iPET: Privacy Enhancing Traffic Perturbations for IoT Communications

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY-NC-SA 4.0

SUBMISSION DATE / POSTED DATE

25-02-2022 / 03-03-2022

CITATION

Shenoi, Akshaye; Karthik, Prasanna; Sabharwal, Kanav; Jialin, Li; Divakaran, Dinil Mon (2022): iPET: Privacy Enhancing Traffic Perturbations for IoT Communications. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.19235742.v1>

DOI

[10.36227/techrxiv.19235742.v1](https://doi.org/10.36227/techrxiv.19235742.v1)

iPET: Privacy Enhancing Traffic Perturbations for IoT Communications

Akshaye Shenoi*
TUMCREATE Singapore
akshaye.shenoi@tum-create.edu.sg

Prasanna Karthik*
National University of Singapore
prasanna@comp.nus.edu.sg

Kanav Sabharwal*
National University of Singapore
kanav.sabharwal@u.nus.edu

Li Jialin
National University of Singapore
lijl@comp.nus.edu.sg

Dinil Mon Divakaran
Trustwave
dinil.divakaran@trustwave.com

Abstract—IoT devices are vulnerable to different kinds of threats and attacks. The devices constantly communicate with servers over the Internet, allowing an attacker to extract sensitive information by passively monitoring the network traffic. Recent research works have shown that a network attacker with a trained machine learning (ML) model can accurately fingerprint IoT devices based on the (encrypted) traffic flows of the devices. Such fingerprinting attacks are capable of revealing the make and model of the devices, which can further be used to extract detailed user activities.

In this work, we develop and propose iPET, a privacy enhancing traffic perturbation technique that counters ML-based fingerprinting attacks. iPET uses adversarial deep learning, specifically, Generative Adversarial Networks (GANs), to generate these perturbations. Unlike conventional GANs, a key idea of iPET is to deliberately introduce stochasticity in the model. This approach inhibits an attacker from recreating an identical perturbation model and using it for fingerprinting. We evaluate the effectiveness of our defense against state-of-the-art fingerprinting models three different attacker capabilities. Our evaluations on synthetic and real-world datasets demonstrate that iPET decreases the accuracy of even the most powerful attacker significantly.

Index Terms—IoT, privacy, fingerprinting, deep learning, adversarial machine learning

I. INTRODUCTION

Internet of Things (IoT) devices are fast penetrating into different markets such as consumer, enterprise, critical infrastructure, education, and healthcare. The number of IoT devices in the world is projected to go up to 41 billion by 2027, a steep rise from 8 billion in 2019 [1]. IoT devices have become part of our daily lives; we use them to track and monitor heart beat rate, sleep quality, sports activities, etc., as well as for smart, simplified and automated operations such as controlling home environment and ordering groceries. This unprecedented growth is, however, accompanied with increasing threats and attacks [2]. In particular, user privacy is vulnerable to eavesdropping attack — a network attacker that has access to the communication traffic of IoT devices can identify the device types and even device activities.

Advances in machine learning (ML) has largely benefited our society; unsurprisingly, they have also become a useful

tool for attackers. Many recent attacks leverage machine learning models to target the privacy of IoT devices [3], [4], [5], [6], [7], [8], [9]. These ML-based *device fingerprinting attacks* are capable of identifying devices based on the communication patterns captured by a trained model, even when the payloads are encrypted. This leads us to an important question – how can we protect the privacy of IoT users from network attackers that use state-of-the-art ML-based fingerprinting attacks?

Researchers have shown that *adversarial inputs* can effectively deceive ML-based classification models [10], [11], [12], [13]. Recent works [14], [15] have extended these techniques to defeat website fingerprinting classifiers [16], [17], [18]. By crafting adversarial inputs in the form of traffic perturbations, these defenses effectively force misclassifications on the attacker’s models. Applying similar techniques to defend IoT fingerprinting, however, is not straightforward: in a realistic scenario, the attacker, knowing how the adversarial inputs are generated by the defense, may retrain their model to beat the new defense. Therefore, a challenge in developing an ML-based defense is to make it impractical for an attacker to learn, retrain, and counter the defense.

In this work, we design and develop iPET, a privacy-enhancing technique for IoT devices, that leverages adversarial ML to perturb network traffic and thereby defeat fingerprinting attacks. iPET employs a deep learning-based Generative Adversarial Network (GAN) model that creates two deep neural networks, namely the generator (of perturbations) and the discriminator (that fingerprints), that compete with each other to create better versions of themselves. However, an inherent drawback of conventional GANs is that an adversary could also recreate the GAN at their end based on the traffic from their own IoT devices, and subsequently use the discriminator for fingerprinting. The key idea of iPET is that the GAN models created at the defense (and also an adversary) can be made to diverge significantly by carefully choosing the objective function. Unlike conventional GANs that have an objective of misclassifying the source label (e.g., Device A) to a particular target label (e.g., Device B), iPET is more accommodating in that it does not restrict misclassification to a particular target label. This design choice introduces more degrees of freedom in coming up with a GAN model, effectively ensuring that it is impractical for an adversary

*Authors contributed equally.

Fig. 1: Threat model, where an IoT device communicates with a server in the Internet. The attacker could be any router on the communication path with access to encrypted traffic exchanged between the IoT devices and their servers.

to accurately recreate the model used for defense. To the best of our knowledge, iPET is the first solution to use adversarial ML to protect privacy of IoT users. We summarise the contributions of our work below.

- We develop and present iPET, a novel technique that exploits the stochasticity between instances of generative models to add adversarial perturbations in a way that undermines ML-based device fingerprinting models.
- We implement iPET in Python using Keras [19] libraries from TensorFlow [20], and make the source code available online¹.
- We carry out comprehensive evaluations using both synthetic data as well as real network traces of IoT devices. We evaluate common defenses such as Tor and VPN, and compare against iPET. Going further, we also consider different attacker capabilities. Our results demonstrate that iPET is not only significantly more effective than Tor and VPN in the face of a network attacker, but is also resilient against a powerful attack who retrain his fingerprinting model based on the knowledge of our defense solution.

Next we present the background necessary for our work and formally define the threat model. In Section III, we present the goals and challenges in developing a defense solution, and provide an overview of iPET. We develop iPET in Section IV and analyze its resilience to various attack scenarios. Section V presents several real-world deployment options of iPET. In Section VI, we pose important research questions on countering device fingerprinting, and evaluate iPET in this context. Related works are discussed in Section VII.

II. BACKGROUND AND THREAT MODEL

A. System: Users and Attacker

We consider a system where an IoT device communicates with servers on the Internet for routine operations. IoT devices connect to the Internet often with the help of a gateway (e.g., home IoT gateway) as shown in Figure 1. The potential victims here are the users of IoT devices, who are concerned about privacy leaks on the Internet links, despite the communications being encrypted.

We assume that an attacker may control any network device (e.g., router) on the communication path between an IoT

device and its server, except the user gateway and the server, which are assumed to be secured and hardened. The attacker may do so by compromising a network device (e.g., a router) belonging to an Internet Service Provider (ISP), or the attacker could just be a malicious ISP. The attacker is capable of passively eavesdropping on both the incoming and outgoing communications at the network device. However, the attacker does not have access to home networks of users and their internal traffic; we consider such attacks out of scope of the current work. We also assume that DDoS attacks on IoT devices or gateways is out of scope of our work, since we consider only a passive attacker and attacks on user privacy.

B. Attacker: Goals and Capabilities

The attacker’s objective is to infer sensitive information about the users of the IoT devices from the traffic captured at the network entity under his control. The attacker uses a machine learning (ML) classifier to carry out *device fingerprinting attack* — a network attack that infers the type (make and model) of the IoT device [3], [4], [5], [6], [7], [8], [9]. Device fingerprinting could be the first step of more serious attacks, such as those that infer fine-grained sensitive information such as IoT user activity [21]. To build this ML classifier, the attacker may purchase his own IoT devices, generate (potentially encrypted) traffic from them, label them to subsequently extract *features* and train a device-fingerprinting model. This trained model is employed on traffic of users to classify and identify IoT devices. Since packet payloads are often encrypted (e.g., due to the increasing adoption of TLS [22]), most of the features for an ML classifier come from the header fields of the packets. Features may be extracted for each connection (e.g., HTTP connection), or for a session (defined by static or dynamic time-window) of a device. To be specific, in this paper, we define two fingerprinting models for identifying IoT devices:

i) $\mathcal{M}_{agg}$: **model based on feature aggregates**. The features of this model are extracted from sessions of fixed-length time-windows of a device. Existing works select time-windows ranging in minutes to achieve high fingerprinting accuracy (e.g., DEFT [5] uses a window length of 15 minutes). For a given session, the features include, the number of packets, statistics (mean, minimum, maximum, and sum) of the packet lengths, etc. The features are aggregated separately for incoming and outgoing packets of a device (IP address) and further grouped by the different protocols (e.g. HTTP, UDP, etc.). Appendix A gives the full list of features. We use a Random Forest classifier to train the model in a supervised way.

ii) $\mathcal{M}_{seq}$: **sequence-based device classifier**. With this model, the goal is to capture the temporal relationships between packets in a device’s traffic session. For readability, we denote the features of a session as a matrix τ of shape $[f, n]$, where each of the n columns corresponds to a discrete time-slot ω_i (say, of length 100ms) in a session. Essentially, τ captures the session as a time-series; e.g., if session-length is 1s, there would be 10 columns in τ . We consider f features (rows of τ); they represent the number of incoming/outgoing packets

¹<https://github.com/akshayeshenoi/ipet>

Fig. 2: Model architecture of $\mathcal{M}_{\text{seq}}$. The fingerprinting model is trained to classify $\mathcal{C}$ devices.

in ω_i , the sum of lengths of incoming/outgoing TCP payloads in ω_i , and the sum of lengths of incoming/outgoing UDP payloads in ω_i . Although these features appear simple, yet they are known to be effective in fingerprinting IoT devices [23], [6]; besides they can be extracted with minimal processing. Additionally, we compute the silence period s between two consecutive sessions (time between the first packet of current session and last packet of previous session) as another feature. We train a Bidirectional Recurrent Neural Network [24] (see Figure 2) to capture the temporal relations within a session. The silence period s is concatenated to the output of the Bi-LSTM layer, which is then fed to a dense layer. Finally, the softmax layer normalizes the probability distribution predicted by the model. As will be seen in Section V, systems such as Tor and VPN hide many of the features used by $\mathcal{M}_{\text{agg}}$. In such scenarios, $\mathcal{M}_{\text{seq}}$ performs better than $\mathcal{M}_{\text{agg}}$. Another advantage of $\mathcal{M}_{\text{seq}}$ model is that it achieves high accuracy with short time-windows (e.g., see [23] and our evaluations in Section VI-A).

Furthermore, we endow our attacker with the knowledge of any defense that an IoT device user may have in place. For instance, when a user employs Tor for defense, we assume that an attacker would be able to infiltrate the overlay network and take control of a Tor relay node. Similarly, if the user employs a traffic perturbation technique for defense, we assume that the attacker may deploy the defense at his own premises, route the traffic of his devices through the defense, obtain the perturbed traffic and subsequently retrain his ML classifier. We elaborate the attack scenarios in Section IV-A.

III. iPET: GOALS, CHALLENGES, AND OVERVIEW

A. Defense Goal and Challenge

Consider the communications between an IoT BP Monitor and its server through the network path, namely, IoT Device $\rightarrow R_0 \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow \text{Server}$, as shown in Figure 3. The attacker is located at R_2 and is capable of eavesdropping the communications along the network path $R_0 \rightarrow \text{Server}$, to perform device fingerprinting. The goal of the IoT user (who uses a defense solution) is to effectively counter the ML-based fingerprinting attacks carried out by the network attacker. This means, the defense should be able to significantly reduce the probability of successful device classification from the currently achievable very high accuracy (of over 99%, as per our experiments in Section VI-A).

Fig. 3: iPET, the proposed traffic perturbation technique employed at the periphery (i.e., R_0 and Server) of the network that is under threat from the attacker.

As mentioned in Section II-B, an attacker trains a fingerprinting model by extracting information from network traffic; such a model learns specific characteristics of traffic (even if the payloads are encrypted), to construct decision boundaries that differentiate the patterns corresponding to different devices. To prevent this, the defense employed by the user should be able to obfuscate traffic by adding noise. For instance, the solution could add noise in the form of cover traffic (i.e., dummy packets with arbitrary but realistic sizes) to counter a fingerprinting model that uses packet count and size as features.

Challenge: The challenge in coming up with a traffic obfuscation technique is that it must be resilient to attackers that are aware of the defense solution. In particular, an attacker that learns that a specific defense solution in place, may retrain his model accordingly to defeat the defense solution. In Section VI-A, we show that Tor [25], a popular privacy enhancing overlay network, is easily defeated with our sequence-based fingerprinting model $\mathcal{M}_{\text{seq}}$ trained on Tor output ($\mathcal{M}_{\text{seq}}$ achieves $\approx 81\%$ accuracy.)

B. Overview of iPET

iPET is a deep learning-based traffic perturbation technique, and consists of a Perturber and an Extractor. iPET Perturber, which adds traffic perturbations, is deployed at the ingress of the network path that is vulnerable to fingerprinting attacks. Likewise, complementary to this, the iPET Extractor removes the perturbations at the egress of the vulnerable network path. A perturbation vector specifies (i) the cover traffic to be added in the form of dummy packets, (ii) the additional bytes to be added to the packets, and (iii) the small and limited delays to be added to the packets from the IoT devices.

For illustration, we assume a simplified deployment model where iPET Perturber is deployed at the home Gateway (R_0) and the iPET Extractor is deployed at the Server as shown in Figure 3. Real-world deployments must consider the trust assumptions of the involved network nodes and, additionally, consider real-world cases that may require routing the traffic through multiple hops. We discuss these in Section V.

We present the design of iPET progressively. iPET-Basic (Section IV-B) is a generative model that creates perturbation vectors, which when added to the original device traffic, causes the attacker's fingerprinting model to misclassify. This means that the fingerprinting model's inference of the make and model of the devices fails with a very high probability. However, this form of defense is still vulnerable against an attacker who retrain his device fingerprinting model by deploying iPET at their end (refer Section IV-B3). To thwart such *retraining attacks*, we present an enhanced defence iPET-Advanced (Section IV-C), that uses an iterative process of training iPET perturbation generator in tandem with the attacker's fingerprinting model, simulating the retraining process over multiple rounds. Our neural network model is designed to enhance stochasticity between trained instances, which makes it extremely difficult for an attacker to recreate the defensive model at their end, thereby preventing retraining attacks. In fact, we go a step further and show that iPET-Advanced is resilient against an even more powerful attacker (discussed in Section IV-A) that may retrain on *multiple* such iPET-Advanced defense models to learn its parameters and architecture and additionally, account for the stochasticity in iPET training.

IV. iPET

iPET uses a deep learning model to generate traffic perturbations, which when added to device traffic, will likely lead to misclassification of the attacker's ML classifier that is used for device fingerprinting. The input to iPET is a one-dimensional vector $\xi \in [0, 1]^k$ ($k = 20$) drawn from a Uniform distribution, and the output is a two-dimensional perturbation matrix δ of shape $[m, n]$; Figure 4 illustrates this. The i^{th} column of this matrix specifies a perturbation vector, which determines how the traffic should be perturbed at a discrete slot ω_i , where $1 \leq i \leq n$. Each row corresponds to a particular type of perturbation (e.g., addition of dummy packets); and m is a function of this. The values in the perturbation matrix are used by an iPET gateway for a session of length $n \times |\omega|$. The perturbation vector for a time-slot stipulates the following types of perturbations: 1) the number and size of dummy packets to be added in every discrete slot, and 2) the amount of delay (in milliseconds) for each packet, and 3) padding (in bytes) to be added for each packet. iPET trains on the traffic of IoT devices to generate perturbations that will obfuscate potentially discernible patterns over sessions of fixed length ($n \times |\omega|$).

We introduce the attack scenarios in Section IV-A. We describe the foundation of iPET design, namely, iPET-Basic in Section IV-B. Subsequently, we develop iPET-Advanced, which is the final model used as iPET.

A. Attack Scenarios

i) Naïve attacker ($\mathcal{M}_{seq}^{naive}$). A Naïve attacker deploys his own IoT devices, captures their (possibly encrypted) traffic at his IoT gateway, and labels them. He uses this labeled dataset for training a sequence-based classifier $\mathcal{M}_{seq}^{naive}$ (same as $\mathcal{M}_{seq}$,

Fig. 4: The input to iPET is a random vector ξ , and the output is a perturbation matrix δ that specifies, for a specific time-slot ω_i , i) the number of dummy packets along with the packet sizes to be added, ii) the amount of bytes to pad on to packets and iii) the delay to be added to each packet.

defined in Section II-B). As we show later in Section VI-A1 sequence-based classifiers perform better than aggregate ones.

ii) Retraining attacker ($\mathcal{M}_{seq}^{retrained}$). Different from the Naïve attacker, a Retraining attacker deploys iPET on an IoT gateway, gaining blackbox access to the iPET gateway. In other words, the attacker deploys a set of devices, captures perturbed traffic from the egress of iPET gateway, and retrain his fingerprinting classifier. The attacker uses this iPET-aware ML classifier on (victim) user traffic for device fingerprinting. We denote this retrained sequence-based classifier as $\mathcal{M}_{seq}^{retrained}$.

iii) Powerful attacker ($\mathcal{M}_{seq}^{powerful}$). In this scenario, the attacker operates *several* iPET gateways. He can now deploy IoT devices at each of his iPET gateways, capture their traffic, and label them. The attacker builds a more robust ML classifier that is trained on the traffic traces from several iPET instances running at the gateway he controls. The new ML classifier thus trained is called $\mathcal{M}_{seq}^{powerful}$. Additionally, we also assume that the attacker has learned the model parameters and hyper parameters of iPET model (Section IV-C) from the several instances that he has access to. However, he does not have access to the exact weights of the neural network models trained at the iPET instances for the different users.

B. iPET-Basic

The design of iPET-Basic is inspired by Generative Adversarial Perturbations [12], which utilizes a generative network (i.e., a generator) to formulate perturbations for defeating an image classifier (i.e., a discriminator) with high confidence. For instance, when a perturbation from the generator is added to an image of a dog, the discriminator will label it as a cat with a high probability. Analogous to this, in our work, iPET-Basic is the generator ($\mathcal{G}$) and the discriminator ($\mathcal{D}$) is the naïve attacker ($\mathcal{M}_{seq}^{naive}$). The basic idea is that $\mathcal{G}$ progressively learns to generate better perturbations that defeat $\mathcal{D}$.

1) Model Design: Model Architecture. The generator's model architecture is described in Fig 5. We utilize a dense layer, where each neuron learns an element in the perturbation matrix δ . For example, the number of packets to be added in the discrete slot is learned by a neuron. The dense layer output is reshaped into the perturbation matrix δ . The custom limit layer is included to scale the perturbation values within

Fig. 5: Model architecture of the generator.

Fig. 6: Overview of generator training.

the constraints imposed either by the user (e.g., for saving bandwidth) or the network protocol (e.g., the maximum allowed payload size). We impose the following constraints in the custom limit layer:

- Control the number of dummy packets to add, and their sizes, per discrete slot (ω_i).
- Ensure that the payload bytes do not result in the packet size exceeding the MTU (Maximum Transmission Unit) imposed by the network.
- Ensure that the delay (in milliseconds) added to original packets is within the user specified limits.

We define control parameters for the above, which help iPET to achieve a trade-off between privacy and bandwidth overheads, as we demonstrate later in Section VI-A3.

Training. The objective of iPET-Basic is to generate perturbations $\mathcal{G}(\xi) = \delta$, which when added to an input time series τ , creates an adversarial input $\tilde{\tau} = \tau \oplus \delta$, that causes the discriminator $\mathcal{D}$ to misclassify the original traffic session τ . An example of the working of $\oplus$ operator is shown in Figure 4. The perturbed traffic, along with the silence period s of the device, is now used as an input to the discriminator, as summarised in Figure 6. The discriminator predicts the device type for the given input. The prediction is used as a feedback to update the generator weights, and this process continues.

There are two well-known approaches to generate perturbations. i) Targeted approach [14], [12]: conventionally, in GAN-based systems, the generator attempts to force the discriminator to classify an input into a specific wrong target class. In such a case, the GAN is trained iteratively against the discriminator until this objective is reached. ii) Untargeted approach [15], [12]: the generator does not have a well-defined goal of forcing a misclassification into a particular class label.

Instead, the generator receives a positive feedback as long as the discriminator misclassifies the input. The open-ended nature of the training objective could likely result in a different iPET generator being created each time the model is trained. This is mainly due to freedom in misclassifying an input into any of the available classes. We therefore choose the untargeted approach for our work. Thus, we train our generator $\mathcal{G}$ such that,

$$\mathcal{D}(\tilde{\tau}, s) \neq c_{\tau, s}, \quad (1)$$

where τ is a traffic session (input time series of feature values), s is the silence period of the IoT device, and $c_{\tau, s}$ is the ground truth label.

Loss function. The loss function can thus be defined as a decreasing function of categorical cross-entropy, denoted by

$$l = \sum_{i=1}^c y_i \log(\hat{y}_i), \quad (2)$$

where, $\hat{y}$ is the model output and y is the corresponding ground truth label, both of which are one-hot encoded. Following common practice, the model is trained for sufficient number of epochs to learn the perturbations that can deceive the discriminator with high confidence.

2) Stochastic Differences between Generator Instances:

In iPET, every user gets a different instance of iPET-Basic installed at their iPET gateway. The different instances (e.g., $\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^{\#users}$) of iPET-Basic have the following similarities. i) The discriminator $\mathcal{D}$ (refer Figure 6), which is used for training the different iPET-Basic generator instances, is the same. ii) The model architecture and the hyper-parameters of the generator instances are also the same. However, due to their initial random weights and the untargeted training objective, the final weights of the generators are likely to be different. Thus, effectively, the perturbations generated by the instances at a user ($\mathcal{G}^{user}$) and an attacker ($\mathcal{G}^{adv}$) are different; this is illustrated in Figure 7(a).

3) **Attackers against iPET-Basic: Naïve attacker.** In this scenario, the attacker uses the discriminator $\bar{\mathcal{D}} = \mathcal{M}_{seq}^{naive}$ to carry out fingerprinting attack against a user that employs iPET-Basic generator at her premises.

Retraining attacker Figure 7(a) depicts generator instances at the user ($\mathcal{G}^{user}$) and the attacker ($\mathcal{G}^{adv}$). Subsequently, as show in Figure 7(b), the attacker, at his end, retrains a new discriminator ($\bar{\mathcal{D}} = \mathcal{M}_{seq}^{retrained}$) using the instance of iPET-Basic generator ($\mathcal{G}^{adv}$) available on his gateway, with an intention of fingerprinting devices deployed by the user.

C. iPET-Advanced

We first present the design of iPET-Advanced followed by a discussion on the attack scenarios against iPET-Advanced.

1) **Model Design:** iPET-Advanced is build with iPET-Basic's generator architecture (refer Section IV-B) as the fundamental building block. The generator uses an untargeted training objective to ensure that the perturbations generated by each iPET-Advanced model is different. The goal of iPET-Advanced is to ensure that the perturbations of the generator instances at the gateways differ *significantly* from one another.

Fig. 7: (a) Two instances of iPET-Basic generators, one at a user and another at an attacker. (b) The Retraining attacker builds a stronger discriminator $\bar{D}$ based on G^{adv} at his own iPET gateway.

Fig. 8: (a) The training process for creating generator instance G_i^{user} , and its deployment at a user. (b) The training process for creating a generator instance G_j^{adv} at an attacker, from which he builds his discriminator $\bar{D}$.

Intuitively, the idea is to make it challenging for an attacker to use the decision boundaries that he learns from his own generator instance for employing against the traffic captured at another iPET user.

As with iPET-Basic, we use a GAN-based architecture for iPET-Advanced. The distinct and important design principle of iPET-Advanced is to chain a series of discriminator-generator sequences such that, at each stage, the randomness of the perturbations increases. For illustration, let us take the example of a generator instance trained specifically for a user, shown in Figure 8(a). The subscript of the model denotes the *stage* in the chain and the superscript denotes the user for

which the model is created. We begin with the discriminator D_0 (same as the D used in Section IV-B) on which the generator (G_0^{user}) is trained. Subsequently, the D_1^{user} is trained on G_0^{user} . This newly trained discriminator is now used to create the second stage generator (G_1^{user}). In each iteration, though the perturbations added are small, the untargeted training objective moves the data points randomly. Continuing this process, a chain of generators and discriminator are created. The key idea of this process is that the stochasticity introduced by the untargeted training objective is magnified by the iterative training process. Observe that from second stage onward, the discriminator on which the user and the attacker train (i.e., D_1^{user} and D_1^{adv}) are different, unlike the first stage (equivalent to iPET-Basic) where both train their generators on the same D_0 . Therefore, a generator $G_i^{user} \forall i \geq 1$ will be suitable for use as the iPET-Advanced generator. Figure 8(a) shows a chain, where G_i^{user} is chosen for installation at a user's iPET gateway. Note that only the generator at the end of the chain is installed at the iPET gateway and the intermediate models are not available to the iPET users. This design also ensures that the generator installed at different users is likely to be different.

2) *Attackers against iPET-Advanced: Naïve attacker.* In this straight-forward attack scenario, the attacker uses the discriminator $\bar{D} = D_0$ to contend against the users' generator G_i^{user} .

Retraining attacker. Figure 8(b) shows the chain of discriminator-generator sequences that leads to the generator instance G_j^{adv} installed at the attacker's iPET gateway. The attacker uses this generator instance to retrain his ML classifier, namely, $\bar{D} = \mathcal{M}_{seq}^{retrained}$.

Powerful attacker. We now describe the attack scenario where the Powerful attacker (refer Section IV-A) creates a very powerful discriminator $\bar{D} = \mathcal{M}_{seq}^{powerful}$ to contend against the users' generator G_i^{user} . Figure 9(a) shows the access that the Powerful attacker has over the several iPET gateways he controls. When the attacker uses the iPET-Advanced at the gateways as a blackbox, he will have access to the several last-stage generators such as G_j^{adv} and G_k^{adv} shown in the Figure. However, if he learns the architecture and parameters of iPET-Advanced, he can even recreate the intermediate stage generators. Figure 9(b) shows the attacker training his $\bar{D} = \mathcal{M}_{seq}^{powerful}$ based on the generators that he has access to. In Figure 9(c), the attacker uses $\bar{D}$ to fingerprint the iPET-Advanced based generator at the user. This attack scenario is evaluated in detail in Section VI-B3. To the best of our knowledge, no previous research works in the IoT domain have evaluated their defenses against an attacker as powerful as $\mathcal{M}_{seq}^{powerful}$.

V. DEPLOYMENT OPTIONS

Deployment of iPET involves placing the iPET Logic (i.e., the iPET Perturber and iPET Extractor modules) at the ingress and egress of the network path that the user considers to be untrustworthy. Implementing iPET Perturber and iPET Extractor at an IoT user's home gateway and the server, respectively, is an option; as shown in Figure 3. However, there

Fig. 9: (a) Different generators that the attacker can build based on its access to several iPET gateways. When the attacker learns the iPET-Advanced architecture, he will also have access to intermediate generators shown here. (b) The attacker builds a discriminator $\bar{D}$ based on the generators available to him (c) The attacker attempts to fingerprint a user who uses a iPET-Advanced based generator G_i^{user} .

is a dependency on servers to modify code at support iPET. Furthermore, several IoT devices can be easily mapped to their servers (i.e., destination IPs), thereby revealing the identity of the devices. We now discuss a few other deployment scenarios that are suitable for real-world use cases.

VPN-based Deployment. An IoT user may leverage a trusted third party service such as a VPN server to implement iPET. Here, the IoT home gateway, the IoT server, and the VPN server are assumed to be trusted. The IoT user's home gateway would establish a TLS connection with the remote VPN server, and iPET logic would be implemented at the home gateway (i.e., R_0) and the VPN server as shown in Figure 10a. Note that the traffic on the VPN $\rightarrow$ Server path is still vulnerable to fingerprinting attacks, yet the attacker will have no way of associating the identified device with the IoT user. However, on the downside, this scenario requires that the user place a high degree of trust on the VPN server since it has access to the undefended traffic (i.e., source, destination and traffic metadata) exchanged between the IoT device and the server.

Tor-based Deployment. Anonymous onion-routing networks such as Tor [26] may be better candidates, since no remote entity in the network has complete knowledge of the source and destination IP addresses. iPET can indeed be implemented as a Tor pluggable-transport [27] where the perturbed traffic is routed through a Tor circuit consisting of multiple peers as shown in Figure 10b. The gateway coordinates with the final hop peer to add and remove perturbations. This prevents a potentially malicious first hop peer from fingerprinting the IoT traffic. While the final peer in the circuit does have access to the undefended traffic, it does not know the true source of

Fig. 10: iPET Deployment Scenarios

the traffic. The design of the protocol ensures that only the IoT home gateway and the IoT server may be assumed to be trustworthy. However, onion encryption schemes encapsulate the messages in multiple layers of encryption adding significant bandwidth overheads.

TEE-based Deployment. Finally, an IoT user can consider a P2P (peer-to-peer) network of IoT gateway devices that support Trusted Execution Environment (TEE) as a deployment option. Gateways that support TEEs (e.g., Intel SGX [28]) form an iPET overlay network to add, route and remove traffic perturbations. Like Tor circuits, an IoT gateway first creates a circuit of multiple peers, each of which implements iPET Logic as shown in Figure 10c. Additionally, the IoT home gateway and the IoT server are, again, the only nodes assumed to be trustworthy. However, unlike Tor, the solution does not rely on the design of the protocol to achieve this, but instead leverages the hardware backed TEE. Each peer receives session parameters from the gateway which include encryption keys and the perturbation vectors. Privacy sensitive operations such as encryption/decryption, traffic perturbations and routing could be performed inside the TEE. This simplifies the design as it eliminates the need for complex protocols such as onion-encryption. Remote Attestation service[29] provided by the TEEs can be used to ensure that the iPET code running

inside the TEE is genuine and tamper-proof.

VI. PERFORMANCE EVALUATIONS

Implementation. We implement iPET in Python, using Keras libraries from TensorFlow v2.5.0 [20]. iPET generates a perturbation matrix (δ) intended to be used for a session, such that the session length is $n \times |\omega| = 4$ seconds, where $|\omega| = 0.1$ s and $n = 40$. We limit the perturbation types to adding dummy packets (cover traffic) to the original traffic session. As for the custom limit layer, we allow a maximum of *four* packets and 100 bytes of additional payload in each traffic direction for a discrete time slot ω_i .

Real-World Dataset. We conduct our experiments on real world IoT device traffic from the UNSW Smart Home Traffic Dataset [8]. The dataset includes traffic traces collected from 30 devices over 20 days. The dataset consists of 20 pcap files, each representing a day's worth of data. We set our vantage point to be the IoT gateway, with the devices communicating via the gateway to servers on the internet. For our evaluations, we do not consider traffic that is internal to the home network that our attacker cannot observe.

Evaluation Metrics. We use weighted F1-score for measuring device fingerprinting accuracy of the attacks.

Evaluation Scenarios. We attempt to answer several research questions (RQs) as part of performance evaluation. In Section VI-A, we measure the performance of iPET in comparison to other defenses. We then compare the efficacy of iPET-Basic and iPET-Advanced against attackers with varying degrees of capabilities in Section VI-B. Finally, we evaluate the *transferability* of iPET perturbations, i.e., we measure how effective iPET is against attackers who may use different ML classification models (Section VI-C).

A. Performance of iPET

Defense Setup. For the experiments in this section, the real-world trace for each device is replayed through various baseline defenses.

First, we compare iPET with the undefended gateway scenario and two other well-known defenses — VPN and Tor. In the case of Tor, we use the actual Tor code v0.4.4.8 [26]. For VPN, we emulate the behaviour by overlaying all packets over another wrapper connection. Given the volume of the traces (30 devices over 20 days), we use the Shadow framework [30] to accelerate the pace of our experiments. Shadow faithfully reproduces the application and network-layer behaviour of the defenses in a realistic virtual network topology. We configure each device as a node where the packets belonging to the respective device in the pcap files are replayed using a modified pcap-replay [31] tool. The packets are routed via another node where the defense is deployed (i.e., the gateway). We capture the trace at the gateway node to evaluate against the fingerprinting models of an attacker.

Next, we compare the efficacy of iPET against three common traffic perturbation techniques that may reduce the attacker's accuracy:

- Constant Rate Padding [3]: Perturbations are added to undefended traffic in such a way that a constant bit-rate of

100KB/s is maintained. Dummy packets of fixed lengths are added to every time-slot $|\omega|$ (100ms) in order to match the overall bit-rate.

- MTU Padding [23]: Each network packet is padded to MTU size. Packets from the link layer, network layer or transport layer may be modified. We choose the transport layer for our evaluations. No dummy packets are added in this scenario.
- Random Padding: Perturbations of random sizes are drawn from a uniform distribution and are added to the undefended traffic. This includes both the addition of dummy packets and padding the legitimate packets. For a fair comparison, we impose the same limits on the perturbation sizes as imposed in iPET, i.e., a maximum of four packets and 100 bytes every 100ms.

We implement the above techniques using Python scripts. Here, we programmatically compute the resultant traffic traces *offline*, as opposed to generating traffic traces on Shadow. This is due to the prohibitively high cost of running the Shadow simulator for the large traffic traces. For instance, replaying packets from a single day for a particular device takes ≈ 60 minutes with the Shadow framework. This means that we need around 600 CPU-hours (30 devices over 20 days) to generate one instance of a perturbed trace.

Specifically, the Python scripts ingest the pcap files from the dataset, iterate through the sequence of packets, apply the perturbations, and finally output the resultant sequence.

Attack setup. We consider two types of attackers, $\mathcal{M}_{agg}^{retrained}$ and $\mathcal{M}_{seq}^{retrained}$, based on their use of or aggregate or sequential features for device fingerprinting (refer Section II-B). The superscript *retrained* denotes that the attacker trains the classifier by employing the defense (e.g., VPN) at their end. The parameters of the attackers are as follows:

- $\mathcal{M}_{agg}^{retrained}$: Following our discussion in Section II-B, we choose a time window of 15 minutes to define a session for $\mathcal{M}_{agg}$. The attacker uses all the features listed in Appendix A. In the undefended scenario, the aforementioned features can be extracted, whereas in the other scenarios, due to overlay routing, we only use the features that are available, i.e., packet count, packet length stats (min, max, average), and the flow duration.
- $\mathcal{M}_{seq}^{retrained}$: We set the time-slot length $|\omega|$ to 100ms, and session length to be 4 seconds (Section II-B). The features used for the undefended scenario are as defined in Section II-B, resulting in a total of $m = 6$ features. For the rest of the defenses, $m = 4$ since all packets flow over a TCP/TLS connection (whether the original packets are TCP or UDP).

For iPET, the experiments in this section are carried out using 3 instances of Retraining attackers, and evaluating their performance against 10 victim users, giving us a total of 30 points. Thus all values reported here are an average of these 30 points.

1) **RQ1: How does iPET perform in comparison to other defenses, against device fingerprinting attacks?**: We first evaluate the efficacy of iPET and the two well-known defenses

	$\mathcal{M}_{agg}^{retrained}$	$\mathcal{M}_{seq}^{retrained}$
GATEWAY	100	99.7
VPN	99.7	99.3
TOR	65.3	81.6
iPET	15.8	28.3

TABLE I: Weighted F1-score achieved by attacker on IoT devices with at least 0.5% support, against well-known defenses.

	$\mathcal{M}_{seq}^{retrained}$
CONSTANT RATE	73.9
PADDING TO MTU	92.4
RANDOM PERTURBATION	94.9
iPET	28.3

TABLE II: Weighted F1-score achieved by attacker on IoT devices with at least 0.5% support, against common traffic perturbation techniques.

against the two ML-based classifiers, namely $\mathcal{M}_{agg}^{retrained}$ and $\mathcal{M}_{seq}^{retrained}$. The results of this experiment are given in Table I. We observe that iPET has a very low F1-score against both $\mathcal{M}_{agg}^{retrained}$ and $\mathcal{M}_{seq}^{retrained}$, thwarting device fingerprinting attacks that use these classifiers. Tor is the most effective among the baselines, yet it has a very high F1-score of 81.6% against $\mathcal{M}_{seq}^{retrained}$. Note that the sequence-based classifier $\mathcal{M}_{seq}^{retrained}$ is comparable or better than the aggregate classifier $\mathcal{M}_{agg}^{retrained}$ against all defenses.

In Figure 11a, we present a finer analysis using a CDF of the F1-scores measured per device when $\mathcal{M}_{seq}^{retrained}$ is employed. Observe that, with iPET the F1-score never exceeds 60% for any device. In fact, 40% of the devices have an F1-score below 40%. In contrast, when Tor is used, 40% of the devices are identified with an F1-score of 80% or more, demonstrating the inability of Tor in changing the IoT device traffic effectively against fingerprinting attacks. As expected, VPN and undefended scenarios perform the worst, with devices being identified with a F1-score close to 100%.

We now evaluate the efficacy of *Constant Rate Padding*, *Random Padding*, and *Padding to MTU* techniques. Table II shows that neither of the three techniques are effective against $\mathcal{M}_{seq}^{retrained}$. Even the most effective defense of the three—Constant Rate Padding—has a high weighted F1-score of 73.9% compared to iPET’s 28.3%. Figure 11b illustrates the effectiveness at a device level. Padding to MTU and Random Padding techniques fail to defend against an attacker in most cases, i.e., all devices are fingerprinted with an F1-score of at least 80%.

2) RQ2: What is the effect of fingerprinting attack on different IoT devices in the presence of iPET defense?:

We now analyze how iPET defends different categories of IoT devices. We group the 30 devices in our dataset into six categories based on their use case. The average F1-score of the devices within each category is plotted in Figure 11c, for both iPET and Tor. We observe that iPET outperforms Tor in all the categories. The fingerprinting attack does not achieve more than $\approx 30\%$ against iPET. Figure 11d provides further insights on the worst performing categories, namely, home automation devices and smart cameras.

3) RQ3: What is the trade-off between the privacy provided by iPET and the additional bandwidth consumed?: In iPET, perturbations added have a direct effect on the efficacy of iPET; in this experiment, we study this effect by varying the bandwidth parameters in iPET. We represent the maximum bandwidth parameters as: [max no. of dummy packets, max payload size of dummy packets] in any given time-slot ω_i (payload size excludes packet header size). Figure 11e plots the F1-score of iPET for different variations of the maximum bandwidth parameters, when evaluated against $\mathcal{M}_{seq}^{retrained}$. We encode these constraints in the custom limit layer of our deep learning model and repeat our experiments for each configuration (point on the X-axis). In the figure, observe that the F1-score of classifying the devices reduces with increasing perturbation. When iPET is allowed to a maximum of 10 dummy packets amounting to maximum of 1KB within $|\omega| = 0.1s$, the weighted average F1-score reduces to 10%. For all the evaluations in this work, we use the perturbation configuration [4,100].

Additionally, we also plot the F1-score for the Random Padding technique described in RQ1 (Section VI-A1). Intuitively, the F1-score drops as we increase the maximum payload size of the dummy packets, yet we observe that despite the increase, it is still outperformed by iPET. In fact, even with high limits [80,10000], the Random Perturbations technique restricts the attacker to an F1-score of 71.1%, significantly higher than the score against iPET configured at limits [4,50].

4) RQ4: What is the bandwidth overhead of iPET?: We measure the additional bandwidth incurred due to iPET. Figure 11f plots the CDF of the IoT devices for the additional bandwidth incurred (in bytes/second). Around 50% of devices do not require an additional bandwidth of more than 500 bytes/s, which is insignificant compared to the home broadband bandwidth available today (the global average is over 55 Mbps, as per a recent report [32]).

B. Evaluating iPET-Basic and iPET-Advanced

Defense and Attack Setup. We compare the effectiveness of iPET-Basic and iPET-Advanced against the attack scenarios that we described in Section IV-A, namely, $\mathcal{M}_{seq}^{naive}$, $\mathcal{M}_{seq}^{retrained}$, and $\mathcal{M}_{seq}^{powerful}$. Comprehensive evaluation of our research questions in this section are also computationally very expensive. Therefore, we compute the traces offline (as described in Section VI-A). As an example, Fig. 12b requires models to be run on 50 traffic traces, each trace consisting of 30 devices emitting packets over 20 days. This makes the use of Shadow prohibitively expensive.

1) RQ5: How do iPET-Basic and iPET-Advanced perform against different attack scenarios?: Figure 12a plots the effectiveness of $\mathcal{M}_{seq}^{naive}$ and $\mathcal{M}_{seq}^{retrained}$ against iPET-Basic and iPET-Advanced. The Naïve attacker $\mathcal{M}_{seq}^{naive}$ is unsuccessful in fingerprinting the devices protected by both iPET-Basic and iPET-Advanced. Whereas, the Retraining attacker $\mathcal{M}_{seq}^{retrained}$ achieves a high F1-score of 89% against iPET-Basic. In comparison, iPET-Advanced is highly effective in countering fingerprinting attack by $\mathcal{M}_{seq}^{retrained}$. We conclude that chaining the discriminator-generator pair in iPET-Advanced helps im-

Fig. 11: Performance of iPET on real-world IoT device traffic (UNSW Smart Home Traffic Dataset [8]).

prove stochasticity of its generator, making it effective against Retraining attacker.

2) **RQ6: How does the length of generator chain affect the performance of iPET-Advanced?**: In this section, we design an experiment to analyze how the number of stages affects the resilience of iPET-Advanced. In Figure 12b, the X-axis denotes the stage of the generator as a defense. We now evaluate the efficacy of these generators when an attacker uses discriminators from different stages $\mathcal{D}_j$, where $0 \leq j \leq 4$. We observe that, the attacker's discriminators do not achieve a weighted F1-score of more than $\approx 20\%$, when the generator is at stage $\mathcal{G}_{i \geq 1}$.

3) **RQ7: Is iPET-Advanced resilient to the Powerful attacker (Section IV-A)?**: **Datasets.** In addition to the real-world dataset, in this set of experiments, we also consider a synthetic dataset to evaluate the capability of iPET-Advanced against a Powerful attacker (Section IV-C2). Our synthetic dataset consists of data points from 50 classes (devices), each with 50,000 samples. The points are sampled from Gaussian Mixture Model (GMM), one for each class. The GMM itself is composed of five Gaussian distributions. We also added sufficient noise to the data and ensure minor overlaps between classes to emulate real-world behavior.

Results. For the Powerful attacker, we train, what we call a super discriminator $\mathcal{D}_{\text{super}} = \mathcal{M}_{\text{seq}}^{\text{powerful}}$, with inputs from several generators from different stages of various chains. The attacker recreates these intermediate generators from different chains at his end from his knowledge of the iPET-Advanced model's architecture and parameters (refer Figure 9 from Section IV). Figure 12b shows that, on the real-world dataset, although $\mathcal{D}_{\text{super}}$ performs better than the Retraining attacker

$\mathcal{D}_j = \mathcal{M}_{\text{seq}}^{\text{retrained}}$, the maximum F1-score converges to $\approx 50\%$. On the synthetic dataset, we get even better results, with the F1-score of $\mathcal{D}_{\text{super}}$ converging to $\approx 5\%$, demonstrating the effectiveness of perturbations generated by iPET-Advanced.

C. Transferability of iPET

Defense and Attack Setup. iPET perturbations are trained against the fingerprinting classifier $\mathcal{M}_{\text{seq}}^{\text{retrained}}$ that employs a Bi-LSTM based architecture to learn the time series feature. For our experiment, we consider a retraining attacker, $\mathcal{M}_{\text{seq}}^{\text{retrained}}$, who uses different training architectures by substituting the Bi-LSTM layers in $\mathcal{M}_{\text{seq}}$ with Gated Recurrent Unit (GRU) / Convolutional Neural Network (CNN) / Multilayer Perceptron (MLP) based $\mathcal{M}_{\text{seq}}$ classifiers. We also consider the scenario where the attacker chooses a fingerprinting model that uses aggregate features (instead of time series) on a RandomForest Classifier, i.e. $\mathcal{M}_{\text{agg}}$. Further, we also evaluate iPET against an attacker who tweaks the parameters (i.e. ω and observation time) of the input time series (τ) to achieve better fingerprinting accuracy.

1) **RQ8: Is iPET effective against an attacker who uses a different ML classification model?**: In Table III, we first benchmark the efficacy of each classification model of the attacker by showcasing the fingerprinting accuracy achieved by the model in the undefended setup. We note that each architecture can accurately identify the devices with an F1-score $\geq 97\%$. When iPET (that uses Bi-LSTM classifier) is used to defend the traffic, the fingerprinting accuracy of each of the classifiers used by attacker reduces considerably. Most notably, the Multilayer Perceptron, which is the most effective ML model shown in Table III, only achieves an F1-score

Fig. 12: (a) iPET-Basic and iPET-Advanced against $\mathcal{M}_{seq}^{naive}$ and $\mathcal{M}_{seq}^{retrained}$. (b) Fingerprinting accuracy of attack discriminators $\bar{D}_j$ and D_{super} (Powerful attacker), against different user generators G_i on Real-world dataset, and (c) Synthetic dataset.

Model	Architecture	F1-score (undefended)	F1-score (defended by iPET)
$\mathcal{M}_{agg}$	Random Forest	100	17.93
$\mathcal{M}_{seq}$	Bi-LSTM	99.7	33.5
$\mathcal{M}_{seq}$	GRU	99.47	38.21
$\mathcal{M}_{seq}$	CNN	97.46	37.14
$\mathcal{M}_{seq}$	MLP	97.63	41.18

TABLE III: Exploring the transferability of iPET by evaluating the attack F1-score an attacker achieves by varying the fingerprinting classifier. The defence is fixed, and tailored against an $\mathcal{M}_{seq}$ attacker with a Bi-LSTM architecture.

$\omega(s)$	0.05	0.1	0.2	0.5	1
F1-score	27.44	26.27	29.28	24.18	28.96

TABLE IV: Weighted F1-score achieved by attacker on Varying ω in captured Time Series. The defence is fixed, adding perturbations for each $\omega = 0.1s$.

Fig. 13: Varying the total observation time for the Time Series as modeled by the attacker and Defense. The sampling rate ω remains constant.

2) **RQ9: Is iPET effective against an attacker who uses different input time series parameters in their ML classification model?**: We first evaluate the advantage an attacker gains by varying the time ω , which is used for calculating the features within the time series input τ . The results for this experiment are shown in Table IV. The iPET model considered generates perturbations for $\omega = 0.1s$. We note that the attacker does not achieve any substantial improvement in the F1-score ($\approx 24 - 29\%$) by changing ω .

Observation time of traffic is an important parameter of ML classification models used by both the attacker as well as the iPET generative model. In Figure 13, the X-axis plots the observation time used for the attacker's ML model. Each line plotted in the graph represents a iPET instance that uses a different observation time for modeling perturbations. The Y-axis plots the F1-score achieved by the attacker against the respective model used by iPET. When iPET uses an observation time of 4 seconds, the attacker achieves a maximum F1-score of $\approx 80\%$ with an observation time of 0.5 seconds and the F1-score reduces with increase in attacker's observation time. However, this behavior becomes less prominent when iPET also used smaller observation time (e.g., 0.5s). Therefore, we recommend that the observation time chosen be as small as possible.

VII. RELATED WORKS

User traffic identification attacks are well-known, and several research works have proposed fingerprinting attacks that identify websites [33], [34], [35], [36], [37], [16], [17], [18], webpages [38], [39], videos [40], [41], [42], [43], browsers [44], [45], and even newer protocols [46] from encrypted communications. In the widely studied website fingerprinting attack, a network attacker identifies the website accessed by a user over a communication channel, with recent research works exploit algorithms from machine learning and deep learning.

Several defenses [37], [35], [16], [43], [47], [48], [14], [17], [49] have been proposed to counter user traffic identification attacks, and website fingerprinting in particular, and all of them use some form of traffic perturbation. Early countermeasures [37], [35], [47], [48], [49] for website fingerprinting used anonymous communication (e.g., Tor, VPN) in conjunction with a traffic perturbation algorithm. However, protecting against attackers [16], [17], [18] that use Machine Learning (ML) and Deep Learning (DL) has been a challenge. Recent countermeasures for website fingerprinting [14], [15], use DL-based traffic perturbations to deceive the ML/DL classifiers

used by the attacker. These countermeasures add perturbations such that the perturbed traffic flows get incorrectly classified by the attacker's ML model. A key challenge is that an attacker can further retrain to learn the added perturbations. In this context, Mockingbird [14] a website fingerprinting defense, proposed the use of a semi-targeted objective function (i.e., chooses a set of target labels for misclassifying a datapoint) to increase the stochasticity of the perturbations, to counter retraining attackers.

In the context of IoT, many research works [3], [4], [5], [6], [7], [50], [8], [9], [23] have shown that it is possible to fingerprint the devices with a very high accuracy, highlighting their vulnerability. Some of the proposed fingerprinting attacks [23] require access to the WLAN traffic of the IoT devices, assuming an in-network attack model. Alternatively, what is more relevant to our work, are the attacks that can fingerprint the devices based on the traffic exchanged over the Internet [3], [4], [5], [6], [7], [50], [8], [9]. These attacks use ML classifiers to identify the IoT devices, but assume that the IoT user has no defense (e.g., Tor, VPN) in place. The solutions [51], [21], [52] that counter such attacks require trusting a third party service provider, namely, the VPN provider, who implements a traffic perturbation algorithm in coordination with the users' IoT gateways to obfuscate traffic patterns. Most importantly, the solutions do not assume a strong attacker who retrains the perturbed traffic as a counter strategy. We push the boundary of defense in the IoT domain by proposing the first deep learning-based perturbation generator that is resilient to powerful attackers.

VIII. CONCLUSIONS

We developed and presented iPET, a new defense solution against IoT device fingerprinting attacks that is resilient to different attack capabilities. iPET uses a novel deep learning-based perturbation technique effectively reducing the accuracy of fingerprinting attacks from a very high 99% to an ineffective $\approx 28\%$. Further, different from previous works, we considered a more powerful attacker with knowledge of the iPET model's architecture and parameters; and our experiments show that such an attacker achieves only 40% accuracy against iPET.

REFERENCES

- [1] The Internet of Things 2020. Available online at: <https://www.businessinsider.com/internet-of-things-report>.
- [2] Francesca Meneghello, Matteo Calore, Daniel Zucchetto, Michele Polese, and Andrea Zanella. IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices. *IEEE Internet of Things Journal*, 2019.
- [3] Noah J. Athorpe, Dillon Reisman, Srikanth Sundaresan, Arvind Narayanan, and Nick Feamster. Spying on the Smart Home: Privacy Attacks and Defenses on Encrypted IoT Traffic. *CoRR*, 2017.
- [4] Antônio J. de Pinheiro, Jeandro M. Bezerra, Caio A. P. Burgardt, and Divanilson R. Campelo. Identifying IoT devices and events based on packet length from encrypted traffic. *Computer Communications*, 2019.
- [5] Vijayanand Thangavelu, Dinil Mon Divakaran, Rishi Sairam, Suman Sankar Bhunia, and Mohan Gurusamy. DEFT: A Distributed IoT Fingerprinting Technique. *IEEE Internet Things Journal*, 2019.
- [6] Shuaike Dong, Zhou Li, Di Tang, Jiongyi Chen, Menghan Sun, and Kehuan Zhang. Your Smart Home Can't Keep a Secret: Towards Automated Fingerprinting of IoT Traffic. In *ACM Asia Conf. on Computer and Communications*, 2020.
- [7] Mustafizur R. Shahid, Gregory Blanc, Zonghua Zhang, and Hervé Debar. IoT Devices Recognition Through Network Traffic Analysis. In *IEEE International Conf. on Big Data*, 2018.
- [8] Arunan Sivanathan, Hassan Habibi Gharakheili, Franco Loi, Adam Radford, Chamith Wijanayake, Arun Vishwanath, and Vijay Sivaraman. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Trans. on Mobile Computing*, 2019.
- [9] Yair Meidan, Michael Bohadana, Asaf Shabtai, Martín Ochoa, Nils Ole Tippenhauer, Juan David Guarnizo, and Yuval Elovici. Detection of Unauthorized IoT Devices Using Machine Learning Techniques. *CoRR*, 2017.
- [10] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing Properties of Neural Networks. *CoRR*, 2014.
- [11] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conf. on Learning Representations (Poster)*, 2015.
- [12] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge J. Belongie. Generative adversarial perturbations. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018.
- [13] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal Adversarial Perturbations. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2017.
- [14] Mohammad Saidur Rahman, Mohsen Imani, Nate Mathews, and Matthew Wright. Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks With Adversarial Traces. *IEEE Trans. on Information Forensics and Security*, 2021.
- [15] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations. In *USENIX Security Symposium*, 2021.
- [16] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *ACM SIGSAC Conf. on Computer and Communications Security*, 2018.
- [17] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *ACM SIGSAC Conf. on Computer and Communications Security*, 2018.
- [18] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proc. on Privacy Enhancing Technologies*, 2019.
- [19] Keras Library. Available online at: <https://keras.io/>.
- [20] TensorFlow v2.5.0. Available online at: https://www.tensorflow.org/versions/r2.5/api_docs/python/tf.
- [21] Noah J. Athorpe, Dillon Reisman, and Nick Feamster. A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic. *CoRR*, 2017.
- [22] Google Transparency Report. Available online at: <https://transparencyreport.google.com/https/overview>.
- [23] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-Level Signatures for Smart Home Devices. In *Network and Distributed System Security Symposium*, 2020.
- [24] Mike Schuster and Kuldip K. Paliwal. Bidirectional Recurrent Neural Networks. *IEEE Trans. on Signal Processing*, 1997.
- [25] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.
- [26] tor - Tor's source code. Available online at: <https://gitweb.torproject.org/tor.git/tree/?h=tor-0.4.4.8>.
- [27] Tor at the Heart: Bridges and Pluggable Transports. Available online at: <https://blog.torproject.org/tor-heart-bridges-and-pluggable-transports/>.
- [28] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Workshop on Hardware and Architectural Support*, 2013.
- [29] Attestation Services for Intel® Software Guard Extensions. Available online at: <https://www.intel.com/content/www/us/en/develop/topics/software-guard-extensions/attestation-services.html>.
- [30] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium*, 2012.
- [31] Extra plug-ins for shadow, 2021. Available online at: <https://github.com/shadow/shadow-plugin-extras>.
- [32] Speedtest Global Index – Internet Speed around the world – Speedtest Global Index, August 2020. Available online at: <https://www.speedtest.net/global-index>.
- [33] Qixiang Sun, Daniel R Simon, Yi-Min Wang, Wilf Russell, Venkata N Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*, 2002.

- [34] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *ACM Workshop on Privacy in the Electronic Society*, 2011.
- [35] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conf. on Computer and Communications Security*, 2012.
- [36] Jamie Hayes and George Danezis. k-fingerprinting: A Robust Scalable Website Fingerprinting Technique. In *USENIX Security Symposium*, 2016.
- [37] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *USENIX Security Symposium*, 2014.
- [38] Meng Shen, Yiting Liu, Liehuang Zhu, Xiaojiang Du, and Jiankun Hu. Fine-Grained Webpage Fingerprinting Using Only Packet Length Information of Encrypted Traffic. *IEEE Trans. on Information Forensics and Security*, 2020.
- [39] Kailong Wang, Junzhe Zhang, Guangdong Bai, Ryan Ko, and Jin Song Dong. It's Not Just the Site, It's the Contents: Intra-domain Fingerprinting Social Media Websites Through CDN Bursts. In *The Web Conference*, 2021.
- [40] Feng Li, Jae Won Chung, and Mark Claypool. *Silhouette: Identifying YouTube Video Flows from Encrypted Traffic*. 2018.
- [41] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security Symposium*.
- [42] Gargi Mitra, Prasanna Karthik Vairam, Patanjali Slpsk, and Nitin Chandrachoodan. White Mirror: Leaking Sensitive Information from Interactive Netflix Movies using Encrypted Traffic Analysis. In *ACM SIGCOMM Conf. Posters and Demos*, 2019.
- [43] Xiaokuan Zhang, Jihun Hamm, Michael K. Reiter, and Yinqian Zhang. Statistical Privacy for Streaming Traffic. In *Network and Distributed System Security Symposium*, 2019.
- [44] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *IEEE Symposium on Security and Privacy*, 2018.
- [45] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Mitigating Browser Fingerprint Tracking: Multi-level Reconfiguration and Diversification. In *IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015.
- [46] Levente Csikor, Himanshu Singh, Min Suk Kang, and Dinil Mon Divakaran. Privacy of DNS-over-HTTPS: Requiem for a Dream? In *IEEE European Symposium on Security and Privacy*, 2021.
- [47] Tao Wang and Ian Goldberg. Walkie-Talkie: An Efficient Defense Against Passive Website Fingerprinting Attacks. In *USENIX Security Symposium*, 2017.
- [48] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. WTF-PAD: Toward an Efficient Website Fingerprinting Defense for Tor. *CoRR*, 2015.
- [49] Experimental Defense for Website Traffic Fingerprinting. Available online at: <https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks>.
- [50] Manuel López Martín, Belén Carro, Antonio Sánchez-Esguevillas, and Jaime Lloret. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*, 2017.
- [51] Noah J. Aphorpe, Danny Yuxing Huang, Dillon Reisman, Arvind Narayanan, and Nick Feamster. Keeping the Smart Home Private with Smart(er) IoT Traffic Shaping. *Proc. on Privacy Enhancing Technologies*, 2019.
- [52] Ahmed Alshehri, Jacob Granley, and Chuan Yue. Attacking and Protecting Tunneled Traffic of Smart Home Devices. In *ACM Conf. on Data and Application Security and Privacy*, 2020.

APPENDIX A

AGGREGATE FEATURES MODEL DETAILS

We list the features that $\mathcal{M}_{agg}$ adversary extracts in Table V. We use “statistics” to cumulatively denote the minimum, maximum and mean values of the feature and “#” to denote the number of occurrences of that feature. As explained in Section II-B, the features are computed over a session of some configured length.

An attacker in the undefended scenario has access to all the features listed in Table V. However, in some deployment

Protocol	Feature Extracted
DNS	# DNS queries
	# DNS errors in reply code
	# DNS 'IN' class queries
	Average DNS response time
	# Packets
	Packet Length statistics
HTTP	# TCP keep Alive
	# Unique HTTP status codes
	# Unique HTTP request methods
	# Unique HTTP request URI
	# Packets
	Packet Length statistics
TCP / UDP / ICMP / TLS / SSDP / NTP / STUN / GQUIC / Other Protocols	# Packets
	Packet Length statistics

TABLE V: Comprehensive list of features utilized.

scenarios (Section V), features like the packet protocol may not be known. For instance, a VPN might overlay all UDP and TCP packets over a single TCP connection. In that case, the attacker computes the count of packets, packet length statistics and the flow duration. The flow duration is the difference between the time at which the last and the first packet were observed in a session. In order to maximise the information drawn from these features, we group them based on the direction of the packets.