AdvTraffic: Obfuscating Encrypted Traffic with Adversarial Examples

Hao Liu* Jimmy Dani* Hongkai Yu† Wenhai Sun‡ Boyang Wang*
*University of Cincinnati, †Cleveland State University, ‡Purdue University
{liu3ho, danijy}@mail.uc.edu, h.yu19@csuohio.edu, whsun@purdue.edu, boyang.wang@uc.edu

2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS) | 978-1-6654-6824-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/IWQoS54832.2022.9812875

***Abstract*—Website fingerprinting can reveal which sensitive website a user visits over encrypted network traffic. Obfuscating encrypted traffic, e.g., adding dummy packets, is considered as a primary approach to defend against website fingerprinting. However, existing defenses relying on traffic obfuscation are either ineffective or introduce significant overheads. As recent website fingerprinting attacks heavily rely on deep neural networks to achieve high accuracy, producing adversarial examples could be utilized as a new way to obfuscate encrypted traffic. Unfortunately, existing adversarial example algorithms are designed for images and do not consider unique challenges for network traffic.**

In this paper, we design a new method, named AdvTraffic, which can customize perturbations produced by any existing adversarial example algorithm on images and derive adversarial examples over encrypted traffic. Our experimental results show that the integration of AdvTraffic, particularly with Generative Adversarial Networks, can effectively mitigate the accuracy of website fingerprinting from 95.0% to 10.2%, even if an attacker retrains a classifier with defended traffic. Compared to other defenses, our method outperforms most of them in mitigating attack accuracy and offers the lowest bandwidth overhead.

***Index Terms*—Website Fingerprinting, Encrypted Traffic, Machine Learning, Adversarial Examples**

I. Introduction

With secure communication protocols (e.g., Tor), a user can hide which website she visits against eavesdroppers as destination IP addresses of network packets are encrypted. However, by analyzing traffic pattern, such as the size and direction of each packet, a website fingerprinting attack can still reveal which sensitive website that a user visits [1]–[14]. For instance, by leveraging neural networks, an attacker can achieve 98% accuracy in website fingerprinting [11].

Due to the potential privacy leakage, many defenses [4], [15]–[19] have been proposed. In general, these defenses obfuscate encrypted traffic by adding dummy packets, buffering packets, or merging traffic of two websites as a super sequence. However, it is still challenging for a defense to be effective against website fingerprinting while introducing lower overheads. Specifically, if a defense introduces low overheads, the attack accuracy remains high, especially when neural networks are leveraged as classifiers. On the other hand, if the attack accuracy is significantly reduced, a defense often requires significant amounts of overheads in bandwidth and/or network delay.

In this paper, we aim to produce *adversarial examples* [20], [21] over encrypted traffic and leverage these adversarial examples to obfuscate encrypted traffic against website fingerprinting. Adversarial examples are perturbed samples, e.g. perturbed images, where the perturbation is small but sufficient to force a well-trained neural network to predict incorrectly. For example, a perturbed image of a panda remains a panda to humans but it forces a well-trained neural network to predict an incorrect label [20]. How to generate adversarial examples have been well studied in the image domain [20]–[27].

However, due to the difference between images and network traffic, we need to tackle several real-world constraints when producing adversarial examples over encrypted traffic. For instance, the sign of a perturbation should be the same as the sign (i.e., direction) of a packet, otherwise original packets need to be buffered, which will interrupt network protocols and introduce unnecessary network delay.

To address the challenge, we propose a new method, referred to as *AdvTraffic*, to customize perturbations generated by existing adversarial example algorithms and produce adversarial examples specifically over encrypted traffic. Our method is compatible and can be integrated with any existing adversarial example algorithm. Our main contributions and findings are summarized below:

- We design a new method, AdvTraffic, which can be leveraged to produce adversarial examples over encrypted traffic. We integrate AdvTraffic with four existing adversarial example algorithms, including Fast Gradient Sign Method [21], DeepFool [23], Projected Gradient Decent [26], and AdvGAN [28] respectively.
- Our experimental results over a large-scale dataset show that, the integration of our method AdvTraffic with AdvGAN (denoted as AdvTraffic-AdvGAN), is able to successfully reduce the attack accuracy of a Convolutional Neural Network from 95.0% to 10.2%, even when the attacker's classifier is retrained with defended traffic. Moreover, the obfuscated traffic generated by AdvTraffic-AdvGAN are also transferable across different neural network classifiers.
- Compared to recent defenses [16], [17], [29]–[31], our method outperforms most of them in terms of mitigating attack accuracy. In addition, our method derives the lowest bandwidth overhead among all the defenses. The comparison results in the closed-world setting is highlighted in Table VI in Sec.V. Moreover, our method also outperforms other defenses in the open-world scenario.

978-1-6654-6824-4/22/$31.00 ©2022 IEEE

Reproduci
at https://gith

System an
is described i
A user conne
All the netwc
encrypted by
user visits o
among traffic

Same as
[13], we assu
encrypted tra
attacker does
addition, it
aims to reve
encrypted tra
learn the dire
a user and th

Data Forn
of outgoing
of packets as
to as a traffi
trace in the *b*
[16], [17], [2
packets in the
number of T
Tor cells[1]. If
a burst are o
negative, it suggests the packets in a burst are incoming (from server to client). A traffic trace in the burst format, in essence, is a vector of integers. For instance, given a sequence of Tor cells $(+1, +1, -1, -1, +1, +1, +1, +1, -1)$, its burst format is $(+2, -2, +4, -1)$. For the ease of presentation, we use the terms "trace" and "sample" interchangeably in the rest of this paper. Note that we do not leverage timestamps of packets in this paper. Some website fingerprinting attacks leveraging timestamps can be found in [32].

Evaluation Metric. By following the previous studies in website fingerprinting, we examine the attack performance of website fingerprinting in two settings, including *closed-world setting* and *open-world setting*. In the closed-world setting, we assume that an attacker possesses a list of *monitored websites* and a user only visits websites within this list. The evaluation in closed-world setting is formulated as a classification problem over multiple classes, where each website is considered as a class/label. The goal of the attacker is to infer which website a user visits. Accuracy is utilized as the metric to measure the attack performance.

In the open-world setting, we assume that an attacker possesses a list of monitored websites but a user can also visits *unmonitored websites* outside of this list. The evaluation in the open-world setting is formulated as a binary classification. The goal of the attacker is to reveal whether a traffic trace belongs to the list of monitored websites or not. Precision, recall, precision-recall curves, and ROC (Receiver Operating Characteristic) curves are utilized to measure the attack performance in the open-world evaluation.

[1]Tor protocol uses fixed-length packets, named cells, to transmit data. How to transform the size of a TCP packet to a corresponding number of cells can be found in [9]

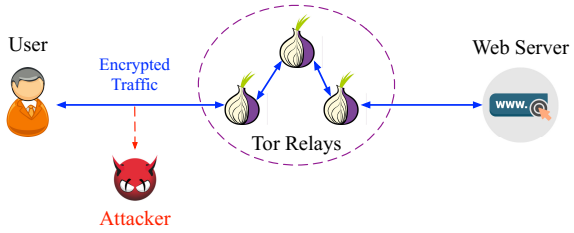

Fig. 1. The system model of website fingerprinting

In this paper, we follow the *standard model* used in previous studies [5]–[7], [10], to measure the performance in the open-world setting. Specifically, given a classifier trained based on monitored websites in the closed-world evaluation, all the traffic traces from the unmonitored websites are considered as samples of a new class, which is added to the classifier as an additional class. The classifier is retrained with data from monitored websites and data from unmonitored websites. During the testing, given an unlabeled traffic trace, if the highest confidence of the classifier belongs to one of the monitored websites and this confidence is higher than a predefined threshold, then this traffic trace will be labeled as a monitored website (even through the highest confidence is not from the correct monitored website). By tuning the predefined threshold, an attacker in the open-world evaluation can maximize precision or recall.

Our Goals in Defense. In this study, our primary goal is to build a defense, which can obfuscate traffic pattern such that an attacker will derive a lower attack performance in both closed-world setting and open-world setting. In addition, we aim to minimize the impacts and overheads of this traffic obfuscation to the current network protocol. For instance, our defense will only add dummy traffic such that there is no need to buffer original packets.

Generative Adversarial Networks. Generative Adversarial Network (GAN) [33], proposed by Goodfellow et al. in 2014, is a generative model which can generate synthetic data that are difficult to distinguish from real data. It can be utilized to produce more data to facilitate the training of classifiers in different applications or output adversarial examples to fool well-trained classifiers. We introduce the background information of GANs in this subsection to facilitate the discussions on an algorithm described in later sections.

A GAN consists of Generator (G) and Discriminator (D). Both the Generator and Discriminator, in essence, are neural networks. Generator takes random vectors from a low-dimensional latent space as inputs and aims to produce generated data, whose distribution is close to the distribution of real data in a high-dimensional space. These random vectors

can often be generated by a uniform distribution or Gaussian distribution. Discriminator takes generated data and real data as inputs and aims to distinguish generated data from real data. It can be considered as a binary classifier.

In the training, Generator and Discriminator update their weights by leveraging backpropagation such that Discriminator maximizes the probability of assigning correct labels to both generated data and real data. On the other hand, Generator minimizes the distance between generated data and real data and makes it more difficult for Discriminator to assign correct labels. More specifically, Generator and Discriminator play a two-player minimax game with a value function $V(D, G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{1}$$

where all the real data $\mathbf{x}$ follows the distribution of $p_d(\mathbf{x})$ and all the random inputs $\mathbf{z}$ of Generator follows the distribution of $p_z(\mathbf{z})$. This loss function was derived from the cross-entropy between generated data and real data.

To avoid vanishing gradients [34], a GAN can utilize Wasserstein distance (also referred to as Earth Mover's distance) rather than cross-entropy to measure the loss in training, which refer to WGAN [34], where the value function is defined as

$$\min_G \max_D V_W(D,G) = \max_{w\in W} \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))] \tag{2}$$

where the max value represents the constraint on the discriminator. In this paper, unless specified, we utilize WGAN to avoid vanishing gradients.

III. Adversarial Example Algorithms

An adversarial example [21] is a perturbed sample, where the perturbation is often small but can force a well-trained neural network to predict into an incorrect class. Many Adversarial Example (AE) algorithms [20]–[27] have been proposed. A comprehensive survey on adversarial examples can be found in [35].

In this section, we introduce the details of several existing *white-box* and *semi-white-box* AE algorithms. A white-box AE algorithm suggests that an adversary knows the structure and hyperparameters of a target model. As for the semi-white-box AE algorithm, it indicates that an adversary can access to the target model (structure and hyperparameters) during the feed-forward network training process. A target model means the model that an adversary aims to fool. There are also black-box AE algorithms [24] that do not need to know the structure and hyperparameters in advance. We focus on white-box/semi-white-box AE algorithms in this paper.

Fast Gradient Sign Method. Fast Gradient Sign Method (FGSM), which was proposed by Goodfellow et. al. [21], is a white-box algorithm that can produce adversarial examples by leveraging gradients of a neural network. Specifically, given a sample x, the algorithm adds a perturbation to this sample based on the sign of the gradient of cost function $J(\theta, x, y)$, where y is the true label of sample x and θ are parameters of a target neural network. The algorithm can be summarized below:

$$x_{adv} = x + \epsilon \cdot \texttt{sign}(\nabla_x J(\theta, x, y)) \tag{3}$$

where x_{adv} is an adversarial example of sample x, ϵ is a parameter to control the size of the perturbation, and $\nabla_x J$ is the gradient of the cost function with respect to sample x.

DeepFool. DeepFool [23] is a white-box algorithm that iteratively computes a perturbation based on the linearization of the target classifier to generate adversarial examples. A perturbation generated by DeepFool is often lower than a perturbation produced by FGSM, but takes a longer time to obtain due to multiple iterations involved. Given a sample x and a target classifier $f(\cdot)$, the algorithm of DeepFool can be described below:

$$x_0 \leftarrow x \quad x_{i+1} \leftarrow x_i + r_i$$
$$r_i \leftarrow \left(-\frac{f(x_i)}{||\nabla f(x_i)||_2^2}\nabla f(x_i)\right) \tag{4}$$

where x_{i+1} represents the intermediate adversarial example obtained after the i-th iteration. The iterations stop when $\texttt{sign}(f(x_i))$ is no longer the same as $\texttt{sign}(f(x_0))$. Once the iterations stop, the algorithm outputs the perturbation $\hat{r} = \sum_i r_i$. The final adversarial example can be represented as $x_{adv} = x + \sum_i r_i$.

Projected Gradient Descent (PGD). PGD [26], in essence, is an iterative version of FGSM. It can produce lower perturbations which are more difficult to notice. In each iteration, PGD generates an intermediate adversarial sample using a small step. The algorithm can be described as below:

$$x_0 \leftarrow x \quad x_{i+1} \leftarrow \texttt{Clip}_\epsilon\left[x_i + \alpha \cdot \texttt{sign}(\nabla_x J(\theta, x, y)\right] \tag{5}$$

where x_{i+1} represents the intermediate adversarial example derived after the i-th iteration, ϵ is a parameter to control the size of the perturbation, element-wise clipping function $\texttt{Clip}_\epsilon[\cdot]$ keeps each element $u_{i,j}$ of an input u_i within the range of $[u_{i,j} - \epsilon, u_{i,j} + \epsilon]$, and α is a parameter to control the size of changes in each iteration. The algorithm stops when the number of iterations reaches a predefined parameter and outputs adversarial example $x_{adv} = x_{i+1}$, where the total number of iterations is i.

AdvGAN. AdvGAN [28] is a semi-white-box algorithm that generates adversarial examples by leveraging a GAN. As shown in Fig. 2, the framework of AdvGAN consists of a Generator G, a Discriminator D, and a target classifier f. Given a sample x as an input, Generator outputs a perturbation $G(x)$. Next, perturbed sample $x + G(x)$ and sample x are used as inputs for Discriminator. In addition, perturbed sample $x + G(x)$ is used as input to the target classifier f to obtain a label t.

During the training, Discriminator aims to distinguish real samples and perturbed samples. Generator aims to produce perturbed samples that are difficult for Discriminator to distinguish from samples. In addition, Generator aims to produce perturbed samples that can be misclassified by f. The loss

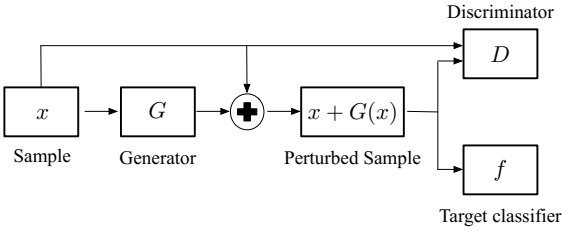

Fig. 2. Framework of AdvGAN

function of the entire AdvGAN is a linear combination of the loss of Discriminator, the loss of classifier f and a soft hinge loss. This soft hinge loss is included to bound the magnitude of the perturbation [28].

After training with sufficient amounts of data, Generator will be extracted and used to produce adversarial perturbations for any input without requiring access to the target model.

IV. Adversarial Examples for Encrypted Traffic

A. Our Proposed Method: AdvTraffic

We aim to produce adversarial examples for encrypted traffic traces such that an attacker who performing website fingerprinting with a neural network will predict incorrect labels. By obfuscating traffic with perturbations, it changes the size of each burst, and therefore, preserves privacy leakage against website fingerprinting.

Challenges for Obfuscating Encrypted Traffic. However, producing adversarial examples on traffic traces is different compared to generating adversarial examples on images, where additional constraints must be considered [29], [36]. Specifically, due to the nature of network traffic, the sign of a perturbation should be the same as the sign of a burst. Put differently, if a burst is positive, then a perturbation should also be positive. If a burst is negative, then its perturbation should also be negative.

If the sign of a perturbation is opposite to the sign of a burst, then it suggests that network packets need to be buffered in order to be consistent with the perturbation. The buffered data will have impacts on all the following packets, which will significantly affect the network performance and the correctness of network protocols. To make it even worse, with different signs, if the absolute value of a perturbation is greater than the size of a burst, the direction of network packets needs to be changed (e.g., an outgoing packet becomes an incoming packet), which is not feasible in the real world.

Our Method: AdvTraffic. By considering the constraints we discussed above, we propose a new method, referred to as AdvTraffic, which can be integrated with any existing AE algorithm to produce adversarial examples on encrypted traffic. The main idea is to adjust perturbations generated by an AE algorithm before calculating adversarial examples on encrypted traffic. Perturbations having the same signs as bursts are kept while others are reset as 0s in the adjusted perturbations.

The details of AdvTraffic can be summarized as follows: given a sample x and a perturbation r generated by an AE algorithm, where $x = (x_1, ..., x_n)$ and $r = (r_1, ..., r_n)$, instead of directly computing a perturbed sample $x_{adv} = x + \epsilon \cdot r$, AdvTraffic first computes an adjusted perturbation $r^* = (r_1^*, ..., r_n^*)$ as,

$$\begin{cases} r_j^* = r_j & \text{if } x_j \cdot r_j \geq 0 \\ r_j^* = 0 & \text{otherwise} \end{cases} \tag{6}$$

for $1 \leq j \leq n$. Next, AdvTraffic computes perturbed sample $x_{adv} = x + \epsilon \cdot r^*$, where parameter ϵ is used to control the magnitude of the perturbation.

For example, assume we have a sample x and a perturbation r as

$$x = (x_1, ..., x_n) = (+2, -2, +4, -1, +6, +8)$$
$$r = (r_1, ..., r_n) = (-1, -1, +2, +2, +1, +1)$$

the adjusted perturbation r^* based on our algorithm is

$$r^* = (r_1^*, ..., r_n^*) = (0, -1, +2, 0, +1, +1)$$

where r_1^* and r_4^* are reset as 0s as $x_1 \cdot r_1^* < 0$ and $x_4 \cdot r_4^* < 0$.

Note that a floor function will be applied for each obfuscated packet size to keep it as integer. Moreover, if an original AE algorithm (e.g., DeepFool or PGD) runs multiple iterations to obtain adversarial examples, AdvTraffic is applied within each iteration accordingly.

Additional Assumptions. In order to leverage AdvTraffic to obfuscate traffic, we have a few additional assumptions. First, to successfully deploy our defense, the defense method has control over both the incoming and outgoing packets. This assumption is also used in WTF-PAD [16] and Mockingbird [29] for enabling website fingerprinting defenses.

Second, a target classifier for generating adversarial examples is the classifier that an attacker uses to predict websites in website fingerprinting. In other words, we assume a user, a server or a proxy who produces defended traffic know the structure and hyperparameters of a classifier that an attacker builds for fingerprinting websites. For example, an attacker may use a classifier from the current literature to pursue a high attack accuracy, where the structure and hyperparameters are public available.

In case of an even stronger attacker, who can utilize a different classifier for website fingerprinting compared to the target classifier used for generating adversarial examples, our defense can still be effective as adversarial examples are *transferable* (i.e., can still change the predictions) over different neural network classifiers. In one of our experiments presented in Sec. V, we will also show that even if the target classifier utilized in the generation of adversarial examples is different from the classifier that a website fingerprinting attacker runs, the attack accuracy can still be significantly reduced when our proposed defense applied.

B. AE Algorithms with AdvTraffic

We describe the details of the variation of each AE algorithm (introduced in Sec. III) that we can generate by integrating our proposed method AdvTraffic. For each variation built

upon AdvTraffic, we focus on the main difference compared to the original AE algorithm.

AdvTraffic-FGSM. The variation of FGSM integrating AdvTraffic can be highlighted as below:

- Compute perturbation $r \leftarrow \texttt{sign}(\nabla_x J(\theta, x, y))$
- Given sample $x = (x_1, ..., x_n)$ and $r = (r_1, ..., r_n)$, output an adjusted perturbation $r^* = (r_1^*, ..., r_n^*)$, where for $1 \leq j \leq n$ compute
  - $\diamond$ $r_j^* = r_j$, if $x_j \cdot r_j \geq 0$
  - $\diamond$ $r_j^* = 0$, otherwise
- Output an adversarial example $x_{adv} = x + \epsilon \cdot r^*$.

AdvTraffic-AdvGAN. The variation of AdvGAN integrating with AdvTraffic can be presented as below.

- Given $G(x)$ outputted by Generator, set $r \leftarrow G(x)$.
- Given sample $x = (x_1, ..., x_n)$ and $r = (r_1, ..., r_n)$, output an adjusted perturbation $r^* = (r_1^*, ..., r_n^*)$, where for $1 \leq j \leq n$ compute
  - $\diamond$ $r_j^* = r_j$, if $x_j \cdot r_j \geq 0$
  - $\diamond$ $r_j^* = 0$, otherwise
- Output an adversarial example $x_{adv} = x + r^*$.

Note that we only present the variations of FGSM and AdvGAN due to the space limitation. The variations of DeepFool and PGD follow the similar logic, where the perturbations applied only when the direction of each perturbation packet is consistent with the direction of original packet.

V. Performance Evaluation

A. Datasets

We leverage one existing large-scale dataset, named DF dataset, to measure the defense performance with our method.

DF dataset. This dataset was collected in [10]. It includes 95 monitored websites with 1,000 traces per website for the closed-world evaluation and 40,716 unmonitored websites with 1 trace per website for the open-world evaluation. The dataset for the closed-world setting is collected by visiting the homepage of each top-100 Alexa site 1,250 times. In the open-world setting, the dataset is collected by visiting Alexa's top 50,000 sites (1 trace per site), excluding the first 100 sites used in closed-world setting. More details can be found in [10].

Data Pre-Processing. Given the dataset, we further pre-process data and remove invalid traces by following the same pre-processing approach as [10]. Specifically, if a trace has less than 50 packets or it starts with an incoming packet, we consider it invalid and remove it.

After this pre-processing, each monitored website in DF dataset has at least 463 traces. For the ease of analyses, we take 460 traces per monitored website for all the 95 monitored websites in our experiments. For the unmonitored websites, 26,296 traces (i.e., 1 trace per website from 26,296 website) remain valid after the pre-processing. We use all the valid 26,296 traces in our evaluation.

In addition, same as previous studies [9]–[11], we also trim or pad each trace to a fixed length. Given this DF dataset (after pre-processing), we found that nearly 80% traces have a size of 512 or higher and over 98% traces have a size of 1024 or higher. We perform fingerprinting attacks with both sizes running a Convolutional Neural Network, named DF model (see in a later subsection), and derive 95% accuracy with length 512 and 93% accuracy with length 1024. For the ease of computation on adversarial examples, we select 512 as the trace length in our experiments.

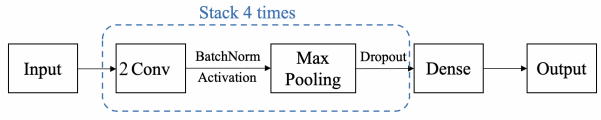

Fig. 3. Architecture of DF model

B. Experimental Setting

We implement the variations of AE algorithms integrating AdvTraffic and neural networks with Pytorch 1.3.0. We run all the experiments on a Linux machine with Ubuntu 18.04 OS 2.8 GHz CPU, 16 GB Memory, and a Nvidia GeForce GTX 1070 GPU.

We leverage NNI (Neural Network Intelligence) [37] to search the hyperparameters of a neural network. We use TPE (Tree-structure Parzen Estimator), which is one of the search algorithms rendered by NNI, as the search algorithm. We perform hyperparameter search with 100 trials at most or stop the search if it takes more than 100 hours.

C. Architectures of Neural Networks

DF model. We use Deep Fingerprinting (DF) model designed by Sirinam et al. [10] as the attack classifier in website fingerprinting. This DF model achieved 98% accuracy in the closed-world setting and outperformed several other models in [10]. Moreover, it can effectively defeat one existing defense (e.g., WTF-PAD). DF model is presented in Fig. 3.

Note that, we implement DF model in Pytorch in our experiment with the same architecture proposed in [10]. The original DF model was implemented in Tensorflow. In our implementation, we re-search the hyperparameters, such as in-channels, out-channels, kernel size, learning rate, and optimizer, etc. The tuned hyperparameters of DF model over DF dataset can be found in Appendix.

The Architecture of AdvGAN. For the Generator, it consists of an encoder, a transformer, and a decoder. This structure is based on the one used in image-to-image translation [38]. For the Discriminator, it is a CNN. More information about the structure of Generator and Discriminator can be found in Appendix. Note that, as the original implementation is designed for images, we modify the code to make it compatible with 1-dimensional data. We also customize some hyperparameters to gain a better performance in our experiments. In addition, rather than using real samples, we use random vectors that are produced by Gaussian distribution as the inputs to generate perturbations at the Generator.

D. Closed-World Evaluation

Experiment 1: Attack Results on Non-Defended Data. We first evaluate the attack performance of DF model

TABLE I
CLOSED-WORLD: ATTACK RESULTS ON NON-DEFENDED DATA

	DF model
Accuracy	95.0%
Training Time	36 minutes

on non-defended data, where we train/validate/test with 70%/10%/20% non-defended data respectively. As shown in Table I, given DF dataset, DF model obtains 95.0% accuracy after training 200 epochs and the training takes 36 minutes.

Experiment 2: Attack Results (Training with Non-Defended Data and Testing with Defended Data.) Next, we investigate the attack performance over defended data, where the classifier is trained using non-defended data and test with defended data. We leverage multiple algorithms, including AdvTraffic-FGSM, AdvTraffic-DeepFool, AdvTraffic-PGD, and AdvTraffic-AdvGAN, to produce defended data (i.e., adversarial examples of encrypted traffic traces). Parameters of each AE algorithm can be found in Appendix. DF model is leveraged accordingly as the target classifier when we produce defended data. As we can see from Table II, website fingerprinting performs poorly if a classifier is trained with non-defended data but tested with defended data. Specifically, except AdvTraffic-FGSM only mitigates the attack accuracy from 95.0% to 62.7%, others significantly reduce the attack accuracy to less than 8%.

TABLE II
CLOSED-WORLD: ATTACK ACCURACY (TRAINING WITH NON-DEFENDED DATA AND TESTING WITH DEFENDED DATA)

AE Algorithm	DF model
AdvTraffic-FGSM	62.7%
AdvTraffic-DeepFool	6.7%
AdvTraffic-PGD	7.9%
AdvTraffic-AdvGAN	**1.9%**

TABLE III
CLOSED-WORLD: ATTACK ACCURACY (TRAINING AND TESTING WITH DEFENDED DATA)

AE Algorithm	DF model
AdvTraffic-FGSM	92.4%
AdvTraffic-DeepPool	90.4%
AdvTraffic-PGD	88.5%
AdvTraffic-AdvGAN	**10.2%**

Experiment 3: Attack Results (Training and Testing with Defended Data). In this experiment, we examine the attack performance over defended data, where the classifier is trained and tested using defended data (i.e., adversarial training). This experiment setting captures a more real-world scenario where a (strong) website fingerprinting attacker learns which defense has been applied and re-trains its classifier over defended data. We still leverage the same AE algorithms from last experiment and use DF model (trained based on non-defended data) as the target classifier to produce defended data. Once we have defended data from each AE algorithm, we retrain DF model accordingly to examine the case of training and testing with defended data.

As reported in Table III, if a website fingerprinting attacker performs adversarial training, it can dramatically regain attack accuracy for most of the AE algorithms. However, if defended data is generated by AdvTraffic-AdvGAN, an attacker fails to regain its attack accuracy, which is only 10.2%. In other words, AdvTraffic-AdvGAN would be the best option for the defense if we aim to utilize adversarial examples to obfuscate traffic.

Experiment 4: Defense Overheads. In this experiment, we examine the overheads of our method when we integrate it with different adversarial examples. We examine the overheads in two aspects, the average generation time of an adversarial example for a traffic trace and the average perturbation size (i.e., the bandwidth overhead) that AE algorithms need to obfuscate a traffic trace. As shown in Table IV, AdvTraffic-AdvGAN is the fastest one in term of producing adversarial examples. It only takes 0.003 seconds on average to produce an adversarial example given a traffic trace. In addition, it produces a relatively low bandwidth overhead. On the other hand, AdvTraffic-AdvGAN needs a one-time pre-training time (1.33 hours) to train a Generator while other algorithms do not.

TABLE IV
DEFENSE OVERHEADS

AE Algorithm	Pre-Training Time	Generation Time	Bandwidth Overhead
AdvTraffic-FGSM	-	0.007 seconds	22.7%
AdvTraffic-DeepFool	-	0.014 seconds	7.6%
AdvTraffic-PGD	-	0.018 seconds	83.4%
AdvTraffic-AdvGAN	1.33 hours	0.003 seconds	23.0%

Experiment 5: The Impact of Cross-Classifiers (transferability). In this experiment, we examine the impact of cross-classifiers. Specifically, we assume that an attacker can further adapt and use a different classifier compared to the one used for generating adversarial examples over encrypted traffic. For instance, defended data are generated by DF model while an attacker utilizes a LSTM (Long Short-Term Memory) as the classifier during the attack. We are aiming to measure whether the defense is transferable in this cross-classifier scenario. In addition to DF model, we built a basic LSTM which includes 2 LSTM layers. With this LSTM, an attacker can achieve 80.5% accuracy on non-defended data over DF dataset. The details of its structure and parameters are listed in Appendix.

Given DF model and this LSTM, we use one of them as the target classifier to generate adversarial examples and use the other one as the attack classifier to perform website fingerprinting. In addition, we retrain the attack classifier with defended data. As we can see from Table V, even an attacker uses a different classifier than the one used in generating adversarial examples, our method AdvTraffic-AdvGAN can still effectively mitigate the attack accuracy. For instance, if the defended data are generated by DF model, it can still mitigate the attack accuracy to 4.6% if an adversary uses the LSTM as the attack classifier. Similarly, if the defended data are generated by LSTM model, it can mitigate the attack accuracy to 4.1% if an adversary uses DF model as the attack classifier. Therefore, we conclude that adversarial examples generated by AdvTraffic-AdvGAN are transferable, where the ones generated by one neural network can also affect other neural networks.

TABLE V
CLOSED-WORLD: ATTACK ACCURACY (TRAINING WITH DEFENDED DATA AND TESTING WITH DEFENDED DATA, BUT TARGET CLASSIFIER IS DIFFERENT FROM ATTACK CLASSIFIER)

AE Algorithm	Target: LSTM Attack: DF	Target: DF Attack: LSTM
AdvTraffic-FGSM	94.0%	60.9%
AdvTraffic-DeepFool	92.6%	24.5%
AdvTraffic-PGD	75.4%	7.2%
AdvTraffic-AdvGAN	**4.1%**	**4.6%**

Experiment 6: Comparison with Existing Defenses. In this experiment, we also compare our method, AdvTraffic-AdvGAN, with five state-of-the-art defenses, including Walkie-Talkie [17], WTF-PAD [16], Mockingbird [29], DFD [30] and Dolos [31]. We evaluate them against website fingerprinting attack with DF model on DF dataset. For the evaluation of accuracy, we focus on the results when adversarial training is involved (i.e., DF model is trained and tested with defended data). As we can see from Table. VI, an attacker can derive only 10.2% accuracy over the defended data generated by AdvTraffic-AdvGAN in our experiment, which outperforms all the existing defenses except one. For instance, both Walkie-Talkie and WTF-PAD fail to effectively reduce the attack accuracy when an adversary retrains the classifier with defended data. This observation is consistent with previous studies. Mockingbird and DFD can offer 33.3% and 12.3% accuracy respectively. Dolos mitigates the attack accuracy to 4%, which is the best defense in the comparison. However, Dolos requires higher bandwidth overhead (30%) than our method (23%).

For Walkie-Talkie, WTF-PAD, and Mockingbird, we implemented and evaluated based on the source code published from previous studies [16], [17], [29]. For DFD and Dolos (marked with *), as their source code are not publicly available, we compare them with the results reported in their papers [30], [31]. Both DFD and Dolos also use the same neural network architecture (DF model) and the same dataset.

TABLE VI
COMPARISON WITH EXISTING DEFENSE.

	Attack Accuracy (on defended traffic)	Bandwidth Overhead
Walkie-Talkie [17]	46.0%	149.6%
WTF-PAD [16]	86.2%	61.1%
Mockingbird [29]	33.3%	57.2%
DFD* [30]	12.3%	$\geq$ 85.0%
Dolos* [31]	**4.0%**	30.0%
AdvTraffic-AdvGAN (Ours)	10.2%	**23.0%**

E. Open-World Evaluation

Experiment 7: Attack Results on Non-Defended Data. In this experiment, we evaluate the open-world setting, which is more realistic in the real world. As mentioned, we use the standard model [5]–[7], [10] to evaluate the open-world setting. In this setting, the 26,296 traces from unmonitored websites are added as traces from an additional class to the classifier obtained from the closed-world setting. We retrain the classifier with traces from monitored websites and traces from unmonitored websites. For this additional class of unmonitored websites, we use 20,000 traces for training and 6,296 for testing. For other classes of monitored websites, we still use 80% per class for training and 20% per class for testing. The classifier aims to decide whether a trace is from monitored websites or unmonitored websites. DF model is utilized as the classifier. We tuned the pre-defined threshold for the confidence to obtain tuned precision or recall.

As we can observe from the results in Table VII, a website fingerprinting attacker can achieve very high precision and recall over non-defended data in the open-world evaluation. Specifically, if we tune the threshold for precision, DF model can achieve 100% precision and 83.9% recall. If we tune the threshold to obtain the highest recall, then DF model can achieve 88.6% recall with 96.9% precision. We also plot the ROC curve for non-defended data in Fig. 4.

TABLE VII
OPEN-WORLD: PRECISION & RECALL (TRAINING AND TESTING WITH NON-DEFENDED DATA)

Classifier	Tuned for Precision		Tuned for Recall	
	Precision	Recall	Precision	Recall
DF model	100%	83.9%	96.9%	88.6%

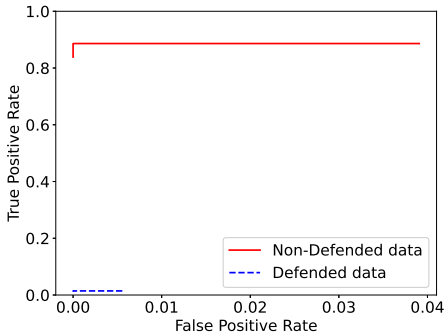

Fig. 4. ROC curve in the open-world setting. Defended data are generated by AdvTraffic-AdvGAN.

Experiment 8: Attack Results (Training and Testing with Defended Data). In this experiment, we examine the open-world setting when data are obfuscated with AE algorithms. As AdvTraffic-AdvGAN derives promising results in the closed-world setting in Experiment 3 while others are not, we only report the results of AdvTraffic-AdvGAN in the open-world evaluation and compare with other defenses in Table VIII. Unfortunately, we were not able to compare with defenses Dolos [31] and DFD [30] in the open-world setting as open-world evaluation was not evaluated in their original papers. The website fingerprinting classifier DF model is retrained over defended data and tested with defended data.

As we can see from Table VIII, AdvTraffic-AdvGAN outperforms the other three defenses in the open-world setting. Moreover, it significantly mitigates a website fingerprinting attacker's performance in the open-world compared to the results reported over non-defended data in Table VIII. For example, an attacker can achieve 100% precision but only derive 1.2% recall if it tunes the threshold for precision, while the state-of-the-art defense Mockingbird [29] achieves 100%

precision but with 30.8% recall. Similarly, if this attacker tunes for recall, it only obtains 78.3% recall and 1.4% precision, while Mockingbird [29] obtains 73.3% precision but with 45.3% recall. In other words, AdvTraffic-AdvGAN is effective as a defense in the open-world setting even when classifier retrains with defended data.

The corresponding ROC curves for DF model over the defended data, which generated by AdvTraffic-AdvGAN, are shown in Fig. 4. From the ROC curve, we can also seen that an attacker's capability is significantly mitigated in the open world setting by our proposed method. We also plot the precision-recall curves for different website fingerprinting defenses in the open-world evaluation in Fig. 5, where the classifier is retrained over defended data each time. It is obvious that AdvTraffic-AdvGAN can significantly alleviate the attack and outperforms other defenses in the open-world setting.

TABLE VIII
OPEN-WORLD: PRECISION & RECALL (TRAINING AND TESTING WITH DEFENDED DATA)

Defense	Tuned for Precision		Tuned for Recall	
	Precision	Recall	Precision	Recall
Walkie-Talkie [17]	100%	1.3%	69.7%	2.6%
WTF-PAD [16]	100%	2.4%	84.5%	4.6%
Mockingbird [29]	100%	30.8%	73.3%	45.3%
AdvTraffic-AdvGAN(ours)	100%	**1.2%**	78.3%	**1.4%**

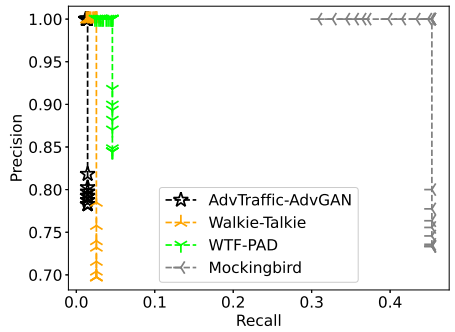

Fig. 5. Precision-Recall curve in the open-world setting over defended data.

VI. RELATED WORK

Website Fingerprinting. Many studies [1]–[8] have examined website fingerprinting. For instance, by manually selecting features and using traditional machine learning algorithms (e.g., kNN or SVM) as classifiers, studies in [5]–[7] can achieve more than 90% in website fingerprinting. Several studies [9]–[13] have proposed to utilize neural networks to promote attack accuracy of website fingerprinting over large-scale datasets. For example, Rimmer et al. [11] proposed website fingerprinting attacks by leveraging Stacked Denoising Autoencoders (SDAE), CNN and Long Short-Term Memory (LSTM). Their CNN achieved 96% in the closed-world setting with 900 websites and 2,500 traces per website while LSTM and SDAE obtained 94% and 95% respectively. Sirinam et al. [39] utilized triplet networks to perform website fingerprinting with few traffic traces. Rahman et al. [32] examined how to utilize timestamps of encrypted packets to infer websites. Wang et al. [40] leveraged adversarial domain adaptation to perform website fingerprinting when training and test traces are collected from different setups. Dani et al. [41] investigated the correlation of website content across traffic traces in website fingerprinting.

Defenses against Website Fingerprinting. Many defenses [4], [15]–[19] have also been proposed to preserve user privacy against website fingerprinting. The main approach is to obfuscate traffic traces such that it is more difficult for an attacker to distinguish traffic pattern. For instance, Juárez et al., [16] proposed a lightweight defense, named WTF-PAD, to protect Tor traffic pattern against website fingerprinting. WTF-PAD requires low overheads but it is not effective if an attacker retrains classifiers with neural networks [10]. Wang et al., [17] proposed a defense, named Walkie-Talkie, against website fingerprinting. This defense merges traffic traces of 2 websites into a super sequence such that an attacker cannot distinguish which website it is between the two. As a result, an attacker can achieve at most 50% accuracy in theory. Li et al. [42] examined an effective defense against cache-based website fingerprinting.

Zhang et al. [43] examined producing adversarial examples over encrypted traffic with FGSM. However, their method fails to preserve privacy if an attacker retrains a classifier with adversarial examples. Besides, they did not consider additional constraints for encrypted traffic in their study. Imani et al. [29] also proposed a method (named Mockingbird) to generate adversarial examples for encrypted traffic. The main idea is to obfuscate traces of monitored websites according to the traces of decoy websites. However, Mockingbird introduces greater bandwidth overhead than our method as its bandwidth relies on decoy websites, which could have high overheads depending on the manual selection. Nasr et al. [36] (referred as Blind) proposed three methods to produce adversarial examples for encrypted traffic depending on which data format is used (e.g., size-based, direction-based, time-based). The main idea of their methods is similar to ours as they utilize a Generator and a classifier to produce adversarial examples. Unfortunately, we are unable to experimentally compare this defense with others as it uses a different metric and its code and (defended) datasets are not publicly available.

VII. LIMITATIONS AND FUTURE WORK

In this section, we discuss the limitations of this study and future work. First, we do not consider perturbations on timestamps. This is because the majority of existing studies in website fingerprinting rely on size and direction rather than timestamps. Our method, however, could be potentially extended to the perturbations on timestamps, where a perturbation on a timestamp has to be non-negative. As the perturbed timestamps will be aggregated to all the later packets in the rest of a traffic trace, optimizations on the network delay will need to be considered. We will leave it as a future work. Second, we focus on white-box and semi-white-box settings in this study, and did not consider black-box setting for generating adversarial examples.

VIII. Conclusions

We propose a method, AdvTraffic, which can customize perturbations in order to produce adversarial examples over encrypted traffic. Our experimental results suggest that (1) it is feasible to produce adversarial examples over encrypted traffic to mitigate privacy leakage against website fingerprinting; (2) Our method built on top of GANs is able to produce obfuscated traffic to preserve privacy effectively and outperforms traditional defenses even when a website fingerprinting attacker adapts and retrains classifiers with defended data; (3) The obfuscated traffic generated by AdvTraffic-AdvGAN are even transferable across different models and architectures.

Acknowledgement

The authors would like to thank the anonymous reviewers for their insightful comments and suggestions. UC authors were partially supported by National Science Foundation (CNS-1947913).

References

[1] M. Liberatore and B. N. Levine, "Inferring the Source of Encrypted HTTP Connections," in *Proc. of ACM CCS'06*, 2006.

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

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

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

[5] T. Wang, X. Cui, R. Nithyannand, R. Johnson, and I. Goldberg, "Effective Attacks on Provable Denfenses for Website Fingerprinting," in *Proc. of 23rd USENIX Security Symposium*, 2014.

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

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

[8] T. Wang and I. Goldberg, "On Realistically Attacking Tor with Website Fingerprinting," in *Proc. of PETS'16*, 2016.

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

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

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

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

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

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

[15] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A Congrestion Sensitive Website Fingerprinting Defense," in *Proc. of 13th ACM Workshop on Privacy in Electronic Society*, 2014.

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

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

[18] S. Henri, G. Garcia-Aviles, P. Serrano, A. Banchs, and P. Thiran, "Protecting against Website Fingerprinting with Multihoming," in *Proc. of PETS'20*, 2020.

[19] J. Gong and T. Wang, "Zero-delay Lightweight Defenses against Website Fingerprinting," in *Proc. of USENIX Security'20*, 2020.

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

[21] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Exmaples," in *Proc. of ICLR'15*, 2015.

[22] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion Attacks against Machine Learning at Test Time," in *Proc. of Joint European Conference on Machine Learning and Knowledge Duscivery in Databases*, 2013.

[23] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks," in *Proc. of CVPR'16*, 2016.

[24] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical Black-Box Attacks against Machine Learning," in *Proc. of ACM ASIACCS'17*, 2017.

[25] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial Machine Learning at Scale," in *Proc. of ICLR 2017*, 2017.

[26] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks," in *Proc. of Workshop on Principled Approaches to Deep Learning*, 2017.

[27] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. McDaniel, "Ensemble Adversarial Training: Attacks and Defenses," in *Proc. of ICLR 2018*, 2018.

[28] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating Adversarial Examples with Adversarial Networks," in *Proc. of 27th International Joint Conference on Artificial Intelligence*, 2018.

[29] M. Imani, M. S. Rahman, N. Mathews, and M. Wright, "Mockingbird: Defending Against Deep-Learning-Based Website Fingerprinting Attacks with Adversarial Traces," 2021, iEEE Transactions on Information Forensics and Security.

[30] A. Abusnaina, R. Jang, A. Khormali, D. Nyang, and A. Mohaisen, "DFD: Adversarial Learning-based Approach to Defend Against Website Fingerprinting," in *Proc. of IEEE INFOCOM'20*, 2020.

[31] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Patch-based defenses against web fingerprinting attacks," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, 2021.

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

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Proc. of the International Conference on Nerual Information Processing Systems (NIPS 2014)*, 2014.

[34] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. of ICML'17*, 2017.

[35] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial Attacks and Defences: A Survey," 2018, https://arxiv.org/pdf/1810.00069.pdf.

[36] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating DNN-Based Traffic Analysis Systems in Real-Time With Blind Adversarial Perturbations." In the Proceedings of the 30th USENIX Security Symposium, 2021.

[37] Microsoft, "NNI: An open source AutoML toolkit for neural architecture search and hyper-parameter tuning," 2017. [Online]. Available: https://github.com/Microsoft/nni

[38] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," in *Proc. of IEEE CVPR'17*, 2017.

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

[40] C. Wang, J. Dani, X. Li, X. Jia, and B. Wang, "Adaptive Fingerprinting: Website Fingerprinting over Few Encrypted Traffic," in *Proc. of ACM CODASPY'21*, 2021.

[41] J. Dani and B. Wang, "HiddenText: Cross-Trace Website Fingerprinting over Encrypted Traffic," in *Proc. of IEEE Conference on Information Reuse and Integration for Data Science (IEEE IRI'21)*, 2021.

[42] H. Li, N. Niu, and B. Wang, "Cache Shaping: An Effective Defense Against Cache-Based Website Fingerprinting," in *Proc. of ACM CODASPY'22*, 2022.
[43] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang, "Statistical Privacy for Streaming Traffic," in *Proc. of NDSS'19*, 2019.
[44] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *European Conference*

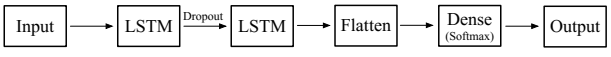

Fig. 11. Architecture of LSTM

Tuned Hyperparameters of DF model. We present the tuned hyperparameters of DF model in Table X.

Tuned Hyperparameters in AdvGAN. In AdvGAN, it includes a Generator, a Discriminator and a classifier. For the classifier, we use DF model, where its hyperparameters have been shown in Table X. The structure of the Generator is the same as the one in [28], [38]. The Generator includes an Encoder, 4 ResNet blocks and a Decoder. The hyperparameters of the Encoder and Decoder in the Generator are summarized in Table XI. We use the default hyperparameters in each ResNet block, and it can be found in [44]. For Discriminator, the hyperparameters are presented in Table XII.

TABLE XI
HYPERPARAMETERS OF ENCODE AND DECODE IN THE GENERATOR

Hyperparameters	Encoder	Decoder
In-channels	[1, 8, 16]	[32, 16, 8]
Out-channels	[8, 16, 32]	[16, 8, 1]
Kernel size	[3, 3, 3]	[3, 3, 6]
Stride	[1, 2, 2]	[2, 2, 1]
Padding	[0, 0, 0]	[0, 0, 0]
Activation function	[ReLU, ReLU, ReLU]	[ReLU, ReLU, Tanh]
Layer Type	Conv1d	ConvTranspose1d
Learning rate	0.001	
Optimizer	Adam	

TABLE XII
HYPERPARAMETERS OF DISCRIMINATOR

Hyperparameters	Size
In-channels	[1, 8, 16, 32]
Out-channels	[8, 16, 32, 1]
Kernel size	[4, 4, 4, 1]
Stride	[2, 2, 2, 1]
Padding	[0, 0, 0, 0]
Activation function	[LeakyReLU, LeakyReLU, LeakyReLU, Sigmoid]
Learning rate	0.001
Optimizer	Adam

Tuned Hyperparameters of CNN/LSTM in Experiment 5. The tuned hyperparameters of CNN (DF model) and LSTM are presented in X and XIII respectively. The details of the LSTM structure describe in Figure 11.

TABLE XIII
TUNED HYPERPARAMETERS OF LSTM

Hyperparameters	Size
Input size	512
Num_lstm_layer	2
Hidden size	256
Dropout	0.4
Learning rate	0.006
Optimizer	Adam
Bidirectional	False