AWA: Adversarial Website Adaptation

Amir Mahdi Sadeghzadeh, Behrad Tajali, and Rasool Jalili

Abstract—One of the most important obligations of privacy-enhancing technologies is to bring confidentiality and privacy to users' browsing activities on the Internet. The website fingerprinting attack enables a local passive eavesdropper to predict the target user's browsing activities even she uses anonymous technologies, such as VPNs, IPsec, and Tor. Recently, the growth of deep learning empowers adversaries to conduct the website fingerprinting attack with higher accuracy. In this paper, we propose a new defense against website fingerprinting attack using adversarial deep learning approaches called Adversarial Website Adaptation (AWA). AWA creates a transformer set in each run so that each website has a unique transformer. Each transformer generates adversarial traces to evade the adversary's classifier. AWA has two versions, including Universal AWA (UAWA) and Non-Universal AWA (NUAWA). Unlike NUAWA, there is no need to access the entire trace of a website in order to generate an adversarial trace in UAWA. We accommodate secret random elements in the training phase of transformers in order for AWA to generate various sets of transformers in each run. We run AWA several times and create multiple sets of transformers. If an adversary and a target user select different sets of transformers, the accuracy of adversary's classifier is almost 19.52% and 31.94% with almost 22.28% and 26.28% bandwidth overhead in UAWA and NUAWA, respectively. If a more powerful adversary generates adversarial traces through multiple sets of transformers and trains a classifier on them, the accuracy of adversary's classifier is almost 49.10% and 25.93% with almost 62.52% and 64.33% bandwidth overhead in UAWA and NUAW, respectively.

Index Terms—Website Fingerprinting, Privacy Enhancing Technologies, Adversarial Deep Learning, Adversarial Example.

I. INTRODUCTION

Website fingerprinting (WF) attack is one of the most serious threats against the anonymity of the users' browsing activities. It enables an adversary to determine which website is visited by a target user even she uses Privacy Enhancing Technologies (PETs), such as VPNs, IPsec, and Tor [1], [2], [3], [4], [5], [6], [7]. The target user is a general term referring to a client-side victim whom the adversary intends to monitor and figure out which specific websites she has been visiting through her browser application. When an adversary visits a website through PETs, she can extract a unique trace associated with the statistical features of that website's network flow, such as the sequence of packets' direction. The adversary can visit various websites several times and collect a trace set from those. To conduct the website fingerprinting attack, the adversary can train a classifier on the collected trace set and uses this classifier to predict the website that has been visited by the target user. The fundamental part of website fingerprinting attack is the classifier being used by the adversary to classify the target user's browsing activities. Recently, Deep

Neural Networks (DNNs) have shown great performance in classifying network traces of various websites[5], [6]. Besides, DNNs-based classifiers have a highly better performance on defenses that has been proposed in the previous studies against website fingerprinting attack, such as WTF-PAD[5].

However, researchers have recently shown that DNNs have a serious vulnerability, which is called adversarial example [8]. An adversarial example is a maliciously crafted input that causes classifiers to predict incorrectly. There are various methods to generate adversarial examples [8], [9], [10], [11], [12]. Adversarial examples are considered as an attack to the classifiers in the literature of adversarial machine learning. However, it can be considered as a defense mechanism against the adversary's classifier in the website fingerprinting domain. PETs can generate adversarial traces using conventional adversarial example generating methods, such as FGSM [9] and C&W [10] to cause the adversary's classifier to predict incorrectly. However, there is a major challenge in using adversarial traces as a defense against website fingerprinting attack. Since PETs are publicly available in the threat model of website fingerprinting attack, an adversary can generate adversarial traces of various websites by her PETs and trains a classifier on them. Since training on adversarial examples, called adversarial training [9], [13], is one of the most effective countermeasures against adversarial examples, the adversary's classifier being trained on adversarial traces can detect the true class of the target user's adversarial traces with a high success rate [14], [15]. Therefore, we need an adversarial trace generating method, which is more resistant against adversarial training in the threat model of website fingerprinting attack.

We propose Adversarial Website Adaptation (AWA), which is a new defense against website fingerprinting attacks. AWA generates adversarial traces that are more resistant to adversarial training. The output of AWA is a transformer set so that each website has a unique transformer. Each transformer consists of a generator that generates adversarial perturbations, which are added to traces to make adversarial traces. We accommodate secret random elements in the training phase of AWA to create various sets of transformers in each run. When two transformer sets are created by various secret random elements, they generate adversarial traces with different distributions. The critical assumption of AWA is that the adversary knows that AWA generates the traces of the target user; however, she has no knowledge about the secret random elements of the target user's transformer set. We demonstrate when an adversary and a target user generate their adversarial traces by different transformer sets, the accuracy of adversary's classifier is low in classifying target user's adversarial traces. AWA has two versions, including Universal AWA (UAWA) and Non-Universal AWA (NUAWA). Transformers in UAWA use adversarial perturbations that are independent of website traces.

The authors are with the Department of Computer Enginnering, Sharif University of Technology, Iran, Tehran, 11365-11155 (e-mail: amsadeghzadeh@ce.sharif.edu; behradtajali@ce.sharif.edu; jalili@sharif.edu).

arXiv:2012.10832v2 [cs.CR] 13 Apr 2021

Hence UAWA can generate adversarial traces on the fly, and there is no need to have access to the entire trace of a website before generating adversarial traces. Transformers in NUAWA use the entire trace of a website to generate adversarial traces.

We run AWA several times and generate multiple sets of transformers to evaluate the performance of AWA. We assume an adversary randomly selects a set of transformers and generates adversarial traces of various websites through them, then she trains a classifier on them. The target user also randomly selects a set of transformers and generates adversarial traces of her browsing activities through them. The results demonstrate that if the transformer sets of an adversary and a target user are different, the accuracy of adversary's classifier is almost 19.52% and 31.94% with almost 22.28% and 26.28% bandwidth overhead in UAWA and NUAWA, respectively. We also evaluate the performance of AWA, encountering a powerful adversary that can generate adversarial traces of various websites through multiple sets of transformers and train a classifier on them. The results indicate that AWA must impose more bandwidth overhead to traces to decrease the accuracy of the adversary's classifier in this setting. UAWA and NUAWA decrease the accuracy of adversary's classifier to almost 49.10% and 25.93% with almost 62.52% and 64.33% bandwidth overhead, respectively.

The main contributions of this paper are as follows:

- We propose AWA as a defense against website fingerprinting attack, which is more resistant against adversarial training. Also, AWA uses adversarial machine learning approaches and generates black-box adversarial traces.
- We present two versions of AWA, including universal AWA and non-universal AWA, and conduct multiple experiments to compare their performance.
- We introduce the concept of secret random elements in the context of generating adversarial traces.
- We propose intra-class distance criterion, which is used to justify the effectiveness of AWA.

The rest of the paper is organized as follows. In Sec. II, the preliminary and the threat model of website fingerprinting are described. Also, the preliminary of DNNs and basic methods for generating adversarial examples are introduced. Sec. III reviews previous studies on website fingerprinting attacks and defenses. In Sec. IV, AWA is presented. We also discuss the constraints of transformers in modifying network traces. Sec V evaluates the performance of UAWA and NUAWA in two scenarios. Sec. VI discusses how to use AWA in practice. Lastly, in Sec. VII, the conclusions of this study will be discussed.

II. BACKGROUND

The Onion Router (Tor) is one of the most popular PETs, which provides anonymity and protection from eavesdropping. Different kinds of research have been conducted previously to breach the anonymity feature of Tor [1], [2], [3], [4], [5], [6], [7]. Website fingerprinting is one such type of traffic analysis attack through extracting traffic patterns from the visited websites can form a unique fingerprint of acquired traffic flow and undermine Tor protection by detecting the website that the user has been just visiting.

Figure 1: A depiction for a typical WF attack scenario.

A. Website Fingerprinting (WF)

Three entities play vital roles in the website fingerprinting attack, an adversary, a target user, and Privacy-Enhancing Technologies (PETs). A target user utilizes PETs to keep her communications confidential and anonymous. Since most PETs are publicly available, an adversary can use them to generate a set of network flows from her favorite websites and train a classifier on them to predict the target user's browsing activities. AS adversary has no access to the label of network flows being generated by the target user, she can not evaluate the performance of her classifier. A website fingerprinting defense mechanism aims to obfuscate the features from which the adversary's classifier learns to recognize and differentiate network flows of a website from those of others. Likewise the previous studies, in this paper, we focus on Tor as the most popular PETs. However, AWA does not depend on the Tor architecture and can be generalized to any PETs.

1) Threat Model: Two initial presumptions are taken for the adversary in WF scenarios. First, the adversary is local, which means she is located somewhere between the Tor client and the Tor bridge where she can access the link. Second, the adversary is passive, meaning that she could only capture the ongoing packets and is not allowed to manipulate, drop, or delay any of them. Any potential eavesdropper conforming such potential features can can play the role of the adversary (e.g., Autonomous Systems, Internet Service Providers, and network administrators). The adversary is also considered to be aware of the client's identity. A common assumption in the literature is that an adversary can parse the network flows of various websites, and isolate it from the other kinds of network traffic. Figure 1 depicts the threat model of website fingerprinting attack.

An essential point about any defense mechanism being implemented on PETs is that an adversary has access to network traffic being generated by the mechanism. For the success of an attack, the adversary needs to capture a set of network flows while visiting a collection of websites, notably those that she is interested to detect. Thereafter, some unique features shall be extracted from each network flow that helps each website become conspicuous to classifiers, amongst other websites. There are loads of such features in WF literature, such as packet size [1], time and volume of traffic [16], edit-distance score [17], and the rate of traffic bursts in both directions [18]. By doing so, the adversary could attain several feature vectors that are used to train her supervised classifier. We consider each feature vector of a website's network flow as a website

trace in the rest of this paper.

There are two settings to evaluate WF attacks and defenses: closed-world and open-world. The closed-world setting assumes that the target user only visits the websites belonging to a monitored set, and the adversary also uses the traces of the same set for training of her classifier. In the more realistic open-world setting, the target user can visit arbitrary website regardless of the monitored set. Hence, the adversary needs to identify if a website belongs to the monitored set and to distinguish between monitored websites. Since the target user is restricted to visit the websites belong to the monitored set in the closed world setting, the adversary has an easier job of classifying her visited websites than in the open-world setting. Hence, defeating the adversary in this setting would be considered as overcoming her in the most advantageous situation. We only evaluate AWA in the close-word setting, which is the worst setting for a defense mechanism.

2) Website Trace Representation: The communication of a client and a server is in the format of a bidirectional flow. A bidirectional flow is a sequence of packets that share the same source and destination IP addresses, port numbers, and protocol. Recent studies [19], [6], [5] have demonstrated that the direction of packets in bidirectional flows is enough for website fingerprinting attacks. Since recent attacks [19], [6], [5] and defenses [14], [20], [21] have focused on the sequence of packets direction, we also use the same features in this study. Each trace of a website is a sequence of packets direction in the order that they are received, which is called Direction Sequence (DS). For DS^i, we have:

$$DS^i = <d_1^i, d_2^i, ..., d_n^i>, \quad s.t. \quad d_j^i \in \{+1, -1\}, \quad (1)$$

where n is the number of packets and d_j^i is the jth packet direction of trace i. The direction of packets from the client to the server is $+1$ and from the server to the client is -1. In this study, we consider each website trace as a Burst Sequence (BS) to simplify making perturbation for each trace. Burst is a sequence of consecutive packets all having the same direction. The burst sign is the sign of its including packets. BS is a sequence of bursts sizes multiplied to the bursts directions. For BS^i, we have:

$$BS^i = <|B_1^i| \times D_1^i, |B_2^i| \times D_2^i, ..., |B_m^i| \times D_m^i>,$$
$$s.t. B_j^i = <d_1^j, d_2^j, ..., d_l^j>, |B_j^i| = l, D_j^i \in \{+1, -1\}, d_k^j = D_j^i, D_p^i = -1 \times D_{p-1}^i, k \in [1, l], j \in [1, m], p \in [2, m] \quad (2)$$

where m is the number of bursts, B_j^i is the jth burst of BS^i, and the D_j^i is the direction of this burst, which is the same as all packets in B_j^i. As an example, suppose that $DS^i = <+1, -1, -1, -1, +1, +1, -1, -1>$. The direction sequence DS^i has four bursts, $B_1^i = <+1>$, $B_2^i = <-1, -1, -1>$, $B_3^i = <+1, +1>$, and $B_4^i = <-1, -1>$. Hence, $|B_1^i| = 1, D_1^i = +1$, $|B_2^i| = 3, D_2^i = -1, |B_3^i| = 2, D_3^i = +1$, and $|B_4^i| = 2, D_4^i = -1$. The burst sequence i is $BS^i = <1 \times (+1), 3 \times (-1), 2 \times (+1), 2 \times (-1)> = <+1, -3, +2, -2>$. In the rest of study, each website trace is a burst sequence, and a trace set is a set of burst sequences.

B. Deep Neural Networks

Deep neural networks (DNNs) are multi-layer functions that receive $x_i \in \mathcal{X}$ as input and output $y_i = f_{DNN}(x_i)$ ($y_i \in \mathcal{Y}$). DNNs are revealed to have high performance on raw data extracted from network traffic and do not need manual feature engineerings to reach high accuracy. Three important DNNs which are mostly used in network traffic classification are Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), and Stacked Denoising Autoencoders (SDAE). Amongst which, One-Dimensional Convolutional Neural Network (1D-CNN) has got the best results in network traffic classification tasks [6], [22], [5]. A neural network consists of three parts: an input layer, a bunch of hidden layers inside, and an output layer. Each layer includes a set of neurons with non-linear activation functions and parameters θ (weights and biases) related to them. Different architectures of deep neural networks have different numbers of parameters. During the training phase, the optimum values for the network parameters are calculated so that the loss function J, which indicates the distance between actual and predicted labels in supervised classification, will be minimized: $\theta^* = \underset{\theta}{\operatorname{argmin}} J(\theta, x, y)$. The optimization is usually done using some common versions of the Stochastic Gradient Descent (SGD). SGD is an iterative method, where θ is initialized randomly and is updated iteratively using the gradient of the loss function with respect to the parameters in every iteration. The gradient is calculated using a mini-batch of training data. The size of a mini-batch is often 64, 128, or 256, and the data included in the batch is often randomly selected from training data. It is notable that putting different data in the mini-batch changes the gradients of the loss function with respect to parameters. Therefore, θ^* depends on the initialization of θ and the order of the training data which SDG is confronted with in every mini-batch.

C. Adversarial Examples

In spite of their remarkable achievements and high performance in complicated tasks, DNNs have been demonstrated having a critical vulnerability [8]. Adversarial examples are intentionally crafted inputs, which cause the victim classifier f to make a mistake about the correct label of the input. Considering f^* as a classifier that always designates the correct label, we define adversarial example x' as:

$$f^*(x') = y, \quad f(x') = y', \quad s.t. \, y' \neq y \quad (3)$$

The typical approach to make adversarial example x' is adding the adversarial perturbation vector η to the real input x like: $x' = x + \eta$. After crafting x', the actual class of x' and x should be the same. Several methods have been introduced to generate adversarial examples so far, of which Fast Gradient Sign Method (FGSM) [9] and Carlini and Wagner (CW) [10] are two popular ones. Moosavi-Dezfooli *et al.* [11] proposed universal adversarial perturbations for the first time where the adversary creates adversarial perturbation independent of a particular input. In this attack, a universal adversarial perturbation is added to a set of samples and cause the target classifier to predict the label of most of samples wrongly.

Adversarial examples are considered as an attack to classifiers in the literature of adversarial machine learning. However, it can be considered as a defense mechanism against the adversary's classifier in the website fingerprinting domain. Tor can implement adversarial example generating methods such as FGSM [9] and C&W [10] and use them to perturb traces of various websites and generate adversarial traces to cause the adversary's classifier to predict incorrectly. Such a defense mechanism does not work because of the threat model of website fingerprinting attacks. Since Tor is publicly available, an adversary can generate adversarial traces of various websites and train a classifier on them. In the adversarial machine learning literature, training on adversarial examples is one of the most effective defenses against adversarial example attacks [9], [13]. Therefore, when an adversary trains a classifier on the adversarial traces, she is doing adversarial training, and her classifier becomes more robust against adversarial traces. Accordingly, where Tor uses adversarial example generating methods as a defense mechanism against adversary's classifier, adversarial training is a part of the threat model of website fingerprinting attack. Saidur et al. [14] and Zhang et al. [15] demonstrate that, because of adversarial training, adversarial traces being generated by FGSM [9] or C&W [10] are not effective against the adversary's classifier. We propose AWA as a new defense mechanism against website fingerprinting attacks that generates adversarial traces which are more resistant against adversarial training.

D. Maximum Mean Discrepancy (MMD)

Given two sets of data $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m\}$, drawn identical independent distribution (i.i.d.) from distributions P and Q respectively, MMD criterion empirically estimates the distance between P and Q in Reproducing Kernel Hilbert Space (RKHS). An RKHS H_k is a space of functions with a kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ such that $f(x) = < f, k(., x) > \forall f \in H_k$. In other words, MMD considers the distance between the embedded mean of two distributions as the distance between them. MMD is defined as follows:

$$MMD(X, Y) = \left\| \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) - \frac{1}{m} \sum_{i=1}^{m} \phi(y_i) \right\|_{\mathcal{H}_k} \quad (4)$$

where $\mathcal{H}$ is a universal RKHS, $\phi(.) \in \mathcal{H}$ is the mapping of input space $\mathcal{X}$ to the RKHS, and $k(.,.) = < \phi(.), \phi(.) >$ is the universal kernel associated with this mapping. MMD can be easily approximated by sampling from distributions P and Q. We use MMD with a Gaussian kernel to estimate the distance between two distributions of websites.

III. RELATED WORKS

This section reviews the most prominent website fingerprinting attacks and defenses presented so far.

A. Website Fingerprinting Attacks

Before the outbreak of deep learning, most of the studies used to apply manual feature engineering and conventional machine learning classifiers. The first WF attack is presented by Hermann et al. [1], where they reach just 3% accuracy for the closed world scnario containing 775 websites. Following that, Panchenko et al. [16] achieve 55% accuracy with enhanced extracted features on the same dataset. WF attacks gradually improve up to 90% success rate using edit-distance [23], [17]. Wang et al. [18] using a k-nearest neighbor classifier on a combination of features achieve 91% accuracy in the closed-world setting containing 100 websites. Panchenko et al. [3] introduce an attack based on a Support Vector Machine (SVM). For the closed-world setting, their method achieves 91% accuracy. Hayes and Danezis [2] suggest k-Fingerprinting attack (k-FP)The k-FP attack achieves 91% accuracy for closed-world setting. Zhuo et al. [7] propose a new attack using Profile Hidden Markov Models. This attack achieves 99% accuracy on SSH and Shadowsocks network traffic in the closed world setting.

Abe and Goto [19] introduce the first WF attack employing DNNs. They use a Stacked Denoising Autoencoder (SDAE) classifier, and instead of whole handcrafted features, used in previous studies, they feed their model just with a sequence of raw packets direction. They report 88% accuracy on the closed-world scenario. Rimer et al. [6] propose Automated Website Fingerprinting (AWF) attack, which utilizes multiple DNN structures, including SDAE, CNN, and LSTM to classify the sequence of packets direction for various website. According to the obtained results, their CNN model outperforms others with 96% accuracy for the closed-word scenario. Sirinam et al. [5] develop a new CNN classifier which surpasses all previous ones and achieves more than 98% accuracy in closed-word settings. The authors collect a dataset for 100 websites to evaluate their classifier. They claim that the sequence of packets direction is enough for classifying websites traces with high accuracy. Furthermore, their attack shows high performance against traces being protected by WTF-PAD [24] and Walkie-Talkie [25] methods. Bhat et al. [26] using the ResNet-18 architecture [27] introduce VAR-CNN that could gain 98.8% accuracy in closed-world settings. VAR-CNN uses the sequence of packets direction, inter-arrival time, and cumulative statistical features to classify traces.

B. Website Fingerprinting Defenses

One of the early defenses against website fingerprinting attack is proposed by Dyer et al. in [28] called Buffered Fixed-Length Obfuscator (BuFLO). BuFLO imposes high bandwidth and latency overhead on traces. Later, some extended versions of BuFLO, such as Tamaraw [29] and CS-BuFLO [30] are proposed to solve this problem. By employing the Adaptive Padding method [31], Juarez et al. introduce WTF-PAD [24]. WTF-PAD tries to obfuscate the inter-arrival time feature by filling the abnormal time gap between two packets of a sequence with dummy ones. Nithyanand et al. in [32] and Wang et al. in [18] almost concurrently propose the idea of finding a representation for all traces that are close to each other and this representation is called super trace [32] or supersequence [18]. Both defenses use clustering to find traces that are close to each other. Then, every trace must be padded so that it

becomes similar to the supersequence of the cluster to which it belongs. Wang *et al.* in another study [18] present Walkie-Talkie (WT) that has two features: Burst-molding and half-duplex communications. WT has 31% and 34% bandwidth and latency overhead, respectively. Since Walkie-Talkie works on the half-duplex links, it has some issues in practice.

Some new methods recently have employed adversarial machine learning approaches to develop a defense against website fingerprinting attack. By leveraging the adversarial example notion, Saidur *et al.* [14] develop their defense method called Mockingbird. The defense procedure starts with selecting a target trace from a pool and calculating the gradient of the distance between the source trace and the target trace. Then it tries to move toward the target iteratively until the detector is deceived. Mocking Bird can reduce the accuracy of Deep Fingerprinting classifier [5] down to 35.2% with almost 56% bandwidth overhead. Zhang *et al.* [15] propose two defenses against video fingerprinting attacks using differential privacy called FPA and d^*. FPA and d^* impose 200% and 600% bandwidth overhead. FPA decreases the accuracy of the target classifier from 94% to almost 20%. As the authors mentioned in their paper, unlike streaming traffic, HTTP traffic is more interactive. Therefore, the proposed defenses are not applicable in the website fingerprinting domain. Nasr *et al.* [20] propose Blind Adversarial Network Perturbations (BANP). The method injects universal perturbations into the traffic stream, leveraging remapping functions. The authors indicate that Deep Fingerprinting [5] attack has 8% accuracy against BANP, and their method imposes 11.11% bandwidth overhead on traces. Abusnaina *et al.* [21] present Deep Fingerprinting Defender (DFD) approach. DFD consists of two modules: the burst observer and the injection module. It works by injecting dummy packets dependent on the passing burst in either one-way or two-way operation mode. With 14.26% bandwidth overhead, they decrease the accuracy of Deep Fingerprinting [5] attack to about 7.29%. Although BANP [20] and DFD [21] evaluate their methods when the adversary is aware of the defense, the procedures of generating adversarial traces for adversary and target user are different in their evaluation. Since we can not choose which versions of PETs belong to adversary and target user in the threat model of website fingerprinting attack, the procedures of generating adversarial traces for them must be the same.

IV. ADVERSARIAL WEBSITE ADAPTATION

We introduce Adversarial Website Adaptation (AWA) as a new defense mechanism against website fingerprinting attacks that generate adversarial traces which are more resistant against adversarial training. We assume statistical features of each website W has a unique distribution D_W, and traces of website W come from this distribution. Also, we have access to the trace set TS and label set Y_{TS}, including traces and labels of all monitored websites. Trace set of website W is called TS_W, and it can be used to estimate the distribution of website W (empirical distribution $\hat{D}_W$). Each website has a unique transformer in AWA, and the goal of a transformer is to transform the distribution of the associated website

to adapt to the transformed distribution of another website. Each transformer has a generator that generates adversarial perturbation to change a website trace, and since the size of adversarial perturbation controls the magnitude of bandwidth overhead, it must be minimized.

AWA has two versions, including Non-Universal AWA (NUAWA), and Universal AWA (UAWA). NUAWA needs to have access to the entire trace of a website before generating adversarial perturbations. UAWA uses universal adversarial perturbations and does not need to access the entire trace of a website before generating adversarial perturbations. Recently, Sadeghzadeh *et al.* [33] and Nasr *et al.* [20] have demonstrated that using universal adversarial perturbation is effective in evading DNNs-based network traffic classifiers. In practice, UAWA generates a set of universal perturbations for each website, and whenever a user wants to visit a website, the pre-made perturbation of that website is added to the trace of user on the fly. The only difference between NUAWA and UAWA is in their generators' inputs. They are fed by the trace set TS in NUAWA and the noise set Z in UAWA. The whole process of transforming the traces of website W is denoted as $T_W(TS_W, Z)$ where T_W is the transformer of website W, TS_W is the trace set of website W, and Z is a noise set.

We explain AWA for a pair of websites and then extend it to K websites. AWA has a framework that changes the distributions of a pair of websites. Suppose that website A has been paired with website B, and the distributions of websites A and B are D_A and D_B, respectively. The transformers T_A and T_B change the distributions D_A and D_B to D'_A to D'_B, respectively, so that D'_A and D'_B become close together. However, the size of change in distributions should be limited due to bandwidth overhead. Therefore, the AWA framework goal is to minimize the distance between $T_A(TS_A, Z)$ and $T_B(TS_B, Z)$ with the minimum amount of change on traces to minimize bandwidth overhead. We utilize parameterized generators G_A and G_B in T_A and T_B, respectively, and a parameterized discriminator D_{AB} to minimize the distance between $T_A(TS_A, Z)$ and $T_B(TS_B, Z)$. The discriminator D_{AB} wants to differentiate between $T_A(TS_A, Z)$ and $T_B(TS_B, Z)$. On the contrary, T_A and T_B want to prevent D_{AB} from differentiating between them. When the discriminator fails to differentiate between $T_A(TS_A, Z)$ and $T_B(TS_B, Z)$, the distributions D'_A and D'_B are adapted. We use an Auxiliary Classifier AC to make transformers move traces of both classes, not just one of them. Auxiliary Classifier is a simple classifier that has been trained on the traces of all websites before transformation. In order to apply the AWA framework for K classes of websites, the following three phases are introduced

- **Pre-training phase:** Auxiliary classifier is trained on the network traces of K websites.
- **Training phase:** First, $K/2$ pairs of websites are randomly selected. Then, generators and discriminator of the proposed framework are trained for each pair on the pre-collected set of network traces. The output of this phase is a transformer set consists of K transformers for each website.
- **Testing Phase:** New clean traces of each website are transformed by the respective transformer.

(a) Training Phase　　　　**(b)** Testing Phase

Figure 2: An overview of Adversarial Website Adaptation (AWA) framework for two websites A and B. In the AWA training phase, the generators G_A and G_B are trained to make the transformers' outputs indistinguishable for discriminator D_{AB} and decrease the logits value of auxiliary classifier AC for the true class of traces. The inputs of transformers are noise set Z in UAWA and trace sets TS_A and TS_B in NUAWA. In the testing phase, the new traces new_trace_A and new_trace_B are transformed by trained transformers T_A and T_B, respectively.

AWA can be run several times to create multiple sets of transformers. Figure 2 shows the overview of AWA framework in training and testing phases for pair websites A and B.

A. Transformers

There are three constraints to transform traces of a website.

1) Transformers can not remove any packets from traces; otherwise, the functionality of network traffic is disrupted. Hence, adding dummy packets is the only way to transform traces.
2) As mentioned in section 2, we consider traces of each website as a burst sequence. The burst sequence consists of integer numbers, and it is discrete. The output of transformers must be integer numbers.
3) If a burst in a trace is broken into two bursts by inserting a dummy packet having the opposite direction in the middle, the latency overhead is imposed on traces. Each breaking of a burst imposes two round trip times overhead on a trace. Since it is not intended to impose latency overhead on traces, transformers only add dummy packets at the end of bursts in the same direction.

Each transformer consists of a generator, which feeds using a noise set or a trace set and generates a perturbation vector. The elements of perturbation vector specifies how many dummy packets are added to each burst of input trace. The first element of perturbation vector is added to the first positive burst of trace, and if a trace starts with a negative burst, the perturbation vector is shifted one element to the right. The value of perturbation vector is all positive, and it is multiplied by the sign of the target trace. Then it is added to the target trace to preserve the constraints 1 and 3. As mentioned, the input of generators is different in NUAWA and UAWA. In NUAWA, the input of a generator is a trace set , and in UAWA, a generator is fed by a noise set. Although there is no need to the noise set Z in NUAWA, we use the same notation $T_W(TS_W, Z)$ for transformers in NUAWA and UAWA for simplicity. For T_W

we have:

$$\text{NUAWA: } T_W(TS_W, Z) = G(TS_W) \times sign(TS_W) + TS_W$$
$$\text{UAWA: } T_W(TS_W, Z) = G(Z) \times sign(TS_W) + TS_W$$
$$(5)$$

Although the output of transformers is not rounded in the training phase, it is rounded in the testing phase to preserve constraint 2. The amount of BandWidth Overhead (BWO) that T_W adds to a trace tr is defined as follows:

$$BWO(\%) = \frac{\| \, |T_W(tr, Z)| - |tr| \, \|_1}{\| \, |tr| \, \|_1} * 100 \qquad (6)$$

where $\| \, |x| \, \|_1$ is the sum of absolute values of x.

B. Loss Functions

The AWA framework has three kinds of loss functions to optimize the parameters of generators, discriminator, and auxiliary classifier. The discriminator's goal is to predict the label of generators correctly. However, the generators' goal is to bring close the distributions of their outputs together to confuse the discriminator and evade the auxiliary classifier. It is improper that generators add a lot of dummy packets to traces to change their distributions. Hence, bandwidth overhead loss restricts generators not to adding high bandwidth overhead to traces. The loss function of the auxiliary classifier is cross-entropy, which is a standard supervised learning loss, and is as follows:

$$\min_{\theta_{AC}} \mathcal{L}_{AC}(TS, Y_{TS}) = -\mathbb{E}_{(x,y)\sim(TS,Y_{TS})} \sum_{k=1}^{K} \mathbb{I}_{(y=k)} \log AC(x)_k$$
$$(7)$$

where K is the number of websites in TS, $\mathbb{I}_{(y=k)}$ is one if $y = k$; otherwise is zero, $AC(.)_k$ is the kth element of AC output, and θ_{AC} is the parameters of AC. The output of discriminator is the label of the transformer from which the input of discriminator has been generated. Because there are

only two transformers, their labels are zero and one. The loss function of D_{AB} is defined as follows:

$$\min_{\theta_{D_{AB}}} \mathcal{L}_{D_{AB}}(TS_A, TS_B, \mathcal{P}) =$$
$$- \mathbb{E}_{x_A \sim TS_A, z \sim \mathcal{P}}[\log D_{AB}(T_A(x_A, z))] \qquad (8)$$
$$- \mathbb{E}_{x_B \sim TS_B, z \sim \mathcal{P}}[\log(1 - D_{AB}(T_B(x_B, z)))]$$

where $\theta_{D_{AB}}$ is the parameters of discriminator D_{AB} and $\mathcal{P}$ is a random noise distribution. The loss function of generators consists of three parts. For simplicity, we only introduce the loss function of G_A, and the loss function of G_B can be defined in the same way. $\mathcal{L}_{G_A\_AC}$ is the value of AC logits for the true class of the output of transformer. Logits is the output of the Penultimate layer in DNNs. The purpose of this loss is to make the AC predict the label of the output of T_A wrongly. We have:

$$\mathcal{L}_{G_A\_AC}(TS_A, Y_{TS_A}, \mathcal{P}) = \mathbb{E}_{(x,y) \sim (TS_A, Y_{TS_A}), z \sim \mathcal{P}}$$
$$\sum_{k=1}^{K} \mathbb{I}_{(y=k)} max(logits_{AC(T_A(x,z))_k}, 0) \qquad (9)$$

where $logits_{AC(T_A(x,z))_k}$ is the kth element of the output of AC logits, and Y_{TS_A} is the label of website A. $\mathcal{L}_{G_A\_OH}$ regulates the magnitude of change that transformers add to traces. The magnitude of change between the input and output of transformers is the bandwidth overhead that AWA imposes on traces, which must be minimized. We have:

$$\mathcal{L}_{G_A\_OH}(TS_A, \mathcal{P}, \tau_{low}, \tau_{high}) = \mathbb{E}_{x \sim TS_A, z \sim \mathcal{P}} \ max($$
$$\frac{\| |T_A(x,z)| - |x| \|_1}{\| |x| \|_1} - \tau_{high}, 0) - min(\frac{\| |T_A(x,z)| - |x| \|_1}{\| |x| \|_1}$$
$$- \tau_{low}, 0)$$
$$(10)$$

where τ_{high} and τ_{low} determine the level of bandwidth overhead inducing no penalty, and $\| |.| \|_1$ is the sum of absolute values. τ_{high} controls the upper bound of bandwidth overhead, and if overhead is more than $\tau_{high} \times 100$, $\mathcal{L}_{G_A\_OH}$ increases. Since sometimes some generators do not move websites distributions, we add a loss term that makes generators move the distributions of websites. This term using τ_{low} makes generators to add some bandwidth overhead to traces, and it increases if overhead is less than $\tau_{low} \times 100$. We use the domain confusion objective that has been proposed by Tzeng $et\ al.$ [34] as the third part of G_A loss function. The discriminator's output is a value between zero and one, and when the output is closer to zero, the discriminator's prediction is the transformer with label zero; otherwise, the discriminator's prediction is the transformer with label one. When the distributions of transformers' output are very close together, the discriminator fails to differentiate between them, and its output becomes 0.5. Therefore, this part of the loss function makes transformers change their outputs so that discriminator's output become 0.5. The domain confusion loss function is defined as follows:

$$\mathcal{L}_{G_A\_DC}(TS_A, \mathcal{P}) = -\mathbb{E}_{x \sim TS_A, z \sim \mathcal{P}}$$
$$[\frac{1}{2} \log D_{AB}(T_A(x,z)) + \frac{1}{2} \log(1 - D_{AB}(T_A(x,z)))] \qquad (11)$$

The complete loss of G_A is as follows:

$$\min_{\theta_{G_A}} \mathcal{L}_{G_A}(TS_A, \mathcal{P}, \tau_{low}, \tau_{high}, \alpha, \beta, \gamma) =$$
$$\alpha \ \mathcal{L}_{G_A\_AC} + \beta \ \mathcal{L}_{G_A\_OH} + \gamma \ \mathcal{L}_{G_A\_DC} \qquad (12)$$

where θ_{G_A} is the paprameters of G_A, and α, β, and γ regulate the impact of each part of loss in $\mathcal{L}_{G_A}$.

C. Secret Random Elements

Although we move the distributions of $K/2$ pairs of websites towards each other, because of the limitation on bandwidth overhead being imposed on traces, websites distributions do not become indistinguishable. AWA must add a lot of bandwidth overhead to the traces of paired websites in order to become indistinguishable. This is impractical in the real world. Suppose that we run AWA once and create a transformer set S to generate adversarial traces, and also website A has been paired by website B in the training phase of AWA. T_A^S and T_B^S are the transformers of websites A and B in S, respectively. T_A^S changes the distribution of website A from D_A to D'_A, and T_B^S also changes the distribution of website B from D_B to D'_B. Since websites A and B were paired in the training process of S, the goal of AWA's framework is to bring D'_A and D'_B close together. If an adversary has no access to AWA, she only can train a classifier on samples of D_A and D_B. In this setting, if the target user has access to AWA and generates her adversarial traces with D'_A and D'_B distributions, the adversary's classifier being trained on the samples of D_A and D_B has low accuracy on the target user traces due to the auxiliary classifier in the AWA's framework.

If we put S in publicly available PETs, an adversary also can generate adversarial traces of different websites through S and train a classifier on them. The target user also uses PETs and generates her adversarial traces through S. In this setting, the adversary has access to D'_A and D'_B, and she can train a classifier on samples of them. Since D'_A and D'_B are not indistinguishable and the target user also generates her traces based on these distributions, the adversary's classifier being trained on samples of D'_A and D'_B has high accuracy on the target user's adversarial traces.

When an adversary trains a classifier on adversarial traces, she is doing adversarial training. Zhang $et\ al.$ [35] demonstrate that adversarial training performance strongly correlates with the distance between the distribution of test data and the distribution of training data. Test samples that are far from the distribution of training data are more likely to be classified wrongly. In the website fingerprinting threat model, the training data of the adversary's classifier and also its test data (which is in fact the traces obtained from target user) are generated by the defense mechanisms of PETs. AWA as a defense mechanism is able to control both the training and the test data distributions of the adversary's classifier. Therefore, if AWA generates adversarial traces with different distributions for the adversary and the target user, the adversary's adversarially trained classifier would fail and get low accuracy on the target user's adversarial traces. However, there are multiple users in the real world, and we do not know which one is an adversary and which one is a benign user. Therefore, AWA

must generate adversarial traces of each user by a unique distribution so that when a user, which can be an adversary, trains a classifier on her adversarial traces, can not classify other users' adversarial traces with a promising success rate. With this purpose, AWA must create multiple sets of transformers so that each transformer set generates adversarial traces with different distribution from others. Also, an individual user should not be aware of what transformer set other users have chosen. Otherwise, an adversary can generate her adversarial traces by the same transformer set that the target user has picked up, and in this setting, the accuracy of the adversary's classifier is high.

We accommodate secret random elements in AWA to achieve these aims by inspiring the concept of keys in the cryptography literature. Based on the Kerckhoffs' principle [36], it is assumed the encryption scheme is known to the adversary, but the key itself remains secret. Here in the website fingerprinting threat model, we have a similar assumption that PETs are publicly available, and thus the adversary is able to generate adversarial traces of any chosen website. However, there is no secret key in the defenses against website fingerprinting attack. Notably, the functionality of keys in the cryptography field and secret random elements are very different. The only similarities between these two concepts are randomness and secrecy. Secret random elements make AWA generate a different set of transformers in each run. Suppose that we run AWA twice with various random elements to create two transformer sets S_1 and S_2. An adversary uses S_1, and a target user uses S_2. As an example, the adversary changes the distribution of website A using $T_A^{s_1}$ from D_A to $D_A^{s_1}$, and the target user changes the distribution of website A using $T_A^{s_2}$ from D_A to $D_A^{s_2}$. If $D_A^{s_1}$ and $D_A^{s_2}$ are far enough from each other, the adversary's classifier being trained on the samples of $D_A^{s_1}$ reaches a low accuracy on the target user's adversarial traces being generated by $D_A^{s_2}$. It is noteworthy to mention that because of the bandwidth limitation $D_A^{s_1}$ and $D_A^{s_2}$ can not be very far from D_A.

As mentioned in section II-B, there are multiple random elements in the Stochastic Gradient Descent (SGD) algorithm, such as parameter initialization and the order of training set. SGD is used to minimize loss functions of generators and discriminators in the AWA training phase. We use the initial parameters of generators and discriminators, as well as the order of training set as the two parts of random elements. It is notable that since we have to use SGD in the training phase of transformers anyway, we do not add any new randomness to transformers. We just consider the initial parameters of generators and discriminators, as well as the order of training set as the random elements and keep them secret.

Besides, with respect to the capabilities of AWA's framework, we use the pair list and noise being fed to generators in UAWA as the third and fourth parts of the secret random elements. The pair list is a list of pairs with size $K/2$, determining which website is paired by which website in the AWA training phase. For example, suppose that we have four websites, A, B, C, and D. The pair list can be {(A, C),(B, D)}, which means that A is paired with C and B is paired with D in the AWA training phase. When the pair lists of an adversary and a target

---

Algorithm 1 Adversarial Website Adaptation (AWA)

Input: Trace set TS, label set Y_{TS}, number of websites K, noise distribution $\mathcal{P}$, batch size bs, number of discriminator iterations D_T, number of generator iterations G_T, overhead threshold OH, overhead controllers τ_{low} and τ_{high}, parameters α, β, γ, and number of training iterations T.
Output: Transformer set S

1: $AC \leftarrow$ train auxiliary classifier on (TS, Y_{TS}) using $\mathcal{L}_{AC}(TS, Y_{TS})$
2: $Pairs \leftarrow$ randomly select $K/2$ pairs among websites
3: $S \leftarrow$ empty transformer set with size K
4: **for** $pair$ in $Pairs$ **do**
5:     $TS_A \leftarrow$ traces of website $pair[0]$ in TS
6:     $TS_B \leftarrow$ traces of website $pair[1]$ in TS
7:     randomly initialize parameters of T_A, T_B, D_{AB}
8:     $selected\_T_A \leftarrow \emptyset$
9:     $selected\_T_B \leftarrow \emptyset$
10:     **for** $t \leftarrow 0, T$ **do**
11:       **for** $i \leftarrow 0, D_T$ **do**
12:         $TB_A \leftarrow$ randomly select bs traces from TS_A
13:         $TB_B \leftarrow$ randomly select bs traces from TS_B
14:         update $\theta_{D_{AB}}$ to minimize $\mathcal{L}_{D_{AB}}(TB_A, TB_B, \mathcal{P})$
15:       **end for**
16:       **for** $i \leftarrow 0, G_T$ **do**
17:         $TB_A \leftarrow$ randomly select bs traces from TS_A
18:         update θ_{G_A} to minimize $\mathcal{L}_{G_A}(TB_A, \mathcal{P}, \tau_{low}, \tau_{high}, \alpha, \beta, \gamma)$
19:       **end for**
20:       **for** $i \leftarrow 0, D_T$ **do**
21:         $TB_A \leftarrow$ randomly select bs traces from TS_A
22:         $TB_B \leftarrow$ randomly select bs traces from TS_B
23:         update $\theta_{D_{AB}}$ to minimize $\mathcal{L}_{D_{AB}}(TB_A, TB_B, \mathcal{P})$
24:       **end for**
25:       **for** $i \leftarrow 0, G_T$ **do**
26:         $TB_B \leftarrow$ randomly select bs traces from TS_B
27:         update θ_{G_B} to minimize $\mathcal{L}_{G_B}(TB_B, \mathcal{P}, \tau_{low}, \tau_{high}, \alpha, \beta, \gamma)$
28:       **end for**
29:       **if** $\frac{\||T_A(TS_A, Z \sim \mathcal{P})| - |TS_A|\||_1}{\||TS_A|\||_1} \leq OH$ **and**
30:                            $\frac{\||T_B(TS_B, Z \sim \mathcal{P})| - |TS_B|\||_1}{\||TS_B|\||_1} \leq OH$ **then**
31:         $selected\_T_A \leftarrow T_A$
32:         $selected\_T_B \leftarrow T_B$
33:       **else if** $t = T - 1$ **and** $selected\_T_A = \emptyset$ **and** $selected\_T_B = \emptyset$ **then**
34:         $selected\_T_A \leftarrow T_A$
35:         $selected\_T_B \leftarrow T_B$
36:       **end if**
37:     **end for**
38:     $S[pair[0]] \leftarrow selected\_T_A$
39:     $S[pair[1]] \leftarrow selected\_T_B$
40: **end for**
41: **return** S

---

user are different, the distribution of the adversarial traces that the adversary collects and trains a classifier on differs from the adversarial traces that the target user generates. For example, website A is paired with website B in the training phase of the adversary's transformer set, and website A is paired with website C in the training phase of the target user's transformer set. Hence, the adversary's classifier is more likely to be vulnerable to the target user's adversarial traces generated by different pair lists.

D. AWA Algorithm

The complete training process of AWA is presented in Algorithm 1. After selecting a list of pairs, two transformers are trained in T iterations for each pair of websites. In each iteration, each generator is trained in G_T iterations, and the discriminator is trained two times between the generators training in D_T iterations. Sometimes during the training of transformers in some iterations, they impose high bandwidth overhead to traces, which is unacceptable. Therefore we check the magnitude of bandwidth overhead during the training phase, and we only choose transformers that have reasonable bandwidth overhead. We specify a parameter OH and only select transformers with bandwidth overhead less than OH

during the training. However, if the bandwidth overhead of transformers in all iterations during the training is more than OH, we select the transformers in the last iteration. The output of the algorithm is a transformer set, which consists of K transformers.

V. EVALUATION

AWA is independent of the adversary's classifier and generates black-box adversarial traces. To the best of our knowledge, Deep Fingerprinting being proposed by Sirinam *et al.* [5] is the best adversary's classifier in the previous studies. We use this classifier to evaluate the performance of AWA. The fundamental assumption of AWA is that although an adversary knows that a transformer set generates the adversarial traces of the target user, she has no knowledge about the details of the target user's transformer set. We run AWA several times with various secret random elements to create several transformer sets for evaluating the performance of AWA in this setting. We consider two scenarios to evaluate the performance of AWA.

1) An adversary randomly selects a set of transformers and trains a classifier on adversarial traces generated by it. The target user also randomly selects a set of transformers and generates her adversarial traces by it.

2) An adversary generates her adversarial traces through multiple sets of transformers. In this scenario, the adversary must collect more adversarial traces and run a classifier on them, which increases the computational cost of the adversary to run the website fingerprinting attack. The target user also randomly selects a set of transformers and generates her adversarial traces by it. We assume the target user's transformer set is different from the adversary's transformer sets in this scenario.

Since an adversary has no access to the label of the target user's adversarial traces, she can not evaluate the performance of her classifier on them. Therefore, the adversary's classifier is selected based on the validation set that has been generated by the adversary's transformer set(s).

A. Dataset and Setup

We use the dataset that has been proposed by Sirinam *et al.* in [5], which consists of traces of 95 different websites, and each website has been visited 1000 times. Each trace of a website is a sequence of packets direction. It is shown in [5] that sequence of packets direction is enough to classify traces with a high success rate. Because we want the number of classes to be even, we select only 94 classes. The dataset is split into four various sets, including the AWA training set, the adversary's training set, the adversary's validation set, and the target user's set. AWA training set consists of 400 traces of each website (37600 traces in total) and is used in the AWA training to train the auxiliary classifier, generators, and discriminators. The adversary training and validation sets consist of 400 and 100 traces of each website (37600 and 9400 traces in total) respectively, and are used to train and validate adversaries' classifier. 100 traces of each website (9400 in total) is used as the traces of the target user's browsing activities.

The architecture of transformers and discriminators are presented in Appendix A. Transformers are trained in 1000 iterations where $T_T = 2$, $D_T = 2$, $bs = 100$, and $K = 94$. The length of traces being in the format of burst sequence is 2000. The architecture of the auxiliary classifier is the same as the discriminator. The type of all DNNs is 1D-CNN, and we use Tensorflow 1.12 and Keras 2.2.5 to implement them. Adam optimizer with learning rate 0.0001 is used to optimize parameters of transformers and discriminators. The auxiliary classifier uses Adam optimizer with learning rate 0.0002 and is trained in 30 epochs with batch size 128. The distribution of noise in all experiments is the standard Gaussian distribution. Hyperparameters of the generators' loss function are $\alpha = 10^3$, $\beta = 10^3$, and $\gamma = 10^2$. We run AWA on a single NVIDIA GeForce GTX 1080 Ti GPU with 11 GB RAM. The training phase of each pair of generators takes about 675 seconds, and generating 100 adversarial traces in the testing phase takes about 18 milliseconds. The computation costs of UAWA and NUAWA are precisely the same.

B. Experiments

We conduct eight experiments to evaluate the performance of AWA. We run the AWA training phase for five times with various secret random elements and create five sets of transformers for each experiment. Each table in Figure 3 shows the adversary's classifier accuracy on all possible ways that an adversary and a target user can pick a set of transformers. Each table also shows the magnitude of BWO that a set of transformers impose on the adversary's traces and the target user's traces. The last column of each table indicates the adversary's classifier accuracy in scenario 2. In this setting, the adversary's training set consists of $400\times4\times94= 150,400$ adversarial traces. The parameters τ_{low}, τ_{high} and OH control the magnitude of BandWidth Overhead (BWO) in experiments and are called BWO controlling parameters. Figure 3 shows the results of each experiment in a single table. Four tables on the left side of Figure 3 indicate the performance of UAWA under the various BWO controlling parameters, and the other four experiments on the right side of Figure 3 show the performance of NUAWA under the various BWO controlling parameters. The tables that are next to each other have the same BWO controlling parameters, and BWO is increasing from the top to the bottom in each column.

The first experiment of UAWA (UAWA-Exp1) which results are shown in Table 3a demonstrates that UAWA is very effective against website fingerprinting attack, and if the transformer sets of an adversary and a target user are different, then the adversary's classifier accuracy is almost 19.52% with almost 22.28% BWO. The last column of Table 3a indicates that the accuracy of adversary's classifier in scenario 2 is almost 63.27%. BWO is increased by changing BWO controlling parameters in the second, third, and fourth experiments of UAWA (UAWA-Exp2, UAWA-Exp3, and UAWA-Exp4), which their results are shown in Table 3c, Table 3e, and Table 3g, respectively. The results of these four experiments demonstrate that UAWA is very effective against the website fingerprinting attack in scenario 1, and the accuracy of adversary's classifier is more

(a) UAWA-Exp1: $\tau_{low} = 0.05$, $\tau_{high} = 0.30$, and $OH = 0.50$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	22.4	20.9	23.46	22.0	22.29	
Set 1	22.43	98.40	18.17	18.76	19.40	19.81	63.52
Set 2	21.03	16.61	98.14	19.59	21.0	20.18	63.68
Set 3	23.57	17.74	20.67	98.56	22.41	20.09	61.70
Set 4	22.04	18.05	20.15	19.77	98.11	17.44	64.79
Set 5	22.35	13.15	23.76	22.71	21.06	98.50	62.68

(b) NUAWA-Exp1: $\tau_{low} = 0.05$, $\tau_{high} = 0.3$, and $OH = 0.5$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	26.17	26.57	26.93	26.5	26.12	
Set 1	25.95	93.53	33.05	27.29	31.01	36.92	61.28
Set 2	26.29	34.68	94.13	31.72	32.98	34.62	64.62
Set 3	27.06	31.0	34.23	93.94	31.12	29.42	58.61
Set 4	26.15	29.87	32.30	28.63	93.75	32.64	57.82
Set 5	25.95	35.85	34.36	27.29	30.0	94.01	57.45

(c) UAWA-Exp2: $\tau_{low} = 0.25$, $\tau_{high} = 0.5$, and $OH = 0.75$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	39.15	39.3	38.47	38.65	39.67	
Set 1	39.28	97.64	6.09	14.73	13.30	18.31	51.17
Set 2	39.34	7.53	98.51	8.92	8.22	9.6	41.94
Set 3	38.38	13.58	8.3	97.75	15.8	11.75	54.09
Set 4	38.65	13.32	7.57	11.44	98.42	9.57	42.48
Set 5	39.65	18.08	7.89	11.03	12.95	97.71	52.82

(d) NUAWA-Exp2: $\tau_{low} = 0.25$, $\tau_{high} = 0.5$, and $OH = 0.75$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	43.3	42.17	41.81	42.0	43.38	
Set 1	43.4	95.47	19.63	17.29	19.38	18.93	42.89
Set 2	41.92	21.41	95.61	16.47	16.63	19.06	42.13
Set 3	41.67	20.86	18.85	93.52	17.09	19.30	39.54
Set 4	41.97	19.17	20.02	14.52	95.37	15.85	39.70
Set 5	43.53	18.80	19.65	17.82	15.18	94.48	39.48

(e) UAWA-Exp3: $\tau_{low} = 0.5$, $\tau_{high} = 0.75$, and $OH = 1.0$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	62.6	61.78	61.1	63.89	63.54	
Set 1	62.61	98.30	6.70	8.58	7.58	8.58	51.65
Set 2	61.68	7.62	98.96	8.30	7.67	13.42	51.09
Set 3	61.07	4.85	6.74	98.77	8.27	11.93	49.10
Set 4	63.73	10.20	5.20	11.61	99.13	8.13	43.85
Set 5	63.52	8.55	11.85	10.37	7.76	98.69	49.84

(f) NUAWA-Exp3: $\tau_{low} = 0.5$, $\tau_{high} = 0.75$, and $OH = 1.00$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	64.43	65.53	63.44	63.77	64.95	
Set 1	64.28	95.57	11.69	12.82	13.22	9.53	27.80
Set 2	65.15	11.54	96.58	13.54	11.57	11.76	26.22
Set 3	63.74	14.87	13.11	97.17	9.00	12.96	26.97
Set 4	63.97	12.73	11.06	10.09	96.12	15.11	23.63
Set 5	64.54	9.89	9.64	13.25	13.87	95.93	25.05

(g) UAWA-Exp4: $\tau_{low} = 0.75$, $\tau_{high} = 1.0$, and $OH = 1.25$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	86.13	87.24	84.92	85.3	86.85	
Set 1	86.25	99.03	6.44	9.87	6.26	2.28	52.02
Set 2	87.15	8.45	98.93	4.42	6.34	5.51	52.34
Set 3	84.88	9.97	6.24	98.28	7.03	9.95	50.14
Set 4	85.45	6.75	6.10	4.47	98.73	8.23	39.88
Set 5	86.77	7.17	6.62	10.63	9.17	98.98	47.46

(h) NUAWA-Exp4: $\tau_{low} = 0.75$, $\tau_{high} = 1.0$, and $OH = 1.25$

Adversary's accuracy(%)		Adversary's transformer set					Combination of all sets but the user's transformer set
		Set 1	Set 2	Set 3	Set 4	Set 5	
User's transformer set	BWO(%)	87.59	87.09	86.76	86.23	89.67	
Set 1	88.08	97.41	8.91	10.55	7.44	9.04	21.32
Set 2	86.72	6.70	97.59	6.95	9.81	10.40	23.26
Set 3	86.89	12.08	10.64	97.17	8.57	7.17	21.84
Set 4	86.32	9.26	7.18	7.35	97.26	12.77	22.97
Set 5	88.97	8.52	9.13	5.63	10.21	97.95	20.44

Figure 3: Each table shows the performance of an experiment. The tables on the left and the right sides of the figure indicate the performance of UAWA and NUAWA, respectively, in various magnitudes of Bandwidth Overhead (BWO). Each table shows the accuracy of the adversary's classifiers in all possible ways that an adversary and a target user can select a set of transformers. Each table also indicates the magnitude of BWO that is imposed on the traces of an adversary and a target user. The last column of each table shows the accuracy of the adversary's classifier when an adversary generates her traces through four various sets of transformers, in which the transformer set of the target user is not included.

Figure 4: The average and standard deviation of the adversary's classifier accuracy over various sizes of BWO in both scenarios.

decreased by adding more BWO. However, when an adversary is more powerful and can generate adversarial traces through multiple sets of transformers, UAWA is not very effective, and the accuracy of adversary's classifier is not decreased too much by increasing BWO between UAWA-Exp1 and UAWA-Exp2,

and it is relatively fixed between UAWA-Exp2, UAWA-Exp3, and UAWA-Exp4.

We replicate four experiments of UAWA with the same parameters for UNAWA. The results of NUAWA-Exp1, NUAWA-Exp2, NUAWA-Exp3, and NUAWA-Exp4 are presented in Tables 3b, 3d, 3f, and 3h, respectively. The results of these four experiments of NUAWA demonstrate that NUAWA is effective against website fingerprinting attack, and the accuracy of the adversary's classifier is more decreased by adding more BWO to traces. Although NUAWA is less effective than UAWA in scenario 1, NUAWA is considerably more effective than UAWA in scenario 2. For example in a comparison between UAWA-Exp3 and NUAWA-Exp3 which their BWO is almost the same, although the adversary's classifier accuracy in UAWA is almost 3.3% less than NUAWA in scenario 1, the adversary's classifier accuracy in NUAWA is almost 23% less than UAWA in scenario 2. The results of all experiments demonstrate that if the transformer sets of an adversary and a target user are the same, the accuracy of adversary's classifier is high. However, an adversary can not evaluate her classifier on the

Figure 5: The average Bandwidth Overhead (BWO) that AWA imposes to the traces of 94 various websites for four different transformer sets.

Figure 6: Minimum and average intra-class distance of UAWA and NUAWA over various sizes of Bandwidth Overhead (BWO)

target user adversarial traces because she has no access to their labels. Therefore, an adversary can not determine whether the transformer sets are the same, and she has no trust in her classifier accuracy.

Figure 4 summarizes the relationship between adversary's classifier accuracy and BWO in all eight experiments in scenarios 1 and 2. The results demonstrate that the standard deviation between the adversary's classifier accuracy in various experiments is low. Hence, transformer sets that are generated by various random elements have similar performance. Figure 5 shows the average BWO of each class for four various experiments. The results demonstrate that BWO controlling parameters are very effective in controlling BWO, and BWO of almost all classes are in the range [τ_{low}, τ_{high}].

C. Intra-Class Distance

Since we have five transformer sets in each experiment, all websites' distribution is transformed into five different distributions. We assume an adversary and a target user each select one of these distributions, and the adversary trains a classifier on the samples of her distribution. As the distance between the adversary's distribution and the target user's distribution is increased, the accuracy of the adversary's classifier is decreased, given to the findings of [35]. We introduce Intra-Class Distance (Intra-CD) to calculate the average and minimum empirical distance between the five distributions that have been formed by five transformer sets. Intra-CD uses MMD to estimate the distance between two distributions. Since there are five different distributions, we have ten pairs of distributions. We use Average Intra-CD (Avg Intra-CD) and Minimum Intra-CD (Min-Intra-CD) to measure the average and minimum distance between each pair of these

distributions. Therefore, Avg and Min Intra-CD indicate the average and minimum distances between the distributions of adversary's adversarial traces and the target user's adversarial traces when they pick different sets of transformers. Avg and Min Intra-CD are defined as follows:

$$MMD_k^{i,j} = MMD(T_k^i(TS_k, z), T_k^j(TS_k, z))$$

$$\text{Avg Intra-CD} = \frac{1}{K} \sum_{k=1}^{K} \frac{2}{S(S-1)} \sum_{i=1}^{S-1} \sum_{j=i+1}^{S} MMD_k^{i,j}$$

$$\text{Min Intra-CD} = \frac{1}{K} \sum_{k=1}^{K} Min(\bigcup_{i=1,j=1}^{S} MMD_k^{i,j})$$

(13)

where T_k^i is the transformer of website k in transformer set i, S is the number of transformer sets, and K is the number of classes. In our experiment, $S = 5$ and $K = 94$. Figure 6 indicates Avg and Min Intra-CD of UAWA and NUAWA over various magnitudes of BWO. Avg Intra-CD of UAWA is more than NUAWA, which means that if an adversary and a target user select different sets of transformers, the average distance between the distributions of their adversarial traces is more when UAWA creates the transformers. However, Min Intra-CD of UAWA and NUAWA are very close to each other. Intra-CD demonstrates that the training data distribution of the adversary's classifier is far from the distribution of the target user's adversarial traces, and this distance is increased by adding more bandwidth overhead to traces. Therefore, the adversary's classifier is more likely to be vulnerable to the adversarial traces of the target user when their sets of transformers are different. The results also justify the better performance of AWA when bandwidth overhead is increased.

D. Average Trace Visualization

We use the average trace as a representation of all traces of a website. The average trace is the average of a set of burst sequences over burst indexes. The first element of the average trace is the average of the first burst sizes of all traces and so on. We use the average trace to visualize the changes that AWA applies to a set of traces. Figure 7 presents the average traces of website A ($\mu_A = Avg(TS_A)$) and website B ($\mu_B = Avg(TS_B)$) before transformation and the average traces of transformed traces of website A ($\mu_A = Avg(T_A^s(TS_A, z))$) and website B ($\mu_B^s = Avg(T_B^s(TS_B, z))$), where T_A^s and T_B^s are in the transformer set s. Each plot of Figure 7 is from a different transformer set, and website A is website 1 in all plots. We select three transformer sets from NUMAW-Exp3, and three

Figure 7: The average traces of six pairs of websites before and after transformation. The three plots on the first and second rows are from UAWA-Exp3 and NUAWA-Exp3, respectively. In each plot, website A is website 1, and website B is the paired website with website 1 in a transformer set.

transformer sets UAWA-EXP3. In each transformer set, there is one website being paired by website 1 in the training phase of AWA, and we select this website as the website B in each plot of Figure 7. Website 1 is selected as an example, and the paired websites were determined randomly in the training phase of AWA. For example, the average traces of websites 1 and 38 before and after transformation are depicted in the first plot. This figure indicates that when various transformers transform the traces of website 1, the average trace of website 1 is different, which shows that the distribution of website 1 after transforming by various transformers is different. It also shows that the style of transformed traces of two websites being paired in the AWA training phase are getting close to each other.

E. Results Analysis

We used Deep Fingerprinting (DF) [5] as the adversary's classifier in the experiments. In addition to DF, there are two classifiers in the previous works which have shown higher performance in classifying user's traces in comparison to the traditional approaches. Rimmer *et al.* [6] propose Automated Website Fingerprinting (AWF) classifier, and they utilize SDAE, LSTM, and CNN as the adversary's classifier. AWF uses the sequence of packets direction to classify traces. Bhat *et al.* [26] present VAR-CNN and use the sequence of packets direction, inter-arrival time, and cumulative statistical features to classify traces. The baseline CNN architecture of VAR-CNN is based on ResNet-18 [27]. VAR-CNN runs in 150 epochs and uses learning rate decay and early-stopping to improve the classifier's performance. Table I indicates the performance of UAWA-Exp1 and NUAWA-Exp1 against AWF and ResNet-18 (VAR-CNN) in Scenario 1. We use the CNN classifier of

Table I: The average and standard deviation of the accuracy of various attacks that have been proposed in previous studies [5], [6], [26] against UAWA-Exp1 and NUAWA-Exp1.

| | UAWA-Exp1 | | NUAWA-Exp1 | |
| | 22.28% BWO | | 26.28% BWO | |
Adversary's name	Avg Acc(%)	STD Acc(%)	Avg Acc(%)	STD Acc(%)
DF [5]	19.52	2.34	31.94	2.71
AWF [6]	15.19	2.29	18.87	1.32
ResNet-18 (VAR-CNN) [26]	16.80	1.96	26.60	2.40

Table II: The accuracy of Deep Fingerprinting (DF) attack [5] against AWA, Mockingbird, and WTF-PAD.

Defense name	BWO(%)	DF accuracy(%)
Mockingbird [14]	58.0*	42.0*
WTF-PAD [24]	64.0*	86.0*
UAWA-Exp1	22.28	19.52
NUAWA-Exp1	26.28	31.94

∗ are from [14]

AWF, and since our dataset only has the sequence of packets direction, we train a ResNet-18 classifier on them for VAR-CNN. The results demonstrate that AWA is effective against all three attacks.

We compare the performance of AWA with Mockingbird [14] and WTF-PAD [24] in Table II. Since Mockingbird requires to have access to the entire trace of a website before generating an adversarial example, it is not universal. WTF-PAD [24] is the most promising defense that does not use adversarial machine learning techniques. Saidur *et al.* in [14] have reported the performance of Mockingbird and WTF-PAD against DF attack on the same dataset that we use in this study. Table II indicates the performance of Mockingbird, WTF-PAD, UAWA-Exp1, and NUAWA-Exp1 against DF attack. The results demonstrate that AWA is more effective than Mockingbird and WTF-PAD

defenses with lower BWO.

VI. AWA In Practice

AWA has three phases: pre-training, training, and testing. The pre-training and training phases can be done offline. The generation process of adversarial traces (testing phase) in UAWA also can be done offline, but in NUAWA, it should be done online. The training process of Deep Neural Networks (DNNs) is often done on Graphical Process Units (GPUs) due to their performance. GPUs reduce the training and testing time by orders of magnitude due to their much more efficient matrix operations. Hence, the pre-training and training phases of AWA should be done on GPUs. It is unrealistic to suppose that all users have access to GPUs. Therefore, PETs should provide a GPU server for three phases of AWA. This server is called AWA server. Nevertheless, if a user has access to GPU, she can run AWA on her GPU. The AWA server must run AWA multiple times with various random elements to create multiple sets of transformers. Since AWA does not need random elements after the training phase, except for the noise input of generators in UAWA, all other parts of secret random elements can be removed after the AWA training phase. The noise input of generators in UAWA also can be removed after generating perturbation vectors. The AWA server must save created transformers sets on a secure storage. Notably, the computational cost of the AWA training phase linearly increases by increasing the number of websites. However, it does not affect the computational cost of the testing phase.

When a user for the first time uses AWA, a transformer set is assigned to that user. There are two solutions for a user to generate adversarial traces. First, she can download transformers and use them on her local machine. The size of each transformer, which is equal to the size of generator parameters, is about 112 KB. The user can utilize a Core Process Unit (CPU) or GPU to generate adversarial traces using downloaded transformers. Generating 100 adversarial traces takes about 0.02 seconds on GPU (GeForce GTX 1080 Ti) and about 0.35 seconds on CPU (Intel(R) Core(TM) i7-4710HQ). In the second solution, which is more convenient, the AWA server is responsible for generating adversarial traces. In UAWA, for each user, the AWA server can generate perturbation vectors (the output of the generator) of various websites through the associated transformer set and send them to the user. In this setting, when a user wants to visit website A, she can use the perturbation vector of website A downloaded beforehand. The elements of perturbation vector determine how many dummy packets must be added to the end of each burst on the fly. Therefore, UAWA has no computation cost in the browsing time. In NUAWA, the entire trace of a website must be fed to the transformer. Therefore, similar to [14], [18], there must be a database of fresh traces of various websites in the AWA server. In this setting, when a user wants to visit website A, the AWA server using the associated transformer generates an adversarial trace for website A and send it to her. The user must send real packets and dummy packets based on the received adversarial trace. Hence, there is a little computation cost for the AWA server to generate adversarial traces in NUAWA.

AWA injects dummy packets to both sides of network traffic, Client to Server (C2S) and Server to Client (S2C). A user can send C2S dummy packets by her machine. The entity that sends S2C dummy packets depends on the infrastructure of PETs. A web server or a middleware can send S2C dummy packets. In Tor, S2C dummy packets can be sent by the first node (bridge node) of the Tor network. Hence, adversarial trace in NUAWA and perturbation vector in UAWA should also be sent to the bridge node.

VII. Conclusion

In this paper, we proposed a novel defense against the website fingerprinting attack called Adversarial Website Adaptation (AWA), which has two versions, Universal AWA (UAWA) and Non-Universal AWA (NUAWA). We considered two scenarios to evaluate the performance of AWA. In the first scenario, we have shown that if an adversary and a target user generate their traces by different sets of transformers, both UAWA and NUAW are highly effective; however, the performance of UAWA is better. In the second scenario, we have indicated that if an adversary uses multiple sets of transformers that are different from the target user's set of transformers, NUAWA is much more effective than UAWA. The results demonstrate when the adversary is more powerful and can collect traces of various websites through multiple sets of transformers, the defense can not use the benefits of UAWA and must impose more bandwidth overhead to traces.

The future work is to provide theoretical analysis for AWA using several recent works [37] on certified robustness against adversarial attacks that can provide formal guarantees of upper and lower bound for required noise.

References

[1] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proc. 1st ACM Cloud Comput. Security Workshop (CCSW), Nov. 13*, 2009, pp. 31–42.

[2] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symp., Aug. 10-12*, 2016, pp. 1187–1203.

[3] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale," in *23rd Annu. Net. and Distrib. Syst. Security Symp. (NDSS), Feb. 21-24*, 2016.

[4] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," in *Proc. Privacy Enhancing Technol.*, vol. 2016, no. 4, 2016, pp. 21–36.

[5] P. Sirinam, M. Imani, M. Juárez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. and Commun. Security (CCS), Oct. 15-19*, 2018, pp. 1928–1943.

[6] V. Rimmer, D. Preuveneers, M. Juárez, T. van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *25th Annu. Net. and Distrib. Syst. Security Symp. (NDSS), Feb. 18-21*, 2018.

[7] Z. Zhuo, Y. Zhang, Z. Zhang, X. Zhang, and J. Zhang, "Website fingerprinting attack on anonymity networks based on profile hidden markov model," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1081–1095, 2018.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd Int. Conf. Learn. Representations (ICLR), Apr. 14-16*, 2014.

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd Int. Conf. Learn. Representations (ICLR), May 7-9*, 2015.

[10] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE Symp. Security and Privacy (SP), May 22-26*, 2017, pp. 39–57.

[11] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *IEEE Conf. Comput. Vision and Pattern Recognition (CVPR), July 21-26*, 2017, pp. 86–94.

[12] C. Xiao, B. Li, J. yan Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proc. 27th Int. Joint Conf. AI, (IJCAI), July 13-19*, 2018, pp. 3905–3911.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th Int. Conf. Learn. Representations (ICLR), Apr. 30 - May 3*, 2018.

[14] M. S. Rahman, M. Imani, N. Mathews, and M. Wright, "Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1594–1609, 2021.

[15] X. Zhang, J. Hamm, M. K. Reiter, and Y. Zhang, "Statistical privacy for streaming traffic," in *26th Annu. Net. and Distrib. Syst. Security Symp. (NDSS), Feb. 24-27*, 2019.

[16] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proc. 10th Annu. ACM workshop Privacy in the Electron. Soc. (WPES), Oct. 17*, 2011, pp. 103–114.

[17] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proc. 12th Annu. ACM Workshop on Privacy in the Electron. Soc. (WPES), Nov. 4,*, 2013, pp. 201–212.

[18] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *Proc. 23rd USENIX Security Symp., Aug. 20-22*, 2014, pp. 143–157.

[19] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," *Proc. Asia-Pac. Adv. Netw.*, vol. 42, pp. 15–20, August 2016.

[20] M. Nasr, A. Bahramali, and A. Houmansadr, "Blind adversarial network perturbations," *arXiv preprint arXiv:2002.06495*, 2020.

[21] A. Abusnaina, R. Jang, A. Khormali, D. Nyang, and David, "Dfd: Adversarial learning-based approach to defend against website fingerprinting," in *IEEE Conf. Comput. Commun. (INFOCOM), July 6-9*, 2020.

[22] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 445–458, 2019.

[23] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: website fingerprinting attacks and defenses," in *the ACM Conf. Comput. and Commun. Security (CCS), Oct. 16-18*, 2012, pp. 605–616.

[24] M. Juárez, M. Imani, M. Perry, C. Díaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *21st European Symp. Research in Comput. Security (ESORICS), Sept. 26-30*, vol. 9878, 2016, pp. 27–46.

[25] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *26th USENIX Security Symp., Aug. 16-18*, 2017, pp. 1375–1390.

[26] S. Bhat, D. Lu, A. Kwon, and S. Devadas, "Var-cnn: A data-efficient website fingerprinting attack based on deep learning," in *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 4, 2019, pp. 292–310.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[28] 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 *IEEE Symp. Security and Privacy (SP), May 21-23*, 2012, pp. 332–346.

[29] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proc. ACM SIGSAC Conf. Comput. and Commun. Security (CCS), Nov. 3-7*, 2014, pp. 227–238.

[30] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop on Privacy in the Electron. Soc. (WPES), Nov. 3*, 2014, pp. 121–130.

[31] V. Shmatikov and M. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *11th European Symp. Research in Comput. Security (ESORICS), Sept. 18-20*, vol. 4189, 2006, pp. 18–33.

[32] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proc. 13th Workshop on Privacy in the Electron. Soc. (WPES), Nov. 3*, 2014, pp. 131–134.

[33] A. M. Sadeghzadeh, S. Shiravi, and R. Jalili, "Adversarial network traffic: Towards evaluating the robustness of deep learning-based network traffic classification," *IEEE Trans. Netw. Service Manag.*, pp. 1–1, 2021.

[34] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko, "Simultaneous deep transfer across domains and tasks," in *IEEE Int. Conf. Comput. Vision (ICCV), Dec. 7-13*, 2015, pp. 4068–4076.

[35] H. Zhang, H. Chen, Z. Song, D. S. Boning, I. S. Dhillon, and C. Hsieh, "The limitations of adversarial training and the blind-spot attack," in *7th Int. Conf. Learn. Representations (ICLR), May 6-9*, 2019.

[36] A. Kerckhoffs, "La cryptographic militaire," *J. Des Sci. Militaires*, pp. 5–38, 1883.

[37] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *IEEE Symp. Security and Privacy (SP)*, 2019, pp. 656–672.

APPENDIX A

(a) Generators architecture

(b) Discriminator architecture

Figure 8: The DNN architectures of generator and discriminator in the AWA framework.

Amir Mahdi Sadeghzadeh received his B.Sc. degree in Information Technology Engineering from Isfahan University of Technology in 2014 and his M.Sc. degree in Computer Engineering from Sharif University of Technology in 2016. He is currently a Ph.D. candidate at the Department of Computer Engineering, Sharif University of Technology. His research interests include Deep Learning Security, Adversarial Deep Learning, and Privacy Enhancing Technologies.

Behrad Tajali received his B.Sc degree from KN Toosi university of technology in 2016, and his M.Sc. degree from Sharif University of Technology in 2020 both majored in computer science. His personal research interests lie in Adversarial Machine Learning, Web Security & Privacy, and Utilizing AI Algorithms in Security-sensitive Applications.

Rasool Jalili received his B.Sc. degree in Computer Science from Ferdowsi University of Mashhad in 1985, and his M.Sc. in Computer Engineering from Sharif University of Technology in 1989. He received his Ph.D. in Computer Science from The University of Sydney, Australia, in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, in 1995. He has published more than 150 papers in Computer Security and Pervasive Computing in international journals and conferences proceedings. He is now an associate professor and the director of Data and Network Security Lab (DNSL) in Sharif University of Technology. His research interests include Access Control, Vulnerability Analysis, Database Security, and Machine Learning Security.