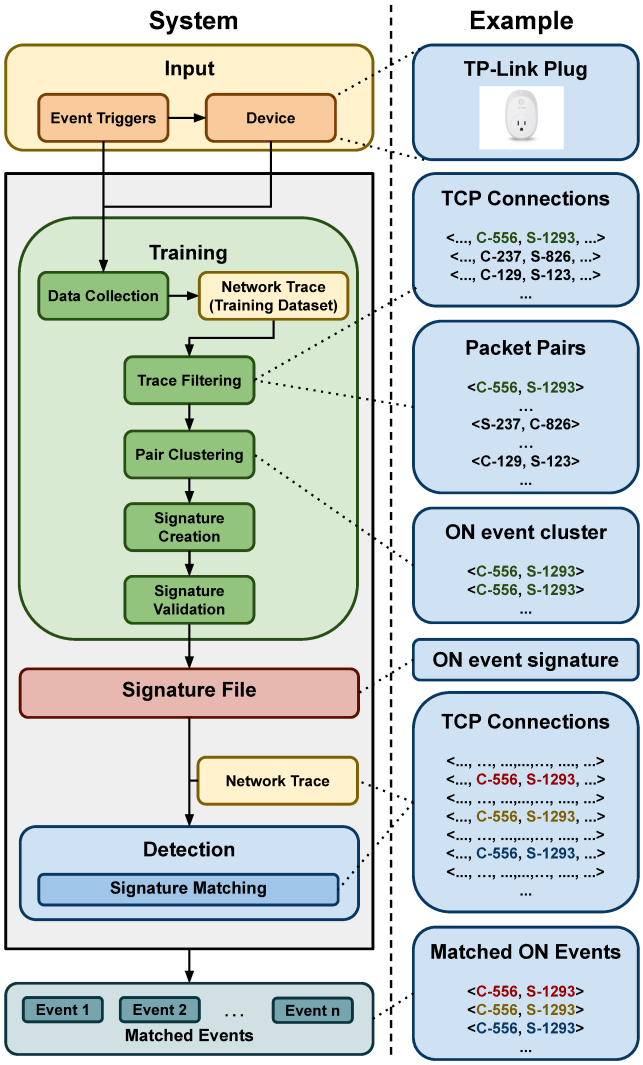

Fig. 2. Left: PingPong Overview. Right: TP-Link plug is used as a running example throughout this section.

each separated by m seconds.[1] The results reported in this paper use $n = 50$ or $n = 100$ depending on the event type (see Section V-A). The script also outputs the current timestamp to a file on the laptop when it issues an event. To collect a training set, we do the following: (1) start `tcpdump` on the router's interfaces; (2) start the script; (3) terminate `tcpdump` after the n-th event has been issued. This leaves us with a set of PCAP files and event timestamps, which constitute our raw training set.

We base our signature generation on the traces collected from the router's local interfaces as they are the vantage points that provide the most comprehensive information: they include both local traffic and Internet traffic. This allows PingPong to exhaustively analyze all network packets generated in the communications between the device, smartphone, and Internet hosts on a per device basis. As signatures are based entirely on packet lengths and directions, signatures present in Internet

[1] We selected $m = 131$ seconds to allow sufficient time such that there is no overlap between events. Section V-G provides more explanation for this choice with respect to other parameters.

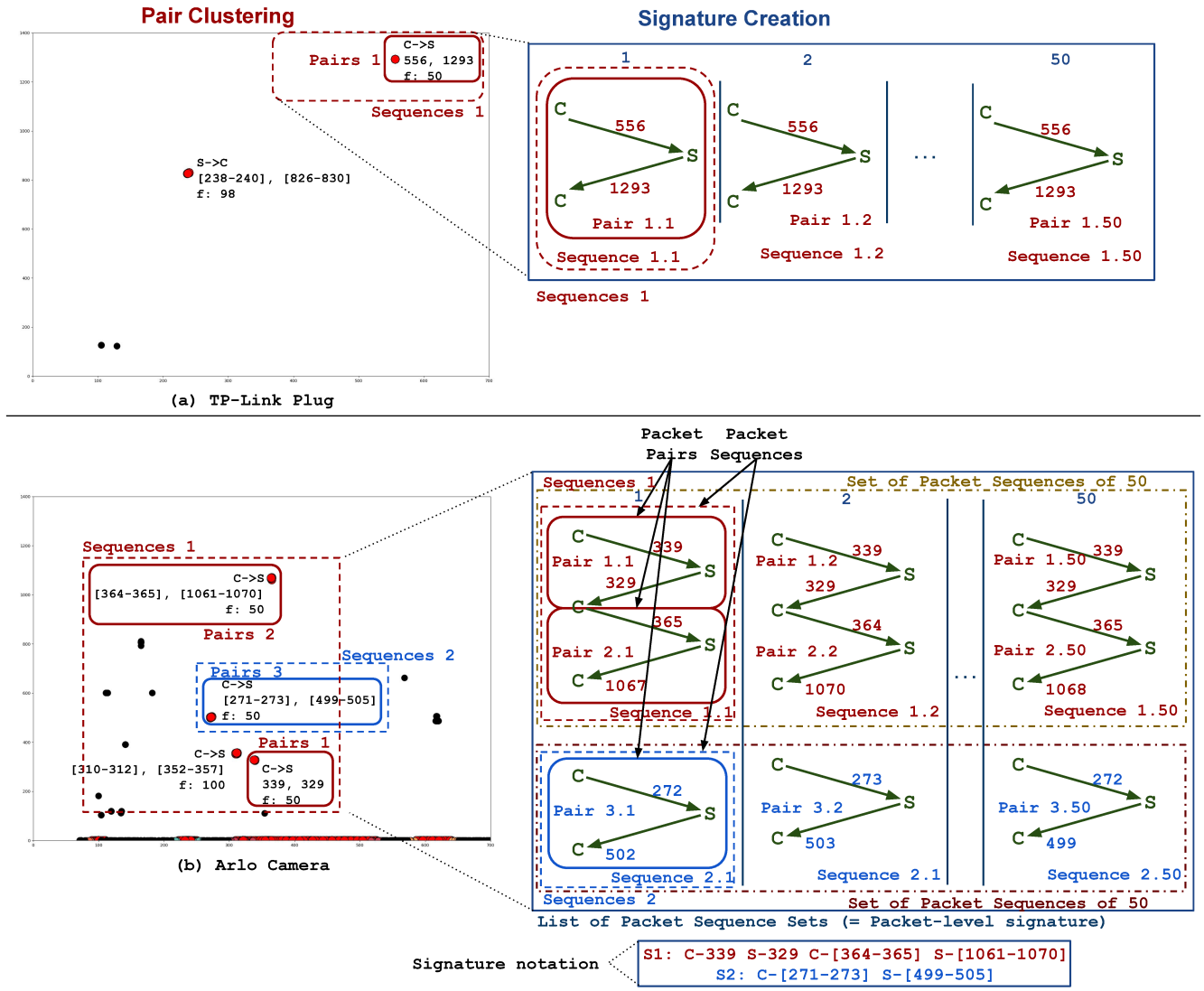

Fig. 3. Pair clustering and signature creation for 2 extreme cases—TP-Link plug has the simplest signature with only 1 pair (see our initial findings in Table III). The Arlo camera has a more complex signature with 1 sequence of 2 pairs and 1 sequence of 1 pair. The left subfigure, in every row, depicts the packet lengths in one packet pair (P_{c_1}, P_{c_2}). Notation: `C->S` means a pair where the first packet's direction is Client-to-Server and the second packet's direction is server-to-client, and vice versa for `S->C`; `f: 50` means that the pair appears in the clustering with a *frequency of 50*; `Signature notation` shows a summary of 2 sets of 50 instances of packet sequences. Example: `C->S 556, 1293 f: 50` means that the pair of packets with lengths 556 (client-to-server) and 1293 (server-to-client) appear 50 times in the cluster.

traffic (*i.e.,* Device-Cloud and Phone-Cloud traffic) are applicable on the WAN side of the router, despite being extracted from traces captured within the local network

Trace Filtering. Next, PINGPONG filters the collected raw training set to discard traffic that is unrelated to a user's operation of a smart home device. All packets, where neither the source nor destination IP matches that of the device or the controlling smartphone, are dropped. Additionally, all packets that do not lie within a time window t after each timestamped event are discarded. We selected $t = 15$ seconds to allow sufficient time for all network traffic related to the event to complete. We also performed a sensitivity study that confirmed this was a conservative choice (see Section V-G).

PINGPONG next reassembles all TCP connections in the filtered trace. Given the set of reassembled TCP connections, we now turn our attention to the packets P that carry the TCP payload. For TLS connections, P is limited further to only be the subset of packets that are labeled as "Application Data" in the unencrypted TLS record header [41]. By only considering packets in P, we ensure that the inherently unpredictable control packets (*e.g.,* TCP ACKs and TLS key negotiation) do not become part of the signature as P only contains packets with application layer payload.

We next construct the set P' by forming *packet pairs* from the packets in P (see Definition IV.1). This is motivated by the following observation: the deterministic sequence of packets that make up packet-level signatures often stem from a *request-reply exchange* between the device, smartphones, and some Internet hosts (see Section III-C). Furthermore, since a packet pair is the simplest possible pattern, and since longer patterns (*i.e.,* packet sequences—see Definition IV.2) can be reconstructed from packet pairs, we look for these packet pairs in the training set. For the TP-Link plug example in Figure 2,

PINGPONG reassembles <..., C-556, S-1293, ...>, <..., C-237, S-826, ...>, etc. as TCP connections. Then, PINGPONG extracts <C-556, S-1293>, <C-237, S-826>, etc. as packet pairs.

Definition IV.1. ***Packet Pair.*** *Let P_c be the ordered set of packets with TCP payload that belong to TCP connection c, let P_{c_i} denote the i-th packet in P_c, and let C and S each denote client-to-server and server-to-client packet directions, respectively, where a client is a smartphone or a device. A packet pair p is then $p = (C{-}P_{c_i}, S{-}P_{c_{i+1}})$ or $p = (S{-}P_{c_i}, C{-}P_{c_{i+1}})$* iff *$P_{c_i}$ and $P_{c_{i+1}}$ go in opposite directions. Otherwise, if P_{c_i} and $P_{c_{i+1}}$ go in the same direction, or if P_{c_i} is the last packet in P_c, the packet pair $p = (C{-}P_{c_i}, \text{nil})$ or $p = (S{-}P_{c_i}, \text{nil})$ is formed, and packet $P_{c_{i+1}}$, if any, is paired with packet $P_{c_{i+2}}$.*

Pair Clustering. After forming a set of packet pairs, relevant packet pairs (*i.e.,* those that consistently occur after an event) must next be separated from irrelevant ones. This selection also needs to take into account that the potentially relevant packet pairs may have slight variations in lengths. Since we do not know in advance the packet lengths in the pairs, we use an unsupervised learning algorithm: DBSCAN [22].

DBSCAN is provided with a distance function for comparing the similarity of two packet pairs, say p_1 and p_2. The distance is maximal if the packet directions are different, *e.g.,* if p_1 is comprised of a packet going from a local device to an Internet host followed by a packet going from an Internet host to a local device, while p_2 is comprised of a packet going from an Internet host to a local device followed by a packet going from a local device to an Internet host. If the packet directions match, the distance is simply the Euclidean distance between the two pairs, *i.e.,* $\sqrt{(p_1^1 - p_2^1)^2 + (p_1^2 - p_2^2)^2}$, where p_j^i refers to the packet length of the i-th element of pair j. DBSCAN's parameters are ϵ and `minPts`, which specify the neighborhood radius to consider when determining core points and the minimum number of points in that neighborhood for a point to become a core point, respectively. We choose $\epsilon = 10$ and `minPts` $= \lfloor n - 0.1n \rfloor$, where n is the total number of events. We allow a slack of $0.1n$ to `minPts` to take into account that event-related traffic could occasionally have missing pairs, for example caused by the phone app not responding to some of the automated events. We study the sensitivity of PINGPONG parameter values in Section V-G.

Figure 3(a) illustrates the pair clustering process for TP-Link plug. There are 50 ON and 50 OFF actions, and there must be at least 45 ($n = 50 \implies$ `minPts` $= \lfloor 50 - 0.1 \times 50 \rfloor = 45$) similar packet pairs to form a cluster. Two clusters are formed among the data points, *i.e.,* those with frequencies `f: 50` and `f: 98`, respectively. Since these two clusters contain similar packet pairs that occur during *t*, this indicates with high confidence that the packets are related to the event.

Signature Creation. Given the output produced by DBSCAN, PINGPONG next drops all clusters whose frequencies are not in the interval $[\lfloor n - 0.1n \rfloor, \lceil n + 0.1n \rceil]$ in order to only include in the signature those clusters whose frequencies align closely with the number of events n. Intuitively, this step is to deal with *chatty devices*, namely devices that communicate continuously/periodically while not generating events. Consequently, PINGPONG only picks the cluster `Pairs 1` with frequency 50 for the TP-Link plug example in Figure 3 as a signature candidate since 50 is in $[\lfloor n - 0.1n \rfloor, \lceil n + 0.1n \rceil] = [45, 55]$ when $n = 50$, whereas 98 is not. As a pair from this cluster occurs exactly once during *t*, there is high confidence that the pair is related to the event.

PINGPONG next attempts to concatenate packet pairs in the clusters so as to reassemble the longest *packet sequences* possible (see Definition IV.2), which increases the odds that a signature is unique. Naturally, packet pair concatenation is only performed when a device has more than one cluster. This is the case for the Arlo camera, but not the TP-Link plug. Packet pairs in clusters x and y are concatenated *iff* for each packet pair p_x in x, there exists a packet pair p_y in y such that p_x and p_y occurred consecutively in the same TCP connection. If there are more pairs in y than in x, the extra pairs of y are simply dropped. The result is referred to as a *set of packet sequences* (see Definition IV.3) and is considered for further concatenation with other clusters if possible.

Definition IV.2. ***Packet Sequence.*** *A packet sequence s is formed by joining packet pairs p_1 and p_2* iff *p_1 and p_2 are both in P_c (same TCP connection)* and *the packets in p_1 occur immediately before the packets in p_2 in P_c. Note that the packet sequence s resulting from joining p_1 and p_2 can be of length 2, 3, or 4, depending on whether or not the second element of p_1 and/or p_2 is nil.*

Definition IV.3. ***Set of Packet Sequences.*** *A set of packet sequences S is a set of similar packet sequences. Two packet sequences s_1 and s_2 are similar and thus belong to the same set S* iff *they (1) contain the same number of packets, (2) the packets at corresponding indices of s_1 and s_2 go in the same direction, and (3) the Euclidean distance between the packet lengths at corresponding indices of s_1 and s_2 is below a threshold—packet lengths in packet sequences inherit the slight variations that stem from packet pairs.*

Figure 3(b) shows how pair clustering produces 3 clusters around the pairs <C-339, S-329> (*i.e.,* cluster `Pairs 1`), <C-[364-365], S-[1061-1070]> (*i.e.,* cluster `Pairs 2`), and <C-[271-273], S-[499-505]> (*i.e.,* cluster `Pairs 3`) for the Arlo camera. The notation `C-[`$l_1 - l_2$`]` or `S-[`$l_1 - l_2$`]` indicates that the packet length may vary in the range between l_1 and l_2. Each pair from cluster `Pairs 1` and each pair from cluster `Pairs 2` are then concatenated into a sequence in `Sequences 1` (a set of packet sequences) as they appear consecutively in the same TCP connection, *i.e.,* `Pair 1.1` with `Pair 2.1`, `Pair 1.2` with `Pair 2.2`, ..., `Pair 1.50` with `Pair 2.50`. The cluster `Pairs 3` is finalized as the set `Sequences 2` as its members appear in different TCP connections than the members of `Sequences 1`. Thus, the initial 3 clusters of packet pairs are reduced to 2 sets of packet sequences. For the TP-Link plug, no concatenation is performed since there is only a single cluster, `Pairs 1`, which is finalized as the set `Sequences 1`.

Finally, PINGPONG sorts the sets of packet sequences based on the timing of the sets' members to form a *list of packet sequence sets* (see Definition IV.4). For example, for the Arlo camera, this step produces a list in which the set `Sequences 1` precedes the set `Sequences 2` because there is always a packet sequence in `Sequences 1` that precedes a packet sequence in `Sequences 2`. The purpose of this step is to make the temporal order of the sets of packet sequences part of the final signature. If no such order can be established, the set with the shorter packet sequences is discarded. Manual inspection of some devices suggests that the earlier sequence will often be the control command sent from an Internet host followed by the device's acknowledgment of the command, while the later sequence will stem from the device initiating communication with some other Internet host to inform that host about its change in status.

Definition IV.4. ***List of Packet Sequence Sets.*** *A list of packet sequence sets is a list that contains sets of packet sequences that are sorted based on the occurrence of the set members in time. Set S_x goes before set S_y* iff *for each sequence s_x in S_x, there exists a sequence s_y in S_y that occurred after s_x within t.*

Signature Validation. Before finalizing the signature, we validate it by running the detection algorithm (see Section IV-B) against the raw training set that was used to generate the signature. If PINGPONG detects at most n events, and the timestamps of detected events match the timestamps for events recorded during training, the signature is finalized as a valid *packet-level signature* (see Definition IV.5) and stored in a signature file. A signature can fail this check if it detects more events than the actual number of events in the training set (*i.e.,* false positives). This can happen if the packet sequences in the signature frequently appear outside t.

Definition IV.5. ***Packet-level Signature.*** *A packet-level signature is then a list of packet sequence sets that has been validated and finalized.*

Signature File. A signature file stores a packet-level signature. Figure 3 shows that the TP-Link plug signature consists of 50 instances of packet sequences in set `Sequences 1`, but only one instance will be used during detection since all 50 are identical. Figure 3(b) shows the signature file (on the right-hand side) for the Arlo camera. It is a list that orders the two sets of packet sequences, `Sequences 1` and `Sequences 2`. `Sequences 1` is comprised of 50 packet sequences, each comprised of two packet pairs. `Sequences 2` is comprised of another 50 packet sequences, each comprised of a single packet pair. Since the sequences vary slightly in each set, all unique variations are considered during detection.

B. Detection

For signature detection, PINGPONG treats a network trace as a stream of packets and presents each individual packet to a set of state machines. A state machine is maintained for each packet sequence of the signature for each flow, i.e., TCP connection for the WAN sniffer or layer-2 flow for the Wi-Fi sniffer. A packet is only presented to the state machines associated with the flow that the packet pertains to. A state machine advances to its next state if the packet matches the next packet (in terms of length and direction) in the modeled packet sequence. The state machines respond differently to packets that do not match the expected next packet depending on whether detection is applied at layer-2 or layer-3. For layer-2, such packets are simply ignored, whereas for layer-3 such packets cause the state machine to discard the current partial match. When a state machine reaches its terminal state, the packet sequence match is reported to a secondary module. This module waits for a packet sequence match for each packet sequence of the signature and verifies the inter-sequence timing constraints before finally declaring a signature match. Please see Appendix A in [48] for a more detailed explanation of the detection.

V. EVALUATION

In this section, we present the evaluation of PINGPONG. In Section V-A, we show that PINGPONG automatically extracted event signatures for 18 devices as summarized in Table IV—11 of which are the most popular devices on Amazon (see Table II). In Section V-B, we used the extracted signatures to detect events in a trace collected from a realistic experiment on our smart home testbed. Section V-C discusses the results of negative control experiments: it demonstrates the uniqueness of the PINGPONG signatures in large (*i.e.,* with hundreds of millions of packets), publicly available, packet traces from smart home and office environments. Section V-D discusses the results of our experiments when devices are triggered remotely from a smartphone and via a home automation service. Section V-E shows the uniqueness of signatures for devices from the same vendor. Section V-F discusses our findings when we used PINGPONG to extract signatures from a public dataset [39]. Finally, Section V-G discusses the selection and sensitivity of the parameters used to extract signatures.

A. Extracting Signatures from Smart Home Devices

Training Dataset. In order to evaluate the generalizability of packet-level signatures, we first used PINGPONG to automate the collection of training sets (see Section IV-A) for all 19 smart home devices (see Table II). Training sets were collected for every device under test, individually without any background traffic (see Figure 1). The automation script generated a total of 100 events for the device. For events with binary values, the script generated $n = 50$ events for each event type (*e.g.,* 50 ON and 50 OFF events). For events with continuous values, the script generated $n = 100$ events (*e.g.,* 100 intensity events for the Sengled light bulb).

Results Summary. For each training set, we used PINGPONG to extract packet-level signatures (see Section IV-A) for each event type of the respective device. In summary, PINGPONG extracted signatures from 18 devices (see Table IV). The signatures span a wide range of event types: binary (*e.g.,* ON/OFF) and non-binary (*e.g.,* light bulb intensity, color, etc.). Similar to our manual observation described in Section III-C, we again see that these events are identifiable by the request-reply pattern.

Table IV presents the signatures that PINGPONG identified. Each line in a signature cell represents a packet sequence set, and the vertical positioning of these lines reflects the ordering

Device	Event	Signature	Communication	Duration (ms) Min./Avg./Max.	Matching (Per 100 Events)			
					WAN Snif.	FPR	Wi-Fi Snif.	FPR
Plugs								
Amazon plug	ON	**S1:** S-[443-445] **S2:** C-1099 S-235	Device-Cloud	1,232 / 2,465 / 4,537	98	0	99	0
	OFF	**S1:** S-[444-446] **S2:** C-1179 S-235 **S3:** C-1514 C-103 S-235						
WeMo plug	ON/OFF	**S1:** PH-259 PH-475 D-246	Phone-Device	33 / 42 / 134	-	-	100	0
WeMo Insight plug	ON/OFF	**S1:** PH-259 PH-475 D-246	Phone-Device	32 / 39 / 97	-	-	99	0
TP-Link plug	ON	**S1:** C-556 S-1293	Device-Cloud	75 / 85 / 204	99	0	-	-
	OFF	**S1:** C-557 S-[1294-1295]						
	ON	**S1:** PH-112 D-115 **S2:** C-556 S-1293	Phone-Device & Device-Cloud	225 / 325 / 3,328	-	-	99	0
	ON	**S1:** PH-112 D-115 **S2:** C-557 S-[1294-1295]						
D-Link plug	ON/OFF	**S1:** S-91 S-1227 C-784 **S2:** C-1052 S-647	Device-Cloud	4 / 1,194 / 8,060	95	0	95	0
	ON	**S1:** C-[1109-1123] S-613	Phone-Cloud	35 / 41 / 176	98	0	98	0
	OFF	**S1:** C-[1110-1124] S-613						
SmartThings plug	ON	**S1:** C-699 S-511 **S2:** S-777 C-136	Phone-Cloud	335 / 537 / 2,223	92	0	92	0
	OFF	**S1:** C-700 S-511 **S2:** S-780 C-136						
Light Bulbs								
Sengled light bulb	ON	**S1:** S-[217-218] C-[209-210] **S2:** C-430 **S3:** C-466	Device-Cloud	4,304 / 6,238 / 8,145	97	0	-	-
	OFF	**S1:** S-[217-218] C-[209-210] **S2:** C-430 **S3:** C-465						
	ON	**S1:** C-211 S-1063 **S2:** S-1277	Phone-Cloud	4,375 / 6,356 / 9,132	93	0	97	0
	OFF	**S1:** C-211 S-1063 S-1276						
	Intensity	**S1:** S-[216-220] C-[208-210]	Device-Cloud	16 / 74 / 824	99	2	-	-
	Intensity	**S1:** C-[215-217] S-[1275-1277]	Phone-Cloud	3,916 / 5,573 / 7,171	99	0	99	0
Hue light bulb	ON	**S1:** C-364 **S2:** D-88	Device-Cloud & Phone-Device	11,019 / 12,787 / 14,353	-	-	-	-
	OFF	**S1:** C-365 **S2:** D-88						
TP-Link light bulb	ON	**S1:** PH-198 D-227	Phone-Device	8 / 77 / 148	-	-	100	4
	OFF	**S1:** PH-198 D-244						
	Intensity	**S1:** PH-[240-242] D-[287-289]	Phone-Device	7 / 84 / 212	-	-	100	0
	Color	**S1:** PH-317 D-287	Phone-Device	6 / 89 / 174	-	-	100	0
Thermostats								
Nest thermostat	Fan ON	**S1:** C-[891-894] S-[830-834]	Phone-Cloud	91 / 111 / 1,072	93	0	93	1
	Fan OFF	**S1:** C-[858-860] S-[829-834]						
Ecobee thermostat	HVAC Auto	**S1:** S-1300 C-640	Phone-Cloud	121 / 229 / 667	100	0	99	0
	HVAC OFF	**S1:** C-1299 C-640						
	Fan ON	**S1:** S-1387 C-640	Phone-Cloud	117 / 232 / 1,776	100	0	100	0
	Fan Auto	**S1:** C-1389 C-640						
Sprinklers								
Rachio sprinkler	Quick Run	**S1:** S-267 C-155	Device-Cloud	1,972 / 2,180 / 2,450	100	0	100	0
	Stop	**S1:** C-496 C-155 C-395						
	Standby/Active	**S1:** S-299 C-155 C-395	Device-Cloud	276 / 690 / 2,538	100	0	100	0
Blossom sprinkler	Quick Run	**S1:** C-326 **S2:** C-177 S-505	Device-Cloud	701 / 3,470 / 8,431	96	0	96	0
	Stop	**S1:** C-326 **S2:** C-177 S-458 **S3:** C-238 C-56 S-388						
	Quick Run	**S1:** C-649 S-459 C-574 S-507 **S2:** S-[135-139]	Phone-Cloud	70 / 956 / 3,337	93	0	93	0
	Stop	**S1:** C-617 S-431						
	Hibernate	**S1:** C-621 S-493	Phone-Cloud	121 / 494 / 1,798	95	0	93	0
	Active	**S1:** C-622 S-494 **S2:** S-599 C-566 S-554 C-566						

Device	Event	Signature	Communication	Duration (ms) Min./Avg./Max.	Matching (Per 100 Events)			
					WAN Snif.	FPR	Wi-Fi Snif.	FPR
Home Security Devices								
Ring alarm	Arm	**S1:** S-99 S-254 C-99 S-[181-183] C-99	Device-Cloud	275 / 410 / 605	98	0	95	0
	Disarm	**S1:** S-99 S-255 C-99 S-[181-183] C-99						
Arlo camera	Stream ON	**S1:** C-[338-339] S-[326-329] C-[364-365] S-[1061-1070] **S2:** C-[271-273] S-[499-505]	Phone-Cloud	46 / 78 / 194	99	2	98	3
	Stream OFF	**S1:** C-[445-449] S-442						
D-Link siren	ON	**S1:** C-1076 S-593	Phone-Cloud	36 / 37 / 65	100	0	98	0
	OFF	**S1:** C-1023 S-613						
Kwikset door lock	Lock	**S1:** C-699 S-511 **S2:** S-639 C-136	Phone-Cloud	173 / 395 / 2,874	100	0	100	0
	Unlock	**S1:** C-701 S-511 **S2:** S-647 C-136						
Others								
Roomba robot	Clean	**S1:** S-[1014-1015] C-105 S-432 C-105	Phone-Cloud	123 / 2,038 / 5,418	91	0	94	0
	Back-to-station	**S1:** S-440 C-105 S-[1018-1024] C-105						
				Average	97.05	0.18	97.48	0.32

TABLE IV. SMART HOME DEVICES FOUND TO EXHIBIT PHONE-CLOUD, DEVICE-CLOUD, AND PHONE-DEVICE SIGNATURES. PREFIX PH INDICATES PHONE-TO-DEVICE DIRECTION AND PREFIX D INDICATES DEVICE-TO-PHONE DIRECTION IN SIGNATURE COLUMN.

of the packet sequence sets in the signature (see Section IV-A for the notation).

PINGPONG performed well in extracting signatures: it has successfully extracted packet-level signatures that are observable in the device's Phone-Cloud, Device-Cloud, and Phone-Device communications (see Table IV). Although the traffic is typically encrypted using TLSv1.2, the event still manifests itself in the form of a packet-level signature in the Phone-Cloud or Device-Cloud communication. PINGPONG also extracted signatures from the Phone-Device communication for some of the devices. These signatures are extracted typically from unencrypted local TCP/HTTP communication between the smartphone and the device.

Smart Plugs. PINGPONG extracted signatures from all 6 plugs: the Amazon, WeMo, WeMo Insight, TP-Link, D-Link, and SmartThings plugs. The Amazon, D-Link, and SmartThings plugs have signatures in the Phone-Cloud or Device-Cloud communication, or both. The TP-Link plug has signatures in both the Device-Cloud and Phone-Device communications. Both the WeMo and WeMo Insight plugs have signatures in the Phone-Device communication. In general, the signatures allow us to differentiate ON from OFF except for the WeMo, WeMo Insight, TP-Link plugs' Phone-Device communication, and D-Link plug's Device-Cloud communication (see Table IV).

Light Bulbs. PINGPONG extracted signatures from 3 light bulbs: the Sengled, Hue, and TP-Link light bulbs. The Sengled light bulb has signatures in both the Phone-Cloud and Device-Cloud communications. The Hue light bulb has signatures in both Device-Cloud and Phone-Device communications. The TP-Link light bulb has signatures only in the Phone-Device communication. Table IV shows that PINGPONG also extracted signatures for events other than ON and OFF: Intensity and Color.

Thermostats. PINGPONG extracted signatures for both the Nest and Ecobee thermostats. Both thermostats have Phone-Cloud signatures. The signatures allow us to differentiate Fan ON/OFF/Auto events. The Ecobee thermostat's signatures also leak information about its HVAC Auto/OFF events.

Sprinklers. PINGPONG extracted signatures from both the Rachio sprinkler and Blossom sprinkler. Both sprinklers have signatures in both the Device-Cloud and Phone-Cloud communications. The signatures allow us to differentiate Quick Run/Stop and Standby/Hibernate/Active events.

Home Security Devices. A highlight is that PINGPONG extracted signatures from home security devices. Notably, the Ring alarm has signatures that allow us to differentiate Arm/Disarm events in the Device-Cloud communication. The Arlo camera has signatures for Stream ON/OFF events, the D-Link siren for ON/OFF events, and the Kwikset lock for Lock/Unlock events in the Phone-Cloud communication.

Roomba robot. Finally, PINGPONG also extracted signatures from the Roomba robot in the Phone-Cloud communication. These signatures allow us to differentiate Clean/Back-to-station events.

Signature Validity. Recall that signature validation rejects a signature candidate whose sequences are present not only in the time window t, but also during the subsequent idle period (see Section IV-A). We saw such a signature candidate for one device, namely the LiFX light bulb. PINGPONG captured a signature candidate that is present also in the idle period of the TCP communication and then rejected the signature during the validation phase. Manual inspection revealed that the LiFX light bulb uses unidirectional UDP communication (*i.e.,* no request-reply pattern) for events.

B. Smart Home Testbed Experiment

Testing Dataset. To evaluate the effectiveness of packet-level signatures in detecting events, we collected a separate set of network traces and used PINGPONG to perform detection on them. We used the setup in Section III-B to collect one dataset for every device. Our smart home setup consists of 13

of the smart home devices presented in Table II: the WeMo plug, WeMo Insight plug, Hue light bulb, LiFX light bulb, Nest thermostat, Arlo camera, TP-Link plug, D-Link plug, D-Link siren, TP-Link light bulb, SmartThings plug, Blossom sprinkler, and Kwikset lock. This fixed set of 13 devices was our initial setup—it gives us the flexibility to test additional devices without changing the smart home setup and needing to rerun all the experiments, yet still includes a variety of devices that generate background traffic. While collecting a dataset, we triggered events for the device under test. At the same time, we also connected the other 12 devices and turned them ON before we started the experiment—this allows the other devices to generate network traffic as they communicate with their cloud servers. However, we did not trigger events for these other devices. For the other 6 devices (the Amazon plug, Sengled light bulb, Ecobee thermostat, Rachio sprinkler, Roomba robot, and Ring alarm), we triggered events for the device under test while having all the 13 devices turned on. To generate additional background traffic as depicted in Figure 1, we set up 3 general purpose devices: a Motorola Moto g[6] phone that would play a YouTube video playlist, a Google Nexus 5 phone that would play a Spotify song playlist, and an Apple MacBook Air that would randomly browse top 10 websites [4] every 10-500 seconds. We used this setup to emulate the network traffic from a smart home with many active devices.

Results Summary. Table IV presents the summary of our results (see column "**Matching**"). We collected a dataset with 100 events for every type of event—for binary events (*e.g.,* ON/OFF), we triggered 50 for each value. We performed the detection for both the WAN sniffer and Wi-Fi sniffer adversaries. For both adversaries, we have a negligible False Positive Rate (FPR) of 0.25 (0.18 for the WAN sniffer and 0.32 for the Wi-Fi sniffer) per 100 events for every event type.

C. Negative Control Experiment

If the packet-level signatures are to be used to detect events in traffic in the wild, they must be sufficiently unique compared to other traffic to avoid generating false positives. We evaluated the uniqueness of the signatures by performing signature detection on 3 datasets. The first 2 datasets serve to evaluate the uniqueness of the signatures among traffic generated by similar devices (*i.e.,* other smart home devices), while the third dataset serves to evaluate the uniqueness of the signatures among traffic generated by general purpose computing devices.

Dataset 1: UNSW Smart Home Traffic Dataset. The first dataset [45] contains network traces for 26 smart home devices that are *different* from the devices that we generated signatures for. The list can be found in [50]. The dataset is a collection of 19 PCAP files, with a total size of 12.5GB and a total of 23,013,502 packets.

Dataset 2: YourThings Smart Home Traffic Dataset. The second dataset [5], [6] contains network traces for 45 smart home devices. The dataset is a collection of 2,880 PCAP files, with a total size of 270.3GB and 407,851,830 packets. There are 3 common devices present in both YourThings and our set of 18 devices: the WeMo plug, Roomba robot, and TP-Link light bulb.

Dataset 3: UNB Simulated Office-Space Traffic Dataset. The third dataset is the Monday trace of the CICIDS2017 dataset [43]. It contains simulated network traffic for an office space with two servers and 10 laptops/desktops with diverse operating systems. The dataset we used is a single PCAP file of 10.82GB, with a total of 11,709,971 packets observed at the WAN interface.

False Positives. For datasets 1 and 3, we performed signature detection for all devices. For dataset 2, we only performed signature detection for the 15 of our devices that are *not* present in YourThings to avoid the potential for true positives. We used WAN sniffer detection for devices with Phone-Cloud and Device-Cloud signatures, and Wi-Fi sniffer detection for all devices.

WAN Sniffer. There were no false positives across 23,013,502 packets in dataset 1, 1 false positive for the Sengled light bulb across 407,851,830 packets in dataset 2, and 1 false positive for the Nest thermostat across 11,709,971 packets in dataset 3.

Wi-Fi Sniffer. PINGPONG detected some false positives due to its more relaxed matching strategy (see Section IV-B). The results show that the extracted packet-level signatures are unique: the average FPR is 11 false positives per signature across a total of 442,575,303 packets from all three datasets (*i.e.,* an average of 1 false positive per 40 million packets).

Further analysis showed that signatures comprised of a single packet pair (*e.g.,* the D-Link plug's Phone-Cloud signatures that only have one request and one reply packet) contributed the most to the average FPR—FPR is primarily impacted by signature length, not device type. Five 3-packet signatures generated 5, 7, 16, 26, and 33 false positives, while one 4-packet signature generated 2 false positives. There were also three outliers: two 4-packet signatures generated 46 and 33 false positives, and a 6-packet signature generated 18 false positives. This anomaly was due to PINGPONG using the range-based matching strategy for these signatures (see Appendix A in [48]). Furthermore, the average of the packet lengths for the signatures that generated false positives is less than 600 bytes: the packet lengths distribution for our negative datasets shows that there are significantly more shorter packets than longer packets.

D. Events Triggered Remotely

Our main dataset, collected using our testbed (see Section V-A), contains events triggered by a smartphone that is part of the local network (*i.e.,* the smart home testbed). However, smart home devices can also be controlled remotely, using home automation frameworks or a remote smartphone. In this section, we summarize our results for these scenarios. Please see Appendix B in [48] for details.

Home Automation Experiment (IFTTT). We integrated IFTTT into our existing infrastructure for triggering device events. IFTTT provides support for 13 out of our 18 devices: no support was provided at the time of the experiment for the Amazon plug, Blossom sprinkler, Roomba robot, Ring alarm, and Nest thermostat. The main finding is that, from the supported 13 devices, PINGPONG successfully extracted Device-Cloud signatures for 9 devices and 12 event types.

Comparison of Device-Cloud Signatures. We also compared the Device-Cloud signatures of the TP-Link plug, the D-Link plug, and the Rachio sprinkler. Our results show that the majority of Device-Cloud signatures are the same or very similar across 3 different ways of triggering the devices: local smartphone, remote smartphone, and IFTTT.

E. Devices from the Same Vendor

Since the signatures reflect protocol behavior, a natural question to ask is whether devices from the same vendor, which probably run similar protocols, have the same signature. In our testbed experiment, we had already extracted signatures from 2 TP-Link devices: the TP-Link plug and TP-Link light bulb (see Table IV). We also acquired, and experimented with, 4 additional devices from TP-Link. We defer the detailed results to Table X in [48]. In summary, we found that packet-level signatures have some similarities (*e.g.,* the TP-Link two-outlet plug and TP-Link power strip have similar functionality and have packet lengths 1412B and 88B). However, they are still distinct across different device models and event types, even for devices with similar functionality (*e.g.,* the TP-Link plug, TP-Link two-outlet plug, and TP-Link power strip).

F. Public Dataset Experiment

In this section, we apply the PINGPONG methodology to a state-of-the-art, publicly available IoT dataset: the Mon(IoT)r dataset [39]. First, we show that PINGPONG successfully extracted signatures from new devices in this dataset, thus validating the generality of the methodology and expanding our coverage of devices. Then, we compare the signatures extracted from the Mon(IoT)r dataset to those extracted from our testbed dataset, for devices that were present in both.

The Mon(IoT)r Dataset. The Mon(IoT)r dataset [39] contains network traces from 55 distinct IoT devices.[2] Each PCAP file in the dataset contains traffic observed for a single device during a short timeframe surrounding a single event on that device. Moreover, the authors provide timestamps for when they performed each event. As a result, we can merge all PCAP files for each device and event type combination into a single PCAP file, and directly apply PINGPONG to extract signatures, similarly to how we extracted signatures from the training set we collected using our testbed.

We only considered a subset of the 55 devices in the Mon(IoT)r dataset, due to a combination of limitations of the dataset and of our methodology. In particular, we did not apply PINGPONG to the following groups of devices in the Mon(IoT)r dataset: (1) 3 devices with nearly all PCAP files empty; (2) 6 devices with a limited number (three or less) of event samples;[3] and (3) 13 devices that only communicate via UDP (PINGPONG's current implementation only considers TCP traffic). Next, we report results from applying PINGPONG to the remaining 33 devices in the Mon(IoT)r dataset. Out of those, 26 are exclusive to the Mon(IoT)r dataset, while seven are common across the Mon(IoT)r dataset and our testbed dataset.

Devices only in the Mon(IoT)r Dataset. We ran PINGPONG's signature extraction on the traces from the 26 new devices from the Mon(IoT)r dataset. PINGPONG successfully extracted signatures for 21 devices and we summarize those signatures in Table V. Some of these devices provide similar functionality as those in our testbed dataset (*e.g.,* bulbs, cameras). Interestingly, we were also able to successfully extract signatures for many new types of devices that we did not have access to during our testbed experiments. Examples include voice-activated devices, smart TVs, and even a fridge. This validates the generality of the PINGPONG methodology and greatly expands our coverage of devices.

There were also 5, out of 26, new devices that PINGPONG originally appeared to not extract signatures from. However, upon closer inspection of their PCAP files and PINGPONG's output logs, we observed that those devices did actually exhibit a new type of signature that we had not previously encountered in our testbed experiments: a sequence of packet pairs with the exact same pair of packet lengths for the same event. The default configuration of PINGPONG would have discarded the clusters of these packet pairs during the signature creation of the training phase (see Section IV-A), because the number of occurrences of these pairs is higher than (in fact a multiple of) the number of events. However, based on this intuitive observation, PINGPONG can easily be adapted to extract those signatures as well: it can take into account the timing of packet pairs in the same cluster instead of only across different clusters, and concatenate them into longer sequences. We note that these frequent packet pairs can be either new signatures for new devices, or can be due to periodic background communication. Unfortunately, the Mon(IoT)r dataset does not include network traces for idle periods (where no events are generated), thus we cannot confirm or reject this hypothesis.

Common Devices. We next report our findings for devices that are present in both the Mon(IoT)r dataset and in our own testbed dataset, referred to as common devices. There were already 6 common devices across the 2 datasets, and we acquired an additional device after consulting with the authors of the paper: the Blink camera. We excluded 2 common devices: (1) the Nest thermostat as it was tested for different event types; and (2) the Hue light bulb as it has a unique signature that PINGPONG cannot use to perform matching—it is a combination of Device-Cloud (visible only to the WAN sniffer) and Phone-Device communications (visible only to the Wi-Fi sniffer). Table VI summarizes the results for the 5 remaining common devices. First, we report the complete signatures extracted from each dataset. The signatures reported in Table IV were obtained from data collected throughout 2018. For the WeMo Insight plug and TP-Link plug, we repeated our testbed data collection and signature extraction in December 2019 to facilitate a better comparison of signatures from the same devices across different points in time. Then, we compare the signatures extracted from the two datasets for the common devices: some of the signatures are identical and some are similar. Such a comparison provides more information than simply training on one dataset and testing on the other.

[2]The paper [39] reports results from 81 physical devices, but 26 device models are present in both the US and the UK testbed, thus there are only 55 distinct models.

[3]We consider this to be too few samples to have confidence in the extracted signatures. In contrast, the traces for the remaining devices generally had 30–40 event samples for each device and event type combination.

Device	Event	Signature	Duration (ms)
Cameras			
Amazon camera	Watch	**S1:** S-[627-634] C-[1229-1236]	203 / 261 / 476
Blink hub	Watch	**S1:** S-199 C-135 C-183 S-135	99 / 158 / 275
	Photo	**S1:** S-199 C-135 C-183 S-135	87 / 173 / 774
Lefun camera	Photo	**S1:** S-258 C-[206-210] S-386 C-206 **S2:** C-222 S-198 C-434 S-446 C-462 S-194 C-1422 S-246 C-262 **S3:** C-182	17,871 / 19,032 / 20,358
	Recording	**S1:** S-258 C-210 S-386 C-206 **S2:** C-222 S-198 C-434 S-446 C-462 S-194	13,209 / 15,279 / 16,302
	Watch	**S1:** S-258 C-210 S-386 C-206 **S2:** C-222 S-198 C-434 S-446 C-462 S-194	14,151 / 15,271 / 16,131
Microseven camera	Watch	**S1:** D-242 PH-118	1 / 5 / 38
ZModo doorbell	Photo	**S1:** C-94 S-88 S-282 C-240 / **S1:** S-282 C-240 C-94 S-88	1,184 / 8,032 / 15,127
	Recording	**S1:** C-94 S-88 S-282 C-240 / **S1:** S-282 C-240 C-94 S-88	305 / 7,739 / 15,137
	Watch	**S1:** C-94 S-88 S-282 C-240 / **S1:** S-282 C-240 C-94 S-88	272 / 7,679 / 15,264
Light Bulbs			
Flex light bulb	ON/OFF	**S1:** PH-140 D-[346-347]	4 / 44 / 78
	Intensity	**S1:** PH-140 D-346	4 / 18 / 118
	Color	**S1:** PH-140 D-346	4 / 12 / 113
Wink hub	ON/OFF	**S1:** PH-204 D-890 PH-188 D-113	43 / 55 / 195
	Intensity	**S1:** PH-204 D-890 PH-188 D-113	43 / 50 / 70
	Color	**S1:** PH-204 D-890 PH-188 D-113	43 / 55 / 106
Voice Command Devices			
Allure speaker	Audio ON/OFF	**S1:** C-658 C-412	89 / 152 / 196
	Volume	**S1:** C-[594-602] **S2:** C-[92-100]	217 / 4,010 / 11,005
Amazon Echo Dot	Voice	**S1:** C-491 S-[148-179]	1 / 23 / 61
	Volume	**S1:** C-[283-290] C-[967-979] **S2:** C-[197-200] C-[147-160]	1,555 / 2,019 / 2,423
Amazon Echo Plus	Audio ON/OFF	**S1:** S-100 C-100	1 / 5 / 28
	Color	**S1:** S-100 C-100	1 / 4 / 18
	Intensity	**S1:** S-100 C-100	1 / 4 / 11
	Voice	**S1:** C-[761-767] S-437 **S2:** C-172 S-434	1,417 / 1,871 / 2,084
	Volume	**S1:** C-172 S-434	2 / 13 / 40
Amazon Echo Spot	Audio ON/OFF	**S1:** S-100 C-100	1 / 8 / 233
	Voice	**S1:** C-246 S-214 **S2:** C-172 S-434	1,220 / 1,465 / 1,813
	Volume	**S1:** C-246 S-214 **S2:** C-172 S-434	1,451 / 1,709 / 1,958
Google Home	Voice	**S1:** C-1434 S-136	9 / 61 / 132
	Volume	**S1:** C-1434 S-[124-151] **S2:** C-521 S-[134-135]	8,020 / 9,732 / 10,002
Google Home Mini	Voice	**S1:** C-1434 S-[127-153]	1 / 29 / 112
	Volume	**S1:** C-1434 S-[135-148]	5 / 47 / 123
Harman Kardon Invoke speaker	Voice	**S1:** S-1494 S-277 C-1494 **S2:** S-159 S-196 C-1494	2,199 / 2,651 / 3,762
	Volume	**S1:** S-159 S-196 C-1418 C-1320 S-277 **S2:** S-196 C-[404-406]	223 / 567 / 793
Smart TVs			
Fire TV	Menu	**S1:** C-468 S-323	16 / 18 / 20
LG TV	Menu	**S1:** PH-204 D-1368 PH-192 D-117	43 / 90 / 235
Roku TV	Remote	**S1:** PH-163 D-[163-165] **S2:** PH-145 D-410 **S2:** PH-147 D-113	578 / 1,000 / 1,262
Samsung TV	Menu	**S1:** PH-[237-242] D-274	2 / 7 / 15
Other Types of Devices			
Honeywell thermostat	ON	**S1:** S-635 C-256 C-795 S-139 C-923 S-139	1,091 / 1,248 / 1,420
	OFF	**S1:** S-651 C-256 C-795 S-139 C-923 S-139	
	Set	**S1:** C-779 S-139	86 / 102 / 132
Insteon hub	ON/OFF	**S1:** S-491 C-623 **S2:** C-784 C-234 S-379	76 / 100 / 1,077
Samsung fridge	Set	**S1:** C-116 S-112	177 / 185 / 185
	View Inside	**S1:** C-116 S-112	177 / 197 / 563

TABLE V. SIGNATURES EXTRACTED FROM THE DEVICES ONLY IN THE MON(IOT)R [39] DATASET.

Identical Signatures. For the WeMo Insight plug and Blink camera, the signatures extracted from the Mon(IoT)r dataset and our dataset (December 2019) were identical. Since the signatures obtained from our own dataset do not have any variations in packet lengths, we used PINGPONG's exact matching strategy (see Section IV-B) to detect events in the Mon(IoT)r dataset, and we observed a recall rate of 97% or higher for both devices (see Table VI).

Device	Event	Signature	Duration (ms) Min./Avg./Max./St.Dev.	Matching			
				WAN Sniffer	FPR	Wi-Fi Sniffer	FPR
WeMo Insight plug	ON/OFF	∗**S1:** PH-475 D-246	29 / 33 / 112 / 9	-	-	98.75%	0
		†**S1:** PH-475 D-246	31 / 42 / 111 / 15				
Blink camera	Watch	∗**S1:** C-331 S-299 C-139	267 / 273 / 331 / 8	100%	0	100%	0
		†**S1:** C-331 S-299 C-139	170 / 269 / 289 / 19				
	Photo	∗**S1:** C-331 C-123 S-139 S-123 S-187 C-1467	281 / 644 / 1,299 / 348	97.37%	0	97.50%	0
		†**S1:** C-331 C-123 S-139 S-123 S-187 C-1467	281 / 742 / 2,493 / 745				
TP-Link plug (Device-Cloud)	ON	∗**S1:** C-592 S-1234 S-100	70 / 74 / 85 / 2	100%	0	-	-
	OFF	∗**S1:** C-593 S-1235 S-100					
	ON	†**S1:** C-605 S-1213 S-100	16 / 19 / 29 / 2				
	OFF	†**S1:** C-606 S-1214 S-100					
TP-Link plug (Phone-Device & Device-Cloud)	ON	∗**S1:** PH-172 D-115 **S2:** C-592 S-1234 S-100	406 / 743 / 10,667 / 1,417	-	-	100%	0
	OFF	∗**S1:** PH-172 D-115 **S2:** C-593 S-1235 S-100					
	ON	†**S1:** PH-172 D-115 **S2:** C-605 S-1213 S-100	197 / 382 / 663 / 165				
	OFF	†**S1:** PH-172 D-115 **S2:** C-606 S-1214 S-100					
Sengled light bulb	ON	∗**S1:** S-[217-218] C-[209-210] **S2:** C-430 **S3:** C-466	4,304 / 6,238 / 8,145 / 886	-	-	-	-
	OFF	∗**S1:** S-[217-218] C-[209-210] **S2:** C-430 **S3:** C-465					
	ON	†**S1:** S-219 C-210 **S2:** C-428 **S3:** C-[478-479]	354 / 2,590 / 3,836 / 859				
	OFF	†**S1:** S-219 C-210 **S2:** C-428 **S3:** C-[478-480]					
TP-Link light bulb	ON	∗**S1:** PH-258 D-288	8 / 77 / 148 / 42	-	-	-	-
	OFF	∗**S1:** PH-258 D-305					
	ON	†**S1:** PH-258 D-227	17 / 92 / 224 / 46				
	OFF	†**S1:** PH-258 D-244					

TABLE VI. COMMON DEVICES IN THE MON(IOT)R AND OUR TESTBED EXPERIMENTS. ∗ SIGNATURE: TRAINING ON OUR TESTBED. † SIGNATURE: TRAINING ON MON(IOT)R [39]. MATCHING: TRAINING ON TESTBED, DETECTION ON MON(IOT)R. THE NUMBER OF EVENTS VARY (AROUND 30-40) PER EVENT TYPE—THE RESULT IS PRESENTED IN % FOR CONVENIENCE.

Similar Signatures. For the TP-Link plug and Sengled light bulb, the signatures extracted from the Mon(IoT)r dataset are slightly different from those extracted from our own dataset: some packet lengths at certain positions in the sequence are different (by a few and up to tens of bytes), and these differences appear to be consistent (*i.e.,* all signatures from both datasets appear to be completely deterministic as they do not contain packet length ranges). For example, the TP-Link plug's ON event is `C-592 S-1234 S-100` in our experiment vs. `C-605 S-1213 S-100` in Mon(IoT)r. To understand the cause of these discrepancies, we examined the TP-Link plug in further detail—between the two devices, its signatures exhibit the largest difference in packet lengths across datasets. Through additional experiments on the TP-Link plug, we identified that changes to configuration parameters (*e.g.,* user credentials of different lengths) could cause the packet lengths to change. However, the packet lengths are deterministic for each particular set of user credentials.

For devices that exhibit this kind of behavior, an attacker must first train PINGPONG multiple times with different user credentials to determine to what extent these configuration changes affect the packet lengths in the signatures. Moreover, the signature matching strategy should not be exact, but must be relaxed to allow for small variations in packet lengths. To this end, we implemented *relaxed matching* that augments the matching strategies discussed in Section IV-B.[4] We ran PINGPONG with relaxed matching on the TP-Link plug with a delta of 21B, and successfully detected 100% of events. Furthermore, by performing the negative control experiments described in Section V-C, we verified that the increase in FPR due to relaxed matching is negligible. For dataset 1, relaxed matching results in two FPs for the Wi-Fi sniffer. For dataset 3, relaxed matching results in seven FPs for the Wi-Fi sniffer and one FP for the WAN sniffer. In comparison, exact matching only produces one false positive for the Wi-Fi sniffer for dataset 3. We note that the total number of packets across these datasets is 440 million. However, relaxed matching may eliminate the ability to distinguish event types for signatures that only differ by a few bytes (*e.g.,* the packet lengths for the TP-Link plug's ON and OFF signatures differ by one byte).

Signature Evolution. We observed that some signatures change over time, presumably due to changes to the device's communication protocol. The WeMo Insight plug's signature changed slightly from our earlier dataset from 2018 (see Table IV) to our latest dataset collected in December 2019 (see Table VI):

[4] In relaxed matching, a delta equal to the greatest variation observed in packet lengths is applied to the packets that vary due to configuration changes. For the TP-Link plug, we observed that the first packets differ by 13B in the the Device-Cloud signatures from the two datasets (*i.e.,* $13 = 605 - 592 = 606 - 593$) and the second packets differ by 21B (*i.e.,* $21 = 1234 - 1213 = 1235 - 1214$), thus a delta of 21B is used.

the first PH-259 packet is no longer part of the signature. Both of these datasets were collected using the same testbed with the same user accounts, but with different device firmware versions. Therefore, the change is probably due to changes in the communication protocol, introduced in firmware updates. This is further backed by the observation that the WeMo Insight plug's signature extracted from the Mon(IoT)r dataset (collected in April 2019) is identical to the signature extracted from our December 2019's dataset. This implies that there has been a protocol change between 2018 and April 2019, but the protocol has then remained unchanged until December 2019.

Similarly, the TP-Link light bulb's signature has changed slightly from our first to our second in-house dataset (see Tables IV and VI), and is also slightly different for the Mon(IoT)r dataset.[5] The signatures from our 2018 dataset and those from the Mon(IoT)r dataset differ in the first packet (`PH-198` vs. `PH-258`, an offset of 60 bytes), and the signatures from the Mon(IoT)r dataset and those from our December 2019 differ in the second packet (`D-227` vs. `D-288` and `D-244` vs. `305`, an offset of 61 bytes). Thus, we also suspect that there is a signature evolution due to firmware updates for the TP-Link light bulb. Signature evolution is a limitation of our approach, and is elaborated on in Section VII. Nevertheless, an attacker can easily overcome this limitation simply by repeating PINGPONG's training to extract the latest signatures from a device right before launching an attack.

G. Parameters Selection and Sensitivity

Clustering Parameters. We empirically examined a range of values for the parameters of the DBSCAN algorithm. We tried all combinations of $\epsilon \in \{1, 2, ..., 10\}$ and `minPts` $\in \{30, 31, ..., 50\}$. For those devices that exhibit no variation in their signature related packet lengths, *e.g.,* the TP-Link plug, the output of the clustering remains stable for all values of ϵ and `minPts` < 50. For such devices, keeping ϵ at a minimum and `minPts` close to the number of events n reduces the number of noise points that become part of the resulting clusters. However, our experiments show that there is a tradeoff in applying strict bounds to devices with more variation in their packet lengths (*e.g.,* the D-Link plug), strict bounds can result in losing clusters that contain packet pairs related to events. For the D-Link plug, this happens if $\epsilon < 7$ and `minPts` > 47. In our experiments, we used our initial values of $\epsilon = 10$ and `minPts` $= 45$ (*i.e.,* `minPts` $= \lfloor n - 0.1n \rfloor$ with n = number of expected events) from our smart plugs experiment (*i.e.,* the TP-Link plug, D-Link plug, and SmartThings plug) that allowed PINGPONG to produce the packet-level signatures we initially observed manually (see Section III-C). We then used them as default parameters for PINGPONG to analyze new devices and extracted packet-level signatures from 15 more devices.

Time Window and Signature Duration. We also measured the duration of our signatures—defined as the time between the first and the last packets of the signature. Table IV reports all the results. The longest signature duration measured is 9,132 ms (less than 10 seconds) for the Sengled light bulb's ON/OFF signatures from the Phone-Cloud communication. This justifies our choice of training time window t = 15 seconds during trace filtering and signature validation (see Section IV-A). This conservative choice also provides slack to accommodate other devices that we have not evaluated and that may have signatures with a longer duration. This implies that events can be generated every 15 seconds or longer. We conservatively chose this duration to be 131 seconds to give ample time for an event to finish, and to easily separate false positives from true positives.

[5]We also repeated our experiments for the TP-Link light bulb to further understand this phenomenon.

VI. POSSIBLE DEFENSES

There are several broad approaches that can obfuscate network traffic to defend against passive inference attacks that analyze network traffic metadata:

1) *Packet padding* adds dummy bytes to each packet to confuse inference techniques that rely on individual packet lengths, and less so volume. Packets can be padded to a fixed length (*e.g.,* MTU) or with a random number of bytes.
2) *Traffic shaping* purposely delays packets to confuse inference techniques that rely on packet inter-arrival times and volume over time.
3) *Traffic injection* adds dummy packets in patterns that look similar (*e.g.,* have the same lengths, inter-arrival times or volume signature etc.) as the real events, thus hiding the real event traffic in a crowd of fake events.

The above approaches can be implemented in different ways and can also be combined (*e.g.,* on the same VPN). Since our signatures rely on unique sequences of individual packet lengths, packet padding is the most natural defense and therefore discussed in depth below. We defer discussion of traffic shaping and traffic injection to Appendix C in [48]. We first provide a brief overview of packet padding in the literature. We then discuss how packet padding may be implemented to obfuscate packet-level signatures. Finally, we evaluate the efficacy of packet padding for the TP-Link plug.

Packet Padding in the Literature. Packet padding has already been studied as a countermeasure for website fingerprinting [15], [16], [21], [28]. Liberatore and Levine [28] showed that padding to MTU drastically reduces the accuracy of a Jaccard coefficient based classifier and a naive Bayes classifier, both of which use a feature set very similar to packet-level signatures: a vector of <direction, packet length> tuples. Dyer *et al*. [21] later showed that such padding is less successful against more advanced classifiers, such as the support vector machine proposed by Panchenko *et al*. [36] that also considers coarse-grained features such as total traffic volume. Cai *et al*. [15], [16] improved [21] by providing a strategy to control traffic flow that better obfuscates the traffic volume as a result. Although applied in a different context, these prior works indicate that packet padding should successfully guard against a packet-level signature attack. The question then becomes where and how to implement the padding mechanism.

Possible Implementations. Next, we discuss the potential benefits and drawbacks of different packet padding implementations. We consider a VPN-based implementation, padding at the application layer, and TLS-based padding.

VPN. One option is to route traffic from the smart home devices and the smartphone through a VPN that pads outbound tunneled packets with dummy bytes and strips the padding

off of inbound tunneled packets: a technique also considered in [10]. The smart home end of the VPN may be implemented either directly on each device and smartphone or on a middlebox, *e.g.,* the home router. The former provides protection against *both* the WAN and Wi-Fi sniffers as the padding is preserved on the local wireless link, whereas the latter only defends against the WAN sniffer. However, an on-device VPN may be impractical on devices with limited software stacks and/or computational resources. The middlebox-based approach may be used to patch existing devices without changes to their software. Pinheiro et al. [38] provide an implementation in which the router is the client-side end of the VPN, and where the padding is added to the Ethernet trailer.

Application Layer and TLS. Another option is to perform the padding at the application layer. This has at least three benefits: (1) it preserves the padding across all links, thus provides protection against both the WAN and Wi-Fi sniffers; (2) it imposes no work on the end user to configure their router to use a VPN; and (3) it can be implemented entirely in software. An example is HTTPOS by Luo et al. [31], which randomizes the lengths of HTTP requests (*e.g.,* by adding superfluous data to the HTTP header). One drawback of application layer padding is that it imposes extra work on the application developer. This may be addressed by including the padding mechanism in libraries for standardized protocols (*e.g.,* OkHttp [47]), but a separate implementation is still required for every proprietary protocol. A better alternative is to add the padding between the network and application layers. This preserves the benefits of application layer padding highlighted above, but eliminates the need for the application developer to handle padding. As suggested in [21], one can use the padding functionality that is already available in TLS [40].

Residual Side-Channel Information. Even after packet padding is applied, there may still be other side-channels, *e.g.,* timing and packet directions, and/or coarse-grained features such as total volume, total number of packets, and burstiness, as demonstrated by [21]. Fortunately, timing information (*e.g.,* packet inter-arrival times and duration of the entire packet exchange) is highly location dependent (see the comparison of signature durations in Table VI), as it is impacted by the propagation delay between the home and the cloud, as well as the queuing and transmission delays on individual links on this path. Exploiting timing information requires a much more powerful adversary: one that uses data obtained from a location close to the smart home under attack. The work of Apthorpe *et al*. on traffic shaping [13] and stochastic traffic padding (STP) [10] may aid in obfuscating timing, volume, and burstiness.

Efficacy of Packet Padding. The discussion has been qualitative so far. Next, we perform a simple test to empirically assess the efficacy of packet padding for the TP-Link plug.

Setup. We simulated padding to the MTU by post-processing the TP-Link plug testbed trace from Section V-B (50 ON and 50 OFF events, mixed with background traffic) using a simplified version of PINGPONG's detection that only considers the order and directions of packets, but pays no attention to the packet lengths. We focus on the WAN sniffer because it is the most powerful adversary: it can separate traffic into individual TCP connections and eliminate the confusion that arises from multiplexing. We used the TP-Link plug's two-packet signatures for ON and OFF events (see Table IV) as the Phone-Device communication is not visible on the WAN. We consider the packet padding's impact on transmission and processing delays to be negligible. We assume that the adversary uses *all* available information to filter out irrelevant traffic. Specifically, the adversary uses the timing information observed during training to only consider request-reply exchanges that comply with the signature duration.[6] Moreover, since the TP-Link plug uses TLSv1.2 (which does not encrypt the SNI), the adversary can filter the trace to only consider TLS Application Data packets to the relevant TP-Link host(s) in the no-VPN scenarios.

VPN-Based Padding. To simulate VPN-based packet padding, we consider all packets in the trace as multiplexed over a single connection and perform signature detection on this tunnel. This results in a total of 193,338 positives, or, put differently, more than 1,900 false positives for every event. This demonstrates that VPN-based packet padding works well for devices with short signatures (*e.g.,* a single packet pair).

TLS-Based Padding. From the training data, the adversary knows that the signature is present in the TP-Link plug's communication with `events.tplinkra.com`. To simulate TLS-based packet padding, we performed signature detection on the TLS Application Data packets of each individual TLSv1.2 connection with said host. As expected, this produced a total of 100 detected events, with no FPs. Intuitively, this is because the only TLS Application Data packets of these connections are exactly the two signature packets, and the device only communicates with this domain when an event occurs.

Hybrid. We next explore how multiplexing all of the TP-Link plug's traffic over a single connection affects the false positives (the plug communicates with other TP-Link hosts). This is conceptually similar to a VPN, but only tunnels application layer protocols and can be implemented in user space (without TUN/TAP support). To simulate such a setup, we filtered the trace to only contain IPv4 unicast traffic to/from the TP-Link plug, and dropped all packets that were not TLS Application Data. We then performed detection on the TLS Application Data packets, treating them as belonging to a single TLS connection. For this scenario, we observed 171 positives. While significantly better than TLS-based padding for individual TLS connections, the attacker still has a high probability (more than 50%) of guessing the occurrence of each event (but cannot distinguish ON from OFF).

Recommendations. Based on the above insights, we recommend VPN-based packet padding due to its additional obfuscation (encryption of the Internet endpoint and multiplexing of IoT traffic with other traffic) as TLS-based padding seems insufficient for devices with simple signatures and little background traffic. For more chatty devices, multiplexing all device traffic over a single TLS connection to a single server may provide sufficient obfuscation at little overhead.

[6] $t = 0.204\text{s} \implies \lceil 0.204 + 0.1 \times 0.204\text{s} \rceil = 0.224\text{s}$ (see Table IV and Appendix A in [48])

VII. Conclusion and Discussion

Summary. We designed, implemented, and evaluated PingPong, a methodology for automatically extracting packet-level signatures for smart home device events from network traffic. Notably, traffic can be encrypted or generated by proprietary or unknown protocols. This work advances the state-of-the-art by: (1) identifying simple packet-level signatures that were not previously known; (2) proposing an automated methodology for extracting these signatures from training datasets; and (3) showing that they are effective in inferring events across a wide range of devices, event types, traces, and attack models (WAN sniffer and Wi-Fi sniffer). We have made PingPong (software and datasets) publicly available at [49]. We note that the new packet-level signatures can be used for several applications, including launching a passive inference attack, anomaly detection, etc. To deal with such attacks, we outlined a simple defense based on packet padding.

Current Limitations and Future Directions. PingPong has its limitations and can be extended in several directions.

First, in order to extract the signature of a new device, one must first acquire the device and apply PingPong to train and extract the corresponding packet-level signatures. This is actually realistic for an attacker with minimal side information, *i.e.,* one who knows what device they want to attack or who wants to distinguish between two different types of devices. One direction for future work is to extend PingPong by finding "similar" known behaviors for a new device, *e.g.,* via relaxed matching of known and unknown signatures.

Second, a signature may evolve over time, *e.g.,* when a software/firmware update changes a device's communication protocol. Whoever maintains the signature (*e.g.,* the attacker) needs to retrain and update the signature. We observed this phenomenon, for example, for the TP-Link plug. This can be handled by relaxed matching since the packet sequences tend to be mostly stable and only evolve by a few bytes (see Section V-F).

Third, there may be inherent variability in some signatures due to configuration parameters (*e.g.,* credentials and device IDs) that are sent to the cloud and may lead to slightly different packet lengths. In Section V-F, we saw that this variability is small: from a few to tens of bytes difference and only for some individual packets within a longer sequence. An attacker could train with different configuration parameters and apply relaxed matching when necessary (only on packets with length variations).

Other possible improvements include: profiling and subtracting background/periodic traffic during signature creation, and unifying the way we account for small variation in the signatures in the training and detection—PingPong currently supports range-based matching (see Appendix A in [48]) and relaxed matching as separate features. Another limitation is that our methodology currently applies only to TCP—not to UDP-based devices that do not follow the request-reply pattern.

Conclusion. We believe that the new packet-level signatures identified by PingPong are a simple, intuitive, and universal means for profiling IoT devices. However, we see PingPong only as a building block, which is part of a bigger toolbox for IoT network traffic analysis. We believe that it can and should be combined with other complementary ideas, *e.g.,* traffic shape/volume-based signatures, semi-supervised learning, etc.

Acknowledgment

This project was supported by the National Science Foundation under grants CNS-1649372, CNS-1703598, OAC-1740210, CNS-1815666, CNS-1900654 and a UCI Seed Funding Award at UCI. The authors would like to thank the anonymous NDSS reviewers for their valuable feedback, which helped significantly improve the paper. We would also like to thank Anastasia Shuba for her insights and advice during the project's early stages.

References

[1] IFTTT. https://www.ifttt.com/, September 2018.

[2] If motion detected by D-Link motion sensor, then turn on D-Link smart plug. https://ifttt.com/applets/393508p-if-motion-detected-by-d-link-motion-sensor-then-turn-on-d-link-smart-plug, January 2020.

[3] 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.

[4] Alexa. Top sites in United States. https://www.alexa.com/topsites/countries/US, November 2018.

[5] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. SoK: Security evaluation of home-based IoT deployments. In *2019 2019 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 208–226.

[6] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose. Yourthings scorecard. https://yourthings.info/, 2019.

[7] Amazon. https://www.amazon.com/smart-home/b/?ie=UTF8&node=6563140011&ref_=sv_hg_7, March 2019.

[8] B. Anderson and D. McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*, AISec '16, pages 35–46, New York, NY, USA, 2016. ACM.

[9] Android.com. Android debug bridge (adb). https://developer.android.com/studio/command-line/adb, November 2018.

[10] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster. Keeping the smart home private with smart(er) IoT traffic shaping. *Proceedings on Privacy Enhancing Technologies*, 2019(3), 2019.

[11] N. Apthorpe, D. Reisman, and N. Feamster. Closing the blinds: Four strategies for protecting smart home privacy from network observers. *CoRR*, abs/1705.06809, 2017.

[12] N. Apthorpe, D. Reisman, and N. Feamster. A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic. *CoRR*, abs/1705.06805, 2017.

[13] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster. Spying on the smart home: Privacy attacks and defenses on encrypted IoT traffic. *CoRR*, abs/1708.05044, 2017.

[14] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine. Privacy vulnerabilities in encrypted HTTP streams. In *Proceedings of the 5th International Conference on Privacy Enhancing Technologies*, PET'05, pages 1–11, Berlin, Heidelberg, 2006. Springer-Verlag.

[15] X. Cai, R. Nithyanand, and R. Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 121–130. ACM, 2014.

[16] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 227–238. ACM, 2014.

[17] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 605–616, New York, NY, USA, 2012. ACM.

[18] 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. IEEE, 2010.

[19] B. Copos, K. Levitt, M. Bishop, and J. Rowe. Is anybody home? Inferring activity from smart home network traffic. In *Security and Privacy Workshops (SPW), 2016 IEEE*, pages 245–251. IEEE, 2016.

[20] R. Doshi, N. Apthorpe, and N. Feamster. Machine learning DDoS detection for consumer internet of things devices. *CoRR*, abs/1804.04159, 2018.

[21] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE symposium on security and privacy*, pages 332–346. IEEE, 2012.

[22] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[23] M. Ghiglieri and E. Tews. A privacy protection system for HbbTV in Smart TVs. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pages 357–362, Jan 2014.

[24] J. Hayes and G. Danezis. K-fingerprinting: A robust scalable website fingerprinting technique. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 1187–1203, Berkeley, CA, USA, 2016. USENIX Association.

[25] D. Herrmann, R. Wendolsky, and H. Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 31–42. ACM, 2009.

[26] Y. Jin, E. Sharafuddin, and Z.-L. Zhang. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 49–60, New York, NY, USA, 2009. ACM.

[27] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. Blinc: Multilevel traffic classification in the dark. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 229–240, New York, NY, USA, 2005. ACM.

[28] M. Liberatore and B. N. Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 255–263. ACM, 2006.

[29] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[30] L. Lu, E.-C. Chang, and M. C. Chan. Website fingerprinting and identification using ordered feature sequences. In *Proceedings of the 15th European Conference on Research in Computer Security*, ESORICS'10, pages 199–214, Berlin, Heidelberg, 2010. Springer-Verlag.

[31] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci. HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2011.

[32] T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *Commun. Surveys Tuts.*, 10(4):56–76, Oct. 2008.

[33] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi. HomeSnitch: Behavior transparency and control for smart home IoT devices. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, pages 128–138, New York, NY, USA, 2019. ACM.

[34] OpenWrt/LEDE Project. https://openwrt.org/about.

[35] A. Panchenko and F. Lanze. Website fingerprinting at internet scale. In *NDSS*, 2016.

[36] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, pages 103–114. ACM, 2011.

[37] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.

[38] A. J. Pinheiro, J. M. Bezerra, and D. R. Campelo. Packet padding for improving privacy in consumer IoT. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00925–00929, June 2018.

[39] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi. Information exposure from consumer IoT devices: A multidimensional, network-informed measurement approach. In *Proceedings of the Internet Measurement Conference*, pages 267–279, 2019.

[40] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, RFC Editor, August 2018.

[41] E. Rescorla and T. Dierks. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Aug. 2008.

[42] T. S. Saponas, J. Lester, C. Hartung, S. Agarwal, and T. Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 5:1–5:16, Berkeley, CA, USA, 2007. USENIX Association.

[43] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. 2018.

[44] A. Sivanathan, H. H. Gharakheili, A. R. Franco Loi, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Transactions on Mobile Computing*, (01):1–1.

[45] 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, May 2017.

[46] Square, Inc. Whats going to happen with IFTTT? https://square.github.io/okhttp/, 2019.

[47] Stacey Higginbotham. OkHttp. https://staceyoniot.com/whats-going-to-happen-with-ifttt/, 2019.

[48] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. Ping-pong: Packet-level signatures for smart home device events. http://arxiv.org/abs/1907.11797.

[49] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. Ping-pong: Packet-level signatures for smart home devices (software and dataset). http://plrg.ics.uci.edu/pingpong/.

[50] UNSW. List of smart home devices. https://iotanalytics.unsw.edu.au/resources/List_Of_Devices.txt, November 2018.

[51] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 143–157, 2014.

[52] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Uncovering spoken phrases in encrypted voice over IP conversations. *ACM Transactions on Information and System Security*, 13(4):35:1–35:30, Dec. 2010.

[53] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson. Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, pages 4:1–4:12, Berkeley, CA, USA, 2007. USENIX Association.

[54] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu. HoMonit: Monitoring smart home apps from encrypted traffic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1074–1088, New York, NY, USA, 2018. ACM.