Statistical Privacy for Streaming Traffic

Xiaokuan Zhang*, Jihun Hamm*, Michael K. Reiter†, Yinqian Zhang*
*{zhang.5840, hamm.95, zhang.834}@osu.edu, The Ohio State University
†reiter@cs.unc.edu, University of North Carolina at Chapel Hill

***Abstract*—Machine learning empowers traffic-analysis attacks that breach users' privacy from their encrypted traffic. Recent advances in deep learning drastically escalate such threats. One prominent example demonstrated recently is a traffic-analysis attack against video streaming by using convolutional neural networks. In this paper, we explore the adaption of techniques previously used in the domains of adversarial machine learning and differential privacy to mitigate the machine-learning-powered analysis of streaming traffic.**

Our findings are twofold. First, constructing adversarial samples effectively confounds an adversary with a predetermined classifier but is less effective when the adversary can adapt to the defense by using alternative classifiers or training the classifier with adversarial samples. Second, differential-privacy guarantees are very effective against such statistical-inference-based traffic analysis, while remaining agnostic to the machine learning classifiers used by the adversary. We propose two mechanisms for enforcing differential privacy for encrypted streaming traffic, and evaluate their security and utility. Our empirical implementation and evaluation suggest that the proposed statistical privacy approaches are promising solutions in the underlying scenarios.

I. Introduction

Machine learning (ML) leverages statistical techniques to enable computer systems to learn from data and act based on inferences without being explicitly instructed. The application of ML in security has made possible many broadly adopted defense techniques, such as intrusion detection, spam filtering, biometric recognition, and malware detection. However, when used with malicious intentions, ML also empowers notable attacks. One such example is the traffic-analysis attack.

Encryption is a widely used approach to protecting the confidentiality of network traffic. For example, HTTPS traffic encrypts the HTTP headers and payloads inside the SSL record protocol, hiding the HTTP semantics (*e.g.*, path of the requested resources) from external observers. Moreover, users may choose to use VPN or Tor to further hide (through encryption) the destinations of the web traffic. Traffic analysis, or more specifically website fingerprinting, aims to breach users' privacy by inferring the HTTP semantics (in the case of SSL) or visited websites (in the cases of VPN and Tor) from the encrypted traffic. What enables such attacks is machine learning. By learning from patterns of encrypted traffic to/from known web pages, the ML algorithm can classify unidentified traffic with reasonable accuracy. With the recent development of deep learning, such traffic analysis has become more powerful, invalidating many previously established defenses against traditional machine learning [60].

A recent study by Schuster *et al.* [55] further extended traffic analysis to SSL or QUIC encrypted video streaming services. They demonstrated that due to the uniqueness of the packet burst patterns of the encrypted video streams, the adversary is able to classify the encrypted video streaming with very high accuracy (*e.g.*, 99% for Youtube videos). Consequently, online video streaming is no longer private from a network observer, regardless of the encryption technology used in the protocols.

Relationship to side-channel attacks. Traffic analysis is a type of side-channel attack, i.e., a method to leverage unconventional means to infer sensitive information in a computer system. Generally speaking, in side-channel attacks, the adversary may learn secrets of a system or an application that are otherwise well protected, by observing traces (*e.g.*, timing, power, or resource usage) of its execution. Although not all side-channel attacks rely on ML, this technology does enable some attacks that are otherwise impossible [13], [16], [36], [48], [73], [74].

With increasing learning capacity, the security threats unleashed by these techniques grow rapidly, which calls for more effective defenses. In this paper, we use streaming traffic analysis as a motivating example and explore generic solutions to these ML-powered attacks.

Learning from adversarial machine learning. Inspired by the recent advances in adversarial machine learning, we first explore the use of adversarial samples to defeat ML adversaries. Unlike the common use of adversarial ML, where the attacker crafts samples deliberately aiming to defeat ML-based defenders, we consider the inverse use of such techniques. We exploit adversarial ML techniques to construct noised samples to thwart ML adversaries. In particular, we utilize the Fast Gradient Sign Method (FGSM) to generate adversarial samples to confuse a convolutional neural network (CNN) classifier, and successfully reduce the accuracy of the classification. However, our results show that adversarial ML techniques are not robust: by choosing a different ML algorithm or training the CNN classifier with adversarial samples, the adversary who aims to perform traffic analysis on encrypted streaming packets to extract sensitive information can still do so, as indicated by the high classification accuracy after making these changes.

Adapting differential privacy as security defenses. The failure in the adoption of adversarial samples to defeat streaming traffic analysis motivated us to seek more *principled* solutions to counter such a powerful adversary. Inspired by

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23210
www.ndss-symposium.org

Xiao *et al.* [72], who exploit d^*-privacy—a variant of differential privacy—to insert random noise to disturb storage side channels in `procfs`, we seek to apply a similar principle as a defense against traffic-analysis attacks. However, compared with differentially private `procfs` proposed by Xiao *et al.*, applying differential privacy on network traffic is fundamentally different as traffic analysis is non-interactive. In contrast, attacks leveraging `procfs` are interactive, because the statistical database is constructed as the attacker queries `procfs`. Thus the Laplacian noise can be inserted in the return values of the `procfs` queries. Differentially private streaming traffic needs to be applied proactively to the entire data streams. The approach to do so and its effectiveness with regard to security guarantees and utility loss (*i.e.*, low bandwidth overhead and small amount of lags) is uncertain.

To enforce differential privacy on streaming traffic, we adapt two mechanisms: FPA_k and d^*. Fourier Perturbation Algorithm (FPA_k) [52] is a differentially private mechanism that answers long query sequences over correlated time series data in a differentially private manner based on the Discrete Fourier Transform (DFT). The d^*-private mechanism extends the mechanism from Chan et al. [11] and applies Laplacian noise on time series data. We evaluate both mechanisms in regards to security and utility. Our evaluation results suggest that with proper choice of parameters, both mechanisms defeat—in a sense that it reduces the classification accuracy to the baseline accuracy of random guessing—all types of classifiers that are trained with either the original data or the noised data. With the same parameters, we also show that the utility metrics, defined as *waste* and *deficit*, are moderate. We further compare FPA_k with a baseline defense mechanism. The result suggests that the `waste` induced by FPA_k is at least one order of magnitude lower than the baseline approach.

To demonstrate the practicality, we implement the FPA_k privacy mechanism in a Chrome extension that proxies the Youtube streaming between the browser and the server. The implementation makes use of the `Xhook` [25] framework, which intercepts and modifies `XMLHttpRequest(XHR)` requests and responses. It also utilizes the `numjs` [44] library and the `Random` library in `SIM.JS` [42] for noise injection. Our evaluation suggests that the extension completely renders the attacks proposed by Schuster *et al.* [55] ineffective.

Contributions. This paper makes the following contributions:

- We demonstrate the first attempt to use adversarial ML for defeating streaming traffic analysis, and explore its limitations.
- We develop two mechanisms for enforcing differential privacy for time-series data, and apply them to protect streaming traffic.
- We perform an extensive evaluation on the two differentially private mechanisms, in terms of both security and utility.
- We develop a browser extension which integrates one of the defense mechanisms, and the evaluation shows promising results.

Besides defeating streaming traffic analysis, the techniques proposed in this paper also shed light on defenses against website fingerprinting attacks and generic side-channel attacks that rely on ML. Our study has provided an important piece of evidence suggesting that the differential privacy is a promising solution to ML-enabled inference attacks.

Roadmap. The rest of the paper is organized as follows. Sec. II summarizes the background knowledge needed in the paper. Sec. III presents a motivating example of identifying video streams. Sec. IV describes an approach of generating adversarial samples to defeat traffic analysis, demonstrating its effectiveness and limitations. Sec. V proposes two differentially-private mechanisms for streaming traffic, and provides the basic attack model assumed in the paper. The two methods are evaluated in Sec. VI in regards to security and utility. Sec. VII demonstrates the implementation of a real-world Chrome extension and how it can effectively defeat the traffic analysis attacks. Sec. VIII discusses limitations and practical issues of our approaches. Sec. IX summarizes related work. We conclude the paper in Sec. X.

II. Background

A. Side-Channel Attacks and Traffic Analysis

Side-channel attacks have been studied for more than two decades. Conventional side-channel attacks usually involve analysis of externally observable characteristics of a computer system to extract sensitive information (*e.g.*, cryptographic keys). Some side-channel attacks extract such information through a single run of the victim program; in other cases, multiple side-channel traces must be collected and used to perform statistical analysis to infer useful information. Traffic analysis attacks are examples of the latter case, by observing the meta-data of the encrypted network traffic to classify the traffic [23], [47], [69].

For our purposes here, a side channel arises from an attacker's observation of a feature x, which may itself consist of multiple components. We let $\mathcal{X}$ denote the space of all possible such x values. Often, the attacker will collect feature vectors x and their associated labels in a *training phase*, to build a machine learning model to which it will apply observations x seen during his attack.

B. Machine Learning

In the past, various ML techniques have been employed in statistical side-channel attacks. For example, support vector machines (SVM) have been used to perform website fingerprinting in the Tor network [48] and infer foreground apps on Android [16]; hidden Markov models (HMM) have been used to infer Android Activity transitions [13] and extract cryptographic keys in a cross-VM setting [74]; k nearest neighbors (kNN) have been used to perform keystroke inference on smartwatch [36] and link Bitcoin addresses to an iOS device [73].

Deep Learning [33] is an ML approach that uses multiple layers of non-linear processing units, each of which transforms the representation at one level into that at a higher, more abstract level. The most representative deep learning model is the Deep Neural Network (DNN), which is an artificial neural network (ANN) with multiple hidden layers between the input and output layers [2]. DNNs are very effective at finding hidden features in high-dimensional data, which is hard for humans. It has been applied to solve various problems, producing promising results in different areas such as image recognition [29], [63], speech recognition [24], [54] , natural language processing [62] , and malware detection [15].

Researchers have developed various kinds of DNNs. One of the most popular DNN models is the Convolutional Neural Network (CNN) [32]. CNN typically applies convolutional operation at lower levels, and is designed to process data that has a form of multi-dimensional arrays. CNN takes into account the spatial structure of data by enforcing a local connectivity pattern, which gives it an excellent performance when dealing with data whose local groups of values are highly correlated, such as 1D signals and 2D images [29]. There are other popular DNN models as well, such as Recurrent Neural Networks (RNNs) [53] and Autoencoders [4].

C. Adversarial Machine Learning

Adversarial machine learning is an emerging research field, which is closely related to both machine learning and computer security. Here, we briefly introduce two topics in adversarial ML: adversarial samples and adversarial training.

An adversarial sample x' is an input that is crafted from a legitimate (untampered) input to make a classifier misclassify x' [64]. More specifically, x' is created to be within some distance threshold from some untampered input x, in the hopes that this will imply that x' remains in the same class as x according to some notion of ground truth. However, x' is manipulated so that the ML classifier will classify x' differently from x. Often the distance measure used is l_1, l_2, or l_∞, and ground truth is as evaluated by a human. (A small distance in one of these senses does not necessarily imply that humans will tend to classify x' and x the same, however [57].) Methods of generating adversarial samples include Fast Gradient Sign Method (FGSM) [21], Deepfool [41], Jacobian-based Saliency Map Attack (JSMA) [50] and the Carlini/Wagner attack (CW) [10].

In response, defenses have been proposed to make classifiers more robust against adversarial samples. To date, the most successful one is adversarial training [21], [64], which basically retrains the classifier using the adversarial samples that were generated to fool the classifier, in order to increase the classification accuracy on these crafted samples. However, its effectiveness highly depends on whether the classifier can generate adversarial samples similar to the ones used by the attacker, which is difficult to guarantee.

D. Privacy

Because an adversarial sample x' generated from x is designed to be misclassified, it might be viewed as a more "privacy preserving" representation of x if correct classification constitutes a privacy violation. For this reason, we explore the generation of adversarial samples as a privacy protection in a specific domain, in Sec. IV. Despite the possibility that adversarial samples so generated might suffice to defeat ML classifiers today, there remains the possibility that future classifiers, or auxiliary information that might be brought to bear by the attacker (classifier), would divulge the correct class of x'.

For this reason, in this paper we also explore a novel application of *differential privacy* [17] to this same domain, which will guarantee that certain classes cannot be distinguished by *any* classifier (that works with the same features). The original definition of differential privacy is specific to statistical databases. More specifically, two databases x, x' are adjacent if they differ in exactly one element. A randomized algorithm $A : \mathcal{X} \rightarrow \mathcal{Z}$ satisfies ϵ-differential privacy if for any adjacent databases x, x' and all $Z \subseteq \mathcal{Z}$,

$$\mathbb{P}\left(A(x) \in Z\right) \leq \exp(\epsilon) \times \mathbb{P}\left(A(x') \in Z\right).$$

Chatzikokolakis et al. [12] proposed a generalization of differential privacy called d-privacy that will be useful here. A *metric* d on a set $\mathcal{X}$ is a function $d : \mathcal{X}^2 \rightarrow [0, \infty)$ satisfying $d(x, x) = 0$, $d(x, x') = d(x', x)$, and $d(x, x'') \leq d(x, x') + d(x', x'')$ for all $x, x', x'' \in \mathcal{X}$. A randomized algorithm $A : \mathcal{X} \rightarrow \mathcal{Z}$ satisfies (d, ϵ)-privacy if for all $Z \subseteq \mathcal{Z}$,

$$\mathbb{P}\left(A(x) \in Z\right) \leq \exp(\epsilon \times d(x, x')) \times \mathbb{P}\left(A(x') \in Z\right).$$

In our context, the application of A to sufficiently close examples x and x' (i.e., $d(x, x')$ is "small") from different classes will ensure that *any* classifier has a similar probability of classifying $A(x)$ and $A(x')$ within any subset Z of classes.

III. A Motivating Example

Side-channel attacks leverage ML to automatically learn critical features of side-channel observations and make statistical inferences to extract secrets. Recent advances in deep learning [29], [33], [63], [65] further empower side-channel attackers to conduct more accurate and efficient inference attacks. One such example is recently demonstrated remote identification of encrypted MPEG-DASH video streams by Schuster *et al.* [55].

A. Remote Identification of Encrypted Video Streams

MPEG-DASH is a video streaming standard that segments video streams to variable segment sizes due to variable-rate encoding, and instruments the request of video content at the granularity of segments. Schuster *et al.* [55] demonstrated that packet burst patterns of the encrypted video streams (an observable side channel that reveals the size of the segments) can be correlated to the content of the videos that are requested from the client. They further developed techniques using convolutional neural networks (CNNs) to fingerprint video streams from YouTube, Netflix, Amazon, and Vimeo with very high detection accuracy. For instance, their techniques identified YouTube videos (from a small dataset of 18 videos) with 0 false positives and 0.988 recall.

In this paper, we used this MPEG-DASH video-stream fingerprinting as a motivating example to explore how side channels using ML can be mitigated. To demonstrate the capability of the attacks, we extended the idea presented in Schuster *et al.* [55] and performed fingerprinting attacks of 40 Youtube videos using a set of five ML classifiers. The attack was performed in a closed-world setting, in which we assumed the video to be classified is one of the 40; this closed-world setting is the most advantageous to the attacker and the least favorable to the defender.

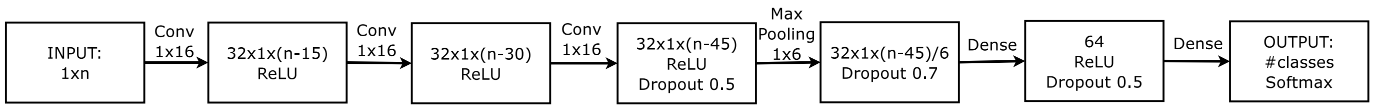


Fig. 1: CNN architecture. n denotes the number of elements of one trace, which is the total time divided by the window size.

B. Data Collection

We manually chose 40 Youtube videos related to four types of sports (basketball, American football, soccer, and hockey) as our dataset. To find these videos, we typed "NBA", "NFL", "MLS" and "NHL" into Youtube search separately, filtered out the short videos that are less than 20 minutes (to make sure the video length is long enough for analysis), and selected 10 videos from each category. Each of the 40 videos was visited from a Chrome browser 100 times during trace collection. Thus 100 traces were collected for each video. Therefore, in total 4000 (*i.e.*, 40×100) traces with 40 distinct labels (*i.e.*, the content of the videos) were included in our dataset. We recorded the *timestamps* and *sizes* of all packets of the first 3 minutes of network traffic after starting to stream each video. The data collection process was automated using `Selenium` [56] and Wireshark's `tshark` [71]. All the data were collected from a desktop running Ubuntu 17.10 connected to our campus network using 1 Gbps Ethernet. The whole process of data collection took about 15 days.

C. Classification

Preprocessing. To convert videos in the dataset into feature vectors of equal length, we aggregated the raw data into 0.25-second bins. Here, 0.25s is the *window size* (w). Each 3-minute video stream was thus abstracted as an array of 720 elements (*i.e.*, bins). Note that we did not filter out the ad traffic that occurs at the beginning of the captures.

Classifiers. We implemented five classifiers, including Support Vector Machine (SVM), Logistic Regression (LR), Random Forest (RF), Neural Net and Convolutional Neural Network (CNN), in Python. Specifically, SVM, LR and RF were implemented using `scikit-learn` [51], Neural Net and CNN were implemented using `Tensorflow` [1] with the `Keras` [14] front end. For Neural Net, we used a single Dense layer with 40 neurons and the Sigmoid function as the activation function. For CNN, we used the same structure as that used by Schuster *et al.* [55]. It consists of three convolutional layers, 1 max pooling layer, and two dense layers. The detailed CNN structure is shown in Fig. 1.

Classification results. We applied the 5 classifiers to classify the 4000 video traces. We used 5-fold cross-validation: each time, a different 20% of the traces were used for testing while the remaining 80% were used for training. The features of the dataset were normalized using the `MinMaxScaler()` method provided by `scikit-learn`. For CNN, we used a batch size of 32 and the model was trained for 40 epochs[1]. As shown in Table I, SVM, LR and RF achieved 0.809, 0.823, and 0.751 classification accuracy, respectively. Neural Net reached 0.831 classification accuracy. CNN had the highest accuracy of 0.944. The classification results had very small variance in the 5-fold tests. These experiments validate the attack demonstrated by Schuster *et al.* [55]. The results suggest that machine learning, and particularly deep learning (*e.g.*, CNN), can empower traffic analysis to easily identify the Youtube video streams from encrypted traffic.

Model	SVM	LR	RF	Neural Net	CNN
Average Accuracy	0.809	0.823	0.751	0.831	0.944
Standard Deviation	0.067	0.063	0.046	0.011	0.004

TABLE I: Classification accuracy with one standard deviation.

IV. Adversarial Machine Learning

Our first attempt is to fool the machine-learning attackers with techniques used in adversarial machine learning.

A. Crafting Adversarial Samples

To generate adversarial samples, we followed the Fast Gradient Sign Method (FGSM) proposed by Goodfellow *et al.* [21]. Let x be the input sample, $g(x;\theta)$ the classifier parameterized by θ, y the true label associated with x, $L(g(x;\theta), y)$ the loss/cost function of the classifier, and η the parameter that controls the amount of perturbation. For untargeted attacks—i.e., the classifier misclassifies a sample as any label but the true label—FGSM generates the following adversarial sample x^* from the clean sample x:

$$x^* = x + \eta\,\mathrm{sign}(\nabla_x L(g(x;\theta), y)).$$

The perturbation $x^* - x$ is the gradient image ∇_x of the given loss L, which by definition is the direction where the loss increases the most. The method then takes only the sign values of the gradient to make it unit l_∞-normed, then multiplies the normalized gradient with the desired perturbation strength η. When η is large, the perturbation is more effective but is more detectable to human eyes or machine classifiers. When η is small it is less effective but is less likely to be detected. In our experiment, we used the `FastGradientMethod()` in the `cleverhans` [49] Python library. We adjusted the level of injected noise (dictated by the `eps` parameter) to generate adversarial samples corresponding to different noise levels. The noise level denotes the maximum distortion of the adversarial sample compared to original input, which is usually a value between 0 and 1. Suppose the original value is v. With `eps` = α, the adversarial value is within the range of $v \pm \alpha v$.

To see how the classifiers perform on adversarial samples, we targeted the CNN model and generated corresponding adversarial test samples using FGSM, with the noise level `eps` = 0.1. Then, we fed these samples to the CNN classifier trained using clean samples. The CNN classifier was unable to classify such samples successfully, with only 0.086 accuracy, which is significantly lower than the original accuracy (0.944). This result suggests that the adversarial samples are very effective against this ML attacker.

[1]The model converged after 40 epochs. Training for 1000 epochs improved the accuracy by only 0.024.

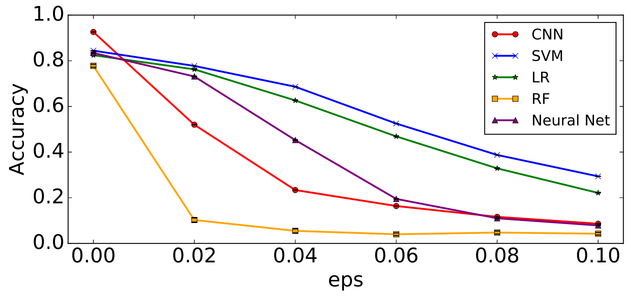


Fig. 2: Test CNN adversarial samples on five classifiers. Each data point is the result of a 5-fold cross-validation.

B. *Limitations of Adversarial Samples*

However, the attacker can also take actions to adapt to these adversarial samples. Here, we study two possible approaches: using a different classifier and conducting adversarial training.

1) Using a different classifier: The adversarial samples generated by the FGSM are designed to fool one particular classifier, which may not be able to deceive other classifiers. To see how the adversarial samples generated for the CNN model affect other classifiers, we fed these samples to other trained models mentioned in Sec. III-C. We adjusted the noise level (*i.e.*, the `eps` parameter) in `FastGradientMethod()` to gain a better understanding of how it would affect the classification accuracy. The higher the noise level, the more distortion in the samples. The result is shown in Fig. 2. Each data point was generated by conducting a 5-fold validation. The mean values of the classification results are shown in the figure, and all standard deviations are below 0.01, which are too small to be visible. From this figure, we can see that when the noise level was 0.05, the CNN model achieved only about 0.20 accuracy, but SVM and LR still reached over 0.6 accuracy. The effectiveness of the adversarial ML techniques against traffic analysis attacks highly depended on the ML algorithms and parameters used by the adversary. Although the accuracy of other ML models also dropped when the noise level was increased (*e.g.*, `eps`=0.1), some classifiers still outperformed CNN significantly (*e.g.*, 0.294 for SVM compared to 0.086 for CNN). Moreover, the attacker could conduct adversarial training to easily circumvent the defense, as shown shortly.

2) Conducting Adversarial Training: Although the defender can generate adversarial samples to fool the attacker, the attacker can utilize the adversarial training technique to reinforce its learning. We used the FGSM method mentioned above to craft the adversarial samples for our training set with the noise level `eps`=0.1, and re-trained the CNN model for 10 epochs. After this process, the new classifier achieved 0.908 accuracy on the adversarial test samples, which is significantly higher than 0.086, the original accuracy.

Of course, the defender could then regenerate adversarial samples against the new model to defeat such attacks; however, the attacker is also capable of adapting accordingly. This arms race can be repeated for many rounds, until they reach an equilibrium (if one exists). For instance, with the dataset we have, we continued the arms race and performed 4 more rounds of the following experiments: In each round, the defender regenerated adversarial samples against the model used by the attacker, and then the attacker trained a new model according to the adversarial samples. The resulting classification accuracy (including the first round) was $[(0.086, 0.908), (0.156, 0.780), (0.272, 0.705), (0.245, 0.536), (0.264, 0.410)]$, where the first element of each 2-tuple is the classification accuracy after the defender's move and the second is the result after the attacker's move. Because the dataset is not huge, the results converged quickly. However, there is no known principled way to find such an equilibrium [22], and it requires a lot of effort to do so empirically. Therefore, unlike the majority of works in adversarial ML where the classifier is the victim and is assumed to stay unchanged, in our setting the adversary is assumed to be aware of any defense strategy that is taken and allowed to adapt accordingly. The defender faces a much harder situation when applying adversarial ML techniques under such an assumption.

V. Differentially Private Streaming

The failure in the adoption of adversarial samples to defeat streaming traffic analysis motivated us to seek more *principled* solutions to counter such a powerful adversary. Differential privacy stands out as a feasible solution. Differential privacy offers a principled privacy guarantee for statistical databases that allows users to query aggregate statistics of elements in the database without leaking individual data elements [17]. It offers strong privacy promises that guarantees statistical indistinguishability of two databases that are different in only one element.

In this section, we would like to develop ϵ-differentially private mechanisms for streaming traffic, which, by adding random noise (dictated by ϵ and a distance threshold t) into the encrypted video streams, render any two videos within distance t to be statistically indistinguishable to each other. In this sense, any two video streams within distance t (which can be selected by the defender) can be intermingled and made indistinguishable with respect to ϵ-differential privacy, though in extreme cases may require adding substantial noise. In this section, we explore two mechanisms, FPA_k and d^*-privacy, to enforce differential privacy on streaming traffic.

A. *Fourier Perturbation Algorithm (FPA_k)*

Rastogi *et al.* [52] proposed the Fourier Perturbation Algorithm (FPA_k), which can answer long query sequences over correlated time-series data in a differentially private manner by using the Discrete Fourier Transform (DFT). A DFT is a linear transform of a length-n real or complex-valued sequence $Q = (Q[1], ..., Q[n])$ into another length-n complex-valued sequence $F = (F[1], ..., F[n])$ where

$$F[j] = \sum_{i=1}^{n} exp(\frac{2\pi\sqrt{-1}}{n}ij)Q[i].$$

The $F[j]$ is called the j-th Fourier coefficient of the $\text{DFT}(Q)$. An Inverse DFT (IDFT) is also a linear transform of a complex-valued sequence $P = (P[1], ..., P[n])$ to another complex-valued sequence $R = (R[1], ..., R[n])$ where

$$R[j] = \frac{1}{n}\sum_{i=1}^{n} exp(\frac{2\pi\sqrt{-1}}{n}ij)P[i].$$

An IDFT has the property $IDFT(DFT(Q)) = Q$.

Let $\mathsf{Lap}(\lambda)$ denote a random variable drawn from the Laplace distribution with scale λ and location $\mu = 0$. Suppose the inputs of the FPA_k algorithm are Q, λ, and k. FPA_k is described as follows:

(a) Keep the first k Fourier coefficients $F[1], ..., F[k]$ after computing $\mathrm{DFT}(Q)$.
(b) Compute $\tilde{F}[i] = F[i] + \mathsf{Lap}(\lambda)$ for $i = 1, ..., k$.
(c) Return $\tilde{Q} = IDFT(PAD^n([\tilde{F}[1], ..., \tilde{F}[k]]))$, where $PAD^n([\tilde{F}[1], ..., \tilde{F}[k]])$ denotes the sequence of length n obtained by appending $n-k$ zeros to $\tilde{F}[1], ..., \tilde{F}[k]$.

Rastogi *et al.* [52] proved that FPA_k (Q, λ) is ϵ-differentially private for $\lambda = \sqrt{k}\Delta_2(\mathbb{Q})/\epsilon$, where $\Delta_2(\mathbb{Q})$ denotes the L2 sensitivity of a set of Qs. Formally, $\Delta_2(\mathbb{Q})$ is the smallest number such that for all $Q, Q' \in \mathbb{Q}$, $|Q - Q'|_2 \leq \Delta_2(\mathbb{Q})$.

B. d^*-private Mechanism

Xiao *et al.* [72] leveraged d-privacy with a particular distance metric d^* on one-dimensional time series. Let x and x' denote two time series. The d^* metric was defined as:

$$d^*(x, x') = \sum_{i \geq 1} |(x[i] - x[i-1]) - (x'[i] - x'[i-1])|$$

To achieve d^*-privacy, Xiao *et al.* [72] extended a mechanism from Chan et al. [11] to implement a d^*-private mechanism as follows: Let $\mathbb{N}$ denote the natural numbers and $D(i) \in \mathbb{N}$ denote the largest power of two that divides i; i.e., $D(i) = 2^j$ if and only if $2^j | i$ and $2^{j+1} \not| \, i$. Note that $i = D(i)$ if and only if i is a power of two. The mechanism A computes a noised value $\tilde{x}[i]$ that is used in place of $x[i]$ using the recurrence

$$\tilde{x}[i] = \tilde{x}[G(i)] + (x[i] - x[G(i)]) + r_i \tag{1}$$

where $x[0] = \tilde{x}[0] = 0$, and

$$G(i) = \begin{cases} 0 & \text{if } i = 1 \\ i/2 & \text{if } i = D(i) \geq 2 \\ i - D(i) & \text{if } i > D(i) \end{cases} \tag{2}$$

$$r_i \sim \begin{cases} \mathsf{Lap}\left(\frac{1}{\epsilon}\right) & \text{if } i = D(i) \\ \mathsf{Lap}\left(\frac{\lfloor \log_2 i \rfloor}{\epsilon}\right) & \text{otherwise} \end{cases} \tag{3}$$

It was proven by Xiao *et al.* [72] that the algorithm in Eqns. 1–3 is $(d^*, 2\epsilon)$-private and $(l_1, 4\epsilon)$-private.

C. Applying Privacy Mechanisms on Streaming Data

The attack scenario in Sec. III motivates the following scenario. A user who watches a Youtube video in a web browser wishes to hide the content of the video. An attacker sitting on the network (*e.g.*, Internet service provider or local network administrator) aims to infer the content of the video by observing only side-channel information. The defender is a network proxy placed between the content provider (*i.e.*, Youtube) and the browser, which obfuscates the network flows from/to the content provider to defeat the fingerprinting attacks. For example, the defender could be implemented as a front-end of the Youtube server or as an extension of the browser. The attacker utilizes features (*e.g.*, bytes per second, packets per second or burst series) of the request and response packets of the MPEG-DASH video streams as side-channel vectors.

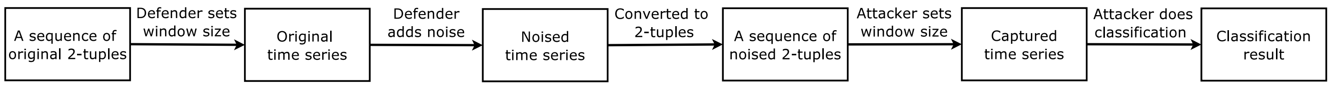
Without loss of generality, the problem can be simplified and abstracted as the following classification problem: An encrypted video stream can be modeled as a sequence of 2-tuples $\{(t_i, s_i)\}_{i\geq 0}$, where (t_i, s_i) represents a video segments of size s_i that is downloaded at time t_i. As t_i is a timestamp represented in continuous time, the adversary needs to discretize the sequence of 2-tuples by grouping all 2-tuples falling in the same time window of length w (*e.g.*, as small as a microsecond or as large as a second) into a single value. As such, each video stream is represented as a time series $x = \{b_j\}_{j\geq 0}$, where b_j is the total size of the downloaded video during time slot j. We let $\mathcal{X}$ denote the space of all possible such x values. Often, the attacker will collect feature vectors x and their associated labels in a *training phase*, to build a machine learning model to which it will apply observations x seen during his attack.

The goal of the defender is to prevent the videos from being identified by the attacker, which is achieved by adding random noise. The workflow of defense and attacks is depicted in Fig. 3. Specifically, the defender takes the following steps to reduce the information leakage. First, she sets a window size w to convert the 2-tuples (t_i, s_i) into a fix-length time series x. Then, she adds random noise, which is dictated by the differentially private mechanisms, to the time series, and generates the noised time series $\tilde{x}$. When the noised time series $\tilde{x}$ is reflected as packets, we assume all packets are transmitted instantaneously; depending on the maximum packet size allowed by the physical network layer, it can be represented as a sequence of 2-tuples $(\tilde{t_i}, \tilde{s_i})$, which are what the attacker observes. Note that the mapping from the time series to the sequence of two tuples is only determined by the network condition which is agnostic to the content of the video. The attacker then chooses his window size (w_A) to generate a new time series, denoted as $\dot{x}$, and performs classification on $\dot{x}$.

As such, when used to obfuscate streaming traffic, both differentially private mechanisms, FPA_k and d^*, require two parameters, w and ϵ. Here, w represents the window size, which also determines the length of $\tilde{x}$. For example, $w = 1s$ means each element of the noised time series represents the total size of downloaded video segments within an interval of $1s$. Parameter ϵ specifies the privacy level of the mechanism: the smaller the ϵ is, the better the privacy would be.

When w_A is different from w, $\tilde{x}$ and $\dot{x}$ may have different lengths. As a result, the attacker may need to merge/split bins in $\tilde{x}$ to create $\dot{x}$. Here, we let $\tilde{x}$ be a time series of n elements and $\dot{x}$ be a time series of n_A elements. Without loss of generality, we only consider cases where w_A mod w = 0 or w mod w_A = 0. The merging and splitting of bins are performed as follows:

- Merging is required when $w_A > w$. Let $r = w_A/w$. Every r bins from $\tilde{x}$ will be merged (summed) into one bin in $\dot{x}$,



A sequence of original 2-tuples
Defender sets window size
Original time series
Defender adds noise
Noised time series
Converted to 2-tuples
A sequence of noised 2-tuples
Attacker sets window size
Captured time series
Attacker does classification
Classification result

Fig. 3: Abstraction of data flow with defense.

i.e.,

$$\dot{x}[i] = \sum_{j=i\times r}^{i\times r+(r-1)} \tilde{x}[j]$$

For instance, when w_A = 2s and w = 1s, $r = 2$, $n_A = \frac{1}{2}n$. $\dot{x}[i] = \tilde{x}[2i] + \tilde{x}[2i+1]$.

- Splitting is required when $w_A < w$. Let $r = w_A/w$. Here we assume that the volume of each bin follows uniform distribution. Therefore, every bin from $\tilde{x}$ will be split (divided) evenly into $1/r$ bins in $\dot{x}$, *i.e.*,

$$\dot{x}[j] = r \times \tilde{x}[i],\ j = \frac{i}{r}, \cdots, \frac{i+1}{r} - 1$$

For instance, when w_A = 1s and w = 2s, $r = \frac{1}{2}$, $n_A = 2n$. $\dot{x}[2i] = \dot{x}[2i+1] = \frac{1}{2}\tilde{x}[i]$.

VI. Evaluation

In this section, we evaluate the security and utility of FPA_k and d^*. We implemented both mechanisms in Python. For FPA_k, k was set to 10, so during the Fourier transformation, only the first 10 Fourier coefficients were kept. FPA_k took a sequence of 2-tuples and parameter w and ϵ as input, discretized it into a time series x with window size w, calculated $\lambda = \sqrt{10}\Delta_2(\mathbb{Q})/\epsilon$ (where $\Delta_2(\mathbb{Q})$ denoted the L2 sensitivity of the set of 40 videos collected in Sec. III), and returned another time series $\tilde{x}$ of the same size after adding noise by following the steps mentioned in Sec. V-A. Similarly, in our implementation of d^*, it took a sequence of 2-tuples, w and ϵ as input, discretized it into a time series x with window size w, and outputted another time series $\tilde{x}$ after adding noise.

The two methods were applied on the 40×100 traces collected in Sec. III. In our experiment, we used ϵ ={5×10^{-8}, 5×10^{-7}, $\cdots$, 50}, w = {0.05s, 0.25s, 0.5s, 1s, 2s}, so there were 50 pairs in total. Each element of the noised time series was truncated by a clip bound of [0, 1GB] to avoid negative volume or enormous volume, because the download size cannot be negative and it is not realistic to complete downloading a large chunk of data within a small window size. Therefore, values less than 0 were changed to 0, and values larger than 1GB were truncated to 1GB.

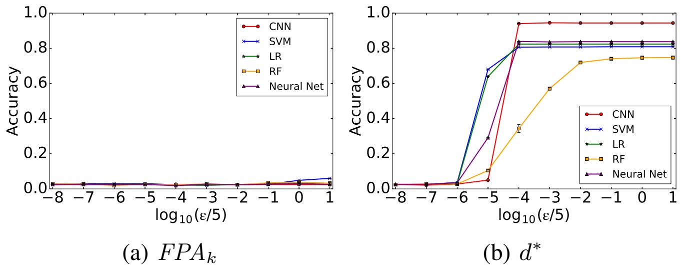


Fig. 4: Classification accuracy of 5-fold cross validation when trained with original traces and tested with noised traces.

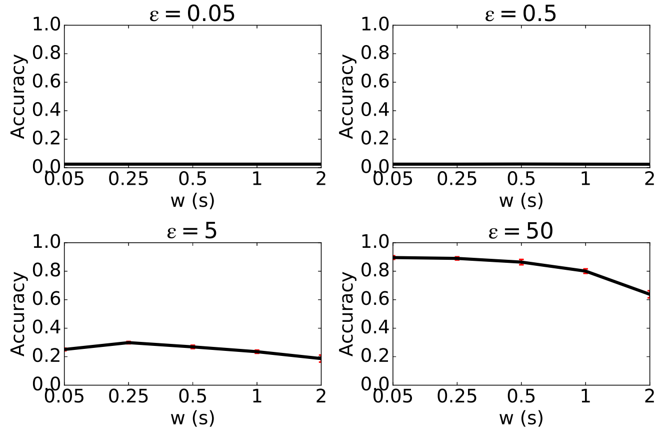
A. Security Evaluation

The security of the differentially private mechanisms are evaluated by classification accuracy. We used the same method mentioned in Sec. III-C to preprocess the data and train the classifiers. According to the dataset used for training and testing, we consider the following cases:

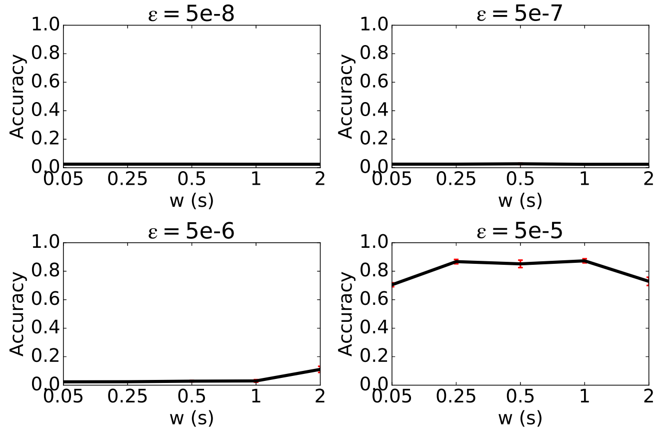
1) Trained with x (clean data), tested with $\tilde{x}$ (noised data): To compare with the defense mechanism of using adversarial samples described in Sec. IV-B, we first used the same 5 classifiers trained with original traces to classify the noised data generated by the two mechanisms with different choices of ϵ, when w = 0.25s. The classification accuracy and standard deviation of a 5-fold cross validation are shown in Fig. 4. For all the data points, the standard deviation is quite small (<0.01), hardly visible in the figures. For FPA_k (Fig. 4a), since it involved the Fourier transformation, the new traces were totally different from the originals, so the classifier could not recognize them for all ϵ values. For d^*, since the noise was added upon the original trace, ϵ played an important role. As shown in Fig. 4b, for $\epsilon \leq 5 \times 10^{-6}$, d^* was effective. When $\epsilon \geq 5 \times 10^{-5}$, the noise added was not enough to deceive the classifiers. These results suggest that with properly selected noise level, both mechanisms can effectively defeat traffic analysis attacks. In the following, we consider a more powerful adversary that could adapt by training the classifiers also with noised data.

2) Trained with $\tilde{x}$ (noised data), tested with $\tilde{x}$ (noised data): We evaluated how the two parameters, w and ϵ, would affect the security of the defense mechanisms by using the CNN classifier mentioned in Sec. III-C as the adversary and measuring the accuracy of the classification. We specifically consider two scenarios: $w_A = w$ and $w_A \neq w$.

• $w_A = w$. First, we consider the scenario where the attacker and the defender use the same w, which means that $\tilde{x}$ = $\dot{x}$. We altered w to see how it would affect the classification accuracy. The results of the classification accuracy and standard deviation of a 5-fold cross validation when $\epsilon = [0.05, 0.5, 5, 50]$ are shown in Fig. 5. The classification accuracy with FPA_k protected data is shown in Fig. 5a. When ϵ was small (*e.g.*, $\epsilon = 0.05$ and $\epsilon = 0.5$), more noise was added during the transformation. The classification accuracy remained low as w increased. However, when ϵ was larger (*e.g.*, $\epsilon = 5$ and $\epsilon = 50$), the noise level was low and w played a more significant role—when $w = 2s$, the classification accuracy went down by about 15%. This is because larger window sizes (used by the adversary during discretization) erased some important features in the data traces, making the classification harder. The classification accuracy with d^* protected data is shown in Fig. 5b. With smaller ϵ values (*e.g.*, $\epsilon = 5 \times 10^{-8}$ and $\epsilon = 5 \times 10^{-7}$), w still had no impact on the classification accuracy at all. A different trend was observed when $\epsilon = 5 \times 10^{-6}$: $w = 2s$ would increase the accuracy to about 25%. We conjecture it was related to the mechanism by which d^* added noise: The amount of noise added had a linear

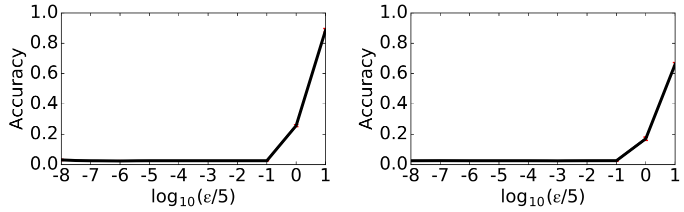


(a) FPA_k

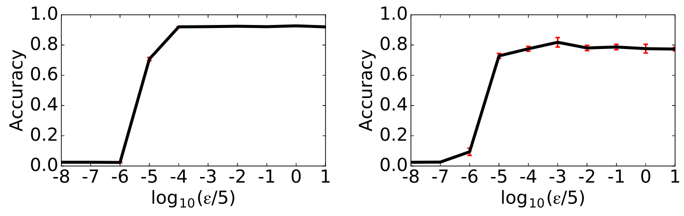


(b) d^*

Fig. 5: w_A = w: effect of w



(a) FPA_k: w = 0.05s (b) FPA_k: w = 2s



(c) d^*: w = 0.05s (d) d^*: w = 2s

Fig. 6: w_A = w: effect of ϵ

relationship with the length of the time series. When w was large, with the video length remaining the same, the time series had fewer elements. Therefore, the noise added was less, which was not enough to confuse the classifier. When $\epsilon = 5 \times 10^{-5}$, the classification accuracy fluctuated as w increases from 0.05 to 2. We believe this was the combined result of two causes: the larger window size reduced the noise level, but also eliminated some of the useful information used by the classifiers. For both methods, the standard deviation of each data point was very small (less than 0.01).

Next, we study the effect of ϵ. The result is shown in Fig. 6. The x axis is $log_{10}(\epsilon/5)$ (*e.g.*, $x = -3$ means that $\epsilon = 5 \times 10^{-3}$). We only show the cases of $w = 0.05s$ and $w = 2s$, since they were the smallest and largest w values we experimented with; result of other w values were similar. Similar to Fig. 5, the standard deviations in Fig. 6 were negligible. From Fig. 6, we can see that in order to keep a low classification accuracy, d^* method required a much smaller ϵ. For example, when $w = 0.05s$, to make sure the classifier had a baseline accuracy (*i.e.*, 2.5%, given 40 videos with 100 traces each), d^* needed $\epsilon \leq 5 \times 10^{-6}$, while FPA_k only required $\epsilon \leq 0.5$. This is because the definitions of ϵ in the two methods are different. We also provide a proof to bridge the two ϵ values in Appendix A.

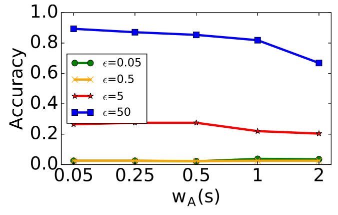
• $w_A \neq w$. Next, we consider the scenario where the attacker and the defender chose different w. To perform the experiment, first, we set ϵ ={5×10^{-8}, 5×10^{-7}, $\cdots$, 50}, respectively. Then, we let w = {0.05s, 0.25s, 0.5s, 1s, 2s}, and tested the classification accuracy when w_A = {0.05s, 0.25s, 0.5s, 1s, 2s}. We only show the results when $w = 0.05s$ and $w = 2s$ in Fig. 7. We can see from the figure that with the same w, when w_A increased, the classification accuracy for both methods decreased. The amount of decrease with d^* was more significant than FPA_k. From this result, it can be inferred that choosing a smaller w_A would benefit the adversary. This is because the larger window size used by the adversary during discretization erased some important features in the data traces.

From the defender's perspective, the choice of w made a difference in the effectiveness of the defense. For FPA_k, $w = 0.05s$ and $w = 2s$ did not differ much (Fig. 7a and Fig. 7b). But for d^*, w mattered: for $w = 0.05s$ (Fig. 7c), $\epsilon = 5 \times 10^{-6}$ was good enough to fool the classifier; but for $w = 2s$ (Fig. 7d), $\epsilon = 5 \times 10^{-6}$, the classifier can achieve an accuracy of 40% when $w_A \leq 0.5s$. Therefore, from the defender's perspective, if the d^* method is chosen, it is better to choose a smaller w to achieve better privacy.

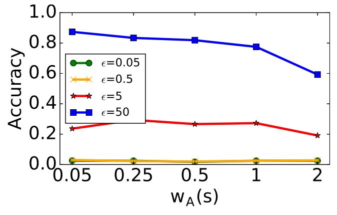
B. Utility Evaluation

We define two metrics, `waste` and `deficit`, to evaluate the utility of the mechanisms. Let the original time series be x and noised time series be $\tilde{x}$. Consider the cumulative traces $A = \sum_1^n x$ and $B = \sum_1^n \tilde{x}$. `waste` and `deficit` are defined as follows:

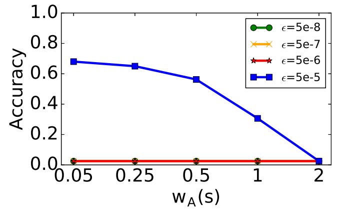
- We define `waste` as the maximum difference between traces A and B when the noised trace B is *above* the



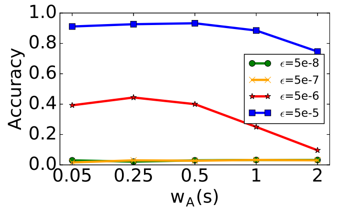
(a) FPA_k: w = 0.05s



(b) FPA_k: w = 2s



(c) d^*: w = 0.05s



(d) d^*: w = 2s

Fig. 7: $w_A \neq w$

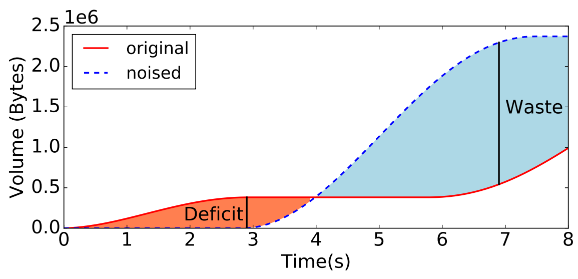


Fig. 8: An example of `waste` and `deficit`.

original trace A.

$$\texttt{waste} = \max_{1 \le i \le n} \{\max(B[i] - A[i], 0)\} \qquad (4)$$

- We define `deficit` as the maximum difference between A and B when the noised trace B is *below* the original trace A.

$$\texttt{deficit} = \max_{1 \le i \le n} \{\max(A[i] - B[i], 0)\} \qquad (5)$$

`waste` means the maximum amount of data that have been downloaded in advance during a time period, and `deficit` means the maximum amount of data that needs to be downloaded to keep streaming during a time period. An example of `waste` and `deficit` is illustrated in Fig. 8. The *red* line represents the cumulative volume of the original trace A, and the *blue* line is that of the noised trace B. The *orange* area means that the noised trace is behind the original one, while the *blue* area means that the noised trace is ahead of it. The `deficit` is the max difference between the two lines in the *orange* area, while the `waste` is that in the *blue* area.

The utility of the two mechanisms was evaluated using the same set of w and ϵ values as in Fig. 5. The `waste` and `deficit` of each noised trace were computed first, and the average `waste` and `deficit` over all traces were calculated and shown in Fig. 9 and Fig. 10. According to Fig. 9a, parameter w did not affect the `waste` of FPA_k much. But when ϵ increased, `waste` would decrease, since there was less noise added. Similarly, for d^*, ϵ was the major factor that affected the `waste` (see Fig. 9b). However, w also had an influence: When ϵ was fixed, larger w indicated fewer `waste` for d^*. We conjecture it was again related to the mechanism by which d^* added noise. The amount of noise added had a linear relationship with the length of the series. When w was larger, the time series was shorter for the same video length. Therefore, less noise was added, which resulted in smaller `waste`.

As shown in Fig. 10a, however, the `deficit` metric of the FPA_k and d^* mechanisms followed a different trend. In FPA_k, `deficit` was less fluctuated when w and ϵ changed (Fig. 10a). For different (w,ϵ) pairs, the average `deficit` stayed within [1.5MB, 3MB]. However, for d^*, changes in either w or ϵ affected the `deficit` significantly. From Fig. 10b, it was clear that when the w was small (*e.g.*, 0.05s), there was no `deficit` at all for all ϵ values; when the w was large (*e.g.*, 2s), the `deficit` could be as large as 0.8MB.

Note that it is possible to take measures to lower the `waste` and `deficit`. For example, one can choose an ϵ value to ensure privacy while keeping a reasonable `waste` and `deficit`. The clip bounds can also be adjusted to limit the maximum/minimum download rate for each bin while maintaining differential privacy. Lowering the upper bound can reduce the `waste`, while increasing the lower bound can remove the `deficit`. Also, the `deficit` can be easily eliminated if buffering the video content upfront for a few seconds.

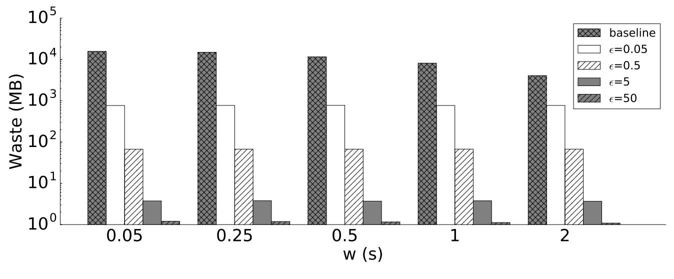
C. FPA_k vs. d^*

To compare FPA_k and d^*, we chose the best parameters for each method, with the baseline accuracy (*i.e.*, 2.5%) and lowest `waste`. For FPA_k, the best parameters were w = 2s, ϵ = 0.5; for d^*, w = 0.5s, $\epsilon = 5 \times 10^{-6}$ were chosen. The `waste` and `deficit` distribution of the 4000 traces after applying the two methods are shown in Fig. 11. From Fig. 11a, it is clear that with the best parameters, FPA_k traces had a median `waste` of about 200%, while that of d^* traces was even higher (600%). For `deficit` (Fig. 11b), however, d^* performed a lot better. More than 80% of d^* traces had a `deficit` less than 1%, while the majority of FPA_k traces (> 50%) had at least 5% `deficit`.

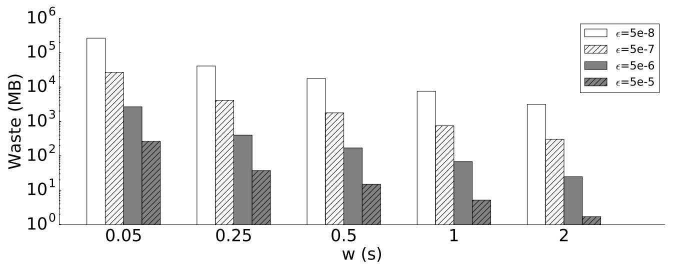
From Fig. 11, we find that FPA_k tended to induce less `waste` (about 200% of the original video size). To achieve similar security protection, d^* had to download 3 times more volume. To achieve differential privacy, some utility loss has to be allowed in either case. Moreover, it is clear that with the same security level (in regards to classification accuracy), if the primary objective is minimize the `waste`, FPA_k is the best choice; if the main goal is to reduce the `deficit`, d^* would be the better option.

D. Comparison with Baseline Defense

In a baseline defense mechanism, the defender could simply download at a constant rate for all videos in the dataset. To make videos with different total data size indistinguishable, smaller videos need to be padded with dummy data to obfuscate the traffic analysis. We designed the following

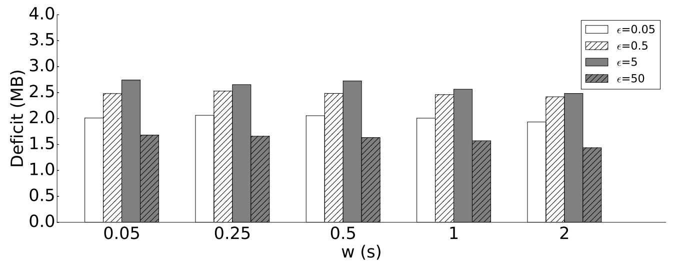


(a) FPA_k

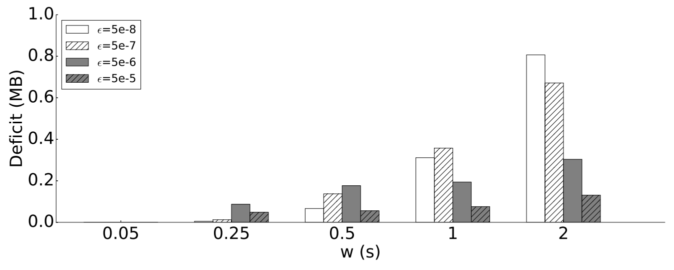


(b) d^*

Fig. 9: `waste` experiment.



(a) FPA_k



(b) d^*

Fig. 10: `deficit` experiment.

mechanism to avoid introducing `deficit` in the resulting streams: With a bin size of w, we divided each time series into multiple bins, and identified the maximum value of downloaded data (denoted as C) for all bins of these 4000 original time series. Then as a baseline defense mechanism, all videos were downloaded at a constant rate of C bytes per w. As such, from the attacker's perspective, all video streams were identical, and no `deficit` would be incurred for the noised video. We evaluated this baseline method with $w = [0.05s, 0.25s, 0.5s, 1s, 2s]$, and the corresponding `waste` are $[15.7GB, 14.9GB, 11.5GB, 8.1GB, 4.1GB]$, which represent the extra data downloaded for a 3-minute video.

We note that it is only fair to compare this baseline approach with FPA_k, because both of them require knowledge of the download profiles of all videos in a dataset (*i.e.*, the set of videos the defender would like to render indistinguishable). By contrast, d^* can be used to add noise on-the-fly. As shown in Fig. 9a, the `waste` induced by FPA_k is at least one order of magnitude lower than the baseline approach. With a tunable privacy level ϵ, *i.e.*, by enforcing statistical indistinguishability rather than absolute indistinguishability, FPA_k can be much more practical (*e.g.*, with less than 10MB `waste` when $\epsilon = 5$).

E. An Optimal Attacker

In the previous subsections, we have evaluated the two differentially private mechanisms with an adversary that could both train and test with noised data. Now we consider a even more powerful attacker who is not only able to train his classifier with noised data, but also has the knowledge of distribution of both clean data and noised data (but not the mapping between the two). With such knowledge, the adversary could first try to remove the noise from the noised time series, and then perform classification (*e.g.*, CNN) to classify the resulting time series. The method to do so is proposed by Naldi *et al.* [43]. Essentially, it requires the attacker to estimate the clean time series by calculating the conditional expectation of each clean data point conditioned on the observed, noised data points.

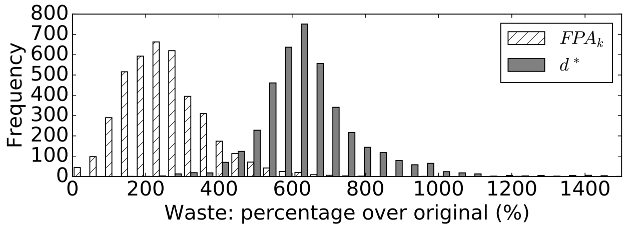
To evaluate the effectiveness of this optimal attacker method, we first calculated the estimated clean dataset from the noised dataset, and then chose 80% of the estimated clean dataset to train the classifier and the rest 20% as the test set to perform classification. The classification accuracy is shown in Table II. Compared with Fig. 5, it appears that the optimal attacker performs slightly better in some cases, but the improvement of the accuracy is no more than 2%. The experiments suggest that even with knowledge of distribution of both clean data and noised data, an optimal attacker cannot significantly improve his attack.

	FPA_k				d^*			
$w(s)$ \ ϵ	0.05	0.5	5	50	5e-8	5e-7	5e-6	5e-5
0.05	0.03	0.03	0.25	0.89	0.03	0.03	0.03	0.72
0.25	0.03	0.03	0.30	0.89	0.03	0.03	0.03	0.89
0.5	0.03	0.03	0.27	0.87	0.02	0.02	0.03	0.86
1	0.03	0.03	0.27	0.80	0.02	0.02	0.11	0.89
2	0.03	0.03	0.17	0.65	0.03	0.03	0.10	0.75

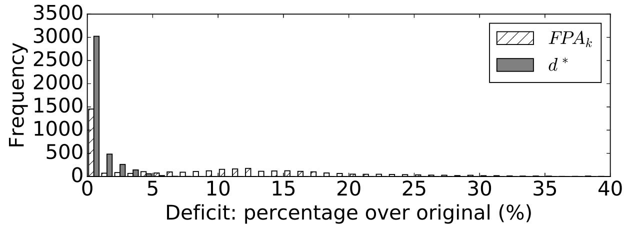
TABLE II: Classification results of the optimal attacker.

VII. Real-world Implementation

To demonstrate the practicality of our approach, we implemented the FPA_k privacy mechanism in a Chrome extension

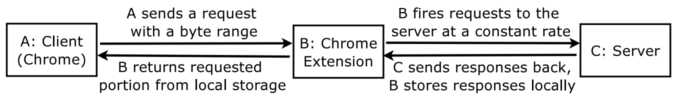


(a) `waste`



(b) `deficit`

Fig. 11: FPA_k (w = 2s, ϵ = 0.5) vs. d^* (w = 0.5s, ϵ = 5e-6)



A: Client (Chrome)
A sends a request with a byte range
B returns requested portion from local storage
B: Chrome Extension
B fires requests to the server at a constant rate
C sends responses back, B stores responses locally
C: Server

Fig. 12: Workflow of the `Chrome` extension.

that proxies Youtube streaming. The workflow of the extension is illustrated in Fig. 12. First, the Youtube client running inside the Chrome browser sends a request to the Youtube server, which is intercepted by the extension. Instead of relaying the request immediately, the proxy sends requests on behalf of the client at a constant rate (*e.g.*, once per second), which is specified by the w parameter of the extension. After receiving the responses from the server, the proxy caches the video chunks locally. If there is a pending request from the Youtube client, the extension returns the requested portion to the client directly from local storage. In this way, the Youtube requests/responses as seen by an external observer are fully controlled by the extension. Since the request pattern from the proxy is differentially private, traffic analysis is thwarted.

The size of the video chunks to be requested is specified as a parameter (*i.e.*, `range`) of the HTTP request header. Youtube video streaming implements a variant of MPEG-DASH [40], which allows the client to specify a chunk of video to be downloaded by setting the `range` parameter to the desired offsets in bytes. The YouTube client adaptively changes this parameter to adjust the requested video chunk size, based on the content of the video and the network condition. To enforce the privacy guarantee, the `range` parameters in the proxy's requests are decoupled from those in the client's requests. The requests sent by the Chrome extension use a `range` parameter dictated by the FPA_k mechanism. To properly watch a Youtube video, both its video stream and its audio stream needs to be downloaded. We applied the differentially private mechanism on both streams.

Implementation. In our implementation, we made use of the `Xhook` [25] framework, which allows us to intercept and modify the `XMLHttpRequest` requests and responses. In our implementation of FPA_k, $k = 10, w = 1s, \epsilon = 0.5$. We used the `numjs` [44] library, which is similar to Python's `numpy`, to implement numeric computation, and used the `Random` library in `SIM.JS` [42] to implement the Laplace distribution. The extension has about 700 lines of Javascript code in total. Note that the use of FPA_k requires the original trace of the video to be known to the proxy beforehand.

Data collection. We used the same methods described in Sec. III to collect traces for 10 videos, and 100 traces for each video, with our extension enabled. Therefore, the network traffic observed is only the communication between the extension and the Youtube server. The traces were collected when the w parameter of the extension was set to 1s, which means that it would send a video request and an audio request to the Youtube server every 1 second.

Effectiveness. To demonstrate that the extension can indeed defeat ML-based traffic analysis, we extracted 12 features which were also time series from the stream and performed classification one by one. The features were: the number bytes per bin (BPB_{up}, BPB_{down}, BPB), the number of packets per bin (PPB_{up}, PPB_{down}, PPB), the average packet length per bin (LPB_{up}, LPB_{down}, LPB), the size of bursts[2] ($BURST_{up}$, $BURST_{down}$, $BURST$). The subscription "up" means packets from client to server; "down" means packets from server to client; no subscription means the sum of "up" and "down". We also evaluated the classification accuracy with all 12 features combined, labeled as ALL.

The dataset (1000 traces) was split into a training set (80%, 800 traces) and a test set (20%, 200 traces). We set w_A = {0.05s, 0.25s, 0.5s, 1s, 2s} to bin the traces, then trained the CNN model in Sec. III for 40 epochs with a batch size of 32 using the training set. After that, the classification was performed on the test set. The results are shown in Table III. As expected, the CNN model can hardly classify these obfuscated traces. For most cases, the classifier only achieved an accuracy of about 15%. Using certain features may increase the classification accuracy (*e.g.*, 23% with $BURST_{up}$ for $w_A = 0.25s$), which were still significantly lower than the values in Table I. This result suggests that differential privacy is effective in defeating machine learning adversaries. According to the Post-processing Lemma [18], the composition of differentially private mechanisms is still differentially private. Therefore, combining the features does not benefit the attacker. As shown in the "ALL" column in Table III, the 12-feature combined classification accuracy remained on the same level as individual features.

Usability. While we cannot directly quantify the user experience, in our evaluation, the video streaming went smoothly without pausing after buffering for roughly 3 seconds at the

[2]A burst is the total size of all packets whose timestamps are no farther apart than a threshold. Here the threshold is set to $0.5s$.

w_A (s)	BPB_{up}	BPB_{down}	BPB	PPB_{up}	PPB_{down}	PPB	LPB_{up}	LPB_{down}	LPB	$BURST_{up}$	$BURST_{down}$	$BURST$	ALL
0.05	0.16	0.12	0.16	0.12	0.16	0.14	0.14	0.13	0.16	0.14	0.15	**0.16**	0.13
0.25	0.20	0.16	0.22	0.18	0.16	0.20	0.12	0.08	0.16	**0.23**	0.14	0.19	0.21
0.5	0.19	0.12	**0.22**	0.14	0.16	0.20	0.14	0.08	0.10	0.19	0.14	0.15	0.20
1	0.16	0.14	0.18	0.14	**0.19**	0.13	0.10	0.10	0.11	0.16	0.14	0.12	0.18
2	0.14	0.12	0.16	0.13	0.14	0.16	0.10	0.10	0.09	0.16	0.16	**0.19**	0.17

TABLE III: Classification result when the `Chrome` extension is enabled. Each column represents the accuracy when trained with the specified feature. The features are up/down/total bytes per bin (BPB), up/down/total packets per bin (PPB), up/down/total average packet length per bin (LPB), up/down/total bursts ($BURST$), and the combination of all 12 features (ALL).

very beginning. We leave a comprehensive user study on the usability as future work.

Nevertheless, an optimal implementation of our statistical privacy mechanisms would enforce the privacy on both the client side and the server side. The browser extension can only control the request rate of the Youtube video streaming, but cannot directly control the response rate from the server. If the server chooses to respond to a request with a packet pattern that is specific to the downloaded video, privacy of the streaming traffic can not be protected by the extension alone. Fortunately, as shown in our experiment, it is not the case—packet patterns in the video download are not content-specific. Therefore, the packet patterns do not leak additional information.

VIII. Discussion

In this section, we discuss the limitation and extension of the statistical privacy approaches.

Leakage through video length. Neither of the two statistically private mechanisms prevents leakage through the length of the videos. Intuitively, to make an 1-minute video indistinguishable from an 1-hour video, considerable amount of noise must be added to hide the difference of the video length, as the L2 sensitivity in this case is prohibitively high. As a result, the utility of the solution will drop significantly. Therefore, in practice, it is more desirable to only make videos with similar length indistinguishable from one another. To do so, grouping the videos by length and padding them to the longest length in each group might be a good solution. For example, all videos of the length between 50 minutes to 1 hour could be considered in one group and padded so that all of them appear to be a 1-hour video.

Comparing FPA_k with d^*. In Sec. VI-C, we compared the utility of the two differentially private mechanisms with certain w and ϵ parameters selected to render the CNN classifier ineffective. However, CNN classification accuracy does not translate directly to security guarantees. As these mechanisms offer different theoretical privacy guarantees, directly comparing them is less meaningful. However, it is worth mentioning that FPA_k mechanism additionally requires the knowledge of the entire time series x before transforming it into the noised version $\tilde{x}$. This additional requirement may be less desirable in scenarios where such information is not available.

Applying differential privacy to website fingerprinting. Although we have shown that differentially private mechanisms are promising countermeasures to streaming traffic analysis attacks, directly applying the same approach to prevent website fingerprinting requires some modifications. Unlike streaming traffic, HTTP traffic is more interactive. For example, an HTML web page may embed a number of objects (*e.g.*, JavaScript files or images) that will be downloaded after the HTML file is parsed by the browser. While streaming allows us to proactively request video contents beforehand and cache data locally, the download of some HTML resources can only start after finishing the download of a previous resource. We plan to address this type of interactive web traffic and expand our approach to WF attacks in future work.

Reducing `waste`. As shown in Fig. 11a, the median `waste` of FPA_k and d^* are 200% and 600%, respectively, compared to the original traces. To be practical, measures must be taken to lower the `waste`. For example, one can lower the security guarantee by increasing the ϵ, so that the amount of noise added is reduced. Another possible approach is to make the upper clip bound smaller. In Fig. 11a, the upper clip bound is set to 1GB, which is far from realistic scenarios, since it is impossible for the server to send 1GB in a single time frame for all the ws we tested. It would be more reasonable to find an empirical clip bound based on real-world statistics.

IX. Related Work

A. Defenses against Side-Channel Attacks

Our work has been influenced by prior studies that insert noise to obfuscate side-channel observations. Many research projects have tried to perturb timers to mitigate timing side-channel attacks [34], [35], [39], [68]. Researchers have also shown that adding noise to shared resources can be an effective defense [5], [28], [59], [75]. Particularly relevant to our work is Xiao *et al.*'s [72], which introduced the d^* algorithm to mitigate storage side channels resulting from `procfs` in the Linux OS, so that statistics reporting through `procfs` satisfies d-privacy for a meaningful distance metric d^*. Their work considered interactive statistical data release, *i.e.*, in which the defender knows exactly when and how the adversary observes the data. In our case, the adversary does not have to interact with the defense system; he only needs to passively observe the streaming traffic, which requires this defense to be more pervasively applied. This, in turn, underscores the importance of measuring its utility impact, as we have done here.

B. Privacy of Time-Series Data

Our work is built upon a number of previous studies that apply differential privacy to time-series data. Rastogi *et al.* [52] proposed the Fourier Perturbation Algorithm (FPA_k) algorithm to ensure differential privacy for time-series data. To avoid relying on a trusted central server, the work also proposed the Distributed Laplace Perturbation Algorithm (DLPA) for distributed time-series data. Shi *et al.* [58] proposed aggregator-oblivious encryption to ensure differential privacy for distributed time-series data. Benhamouda *et al.* [3] extended this work to introduce a general framework for constructing privacy-preserving aggregator-oblivious encryption schemes. Fan *et al.* [20] presented a framework, FAST, to release real-time aggregate statistics under differential privacy based on filtering and adaptive sampling. Cao *et al.* [9] proposed two methods to answer a subset of representative sliding window queries with differential privacy. Kellaris *et al.* [27] introduced ω-event privacy over infinite streams, which protects any event sequence occurring in ω successive timestamps. None of these works has considered applying differential privacy to defeat traffic analysis, however.

C. Website Fingerprinting Defenses

One important branch of traffic analysis is website fingerprinting (WF) on encrypted channels or anonymity networks (*e.g.*, Tor). In a typical WF attack, the adversary utilizes supervised machine learning techniques to train a classifier with encrypted network traffic to/from a set of websites of interest and then classify unknown traffic captured from the victim. Prior works have shown effectiveness of such attacks [8], [23], [47], [48], [60], [61], [69].

Accordingly, many research projects have explored mechanisms to address this security threat. Panchenko *et al.* [48] proposed a defense called Decoy, which loads two webpages at the same time so that the adversary is confused. Luo *et al.* [37] published HTTPOS (HTTP Obfuscation). The defense was implemented on the client side by splitting the traffic into packets with random size using the HTTP `range` header. Dyer *et al.* [19] provide a comprehensive study of countermeasures of traffic analysis and proposes a mechanism dubbed BuFLO that modifies the traffic to enforce a constant rate. Two follow-up works by Tamaraw *et al.* [7] and Cai *et al.* [6] aimed to reduce the overhead of BuFLO by grouping websites with similar traffic patterns and padding them according to the one with greatest size in each group. This grouping-and-padding method was used by other papers as well [45], [69]. Juarez *et al.* [26] proposed a system called WTF-PAD. It deploys adaptive padding for WF defense in Tor, which only adds padding when the usage of the channel is low, so that the bursts of traffic are eliminated. Recently, a defense called Walkie-Talkie was proposed by Wang *et al.* [70], which uses half-duplex communication to ensure one way communication at a time and adds dummy packets and inserts delays to make the traffic of different websites look alike. The major difference between these work and ours is that our method is designed with a theoretical privacy guarantee. We believe our solution can be applied to WF attacks as well. However, unlike streaming traffic, which is essentially non-interactive, additional care must be taken to eliminate leakage through interactive traffic patterns. We plan to expand our approach to WF attacks in future work.

D. Private Messaging Systems

Prior works have applied differential privacy techniques in private messaging systems. One of the first systems is Vuvuzela [67], which is a large-scale private messaging system that protects against both passive and active adversaries with differential privacy guarantee. Alpenhorn [31] extends Vuvuzela by providing strong privacy and forward secrecy guarantees for metadata. Alpenhorn uses the mixnet design provided by Vuvuzela, which guarantees its differential privacy. A recently published system, called Stadium [66], extends Vuvuzela horizontally by providing point-to-point data privacy while maintaining a low latency. Similar to Vuvuzela, Stadium has a security guarantee of differential privacy by verifiable shuffling and adding dummy messages. Another private messaging system, Atom [30], also employs the differential privacy technique proposed by Vuvuzela to hide the number of dialing calls a user receives. Moreover, Atom guarantees the users can have anonymity among honest users besides differential privacy. Although these works also applied differential privacy to prevent traffic analysis, however, their scenarios are completely different. In these private messaging systems, the information they are trying to hide is the participants of communications, *i.e.*, who is talking to whom in the system. In our scenario, the two parties involved are obviously known—the client and the server; however, we strive to prevent traffic analysis from divulging the content that is being streamed from the server to the client.

E. Privacy Using Adversarial ML

The possibility that adversarial ML might be leveraged to improve privacy by interfering with automated classification of observations is a relatively new idea, and one that has been explored only in the context of image classification. Oh et al. [46] specifically considered methods to interfere with automated person recognition in an image. Marohn et al. [38] similarly explored the effectiveness of an image-obfuscation technique dubbed "thumbnail preserving encryption" against ML classifiers. To our knowledge, our work is the first to explore adversarial ML as a privacy protection in the domain of traffic analysis.

X. Conclusion

In this paper, we borrowed techniques from adversarial machine learning and differential privacy to address privacy concerns of streaming traffic. Our findings suggest that constructing adversarial samples effectively confounds an adversary with a predetermined classifier but is less effective when the adversary can adapt to the defense, either by using alternative classifiers or training the classifier with adversarial samples. On the other hand, differential privacy effectively defeats statistical-inference-based traffic analysis, while remains agnostic to the machine learning classifiers used by the adversary. Our evaluation suggests that the two differentially private mechanisms used in the paper offer good security protection with moderate utility loss.

Acknowledgement

We thank the anonymous reviewers for their valuable comments. This project is supported in part by NSF grants 1718084, 1750809, 1801494, and grant W911NF-17-1-0370 from the Army Research Office. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[2] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, 2009.

[3] F. Benhamouda, M. Joye, and B. Libert, "A new framework for privacy-preserving aggregation of time-series data," *ACM Transactions on Information and System Security (TISSEC)*, 2016.

[4] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, 1988.

[5] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert, "Software mitigations to hedge aes against cache-based software side channel vulnerabilities." *IACR Cryptology ePrint Archive*, 2006.

[6] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014.

[7] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[9] J. Cao, Q. Xiao, G. Ghinita, N. Li, E. Bertino, and K.-L. Tan, "Efficient and accurate strategies for differentially-private sliding window queries," in *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 2013.

[10] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017.

[11] T.-H. H. Chan, E. Shi, and D. Song, "Private and continual release of statistics," *ACM Transactions on Information and System Security (TISSEC)*, 2011.

[12] K. Chatzikokolakis, M. E. Andrés, N. E. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2013.

[13] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it: Ui state inference and novel android attacks." in *USENIX Security Symposium*, 2014.

[14] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[15] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013.

[16] W. Diao, X. Liu, Z. Li, and K. Zhang, "No pardon for the interruption: New inference attacks on android through interrupt timing analysis," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016.

[17] C. Dwork, "Differential privacy," in *Proceedings of the 33rd International Conference on Automata, Languages and Programming (ICALP)*, 2006.

[18] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, 2014.

[19] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *2012 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2012.

[20] L. Fan and L. Xiong, "An adaptive approach to real-time aggregate monitoring with differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, 2014.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[22] J. Hamm, "Machine vs machine: Minimax-optimal defense against adversarial examples," *arXiv preprint arXiv:1711.04368*, 2017.

[23] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique." in *USENIX Security Symposium*, 2016.

[24] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, 2012.

[25] jpillora, "Xhook: Easily intercept and modify xhr request and response." [Online]. Available: https://github.com/jpillora/xhook

[26] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *European Symposium on Research in Computer Security*. Springer, 2016.

[27] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias, "Differentially private event sequences over infinite streams," *Proceedings of the VLDB Endowment*, 2014.

[28] G. Keramidas, A. Antonopoulos, D. N. Serpanos, and S. Kaxiras, "Non deterministic caches: A simple and effective defense against side channel attacks," *Design Automation for Embedded Systems*, 2008.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012.

[30] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally scaling strong anonymity," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017.

[31] D. Lazar and N. Zeldovich, "Alpenhorn: Bootstrapping secure communication without leaking metadata." in *OSDI*, 2016.

[32] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, 1995.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[34] P. Li, D. Gao, and M. K. Reiter, "Mitigating access-driven timing channels in clouds using stopwatch," in *Dependable systems and networks (DSN), 2013 43rd Annual IEEE/IFIP international conference on*. IEEE, 2013.

[35] W. Liu, D. Gao, and M. K. Reiter, "On-demand time blurring to support side-channel defense," in *European Symposium on Research in Computer Security*. Springer, 2017.

[36] X. Liu, Z. Zhou, W. Diao, Z. Li, and K. Zhang, "When good becomes evil: Keystroke inference with smartwatch," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[37] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci, "Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows," in *NDSS*, 2011.

[38] B. Marohn, C. V. Wright, W.-C. Feng, M. Rosulek, and R. B. Bobba, "Approximate thumbnail preserving encryption," in *1st International Workshop on Multimedia Privacy and Security*, Oct. 2017.

[39] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," *ACM SIGARCH Computer Architecture News*, 2012.

[40] A. Mondal, S. Sengupta, B. R. Reddy, M. Koundinya, C. Govindarajan, P. De, N. Ganguly, and S. Chakraborty, "Candid with youtube: Adaptive streaming behavior and implications on data consumption," in *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. ACM, 2017.

[41] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. EPFL-CONF-218057, 2016.

[42] mvarshney, "A general-purpose discrete event simulation library written entirely in javascript." [Online]. Available: https://github.com/mvarshney/simjs-source

[43] M. Naldi and G. D'Acquisto, "Differential privacy for counting queries: can bayes estimation help uncover the true value?" *arXiv preprint arXiv:1407.0116*, 2014.

[44] nicolaspanel, "Numjs: Like numpy, in javascript." [Online]. Available: https://github.com/nicolaspanel/numjs

[45] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014.

[46] S. J. Oh, M. Fritz, and B. Schiele, "Adversarial image perturbation for privacy protection – a game theory perspective," in *IEEE International Conference on Computer Vision*, Oct. 2017.

[47] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

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

[49] N. Papernot, N. Carlini, I. Goodfellow, R. Feinman, F. Faghri, A. Matyasko, K. Hambardzumyan, Y.-L. Juang, A. Kurakin, R. Sheatsley, A. Garg, and Y.-C. Lin, "cleverhans v2.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2016.

[50] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016.

[51] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, 2011.

[52] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010.

[53] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[54] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on Acoustics, speech and signal processing (ICASSP)*. IEEE, 2013.

[55] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *USENIX Security Symposium*, 2017.

[56] Selenium, "Selenium - web browser automation." [Online]. Available: https://www.seleniumhq.org

[57] M. Sharif, L. Bauer, and M. K. Reiter, "On the suitability of l_p-norms for creating adversarial examples," *arXiv preprint arXiv:1802.09653*, 2018.

[58] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *NDSS*, 2011.

[59] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2011.

[60] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," *arXiv preprint arXiv:1801.02265*, 2018.

[61] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*. IEEE, 2002.

[62] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014.

[63] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions." CVPR, 2015.

[64] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[65] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," in *Advances in neural information processing systems*, 2014.

[66] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017.

[67] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM, 2015.

[68] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM, 2011.

[69] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting," in *USENIX Security Symposium*, 2014.

[70] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *USENIX Security Symposium*, 2017.

[71] Wireshark, "tshark - the wireshark network analyzer 2.4.6." [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[72] Q. Xiao, M. K. Reiter, and Y. Zhang, "Mitigating storage side channels using statistical privacy mechanisms," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015.

[73] X. Zhang, X. Wang, X. Bai, Y. Zhang, and X. Wang, "Os-level side channels without procfs: Exploring cross-app information leakage on ios," in *NDSS*, 2018.

[74] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM side channels and their use to extract private keys," in *19th ACM conference on Computer and communications security*. ACM, 2012.

[75] Y. Zhang and M. K. Reiter, "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

Appendix

Theorem 1. *Suppose for database $\mathbb{D}$, we have method A that is ϵ-private, and method B that is (d^*, ϵ)-private. We denote the maximum and minimum d^* distance in $\mathbb{D}$ as d_{max} and d_{min}. Then we have:*

(1) If B is (d^*, ϵ)-private, then B is (ϵd_{max})-private.

(2) If A is ϵ-private, then A is $(d^*, \frac{\epsilon}{d_{min}})$-private.

PROOF. According to the definitions, we have:

$$A : \mathbb{P}(A(x) \in Z) \leq exp(\epsilon_A) \times \mathbb{P}(A(x') \in Z) \tag{6}$$

$$B : \mathbb{P}(A(x) \in Z) \leq exp(\epsilon_B \times d^*(x, x')) \times \mathbb{P}(A(x') \in Z) \tag{7}$$

For B, we have:

$$\epsilon_B \times d_{min} \leq \epsilon_B \times d^*(x, x') \leq \epsilon_B \times d_{max} \tag{8}$$

If B is (d^*, ϵ)-private,

$$\frac{\mathbb{P}(A(x) \in Z)}{\mathbb{P}(A(x') \in Z)} = exp(\epsilon \times d^*(x, x')) \leq exp(\epsilon \times d_{max}) \tag{9}$$

So B is at least (ϵd_{max})-private. Similarly, if A is ϵ-private, let $\epsilon = \epsilon' \times d^*(x, x')$, we have:

$$\epsilon' = \frac{\epsilon}{d^*(x, x')} \leq \frac{\epsilon}{d_{min}} \tag{10}$$

So A is at least $(d^*, \frac{\epsilon}{d_{min}})$-private.