Fingerprinting Encrypted Voice Traffic on Smart Speakers with Deep Learning

Chenggang Wang*
University of Cincinnati, USA
wang2c9@mail.uc.edu

Sean Kennedy
University of Cincinnati, USA
kenneds6@mail.uc.edu

Haipeng Li
University of Cincinnati, USA
li2hp@mail.uc.edu

King Hudson
University of Cincinnati, USA
hudsonk4@mail.uc.edu

Gowtham Atluri
University of Cincinnati, USA
atlurigm@ucmail.uc.edu

Xuetao Wei
Southern University of Science and Technology, China
weixt@sustech.edu.cn

Wenhai Sun
Purdue University, USA
whsun@purdue.edu

Boyang Wang
University of Cincinnati, USA
boyang.wang@uc.edu

ABSTRACT

This paper investigates the privacy leakage of smart speakers under an encrypted traffic analysis attack, referred to as voice command fingerprinting. In this attack, an adversary can eavesdrop both outgoing and incoming encrypted voice traffic of a smart speaker, and infers which voice command a user says over encrypted traffic. We first built an automatic voice traffic collection tool and collected two large-scale datasets on two smart speakers, Amazon Echo and Google Home. Then, we implemented proof-of-concept attacks by leveraging deep learning. Our experimental results over the two datasets indicate disturbing privacy concerns. Specifically, compared to 1% accuracy with random guess, our attacks can correctly infer voice commands over encrypted traffic with 92.89% accuracy on Amazon Echo.

Despite variances that human voices may cause on outgoing traffic, our proof-of-concept attacks remain effective even only leveraging incoming traffic (i.e., the traffic from the server). This is because the AI-based voice services running on the server side response commands in the same voice and with a deterministic or predictable manner in text, which leave distinguishable pattern over encrypted traffic. We also built a proof-of-concept defense to obfuscate encrypted traffic. Our results show that the defense can effectively mitigate attack accuracy on Amazon Echo to 32.18%.

CCS CONCEPTS

• **Security and privacy → Network security**.

*The first two authors contributed equally to this research.

KEYWORDS

machine learning, encrypted traffic analysis, smart speaker

ACM Reference Format:
Chenggang Wang, Sean Kennedy, Haipeng Li, King Hudson, Gowtham Atluri, Xuetao Wei, Wenhai Sun, and Boyang Wang. 2020. Fingerprinting Encrypted Voice Traffic on Smart Speakers with Deep Learning. In *13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), July 8–10, 2020, Linz (Virtual Event), Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3395351.3399357

1 INTRODUCTION

Smart speakers, such as Amazon Echo, Google Home and Apple HomePod, are being increasingly adopted in the U.S. with sales surpassing 133 million [2]. However, privacy remains one of the major concerns limiting a more widespread adoption among consumers. This includes two types of concerns: (1) privacy disclosed to voice service providers [1], and (2) the focus of this research, sensitive information that can be revealed by external attackers.

We investigated the privacy leakage of smart speakers by considering an external attacker that runs voice command fingerprinting attacks [18]. In this attack, an attacker eavesdrops encrypted voice traffic of a smart speaker, and leverages side-channel information, including the size, direction, and order of encrypted packets, to infer a user's voice command without decryption. For instance, an attacker can be a *local eavesdropper* on a victim's WiFi network or a compromised WiFi access point. As the content of a response from the server is correlated with a voice command, an attacker leverages both outgoing traffic (the encrypted packets of a *voice command*) and incoming traffic (the encrypted packets of a *response*) to infer a voice command in this attack.

Revealing voice commands can uncover users' activities, lead to unauthorized disclosure, and compromise privacy of millions of users. Moreover, an attacker could leverage voice command fingerprinting to assist malicious attacks (such as *skill squatting* [20, 50]) to attack specific targets. For instance, an attacker could infer which voice commands a specific victim often says by leveraging voice command fingerprinting and then create malicious skills using skill squatting, where the names of malicious skills share similar pronunciations of words appeared in those voice commands. An attacker could further record user conversations through smart speakers using malicious skills and steal sensitive information such as passwords and credit card information [20, 50].

Similar to website fingerprinting [5, 11–14, 19, 22, 25, 27, 28, 30, 34–36, 41] voice command fingerprinting is an encrypted traffic analysis attack, which can be formulated as supervised learning problem. Machine-learning-based encrypted traffic analysis can also be used to fingerprint devices [8, 40]. Kennedy et al. [18] previously studied voice command fingerprinting over a small dataset, which consists of 100 commands with 10 traffic traces per command. By manually selecting features and utilizing AdaBoost as the classifier, their attack can achieve 33.8% accuracy.

This paper aims to improve attack accuracy and advance our understanding of the privacy leakage. Deep learning techniques [21] that have been found to improve attack accuracy of encrypted traffic analysis in website fingerprinting [5, 25, 30, 34, 35], are potential candidates to address the problem of voice command fingerprinting. However, deep learning techniques require large datasets with thousands to millions of samples. Currently, there are no large-scale research datasets available that can be harnessed in the context of voice command fingerprinting.

In this paper, we report the advances we made in bridging the gap in the understanding of privacy impacts of smart speakers. Our experimental results derived from neural networks over two large-scale datasets indicate disturbing privacy concerns on smart speakers as well as the driving force of smart speakers — the AI-based voice services. According to our results, despite high variances that human voices may cause on the outgoing traffic of a smart speaker, **our proof-of-concept attacks remain effective by only leveraging the incoming traffic from the server side.** *This is mainly because the AI-based voice services running on the server side response commands in the same voice and with a deterministic or predictable manner in text, which leave distinguishable network traffic pattern.*

Contributions. Our main contributions are summarized below:

- We built an automatic tool to collect encrypted voice traffic on a smart speaker. It is capable of collecting approximately 3,000 traffic traces per day, which addresses the limitation of data collection in voice command fingerprinting.
- We collected two large-scale datasets on two popular smart speakers, Amazon Echo and Google Home, using 5 automated voices rendered by public text-to-speech APIs. Each dataset consists of 150,000 traffic traces including 100 commands/classes and 1,500 traffic traces per class.
- We performed the attack utilizing Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) and Stacked Autoencoder (SAE) respectively. Our results using bidirectional traffic of Amazon Echo in *the closed-world setting*[1] show that, CNN attained **89.05%** accuracy, LSTM achieved 88.65% and SAE obtained 75.98%. Both CNN and LSTM outperformed the previous attack in [18]. Our attack using ensemble learning [26] further improved accuracy to **92.89%**.
- As human voices vary in practice due to age, gender and accent, which could cause higher variances on outgoing traffic than the automated voices we utilized in our data collection. We also demonstrated that our attacks are effective using incoming traffic only. Specifically, both CNN and LSTM still achieved over **81%** accuracy in the closed world-setting on Amazon Echo dataset. Our attack results in *the open-world setting* are also highly effective.
- We designed a proof-of-concept defense, which obfuscates traffic and mitigates privacy leakage against voice command fingerprinting. According to our results on Amazon Echo dataset, with privacy parameter $\epsilon = 0.005$, our defense can reduce attack accuracy to **1.23%** if an attacker trains models with original traffic and tests with obfuscated traffic. If an attacker adapts, in which it trains and tests with obfuscated traffic, attack accuracy can still be reduced to **32.18%**.

[1]In the closed world setting, an attacker knows a traffic is associated with a given list of commands and infers which command it is. In the open-world setting, an attacker decides whether a traffic is associated with a given list of commands.

2 RELATED WORK

Website Fingerprinting. The purpose of website fingerprinting is to infer which website a user visits over encrypted traffic [12–14, 22, 27, 28, 30, 35, 41]. Early-stage research in website fingerprinting focused on manually extracting features from encrypted traffic and harnessing different conventional machine learning algorithms to achieve higher accuracy. Among these studies, Panchenko et al. [27] proposed CUMUL by considering the cumulative sum function of traffic size. This attack leveraged Support Vector Machine as the classifier and outperformed other methods [30]. This attack achieved an accuracy of 93%, and is even comparable in performance with deep-learning-based attacks [30, 35].

Recent work in website fingerprinting attacks used deep learning models to automatically extract features and resulted in higher accuracy. Sirinam et al. [35] leveraged Convolutional Neural Network (CNN) and attained 98% accuracy in the closed-world setting with 95 websites. Rimmer et al. [30] investigated website fingerprinting with CNN, Long Short-Term Memory (LSTM) and Stacked Denoising Autoencoder (SDAE). SDAE achieved 94% accuracy on a dataset with 900 classes while CNN and LSTM reached 91% and 88%.

Different defense methods have also been proposed. BuFLO [12] sends packets at a fixed size with fixed intervals but introduces high latency. Juarez et al. [17] designed WTF-PAD to obfuscate traffic pattern by using adaptive padding [33], which hides traffic gaps and introduces no latency. Wang et al. devised Walkie-Talkie [42], which applied half-duplex model and burst molding to Tor traffic.

Both WTF-PAD [17] and Walkie-Talkie [42] cause low latency, but can be compromised by CNN-based attacks [35]. A CNN-based attack achieved 90% accuracy against WTF-PAD and 49.7% accuracy against Walkie-Talkie. Imani et al. [15] proposed to leverage adversarial examples [37] as a defense against website fingerprinting. However, it requires the knowledge of entire traffic traces in advance, and hence cannot obfuscate traffic on the fly.

Video Stream Fingerprinting. Schuster et al. [32] showed that traffic bursts are useful to identify encrypted MPEG-DASH video streams. Zhang et al. [51] demonstrated the threat of video stream fingerprinting with 40 YouTube videos with 100 traces per video. CNN achieved 95% accuracy and outperformed competing approaches. On the other hand, the authors showed that leveraging

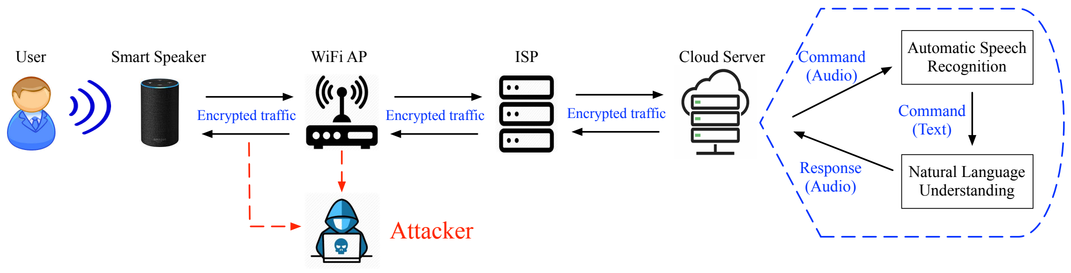


Figure 1: The system model of voice command fingerprinting attacks.

differential privacy on time-series data (e.g., d^*-privacy [47]) can obfuscate traffic pattern and preserve privacy over encrypted traffic. In contrast to website fingerprinting, which extracts features on bi-directional traffic, video stream fingerprinting leverages one-way traffic (i.e., encrypted video traffic sent by the server).

Voice Command Fingerprinting. Kennedy et al. investigated voice command fingerprinting attacks over a small dataset, which includes 100 classes and 10 traces per class [18]. The authors examined several traditional machine learning methods in website fingerprinting and applied to encrypted voice traffic on smart speakers. A method leveraging AdaBoost achieved over 33.8% accuracy in the closed-world setting and outperformed others. The features that the authors examined in [18] include the size of each burst, total transmitted bytes, the number of bursts, occurring packet sizes, percentage incoming packets, and the number of packets.

Apthorpe et al. [8] inferred user activities at home by identifying different smart home devices (including smart speakers) through encrypted traffic pattern. In their study, the authors inferred 3 voice commands (with 3 samples per command) on Amazon Echo based on traffic rate. The dataset and attack methods in [8] are not as comprehensive as our study.

Fingerprinting on Encrypted VoIP Traffic. Several previous studies examined the privacy of encrypted VoIP (Voice over IP traffic) [9, 43–46]. These studies leverage packet size as the only fingerprint because VoIP uses a combination of Variable Bit Rate encoding with stream cipher, but are not effective if data is encrypted with block cipher [46]. These attacks are not applicable to voice command fingerprinting as voice traffic on smart speakers is encrypted with block cipher.

Fingerprinting IoT Devices. Many research studies [6–8, 10, 16, 23, 24, 38, 39] have investigated how to identify IoT devices as well as associated events within a smart home by analyzing encrypted traffic. For instance, Acar et. al. proposed a multi-stage attack, which can achieve over 90% accuracy inferring smart home devices. Different from these studies, our attack focuses on analyzing privacy within a single type of smart home devices.

Other Attacks on Smart Speakers. Injection attacks can inject voice commands through similar pronunciations [20, 50], audible sounds [31, 49], or songs [48]. These attacks focus on the vulnerabilities of Automatic Speech Recognition. Abdullah et al. [4] explored the vulnerabilities of the signal processing step before Automatic Speech Recognition. Zhang et al. [52] investigated the vulnerability of Natural Language Understanding. In contrast to these active attacks, voice command fingerprinting is passive.

3 BACKGROUND

System Model. The system model is illustrated in Fig. 1. It includes four entities: a smart speaker, a WiFi access point, an Internet Service Provider and a server. When a smart speaker receives a user's voice command upon hearing a wake word, it records the voice data and then forwards data to its cloud server. The voice services on the server side will produce responses to a voice command. The voice data traffic between a smart speaker and the server is protected by off-the-shelf encryption technology. For instance, Amazon Echo leverages TLS (Transport Layer Security) 1.2 and all traffic is encrypted with AES (Advanced Standard Encryption) [18].

Threat Model. We assume an attacker is a *local eavesdropper* who can sniff the network traffic of a smart speaker. For instance, an attacker can be an eavesdropper on the victim's WiFi network or a compromised WiFi access point. This attacker cannot decrypt encrypted packets. In addition, this attacker does not drop, change or inject packets.

We assume there is one smart speaker in the victim's WiFi network. We assume the attacker knows the model of a smart speaker, e.g., Amazon Echo or Google Home. We assume the attacker can infer the IP address of a smart speaker as well as the IP address of the server running voice services. With the IP address of a smart speaker, the attacker can filter out traffic from other devices connecting to the same WiFi access point [8].

Inferring the IP address of a smart speaker. Inferring the IP of a smart speaker is feasible for a local eavesdropper. Many existing studies [6–8, 10, 16, 23, 24, 38, 39] have shown that it is possible to distinguish the traffic of a smart speaker (therefore its IP address) from other smart home devices over encrypted traffic.

Once the IP address of a smart speaker is inferred, the server's IP address can be easily observed as a smart speaker mainly communicates with its voice services. Moreover, DNS queries from a smart speaker can also be used to infer the IP addresses of its voice services. For instance, Amazon Echo in our data collection sent DNS queries to resolve the IP addresses of domain name `unagi-na.amazon.com`. As DNS queries and responses are not encrypted, and the IP addresses in the answer section of a DNS response can be easily obtained (e.g., using `Wireshark` or `dig` command).

A traffic trace contains all of the packets for a voice command and its response. We assume packets that are sent to the server are outgoing packets (packets containing a voice command), and packets that are sent to a smart speaker are incoming packets (packets containing a response). We assume that an attacker can infer the start time and the end time of each traffic trace[2]. An attacker can learn side-channel information, including direction, packet size, and timestamp. A traffic trace of a voice command C and its response can be described as

$$T_C = \langle (b_1, s_1, t_1), ..., (b_n, s_n, t_n) \rangle \quad (1)$$

where n is the number of packets in this trace. Each direction b_i is either +1 (outgoing) or −1 (incoming), each packet size s_i is in bytes, and each timestamp t_i is represented in milliseconds.

Closed-World Setting and Open-World Setting. We investigate voice command fingerprinting attacks in both the closed-world setting and the open-world setting.

In the closed-world setting, we assume that an attacker has a prior set of voice commands. For example, this prior set of voice commands can be a set of popular commands that users would ask. Given this prior set, an attacker can harvest labeled traffic traces for each voice command by itself. An attacker can capture an unlabeled traffic trace from a user's smart speaker. This unlabeled traffic trace is associated with one of the voice commands in the prior set. The objective of this attacker is to infer which voice command this unlabeled trace is associated with.

In the open-world setting, an unlabeled traffic trace may not be in the prior set. The objective of this attacker is to infer whether the voice command of this unlabeled traffic trace is in the prior set.

Privacy Metric. We leverage the accuracy of the classification to determine the privacy leakage under voice command fingerprinting in the closed-world setting. For the open-world setting, we leverage true positive rate and false positive rate. This privacy metric is often used in the literature of encrypted traffic analysis.

4 VOICE COMMAND FINGERPRINTING

4.1 Automatic Traffic Collection Tool

Our first challenge is the lack of sufficient training data (or the lack of tools to collect sufficient training data). Existing data collection methods can automatically capture web traffic. However, they do not directly apply to the traffic collection on smart speakers, where *voice interactions are required.*

To automatically collect traffic traces for the study of voice command fingerprinting, we designed a *voice command traffic collection tool* as shown in Fig. 2. This tool consists of three main components: a Raspberry Pi, a regular speaker and a smart speaker. The Raspberry Pi is leveraged as a compromised WiFi Access Point, which can capture and store traffic traces. It is connected to the Internet through an Ethernet cable. A smart speaker is connected to the Raspberry Pi through WiFi. The regular speaker is connected to the Raspberry Pi via an audio cable. An Amazon Echo (2nd generation) is used as an example in Fig. 2. The system is generic, and also works for other smart speakers, such as Google Home.

We prepared a list of commands in text and utilized text-to-speech APIs to generate audio files of voice commands. Specifically, we leveraged Google Cloud text-to-speech API and Amazon Polly to generate multiple audio files for each command. We utilized multiple different automated voices (in US English), including 3

[2]It is feasible for an attacker to infer the start time of each trace on a smart speaker. For example, through our data collection, there is a significant amount of outgoing traffic initiated around the start time of a traffic trace. The end time could be identified once there is no significant volume of traffic after certain time frame, e.g., 2∼3 seconds.

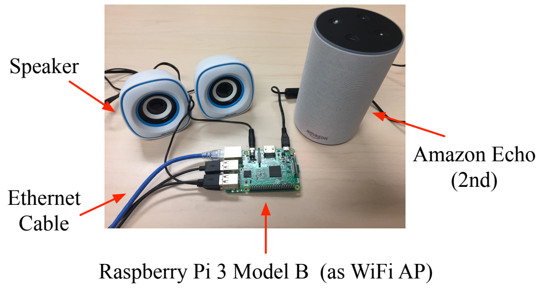


Figure 2: Our automatic voice traffic collection tool.

female voices (Google, Joanna, Salli) and 2 male voices (Joey and Matthew). Google voice is from Google Cloud text-to-speech API and the other four voices are selected from Amazon Polly. For each voice command, our tool generates five audio files, one for each voice. More voices can be supported in this tool if needed.

We used a Raspberry Pi (Pi 3 Model B) running the Raspian OS. We developed a `Python` script containing 80 lines of code and run the script on the Raspberry Pi to automatically play audio files one by one. Upon receiving each voice command from the regular speaker, the smart speaker forwards the command to its cloud server and returns a response. A pre-defined time interval is estimated and applied between the start time of two audio files to ensure that the smart speaker can complete each response. `tcpdump` is executed on the Raspberry Pi to automatically capture traffic traces for each voice command and its response.

To the best of our knowledge, this is the first automatic tool that can be utilized for encrypted traffic analysis on smart speakers. Based on our tests, this tool is capable of automatically collecting approximately 3,000 traffic traces per day without any human interaction. We leveraged this tool and collected two large datasets. Details of these two datasets are presented in Sec. 5.

4.2 Data Format

A raw traffic trace captured by `tcpdump` is first converted to a sequence of tuples as described in Eq. 1. Since this format cannot be directly passed to neural networks, we further transformed the data into two different formats.

Binary Format. Given a traffic trace $T_C = \langle (b_1, s_1, t_1), ..., (b_m, s_m, t_m) \rangle$, its binary format is $T_C = (b_1, ..., b_m)$, which keeps only the direction of each packet.

Numeric Format. Given a traffic trace $T_C = \langle (b_1, s_1, t_1), ..., (b_m, s_m, t_m) \rangle$, its numeric format is $T_C = (b_1 \times s_1, ..., b_m \times s_m)$, which keeps direction and packet size.

For instance, given a traffic trace $T_C = \langle (1, 20, 0.5), (1, 50, 2.1), (-1, 250, 5.3), (1, 100, 6.7) \rangle$, its binary format is $(1, 1, -1, 1)$ and its numeric format is $(20, 50, -250, 100)$.

It is worth mentioning that previous studies in deep-learning-based website fingerprinting often examine binary format only, as Tor networks send fixed-size packets (i.e., *cells*) [35]. In our study, we observed that the packet size of voice traffic on smart speakers varies, so we examined results in both formats. Data in the numeric format is further normalized using MinMaxScaler (with `scikit-learn` library) before being used for neural networks, as the inputs for deep learning should be within the range of $[-1, 1]$.

Since neural networks require the same input length for different classes, different traffic traces are adjusted to an identical vector size. If the original length is smaller than the uniform vector size, we padded the input with 0s; if it is greater, we trimmed the input by dropping data after the uniform vector size. This uniform vector size is one of the hyperparameters we tuned in our experiments. Padding traces to an identical size is commonly used in other attacks, such as website fingerprinting [30, 35], if neural networks are used as classifiers.

4.3 Neural Networks

We implement proof-of-concept attacks using three neural networks, including Convolutional Neural Networks, Long Short-Term Memory and Stacked Autoencoder, respectively. For each type of neural networks, we exploited several different structures, and reported the structure that achieved the highest accuracy.

Convolutional Neural Network (CNN). CNN has been widely used in various classification problems. According to studies in other research areas, especially image classification [21], the accuracy of CNN often outperforms other neural networks. It is likely that CNN would produce higher accuracy than others in voice command fingerprinting.

The structure of our CNN. Our CNN (described in Fig. 8 in Appendix) consists of 11 layers, including 1 input layer, 4 convolutional layers, 5 pooling layers and 1 output layer.

Long Short-Term Memory (LSTM). Long Short-Term Memory, is an advanced version of a Recursive Neural Network. It can mitigate the vanishing gradient and exploding gradient problem in (vanilla) RNNs. LSTM performs better over *time-series data*. Encrypted traffic, in essence, is time-series data, which implies LSTM could outperform others in voice command fingerprinting.

The structure of our LSTM. Our LSTM (illustrated in Fig. 9 in Appendix) consists of 1 input layer, 5 LSTM layers, and 1 output layer. Each LSTM layer consists of multiple LSTM units.

Stacked AutoEncoder (SAE). SAE includes an encoder, a code and a decoder. SAE first compresses data to a smaller number of dimensions and then produces the output by decoding the compressed data. SAE can efficiently extract features from a great number of dimensions and increase accuracy in classification. A traffic trace includes hundreds of packets, where the side-channel information of one packet is a dimension. Leveraging SAE in voice command fingerprinting could attain better attack results.

The structure of our SAE. Our SAE (described in Fig. 10 in Appendix) consists of 9 layers, including 4 layers for encoder, 1 layer for code and 4 layers for decoder. Once it is trained, the encoder and the code are extracted and one dense layer is attached to the end in order to perform classification.

Ensemble Learning. In addition to the three neural networks, we also harness *ensemble learning* [26]. Ensemble learning takes the output of the output layer (i.e., `softmax` function) of each single network, calculates a summation and obtains a prediction with an `argmax` function. We assign weights w_1, w_2 and w_3 for the three neural networks. If the weight for each single network is the same, then it is called *average ensemble*; otherwise, it is referred to as *weighted ensemble* [26].

5 ENCRYPTED VOICE TRAFFIC DATASETS

Overview. We collected two datasets with the tool described in Sec. 4. We refer to the two datasets as *Amazon Echo Dataset* and *Google Home Dataset*. We ran our data collection tool in a 200 square-foot room on campus with a reasonable level of background noise and human activities. Those include a person working on a laptop/PC and typing next to the tool, opening and closing the door occasionally as regular office hours, and students passing the door and the windows of the room from the hallway. The two datasets were collected from March 2019 to August 2019.

Amazon Echo Dataset. For the closed-world setting, we collected 150,000 encrypted traffic traces on Amazon Echo (2nd generation). This dataset includes 100 voice commands and 1,500 traffic traces per command. This list of commands is referred to as ***the monitored list*** by following the literature in website fingerprinting [30, 35]. For the open-world setting, we chose another 100 voice commands and collected 200 traffic traces for each class. This list of commands is referred to as ***the unmonitored list***. We leveraged 5 different voices (Google, Joanna, Joey, Matt, and Salli) for each command in both the closed-world and the open-world setting. Each voice is associated with 20% of traffic traces. This effort lasted approximately 8 weeks in total to complete all the traffic traces in the closed-world setting and approximately 3 weeks to finish the traffic traces in the open-world setting of this dataset. The size of this data is 26.05 GBs (closed-world) and 4.07 GBs (open-world).

The voice commands in our study were selected based on Amazon Echo weekly emails. These emails include popular voice commands that Amazon Echo users often ask. We selected voice commands based on Amazon Echo weekly emails from December 2018 to March 2019. A full list of those commands can be found at [3].

Google Home Dataset. We collected another dataset on Google Home. For this dataset, we only chose a list of 100 voice commands for the study of the closed-world setting. As 21 voice commands in the monitored list of Amazon Echo dataset did not work on Google Home, we chose another 21 new commands and added them to the monitored list of this dataset. Since the wake word is different, we regenerated the audio files with the same voices. For each command, we still collected 1,500 traffic traces with 20% traces per voice. This effort lasted 9 weeks and resulted in a dataset of size 33.57 GBs.

Removing Invalid Traffic Traces. We removed *invalid* traffic traces due to unexpected errors of a smart speaker. For example, an invalid traffic trace could happen when a voice command was correctly played but there was no response from the server. Removing invalid traffic traces is also common in the data collection of other encrypted traffic analysis, such as website fingerprinting [36].

Only a very small number of traces are invalid and removed. For the Amazon Echo dataset, it has 148,770 valid traffic traces (99.18%) for the closed-world setting. The minimal number of traces belonging to one class is 1,340, and the maximum number is 1,500. There are 19,953 valid traffic traces (99.77%) for the open-world setting. The Google Home dataset has 149,745 valid traffic traces (99.83%) for the closed-world setting.

Categories of Voice Commands. Based on the responses of each command, we grouped commands into three categories, referred to as *single response commands*, *time-sensitive response commands*, and *multiple response commands*.

A single response command indicates that the response was always (or almost) the same during our data collection. For example, for voice command "*Where is Mount Rushmore?*" the response from the Amazon server was always the same in our study.

A time-sensitive response command implies that the response changed overtime. For example, "*What is the weather today?*" the Amazon server replied a different answer each day.

A *multiple response* command suggests that we received a number of different responses. However, the content of each response did not change over time. For instance, for the command "*Tell me a barbecue joke.*" the Amazon server randomly returned one of five possible jokes during our data collection.

The ratio of each command category of the monitored list in our Amazon Echo dataset is elaborated in Table. 1. We further discuss the impact of categories on the attack results in the next section.

Table 1: Ratios of Voice Commands in The Three Categories (Amazon Echo Dataset, The Monitored List)

Single	Time-Sensitive	Multiple
45%	21%	34%

Data Visualization. Before we evaluated the results of our neural networks, we first visualized some traffic traces from Amazon Echo dataset by generating heat maps of traffic traces from each command. The main purpose of this step is to demonstrate that it is feasible to fingerprint voice commands based on encrypted traffic of smart speakers. Due to space limitation, we only present the heat maps of 4 voice commands in Fig. 3 and each heat map only contains 10 traffic traces in Amazon Echo dataset.

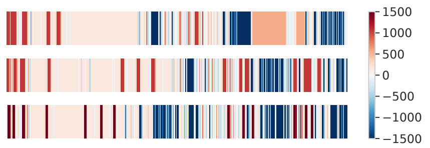
First, we observe that it is indeed viable to infer voice commands based on encrypted traffic. Specifically, the traffic traces of one command are not exactly the same in each heat map, but are very similar. In addition, the traffic pattern of some commands are completely different. For instance, the traffic traces from the first three heat maps, including heat map (a), (b) and (c), are obviously distinguishable. On the other hand, we also notice that it is not trivial to distinguish all the commands based on heat maps as the traffic pattern of some commands are similar. For example, the difference of traffic traces between (c) and (d) in Fig. 3 are not obvious. In fact, most of the classes we investigate have similar pattern as heat map (c) and (d). *This creates the need for sophisticated neural networks*

6 ATTACK EVALUATION

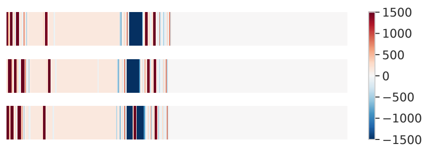
Experiment Setting. We implement our proof-of-concept attack in `Python`. We used `Keras` as the front end and `Tensorflow` as the back end to implement neural networks. We ran our experiments on a Linux machine with Ubuntu 18.04 OS, 2.8 GHz CPU, 16 GB Memory, and a GPU (NIVIDA GeForce GTX 1070). We ran 5-fold cross validation. We used 64% of the data for training, 16% for validation, and 20% for testing.

Hyperparameter Tuning. We searched hyperparameters for neural networks with NNI (Neural Network Intelligence), a Microsoft open-source toolkit. We used TPE (Tree-structured Parzen Estimator) as our search/tuning algorithm, which is one of the tuning algorithms provided by NNI.

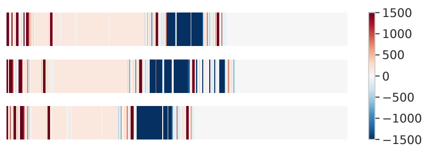
For each neural network, we ran NNI with 50 iterations at most or stopped the search if it took longer than 200 hours. After we



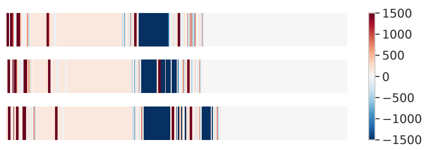
(a) What is my sports update?



(b) What is the date tomorrow?



(c) What is my traffic report?



(d) What is the price of Bitcoin?

Figure 3: Each heat map includes 3 encrypted traffic traces of one voice command. Traffic traces were generated by an Alexa Echo. Outgoing traffic are marked with red color and incoming traffic are marked with blue color. A darker color indicates a packet has a greater packet size.

found a set of tuned hyperparameters with the highest accuracy score, we recorded the tuned hyperparameters and trained our networks using these optimal hyperparameters. When we trained each network, we limited the number of training epochs to 500 or stopped the training early if the accuracy did not continue to improve after 10 consecutive epochs. The search space and tuned hyperparameters can be found in Appendix.

6.1 Closed-World: Outgoing & Incoming Traffic

We first evaluate attack results in the closed-world setting leveraging both outgoing and incoming traffic.

Which Input Format is More Effective? We first compared the attack results between the binary format and numeric format for each model. As shown in Table 2, *the accuracy of the numeric format is much higher than the accuracy of binary format*. This indicates that data in the numeric format leaves more identifiable fingerprints in the encrypted traffic.

Which Neural Network is More Effective? We observed that CNN and LSTM resulted in very similar results and were both significantly higher than SAE in the closed-world setting. The variance of accuracy in each network is small in 5-fold cross validation, which suggests the neural networks are stable.

The results on Google Home dataset indicated similar observations as the ones derived from Amazon Echo dataset. For the attacks on Google Home dataset, we used the same hyperparameters for

Table 2: Attack Results in the Closed-World Setting Based on Both Outgoing and Incoming Traffic (AE: Averaging Ensemble; WE: Weighted Ensemble; ACC: Accuracy; VAR: Variance)

Dataset	Format	CNN		LSTM		SAE		AE	WE
		ACC	VAR	ACC	VAR	ACC	VAR	ACC	ACC
Amazon Echo	Binary	75.88%	0.56×10^{-5}	**77.51%**	4.68×10^{-5}	66.59%	0.44×10^{-5}	85.49%	**85.70%**
	Numeric	**89.05%**	1.50×10^{-5}	88.65%	0.49×10^{-5}	75.98%	0.48×10^{-5}	89.41%	**92.89%**
Google Home	Binary	95.17%	1.62×10^{-5}	**96.90%**	2.48×10^{-5}	90.20%	0.47×10^{-5}	–	–
	Numeric	**99.22%**	0.06×10^{-5}	98.62%	0.13×10^{-5}	92.34%	0.28×10^{-5}	–	–

Table 3: The Comparison with Previous Methods in the Closed-World Setting Based on Both Outgoing and Incoming Traffic of Amazon Echo Dataset

Attack Method	CNN	LSTM	SAE	CUMUL [27]	CNS19 [18]	Random Guess
Accuracy	**89.05%**	88.65%	75.98%	61.44%	76.32%	1%
Training Time (second)	5,327	23,967	1,800	6,073	4,421	N/A

each neural network as the ones in Amazon Echo dataset. This suggests that our neural networks are *transferable* across encrypted traffic from different smart speakers.

Can Ensemble Learning Improve Accuracy? As presented in Table 2, our results demonstrate that ensemble learning can improve the attack accuracy nearly 4% and outperforms each single network in the closed-world setting on Amazon Echo dataset.

For weighted ensemble, we calculated the normalized weights using the accuracy on the validation data and reported the attack accuracy based on test data. The normalized weights we derived for CNN, LSTM and SAE were 0.35, 0.35, and 0.30 respectively on Amazon Echo dataset. We did not run 5-fold cross validation in ensemble learning as the variance of each single network was very low. For Google Home dataset, as CNN already achieved extremely high accuracy with over 99%, we did not use ensemble learning.

Comparison with Previous Studies. We compared our results with previous studies. Particularly, we compared our neural networks running the numeric format with two conventional machine learning attack methods, CUMUL [27] and CNS19 [18], on Amazon Echo dataset in the closed-world setting. We chose CUMUL as it is one of the most effective attack methods, and its accuracy is comparable with deep-learning-based methods in website fingerprinting [30]. CNS19 [18] manually selected a feature set and implemented the classifier with AdaBoost. We implemented both CUMUL and CNS19 with `Python` in our comparison.

As shown in Table 3, CNS19 achieved 76.32% accuracy on our Amazon Echo dataset in the closed-world setting, which is significantly higher than the accuracy of 33.8% reported in [18]. Since we applied the same feature set and same classifier as CNS19, this accuracy increase is likely because the size of our dataset is significantly greater than the dataset utilized in CNS19. Specifically, our dataset has 1,500 traces per class while the dataset in CNS19 only has 10 traces per class. CNS19 outperformed CUMUL and SAE but was outperformed by our CNN and LSTM in the comparison.

The Impact of The Number of Traces. Next, we evaluated attack accuracy with different sizes of data. Specifically, we kept the same 100 commands in the monitored list of Amazon Echo dataset, but we randomly selected a subset of traffic traces from each command based on a given number of traces per class. We

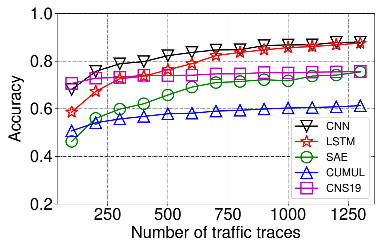


Figure 4: The impact of data size on attack accuracy.

increased the number of traces per class from 100 to 1,300 with an interval of 100. We tested attack accuracy of five methods, including CNN, LSTM, SAE, CNS19, and CUMUL for each different size. For different sizes, we used the same hyperparameters, retrained the neural networks each time based on the corresponding data.

As shown in Fig. 4, for neural networks, we observed significant improvements by utilizing a greater number of traffic traces. CNN performed the best among all the three deep learning models with every size we tested. CNS19 outperformed others only when the number of traces for each class was 100. The accuracy of CNS19 and CUMUL increased slowly and did not gain much improvement when the number of traces per class approached 1,300.

We could obtained more fine-grained results in Fig. 4 if we increase the data size with a smaller interval (e.g., 10). In that case, we would need to re-train hundreds of neural networks, which is time-consuming even with a GPU. Using the interval of 100 was sufficient for us to observe the impact of data size on attack accuracy.

6.2 Closed-World: Incoming Traffic Only

Will Our Attacks Be Effective on Human Voices? Our attack results using both outgoing and incoming traffic are promising. However, a key question we have not investigated is whether our attacks will be effective on human voices in the real world.

The encrypted traffic traces in our two datasets were triggered with five different automated voices from public Text-to-Speech APIs. Although these automated voices render a certain degree of variance in voices, human voices vary significantly due to multiple factors, including gender, age, and accent. These factors could lead to high variances in the voice data of the same voice command,

Table 4: Attack Results in the Closed-World Setting Based on Incoming Traffic Only (AE: Averaging Ensemble; WE: Weighted Ensemble; ACC: Accuracy; VAR: Variance)

Dataset	Format	CNN		LSTM		SAE		AE	WE
		ACC	VAR	ACC	VAR	ACC	VAR	ACC	ACC
Amazon Echo	Binary	24.40%	0.18×10^{-5}	24.38%	0.28×10^{-5}	24.65%	1.72×10^{-5}	24.44%	24.16%
	Numeric	81.69%	1.70×10^{-5}	**85.09%**	1.93×10^{-5}	73.77%	2.37×10^{-5}	84.41%	**86.09%**
Google Home	Binary	8.90%	0.88×10^{-5}	9.25%	0.21×10^{-5}	8.92%	0.35×10^{-5}	9.26%	**9.35%**
	Numeric	88.50%	6.97×10^{-5}	**92.24%**	7.84×10^{-5}	81.57%	3.01×10^{-5}	91.66%	**92.48%**

which may change the pattern of the traffic to the server and affect the accuracy of our attacks.

In addition, humans can ask the same intent with variations in text. For example, a human can ask *"Will it rain tomorrow?"* or *"Is it going to rain tomorrow?"*, where both commands have the same semantic intent and will receive the same response. These variations in text could also affect in the pattern of outgoing traffic.

While investigating all the variances in human voices and texts is obviously challenging (and nearly impossible), ***one of our key observations is that all these variances do not affect the incoming traffic from the server.*** Specifically, humans can ask the same commands with different voices and texts, but as long as the AI-based voice services on the server side understand correctly, the responses (as well as the incoming traffic) are not affected by these variances in voices and texts. Thus, we further conducted experiments using incoming traffic only to prove our attacks would be effective in the real world.

Our results in Table 4 shown that our neural networks are effective even considering incoming traffic only. For instance, LSTM still achieved 85.09% accuracy with the numeric format on Amazon Echo dataset. Between the numeric format and binary format, we observed that the numeric format were more effective.

For each neural network, we re-tuned hyperparameters based on incoming traffic only. The tuned hyperparameters can be found in Appendix. Our results based on incoming traffic only also indicated that the primary reason causing identifiable voice commands over encrypted traffic on smart speakers is likely because their AI-based voice services response in a deterministic or predictable manner, which leave distinguishable fingerprints in encrypted traffic.

The Impact of Different Voice Command Categories. We also evaluated the impact of the categories on attack accuracy using incoming traffic only. We separated Amazon Echo dataset into three subdatasets based on the categories of each command. We evaluated attack accuracy based on each subdataset.

According to the results, single response commands and time-sensitive response commands were easier to infer. For instance, CNN achieved 87.57% accuracy for single response commands and 88.94% accuracy for time-sensitive response commands. For commands with multiple responses, CNN still revealed significant private information with over 75% accuracy.

6.3 Open-World: Incoming Traffic Only

We evaluated the open-world setting with our Amazon Echo dataset. To keep the data balanced, we used 200 valid traces per class in the monitored list and we used all the valid traces of each class in the unmonitored list. We retrained each neural network under

Table 5: Attack Results for Different Categories in the Closed-World Setting (Incoming Traffic, Numeric Format)

	CNN	LSTM	SAE
Single	87.57%	81.50%	80.92%
Time-Sensitive	88.94%	86.67%	83.95%
Multiple	75.92%	74.46%	68.41%

the assumptions of the open-world scenario, which is a binary classification with the aim to decide whether or not a traffic trace is associated with the monitored list. We only reported the results in the open-world setting with incoming traffic only.

Table 6: Attack Results in The Open-World Setting on Amazon Echo (Incoming Traffic Only)

Format	Metric	CNN	LSTM	SAE	AE
Numeric	ACC	99.94%	100%	99.92%	**100%**
	TPR	100%	100%	99.93%	**100%**
	FPR	0.12%	0.00%	0.08%	**0.00%**
Binary	ACC	57.09%	57.56%	50.54%	56.33%
	TPR	66.04%	56.46%	47.41%	57.31%
	FPR	51.98%	41.32%	46.28%	44.67%

Our results in Table 6 show that, with data in numeric format, an attacker can decide whether a traffic trace is associated with the monitored list with an extremely high true positive rate and a very low false positive rate.

7 A DEFENSE AGAINST FINGERPRINTING

We present a proof-of-concept defense to mitigate the privacy leakage against voice command fingerprinting. It integrates two existing primitives, including adaptive padding [33] and differential privacy [47], to obfuscate traffic pattern.

Defense Details. To minimize latency, we first deploy adaptive padding in our defense. Adaptive padding, which was proposed in [33], adds dummy packets and introduces no latency. Dummy packets are inserted based on the distribution of interarrival time and each real packet is still sent at the original timestamp. As a result, it hides traffic bursts and traffic gaps. Details of adaptive padding can be found in [33]. This primitive has been used in WTF-PAD [17] as a defense in website fingerprinting. However, adaptive padding does not hide other traffic fingerprints, such as traffic length or packet size. Recent studies [35] have shown that *leveraging adaptive padding alone* is not effective against deep-learning-based attacks.

To maintain efficacy, we further obfuscate fingerprints that are not well protected by adaptive padding. First, we randomly determine the size of dummy packets based on the distribution of real packet size. Second, we extend the length of different traffic traces to identical obfuscated traffic length. Specifically, after sending the last real packet in each trace, our defense will keep producing dummy packets with adaptive padding until it reaches an obfuscated traffic length. Instead of padding all traffic traces to the same length, which is less efficient, our defense extends a trace with the traffic length of m (i.e., the total number of packets) to an obfuscated traffic length of m', where $2^{a-1} < m \leq m' = 2^a$ and a is an integer.

Third, our defense applies differential privacy to obfuscate packet size on the fly. Specifically, we leverage d^*-privacy [47] to add noise to modify packet size, where d^*-privacy is a variation of differential privacy on time-series data. With d^*-privacy, the obfuscated outputs of two identical length sequences with a distance of d are indistinguishable. Due to space limitation, details of d^*-privacy can be found in [47]. A recent study [51] has demonstrated that d^*-privacy is effective in obfuscating one-way encrypted traffic in video streams. Building upon [51], we utilize this primitive to obfuscate bi-directional traffic in our study. A high-level description of our defense of obfuscating each packet size is described in Fig. 5.

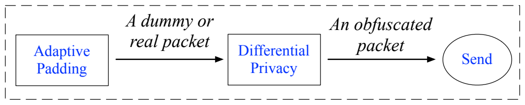


Figure 5: Our defense obfuscates each packet on the fly.

Given a (real or dummy) packet, if noise produced by d^*-privacy is positive (e.g., σ), then additional dummy data is inserted to increase the packet size (e.g., $l' = l + \sigma$); if noise is negative (e.g., $-\sigma$), then the packet size will be reduced (e.g., $l' = l - \sigma$) and a corresponding portion of a packet (e.g., σ) will be buffered until a subsequent real or dummy packet is available in a traffic trace. We implement the buffer as a queue, which sends buffered data before sending new data.

Unlike [51], which operates d^*-privacy (or differential privacy in general) over *bins*, our defense adds the noise on packets. A bin consists of packets within a fixed-size interval. It serves better for one-direction traffic in [51]. For bi-directional traffic, one time interval may have traffic on both directions, which makes it hard to aggregate as one bin and apply noise to traffic. Adding noise directly on packets is more suitable for the bi-directional traffic in our problem. Besides, this minimizes latency, as our defense does not need to buffer the packets in each bin before inserting noise.

Discussions. Adaptive padding complements differential privacy in two aspects. First, it hides traffic bursts and traffic gaps, which differential privacy alone does not. Second, for buffered data caused by negative noise, the dummy packets produced by adaptive padding can send buffered data sooner, which minimizes latency.

Assumptions on Defense. We apply our defense on both incoming and outgoing traffic to minimize the privacy leakage from the encrypted traffic. While our results in the previous section showed that incoming traffic plays a dominating role in the attacks, it is still necessary to obfuscate outgoing traffic to preserve privacy against attacks based on traditional machine learning algorithms, such as CUMUL and CNS19.

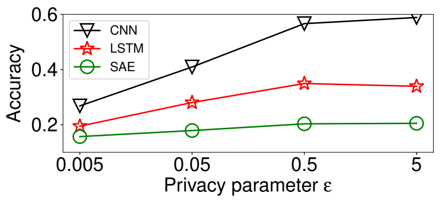


Figure 6: The impact of privacy parameter on defense with inputs in the numeric format.

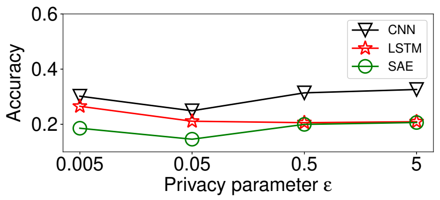


Figure 7: The impact of privacy parameter on defense with inputs in the binary format.

We assume the server will obfuscate incoming traffic, and the smart speaker (or a proxy) will obfuscate outgoing traffic. In practice, the server can calculate the distribution information of interarrival time and packet size, and other information that are required to perform the defense, and forward these information to a smart speaker. As we do not have the capability to change the current network protocol, we run simulations of our defense to generate obfuscated traffic from real traffic and demonstrate its efficacy.

8 DEFENSE EVALUATION

We implemented our defense in Python. We produced obfuscated traffic traces based on Amazon Echo dataset. The distribution of interarrival time and packet size we used in adaptive padding are generated based on Amazon Echo dataset. For differential privacy, we generated multiple versions of obfuscated datasets based on different values of privacy parameter ϵ. Privacy parameter ϵ decides the privacy protection (i.e., the noise level) rendered by differential privacy. A smaller value of ϵ generates higher noise produced and offers stronger privacy protection.

We assessed the performance of the defense in two cases: (1) ***Training with original traffic***. In this case, neural networks are trained based on original traffic traces, but test data are obfuscated; (2) ***Training with obfuscated traffic.*** In this scenario, we assume an attacker adapts to the defense, where it trains neural networks with obfuscated traffic traces and tests with obfuscated traffic traces.

As shown in Table 7, when training with original traffic, our defense can suppress attack accuracy of CNN to 1.23% in the closed-world setting, which is nearly the same as random guess. If an attacker trains a CNN with obfuscated traffic traces, it can improve its accuracy back to 26.81%, which is still significantly lower compared to the accuracy without defense. With average ensemble, an attacker can attain 28.41% accuracy (v.s. 89.41% with no defense). Our defense is also effective against CUML and CNS19.

The Impact of Privacy Parameter. In Fig. 6, we show the impact of ϵ on attack accuracy where the inputs are in the numeric

Table 7: Defense Results in The Closed-World Setting among Different Methods with ϵ=0.005

	CNN	LSTM	SAE	AE	CUMUL	CNS19
No Defense	89.05%	88.65%	75.98%	**89.41%**	61.44%	76.32%
Training with original traffic	**1.23%**	1.05%	1.12%	1.07%	1.97%	1.77%
Training with obfuscated traffic	26.81%	19.48%	15.69%	**28.42%**	17.14%	14.59%

Table 8: Tradeoffs with Different Privacy Parameter

Privacy	Latency		Bandwidth
Parameter ϵ	Per Packet (ms)	Per Trace (ms)	Overhead (KB)
0.005	16.5	136.0 (2.6%)	55.82 (138.7%)
0.05	10.4	31.4 (0.6%)	66.34 (146.0%)
0.5	7.0	17.4 (0.3%)	70.38 (148.8%)

format. When ϵ decreases, the noise level generated by differential privacy increases, which can reduce attack accuracy.

We also studied the impact of privacy parameter ϵ on attack accuracy with inputs in the binary format. As illustrated in Fig 7, we found that attack accuracy remained relatively stable when we changed privacy parameter. The reason is that changing privacy parameter ϵ does not have effect on the inputs if they are in the binary format. There are still some minor changes in accuracy as the privacy parameter changes in Fig. 7. It is because the obfuscated datasets in binary format are not exactly the same across different values of ϵ as adaptive padding is a probabilistic algorithm.

It is worth mentioning that when $\epsilon = 0.005$, *accuracy with inputs in the binary format is slightly higher than the one in the numeric format.* For instance, CNN achieved 30.18% accuracy compared to 26.81% in the numeric format. Average ensemble attained **32.18%** accuracy compared to 28.42% in the numeric format.

Latency. We examined the latency of our defense by evaluating *latency per packet* and also *latency per traffic trace*. Latency per packet indicates how many milliseconds it takes to clear the (potential) buffered data for each real packet. Latency per traffic trace suggests how many extra milliseconds it takes to complete sending all the real packets of a traffic trace. Our results in Table 8 show that the defense introduced minimal latency. The latency per packet hardly aggregated over packets as the buffered data is cleared rather soon either by dummy packets or the next real packet.

Bandwidth. The bandwidth overhead introduced by the defense is affordable. If ϵ decreases, the latency increases but bandwidth overhead decreases. The reason is that when privacy parameter is lower, noise generated by differential privacy is higher, which causes more buffered data per packet and therefore a longer latency on average. On the other hand, more buffered real data are sent by dummy packets generated by adaptive padding, which reduces the overall dummy data needed in each traffic trace.

9 LIMITATIONS AND FUTURE WORK

Human Voices. In this study, we leveraged automated voices but not human voices to trigger encrypted traffic on a smart speaker during our data collection. One of our future work is to evaluate voice command fingerprinting with human voices by considering different genders, ages and accents.

Voice Commands. We did not study popular voice commands that require interactions with other IoT devices or involve credit card transactions. For instance, asking a smart speaker to successfully order an item online with 1,500 times is challenging to perform in a lab setting.

We studied 100 popular voice commands in the closed-world setting. On the other hand, we acknowledge that the number of voice commands users could ask in practice is much greater than 100. It would be interesting to assess the privacy leakage of voice command fingerprinting on data with a much greater number of voice commands (e.g., 1,000). A more effective way of collecting data will be needed in that case.

Packet Timing. We did not leverage packet timing information [29] in our attack. It would be interesting to examine how packet timing information could be utilized in voice command fingerprinting. We will leave it as a future work.

Different Prior Probabilities. In this study, as other existing fingerprinting attacks, we assume that each class in the closed-world setting has a uniform prior probability. However, this is not the most accurate way to formulate the problem. Different voice commands could have different prior probabilities if an attacker takes into account additional background information.

For instance, given two voice commands, *Q1: "How many days until Thanksgiving?"* and *Q2: "How many days until Tax Day?"*, if it is in October, the probability of asking Q1 is obviously greater than the probability of asking Q2. On the contrary, if it is in March, then the probability of asking Q2 is clearly greater than the probability of asking Q1. Without the statistic information from voice service providers, accurately formulating the prior probabilities of different voice commands is challenging.

10 CONCLUSION

We advance the understanding of privacy impacts of smart speakers by investigating voice command fingerprinting attacks using neural networks. Our attack results show worrying privacy concerns especially using incoming traffic only. The experimental results show that our proposed defense can mitigate privacy leakage.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our shepherd, Dr. Selcuk Uluagac, for their insightful comments on this paper. UC authors were partially supported by National Science Foundation (CNS-1947913), UC Office of the Vice President for Research - Pilot Program, and Ohio Cyber Range at UC. King Hudson was partially supported by NSF LSAMP (Louis Stokes Alliances for Minority Participation) program. Wenhai Sun was partially supported by Purdue Research Foundation Summer Faculty Grant. The code, datasets, and other additional information of this study can be found at [3].

REFERENCES

[1] [n.d.]. Amazon Staff Are Listening To Alexa Conversations. https://www.forbes.com/sites/kateoflahertyuk/2019/04/12/amazon-staff-are-

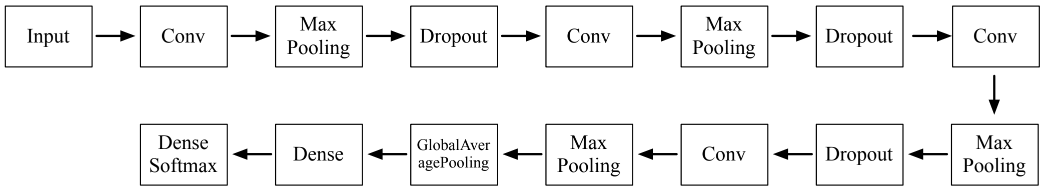


Figure 8: The structure of our CNN.

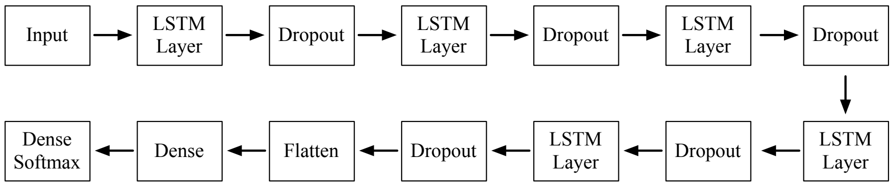


Figure 9: The structure of our LSTM.

Table 9: Tuned Hyperparameters of Each Neural Network with Both Outgoing and Incoming Traffic in the Numerical Format

Hyperparameters	Search Space	CNN	LSTM	SAE
Input Dimension	{300, 325, 350, ..., 575, 600}	475	350	375
Optimizer	{Adam, SGD, Adamax, Adadelta}	Adamax	Adamax	Adam
Learning Rate	{0.001, 0.002, 0.01, 0.05, 0.1}	0.002	0.002	0.001
Decay	{0.00, 0.01, 0.02, ..., 0.50}	0.13	0.19	0.30
Batch Size	{30, 40, 50, ..., 120, 130}	70	130	110
Activation Function	{softsigh, tanh, elu, selu}	[tanh; elu; selu || selu]	[tanh; tanh; tanh; tanh || selu]	[elu; tanh; selu; elu; softsign || tanh]
Dropout	{0.0, 0.1, 0.2, 0.3, 0.4, 0.5}	[0.1; 0.3; 0.1; 0.0]	[0.4; 0.1; 0.1; 0.3; 0.5]	[0.2; 0.0; 0.0; 0.3]
Dense Layer Size	{100, 110, 120, ..., 170, 180}	180	70	130
Convolution Number	{16, 32, 64, 128, 256}	[128; 128; 64; 256]	–	–
Filter Size	{7, 9, 11, ..., 25, 27}	[7; 19; 13; 23]	–	–
Pool Size	{1, 3, 5, 7}	[1; 1; 1; 1]	–	–
LSTM Layer Size	{90, 100, 110,..., 300, 310}	–	[210; 190; 190; 190; 130]	–
SAE Encoder Layer Size	{200, 210, ..., 390, 400}	–	–	[330; 260; 330; 280; 250]

Table 10: Tuned Hyperparameters of Each Model with Incoming Traffic Only in the Numerical Format

Hyperparameters	Search Space	CNN	LSTM	SAE
Input Dimension	{300, 325, 350, ..., 575, 600}	450	500	350
Optimizer	{Adam, SGD, Adamax, Adadelta}	Adam	Adamax	Adadelta
Learning Rate	{0.001, 0.002, 0.01, 0.05, 0.1}	0.002	0.002	1.0
Decay	{0.00, 0.01, 0.02, ..., 0.50}	0.50	0.20	0.30
Batch Size	{30, 40, 50, ..., 120, 130}	150	170	130
Activation Function	{softsigh, tanh, elu, selu}	[tanh; selu; elu; selu || selu]	[tanh; tanh; tanh; tanh || elu]	[elu; selu; selu; softsign; tanh || elu]
Dropout	{0.0, 0.1, 0.2, 0.3, 0.4, 0.5}	[0.2; 0.1; 0.4; 0.5]	[0.1; 0; 0.1; 0; 0.1, 0.5]	[0.1; 0.0; 0.0; 0.0]
Dense Layer Size	{100, 110, 120, ..., 170, 180}	140	150	160
Convolution Number	{16, 32, 64, 128, 256}	[256; 32; 128; 32]	–	–
Filter Size	{7, 9, 11, ..., 25, 27}	[9; 9; 11; 15]	–	–
Pool Size	{1, 3, 5, 7}	[3; 2; 1; 2]	–	–
LSTM Layer Size	{90, 100, 110,..., 300, 310}	–	[170; 290; 170; 90; 250]	–
SAE Encoder Layer Size	{200, 210, ..., 390, 400}	–	–	[330; 290; 270; 250; 220]

listening-to-alexa-conversations-heres-what-to-do

[2] [n.d.]. Smart Speaker Market, Worth USD 11.79 Billion by 2023. https://www.marketsandmarkets.com/

[3] [n.d.]. Voice Command Fingerprinting with Deep Learning. https://github.com/SmartHomePrivacyProject/DeepVCFingerprinting

[4] H. Abdullah, W. Garcia, C. Peeters, P. Traynor, K. R. B. Butler, and J. Wilson. 2019. Practical Hidden Voice Attacks against Speech and Speaker Recognition Systems. In *Proc. of NDSS'19.*

[5] K. Abe and S. Goto. 2016. Fingerprinting Attack on Tor Anonymity Using Deep Learning. In *Proc. of Aisa Pacific Advanced Network (APAN).*

[6] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A. R. Sadeghi, and A. Selcuk Uluagac. [n.d.]. Peek-a-Boo: I see your smart home activities, even encrypted! ([n. d.]). https://arxiv.org/pdf/1808.02741.pdf.

[7] A. Alshehri, J. Granley, and C. Yue. 2020. Attacking and Protecting Tunneled Traffic of Smart Home Devices. In *Proc. of ACM CODASPY'20.*

[8] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster. 2019. Keeping the Smart Home Private with Smart(er) IoT Traffic Shaping. In *Proc. of PETS'19.*

[9] M. Backes, G. Doychev, M. Durmuth, and B. Kopf. 2010. Speaker Recognition in Encrypted Voice Streams. In *Proc. of ESORICS'10.*

[10] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. 2018. Behavioral Fingerprinting of IoT Devices. In *Proc. of Workshop on Attacks and Solutions in Hardware Security.*

[11] S. Bhat, D. Lu, A. Kwon, and S. Devadas. 2019. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. In *Proc. of PETS'19.*

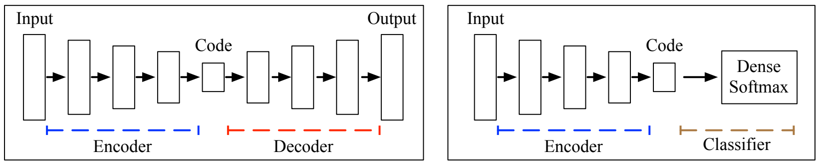


Figure 10: The structure of our SAE. Encoder and code are trained in training (left) and used in classification (right).

[12] Kevin P. Dyer, Scott E. Coull, T. Ristenpart, and T. Shrimpton. 2012. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *Proc. of IEEE S&P'12*.

[13] J. Hayes and G. Danezis. 2016. K-Fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *Proc. of USENIX Security'16*.

[14] D. Hermann, R. Wendolsky, and H. Federrath. 2009. Website Fingertinging: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Proc. of ACM Workshop on Cloud Computing Security.*

[15] M. Imani, M. S. Rahman, N. Mathews, and M. Wright. [n.d.]. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks with Adversarial Traces. ([n. d.]). https://arxiv.org/abs/1902.06626.

[16] H. Jafari, O. Omeotere, D. Adesina, H. Wu, and L. Qian. 2018. IoT Devices Fingerprint Using Deep Learning. In *Proc. of IEEE MILCOM'18*.

[17] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright. 2016. Toward an Efficient Website Fingerprinting Defense. In *Proc. of ESORICS'16*.

[18] S. Kennedy, H. Li, C. Wang, H. Liu, B. Wang, and W. Sun. 2019. I Can Hear Your Alexa: Voice Command Fingerprinting on Smart Home Speakers. In *Proc. of IEEE CNS'19*.

[19] K. Kohls, D. Rupprecht, T. Holz, and C. Popper. 2019. Lost Traffic Encryption: Fingerprinting LET/4G Traffic on Layer Two. In *Proc. of Wisec'19*.

[20] D. Kumar, R. Paccagnella, P. Murley, E. Hennenfent, J. Mason, A. Bates, and M. Bailey. 2018. Skill Squatting Attacks on Amazon Alexa. In *Proc. of USENIX Security'18*.

[21] Y. LeCun, Y. Bengio, and G. E. Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436 – 444.

[22] M. Liberatore and B. N. Levine. 2006. Inferring the Source of Encrypted HTTP Connections. In *Proc. of ACM CCS'06*.

[23] M. H. Mazhar and Z. Shafiq. [n.d.]. Characterizing Smart Home IoT Traffic in the Wild. ([n. d.]). https://arxiv.org/pdf/2001.08288.pdf.

[24] N. Msadek, R. Soua, and T. Engel. 2019. IoT Device Fingerprinting: Machine Learning based Encrypted Traffic Analysis. In *Proc. of IEEE WCNC'19*.

[25] S. E. Oh, S. Sunkam, and N. Hopper. 2019. p-FP: Extraction, Classification, and Predication of Website Fingerprints. In *Proc. of PETS'19*.

[26] D. Opitz and R. Maclin. 1999. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* (1999).

[27] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Penekamp, K. Wehrle, and T. Engel. 2016. Website Fingerprinting at Internet Scale. In *Proc. of NDSS'16*.

[28] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. 2011. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proc. of Workshop on Privacy in the Electronic Society.*

[29] M. S. Rahman, P. Sirinam, N. Mathews, K. G. Gangadhara, and M. Wright. 2020. Tik-Tok: The Utility of Packet Time in Website Fingerprinting Attacks. In *Proc. of PETS'20*.

[30] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen. 2018. Automated Website Fingerprinting through Deep Learning. In *Proc. of NDSS'18*.

[31] N. Roy, S. Shen, H. Hassanieh, and R. R. Choudhury. 2018. Inaudible Voice Commands: The Long-Range Attack and Defense. In *Proc. of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18).*

[32] R. Schuster, V. Shmatikov, and E. Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *Proc. of USENIX Security'17*.

[33] V. Shamtikov and M. H. Wang. 2006. Timing Analysis in Low-Latency Mix Networks: Attacks and Defenses. In *Proc. of ESORICS'06*.

[34] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom. 2019. Robust Website Fingerprinting Through the Cache Occupancy Channel. In *Proc. of USENIX Security'19*.

[35] P. Sirinam, M. Imani, M. Juarez, and M. Wright. 2018. Deep Fingerprinting: Understanding Website Fingerprinting Defenses with Deep Learning. In *Proc. of ACM CCS'18*.

[36] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright. 2019. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning. In *Proc. of ACM CCS'19*.

[37] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing Properties of Neural Networks. In *Proc. of ICLR'14*.

[38] V. Thangavelu, D. M. Divakaran, R. Sairam, S. S. Bhunia, and M. Gurusamy. 2019. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet of Things Journal* (2019).

[39] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky. 2020. Packet-Level Signatures for Smart Home Devices. In *Proc. of NDSS'20*.

[40] A. S. Uluagac, S. V. Radhakrishnan, C. Corbett, A. Baca, and R. Beyah. 2013. A Passive Technique for Fingerprinting Wireless Devices with Wired-Side Observations. In *Proc. of IEEE CNS'13*.

[41] T. Wang, X. Cui, R. Nithyannand, R. Johnson, and I. Goldberg. 2014. Effective Attacks on Proable Defenses for Website Fingerprinting. In *Proc. of 23rd USENIX Security Symposium.*

[42] T. Wang and I. Goldberg. 2017. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *Proc. of USENIX Security'17*.

[43] X. Wang, S. Chen, and S. Jajodia. 2005. Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In *Proc. of ACM CCS'05*.

[44] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose. 2011. Phonotactic Reconstruction of Encrypted VoIP Conversations: Hookt on Fon-iks. In *Proc. of IEEE S&P'11*.

[45] C. Wright, L. Ballard, F. Monrose, and G. M. Masson. 2007. Language Identification of Encrypted VoIP Traffic: Alejandra y Roberto or Alice and Bob?. In *Proc. of USENIX Security'07*.

[46] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. 2008. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *Proc. of IEEE S&P'08*.

[47] Q. Xiao, M. K. Reiter, and Y. Zhang. 2015. Mitigating Storage Side Channels Using Statistical Privacy Mechanisms. In *Proc. of ACM CCS'15*.

[48] X. Yuan, Y Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter. 2018. CommandSong: A Systematic Approach for Practical Adversarial Voice Recognition. In *Proc. of USENIX Security'18*.

[49] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. 2017. DolphinAttack: Inaudible Voice Commands. In *Proc. of ACM CCS'17*.

[50] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian. 2019. Dangerous Skills: Understanding and Mitigating Security Risks of Voice-Controlled Third-Party Functions on Virtual Personal Assistant Systems. In *Proc. of IEEE S&P'19*.

[51] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang. 2019. Statistical Privacy for Streaming Traffic. In *Proc. of NDSS'19*.

[52] Y. Zhang, L. Xu, A. Mendoza, G. Yang, P. Chinprutthiwong, and G. Gu. 2019. Life after Speech Recognition: Fuzzing Semantic Misinterpretation for Voice Assistant Applications. In *Proc. of NDSS'19*.

APPENDIX

Tuned Hyperparameters. For the search space of each hyperparameter in Table 9 and Table 10, we represent it as a set. We searched for learning rate and decay values if the optimizer is Stochastic Gradient Decent (SGD). If the tuned optimizer is not SGD, we used the default learning rate and decay provided by `Keras`. For the activation functions, dropout, filter size and pool size, we searched for hyperparameters at each layer. For each of these, the tuned parameters we report in the table are presented as a sequence of values by following the order of layers we presented in Fig. 8, Fig. 9, and Fig. 10. For instance, for our CNN, the tuned activation functions are `tanh` (1st Conv), `elu` (2nd Conv), `elu` (3rd Conv) and `selu` (4th Conv). The `selu` after symbol || in the table means that the second to last dense layer in our CNN uses `selu` as its activation function. We did not include `relu` as one of the activation functions in the search space. It is because `relu` maps all the negative values (the sizes of all the incoming packets) to 0s, which is not suitable for traffic analysis and has been pointed out in a previous study [35]. Due to space limitation, we skip the tuned hyperparameters in the binary format.