International Journal of Information Security
https://doi.org/10.1007/s10207-021-00574-3

REGULAR CONTRIBUTION

Defeating traffic analysis via differential privacy: a case study on streaming traffic

Xiaokuan Zhang[1] · Jihun Hamm[2] · Michael K. Reiter[3] · Yinqian Zhang[4]

Abstract
In this paper, we explore the adaption of techniques previously used in the domains of adversarial machine learning and differential privacy to mitigate the ML-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 three 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

Keywords Differential privacy · Traffic analysis

1 Introduction

Machine learning (ML) is a powerful tool which can extract implicit information from massive data, which has shown promising results in many areas such as image recognition [17,26,54,56] and natural language processing [22,38,53]. However, when used with malicious intentions, ML also empowers notable attacks. One such example is the traffic-analysis attack, which is a type of side-channel attack.

✉ Yinqian Zhang
yinqianz@acm.org

Xiaokuan Zhang
zhang.5840@osu.edu

Jihun Hamm
jhamm3@tulane.edu

Michael K. Reiter
michael.reiter@duke.edu

1 Ohio State University, Columbus, Ohio, USA

2 Tulane University, New Orleans, LA, USA

3 Duke University, Durham, NC, USA

4 Research Institute of Trustworthy Autonomous Systems and Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, Guangdong, China

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. Traffic analysis, or more specifically website fingerprinting, aims to breach users' privacy by inferring visited websites from the encrypted traffic. 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 [51].

A recent work [49] in USENIX Security shows that a traffic analysis attacker, by merely observing the encrypted network packet sequence of the victim, is able to infer the video that the victim is watching on popular online video streaming platforms (*e.g.*, Youtube, Netflix). They show that by extracting the packet burst patterns of the encrypted video streams, the adversary can achieve very high classification accuracy (*e.g.*, 99% for Youtube videos). This indicates that the online video streaming is no longer private from a passive network observer, even if proper encryption mechanism is applied. 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

Published online: 30 January 2022

and explore generic solutions to ML-powered side-channel attacks [34,44,64,65].

Adversarial samples as defenses Inspired by the recent advances in adversarial machine learning, we first explore the use of adversarial samples to defeat ML adversaries. Usually, adversarial samples are utilized by the attacker to fool ML defenders. However, in our scenario, we consider the inverse use: utilize the adversarial examples to confuse ML attackers. To defeat the ML attacker, which is a convolutional neural network (CNN) classifier, we apply the Projected Gradient Descent (PGD) [35] to generate adversarial samples, and we successfully lower the classification accuracy to the baseline. Nevertheless, adversarial samples are fragile: they do not transfer well, and the classifier can also adapt to them. by choosing a different classifier or applying adversarial training, the adversary who aims to perform traffic analysis on encrypted streaming packets can still achieve high accuracy.

Differential privacy as 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 Xiao et al. [62], 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 protect streaming traffic by applying differential privacy, we adapt three mechanisms: (1) Fourier Perturbation Algorithm (FPA_k) [47], 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); (2) d^*-private mechanism (d^*), which extends the d-privacy mechanism from Chan et al. [8] and applies Laplacian noise on time series data; (3) d_{L1}-private mechanism (d_{L1}), which achieves d-privacy with regard to L1 distance. We perform extensive experiment on these three mechanisms to evaluate their security as well as utility. For security, we show that by selecting proper parameters, all these mechanisms can effectively defeat all types of classifiers, *i.e.*, can reduce their classification accuracy to the baseline of random guessing. For utility, we demonstrate that the utility cost, defined as *waste* and *deficit*, is moderate. We also compare the three mechanisms empirically, and we further compare FPA_k with a baseline defense mechanism, which shows 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, which intercepts and modifies `XMLHttpRequest(XHR)` requests and responses. Our evaluation suggests that the extension completely renders the attacks proposed by Schuster et al. [49] ineffective. The techniques proposed in this paper also shed light on defenses against website fingerprinting attacks and generic side-channel attacks that rely on machine learning classifiers. Our study has provided an important piece of evidence, suggesting that the differential privacy is a promising solution to ML-enabled inference attacks.

2 Background

Side-channel attacks and traffic analysis Side-channel attacks usually involve analysis of externally observable characteristics of a computer system to extract sensitive information (*e.g.*, cryptographic keys). Traffic analysis attacks are side-channel attacks that observe the meta-data of the encrypted network traffic to classify the traffic [20,43,60]. 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.

Deep learning In the past, various ML techniques have been employed in statistical side-channel attacks [12,44,64,65]. For example, support vector machines (SVM) have been used to perform website fingerprinting in the Tor network [44] and infer foreground apps on Android [12]; hidden Markov models (HMM) have been used to infer Android Activity transitions [10] and extract cryptographic keys in a cross-VM setting [65]; k nearest neighbors (kNN) have been used to perform keystroke inference on smartwatch [34] and link Bitcoin addresses to an iOS device [64]. Deep Learning [29] is an ML approach that uses multiple layers of nonlinear 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) [2], 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 [17,26,54,56], speech recognition [21,37,48], natural language processing [22,38,53], and malware detection [11]. One of the most popular DNN models is the Convolutional Neural Network (CNN) [30]. CNN typically applies convolutional operation at lower levels and is designed to process data that has a form of multi-dimensional arrays.

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' [55]. 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. Methods of generating adversarial samples include Fast Gradient Sign Method (FGSM) [18], Projected Gradient Descent [35], Carlini/Wagner attack (CW) [7], *etc*.

In response, defenses have been proposed to make classifiers more robust against adversarial samples. To date, the most successful one is adversarial training [18,35,55], 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.

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 Sect. 5. 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* [14] 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 $A : \mathcal{X} \to \mathcal{Z}$ satisfies ϵ-differential privacy if for any adjacent databases x, x' and all $Z \subseteq \mathcal{Z}$,

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

Chatzikokolakis et al. [9] 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 $d : \mathcal{X}^2 \to [0, \infty)$ satisfying $d(x, x) = 0$, $d(x, x') = d(x', x)$, and $d(x, x'') \le d(x, x') + d(x', x'')$ for all $x, x', x'' \in \mathcal{X}$. A randomized algorithm $A : \mathcal{X} \to \mathcal{Z}$ satisfies (d, ϵ)-privacy if for all $Z \subseteq \mathcal{Z}$,

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

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.

MPEG-DASH standard 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. 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 [39], 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.

3 A motivating example

A recent study by Schuster et al. [49] demonstrated that packet burst patterns of the encrypted video streams (an observable side channel that reveals the size of the segments) can be used to fingerprint video streams from providers such as Youtube with very high detection accuracy. 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 their idea [49] 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.

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. 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

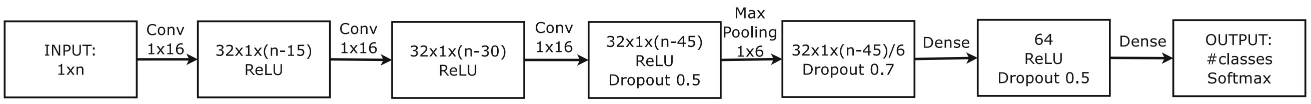


Fig. 1 CNN architecture; n denotes the number of elements of one trace, *i.e.*, the total time divided by the window size

Table 1 Accuracy with one standard deviation

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

process was automated using `Selenium` and Wireshark's `tshark`. All the data were collected from a desktop connected to our campus network using 1 Gbps Ethernet. The whole process of data collection took about 15 days.

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*. 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` [45], and Neural Net and CNN were implemented using `Tensorflow` [1] with the `Keras` 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. [49]. 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 fivefold 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 1, 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 fivefold tests. These experiments validate the attack demonstrated by Schuster et al. [49]. 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.

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

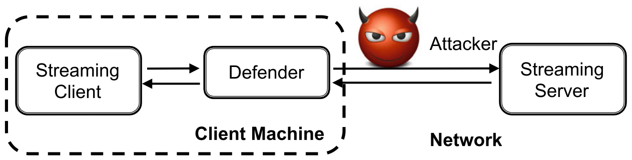


Fig. 2 Threat model

4 Threat model

The attack described in Sect. 3 motivates the following scenario. A user who watches a Youtube video in a web browser (streaming client) 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 service provider (*i.e.*, Youtube) and the browser, which obfuscates the network flows from/to the content provider to defeat the fingerprinting attacks. 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.

Attacker We consider the attacker to be a passive traffic analysis attacker, who can observe the encrypted packet sequences transmitted between the client machine and the streaming server via the network.

Defender The defender intercepts all connections between the client and the server and modifies the requests according to the defense mechanisms, so that the responses sent by the server would follow the pattern dictated by the requests. The attacker can only observe the traffic after the defender's perturbation.

5 Adversarial machine learning

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

5.1 Crafting adversarial samples

To generate adversarial samples, we followed the Projected Gradient Descent (PGD) method [35], which is a multi-step variant of the Fast Gradient Sign Method (FGSM) [18].

FGSM 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:

$$x^* = x + \eta\, \text{sign}(\nabla_x L(g(x;\theta), y)) \quad (1)$$

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.

PGD PGD can be considered as a universal first-order adversary. In each iteration, PGD follows the following rule: $x'_{i+1} = F_{clip}(FGSM(x'_i))$, where $FGSM(x'_i)$ represents the output of FGSM as in (1). F_{clip} keeps x'_{i+1} within a predefined perturbation range.

In our experiment, we used the PGD implementation in the `adversarial-robustness-toolbox` [41] Python library. We adjusted the level of injected noise (dictated by the `eps` parameter, `eps` $\in$ [0, 1]) to generate adversarial samples corresponding to different noise levels. Suppose the original value is v. With `eps` $= \alpha$, 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 PGD, with the noise level `eps` = 0.1 and the max number of iterations (*max_iter*) set to 100. Then, we fed these samples to the CNN classifier trained using clean samples. The CNN classifier was unable to classify such samples, with only 0.003 accuracy, which is significantly lower than the original accuracy (0.944). This result suggests that the adversarial samples are very effective against the CNN classifier.

5.2 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.

Table 2 Transferability test. The numbers are classification accuracy of adversarial examples on different architectures

Test / Base	CNN	SVM	LR	Neural Net
CNN	0.003	0.421	0.344	0.148
SVM	0.048	0.009	0.008	0.028
LR	0.190	0.130	0.263	0.001
Neural net	0.530	0.164	0.088	0.028

Using a different classifier The adversarial samples generated by PGD are designed to fool one particular classifier, which may not be able to deceive other classifiers (may not transfer well). To see whether adversarial samples can transfer, we conduct a transferability study on the classifiers. Since RF does not provide gradients, we test SVM, LR, CNN and Neural Net, and the `eps` is set to 0.1 for PGD. The average results of 5 rounds are shown in Table 2. As shown in the table, the adversarial samples targeted the CNN model do not work well on others; SVM can still achieve 0.570 accuracy. The adversarial samples based on SVM seem to work well on all other classifiers; however, as shown shortly, the attacker could conduct adversarial training to easily circumvent the defense.

Conducting adversarial training If the attacker is aware of the defender's strategy of using adversarial examples, it can adapt to such a setting and regain accuracy by training with those adversarial examples. Similarly, the defender can also generate adversarial examples, knowing that the attacker will try to make the classification robust. Although finding an exact equilibrium is difficult [19], the attacker can practically utilize the adversarial training technique by Madry et al. [35] to make the classification robust.

We conducted the adversarial training on the CNN model for 10 epochs using their method; the `eps` for adversarial training was set to 0.1. Then, we re-generated the adversarial samples targeting the new CNN model. After this process, the new classifier achieved 0.791 accuracy on the new adversarial samples, which indicates that the adversarially trained CNN model is robust against adversarial samples. 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.

6 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 [14] 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. It offers strong privacy promises that guarantees statistical indistinguishability of two databases that are different in only one element.

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 three mechanisms, FPA_k, d^*-privacy and d_{L1}-privacy, to enforce differential privacy on streaming traffic.

6.1 Differentially private mechanisms

Fourier perturbation algorithm (FPA_k). Rastogi et al. [47] 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 $Q = (Q[1], \ldots, Q[n])$ into another length-n complex-valued sequence $F = (F[1], \ldots, F[n])$ 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 DFT(Q). An Inverse DFT (IDFT) is also a linear transform of a complex-valued sequence $P = (P[1], \ldots, P[n])$ to another complex-valued sequence $R = (R[1], \ldots, R[n])$ 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 $F[1], \ldots, F[k]$ after computing DFT(Q).
(b) Compute $\tilde{F}[i] = F[i] + \mathsf{Lap}\,(\lambda)$ for $i = 1, \ldots, k$.
(c) Return $\tilde{Q} = IDFT(PAD^n([\tilde{F}[1], \ldots, \tilde{F}[k]]))$, where $PAD^n([\tilde{F}[1], \ldots, \tilde{F}[k]])$ denotes the sequence of length n obtained by appending $n - k$ zeros to $\tilde{F}[1], \ldots, \tilde{F}[k]$.

Rastogi et al. [47] 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 $Q, Q' \in \mathbb{Q}$, $|Q - Q'|_2 \leq \Delta_2(\mathbb{Q})$.
d^*-private mechanism Xiao et al. [62] 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. [62] extended a mechanism from Chan et al. [8] 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

$$\tilde{x}[i] = \tilde{x}[G(i)] + (x[i] - x[G(i)]) + r_i \tag{2}$$

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

$$G(i) = \begin{cases} 0 & \text{if } \ i = 1 \\ i/2 & \text{if } \ i = D(i) \geq 2 \\ i - D(i) & \text{if } \ i > D(i) \end{cases} \tag{3}$$

$$r_i \sim \begin{cases} \mathsf{Lap}\left(\frac{1}{\epsilon}\right) & \text{if } \ i = D(i) \\ \mathsf{Lap}\left(\frac{\lfloor \log_2 i \rfloor}{\epsilon}\right) & \text{otherwise} \end{cases} \tag{4}$$

It was proven by Xiao et al. [62] that the algorithm in Eqns. 2eqn:recurrence:noise is $(d^*, 2\epsilon)$-private.
d_{L1}-private mechanism L1 distance is a common metric for measuring the similarities between two time series [13,63]. Let x and x' denote two time series. The L1 distance is defined as: $d_{\mathrm{L1}}(x, x') = \sum_i |x[i] - x'[i]|$. To achieve d_{L1}-privacy, the mechanism to add noise is described as follows [62]: The mechanism A computes a noised value $\tilde{x}[i]$ that is used in place of $x[i]$ as

$$\tilde{x}[i] = x[i] + r_i \tag{5}$$

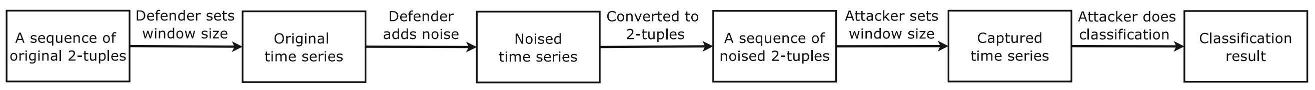

A sequence of original 2-tuples
Defender sets window size
Original time series
Defender adds noise
Noised time series
Converted to 2-tuples
A sequence of noised 2-tuples
Attacker sets window size
Captured time series
Attacker does classification
Classification result

Fig. 3 Abstraction of data flow with defense

where

$$r_i \sim \text{Lap}\left(\frac{1}{\epsilon}\right) \tag{6}$$

The algorithm in Eqns. 5eqn:recurrence-L1:noise is (d_{L1}, ϵ)-private [62].

6.2 Applying privacy mechanisms on streaming data

Without loss of generality, the problem specified in Sect. 4 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, the three differentially private mechanisms, FPA_k, d^* and d_{L1}, 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 $w_A \bmod w = 0$ or $w \bmod w_A = 0$. 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 $\dot{x}$, *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.*,

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

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

7 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 Sect. 3), and returned another time series $\tilde{x}$ of the same size after adding noise by following the steps mentioned in Sect. 6.1. Similarly, in our

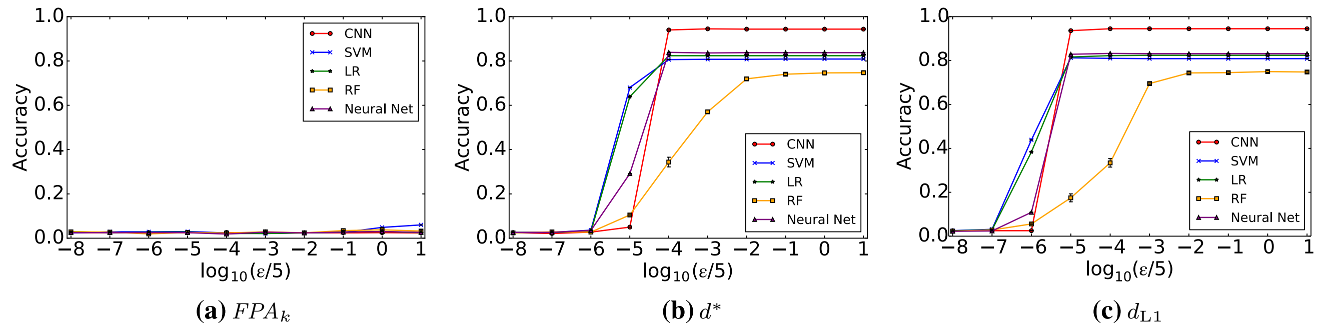


Fig. 4 Classification accuracy of fivefold cross-validation when trained with original traces and tested with noised traces

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 Sect. 3. In our experiment, we used $\epsilon = \{5 \times 10^{-8}, 5 \times 10^{-7}, \ldots, 50\}$, $w = \{0.05s, 0.25s, 0.5s, 1s, 2s\}$, 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.

7.1 Security evaluation

The security of the differentially private mechanisms is evaluated by classification accuracy. We used the same method mentioned in Sect. 3 to preprocess the data and train the classifiers. According to the dataset used for training and testing, we consider the following cases:

7.1.1 Trained with x (clean data), tested with $\tilde{x}$ (noised data)

To compare with the defense mechanism of using adversarial samples described in Sect. 5.2, 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 $w = 0.25$s. The classification accuracy and standard deviation of a fivefold 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. The trend of d_{L1} (Fig. 4c) is very similar to that of d^*, but the settings need to be $\epsilon \leq 5 \times 10^{-7}$ in order to maintain the baseline accuracy for all classifiers, which is one order of magnitude smaller than the settings of d^*. 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.

7.1.2 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 Sect. 3 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 fivefold cross-validation are shown in Fig. 5. The classification accuracy with FPA_k protected data is shown in Fig. 5a. When ϵ was smaller (*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 relationship with the length of the time series. When w was large, with the video length remaining the same, the

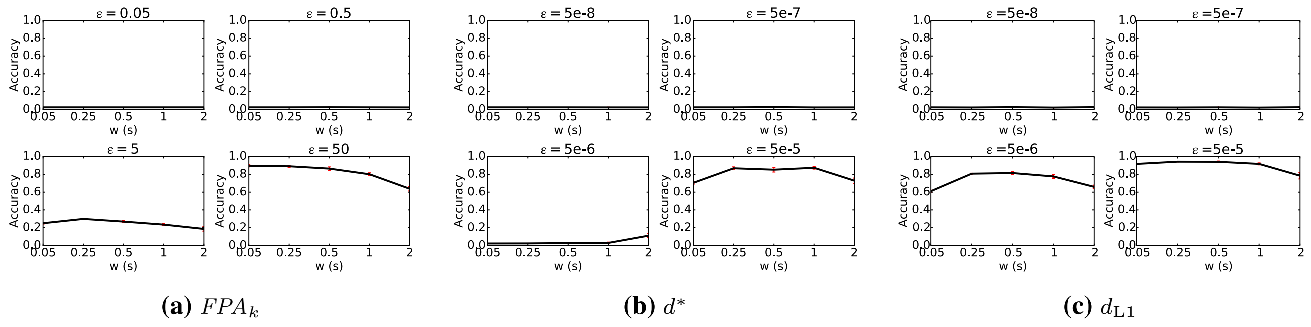


Fig. 5 $w_A = w$: effect of w

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. The classification accuracy with d_{L1} protected data is shown in Fig. 5c. The trend is very similar to that of d^*, the main difference is that to maintain the baseline accuracy, d_{L1} requires $\epsilon \leq 5 \times 10^{-7}$, which is one order of magnitude smaller than the threshold of d^*. The reason is that d^* and d_{L1} have different distance metrics and different noise-adding mechanisms. For all 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, $d^*$and d_{L1} methods 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 $\epsilon \leq 5\times10^{-6}$ and d_{L1} needed $\epsilon \leq 5\times10^{-7}$, while FPA_k only required $\epsilon \leq 0.5$. This is because the definitions of ϵ in the methods are different. We also provide a proof to bridge the ϵ values between traditional differential privacy and d-privacy in Appendix 1.

• $w_A \neq w$. Next, we consider the scenario where the attacker and the defender chose different w. To perform the experiment, first, we set $\epsilon = \{5 \times 10^{-8}, 5 \times 10^{-7}, \ldots, 50\}$, 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

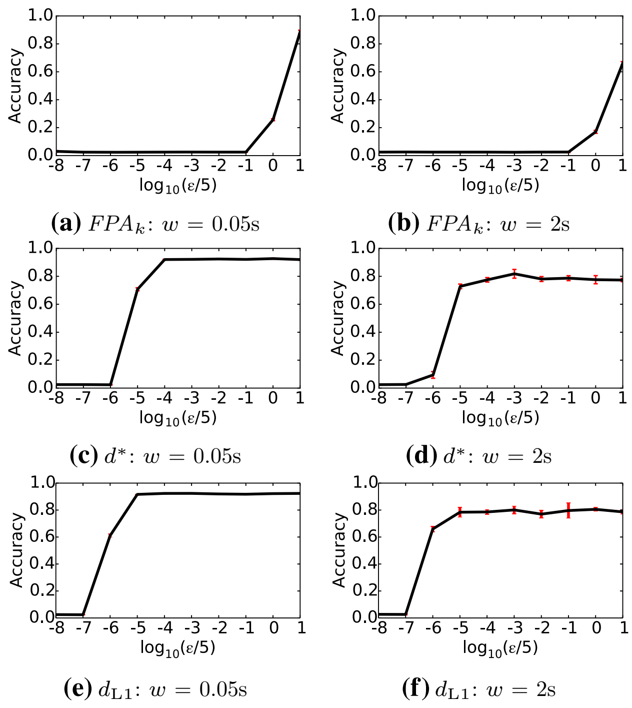
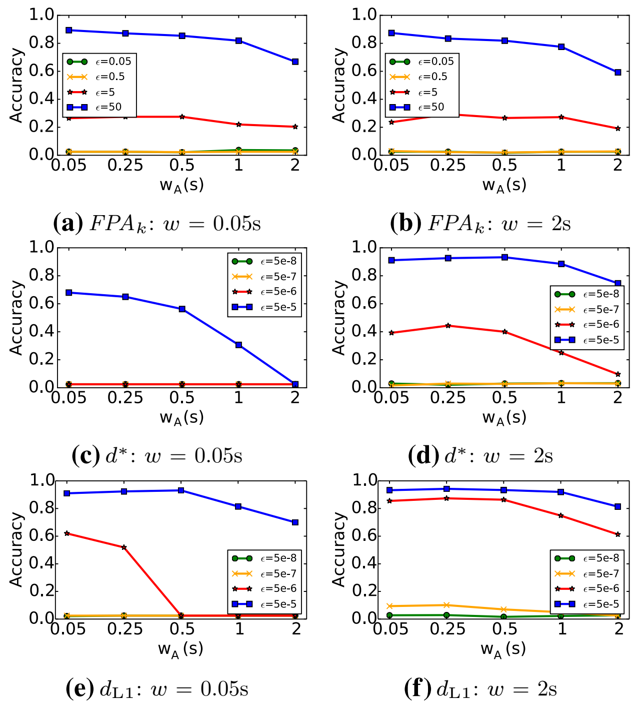


Fig. 6 $w_A = w$: effect of ϵ

$d^*$and d_{L1} 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, b). But for d^*, w mattered: for $w = 0.05s$ (Fig. 7c), $\epsilon = 5\times10^{-6}$ 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 $w_A \leq 0.5s$. Smaller w also made d_{L1} more effective against the classifier, as shown in Fig. 7e, f. Therefore, from the defender's perspective, if the d^* or d_{L1} method is chosen, it is better to choose a smaller w to achieve better privacy.



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

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

(e) d_{L1}: $w = 0.05$s **(f)** d_{L1}: $w = 2$s

Fig. 7 $w_A \neq w$

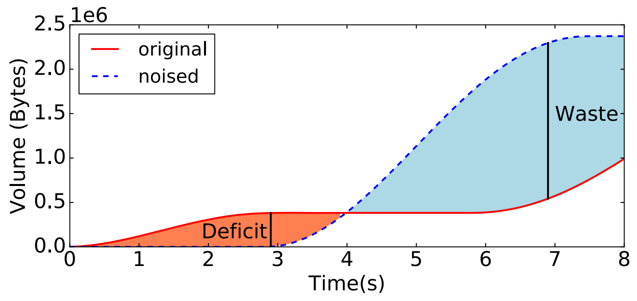


Fig. 8 An example of *waste* and *deficit*

7.2 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 $B = \sum_1^n \tilde{x}$. We define *waste* as the maximum difference between traces A and B when the noised trace B is *above* the original trace A.

$$waste = \max_{1 \le i \le n} \{\max(B[i] - A[i], 0)\} \quad (7)$$

deficit is defined as the maximum difference between A and B when the noised trace B is *below* the original trace A.

$$deficit = \max_{1 \le i \le n} \{\max(A[i] - B[i], 0)\} \quad (8)$$

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 *deficit* is the max difference between the two lines in the *orange* area, while the *waste* is that in the *blue* area.

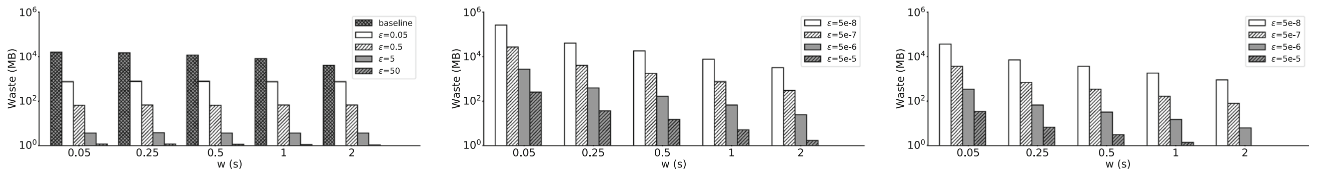
The utility of the three mechanisms was evaluated using the same set of w and ϵ values as in Fig. 5. The *waste* and *deficit* of each noised trace were computed first, and the average *waste* and *deficit* over all traces were calculated and shown in Figs. 9 and 10. According to Fig. 9a, parameter w 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^* and d_{L1}, ϵ was the major factor that affected the *waste* (see Fig. 9b, c). However, w also had an influence: When ϵ was fixed, larger w indicated fewer *waste* for d^* and d_{L1}. We conjecture it was again related to the mechanism by which d^* and d_{L1} 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*.

However, the *deficit* metric of the three mechanisms followed a different trend (Fig. 10). 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. The *deficit* of d_{L1} (Fig. 10c) was quite similar to that of d^*: the *deficit* was negligible when w was small and stayed within 1MB when w was large. Overall, FPA_k cost less *waste* but incurred more *deficit*, while d^* and d_{L1} had fewer *deficit* with higher *waste*.

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*. 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.

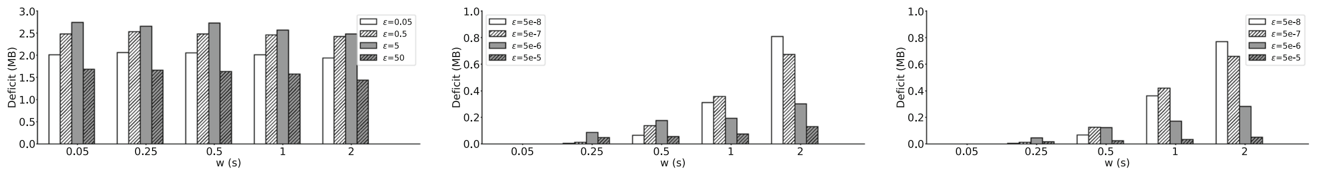
7.3 Comparison of mechanisms

To compare the three differentially private mechanisms, we chose the best (w,ϵ) pair for each method, which achieves the baseline accuracy (*i.e.*, 2.5%) and lowest *waste*. According to the experiment results presented in Sects. 7.1 and 7.2, for FPA_k, the best parameters were w = 2s, ϵ = 0.5; for d^*, w =



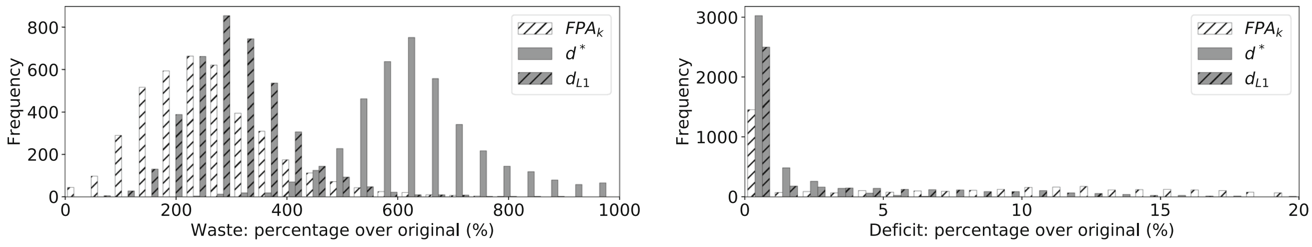
(a) FPA_k **(b)** d^* **(c)** d_{L1}

Fig. 9 *waste* experiment



(a) FPA_k **(b)** d^* **(c)** d_{L1}

Fig. 10 *deficit* experiment



(a) *waste* **(b)** *deficit*

Fig. 11 Utility Comparison: FPA_k (w = 2s, ϵ = 0.5); d^* (w = 0.5s, ϵ = 5e-6); d_{L1} (w = 2s, ϵ = 5e-7)

0.5s, $\epsilon = 5 \times 10^{-6}$ were chosen; for d_{L1}, we used w = 2s, $\epsilon = 5 \times 10^{-7}$. The *waste* and *deficit* distribution of the 4000 traces after applying the three 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%; median *waste* of d_{L1} traces was about 300%; that of d^* traces was even higher (600%). For *deficit* (Fig. 11b), however, more than 80% of d^* traces had a *deficit* less than 1%; most of d_{L1} traces had a *deficit* less than 5%; 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_{L1} cost about 300% *waste* (1.5 times of FPA_k), while d^* had to download 3 times more volume than FPA_k. To achieve differential privacy, some utility loss has to be allowed when applying the three mechanisms. Moreover, it is clear that with the same security level (in regards to classification accuracy), if the primary objective is to 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_{L1} is somewhere in the middle, which is more balanced.

We also studied the *waste* added per window for the three mechanisms when choosing the best parameters. The *waste* added in the ith window is denoted as $\delta[i]$. Following the notions in Sect. 6, we have $\delta[i] = \tilde{x}[i] - x[i]$.

For FPA_k, $\tilde{x}[i] = \tilde{Q}[i]$, so

$$\delta[i] = \tilde{Q}[i] - x[i].$$

For d_{L1},

$$\delta[i] \sim \mathsf{Lap}\left(\frac{1}{\epsilon}\right).$$

For d^*, based on the mechanism presented in Sect. 6.1,

$$\delta[i] = \tilde{x}[i] - x[i] = r_i + r_{G[i]} + r_{G[G[i]]} + \cdots + r_1.$$

The average *waste* added per window of the 4000 traces when choosing the best parameters in each setting were presented in Fig. 12. We find that the *waste* added per window stayed at roughly 0.7MB and 0.9MB for FPA_k and d_{L1}, respectively, while that of d^* was increasing gradually from 0.1MB to 0.8MB. This is consistent with their noise adding mechanisms presented in Sect. 6. Based on this observation, when the length of the video increases, the noise of d^* will accumulate, costing more *waste* than FPA_k and d_{L1}.

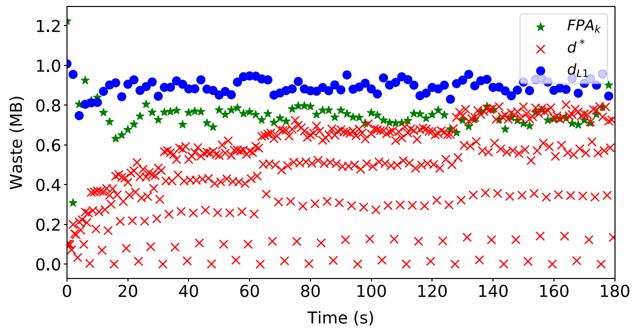


Fig. 12 The *waste* added per window with the best parameters: FPA_k (w = 2s, ϵ = 0.5); d^* (w = 0.5s, ϵ = 5e-6); d_{L1} (w = 2s, ϵ = 5e-7)

7.4 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 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, $d^*$and d_{L1} 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 $\epsilon = 5$).

8 Impacts of video lengths

In the previous sections, the video length (l) was set to 3 minutes (180s); in this section, we further studied how the video length would affect the security. Specifically, we evaluated the security with l = [30s, 60s, 90s, 120s, 150s]. We used the same 40×100 traces. Here, we set $w_A = w = 0.25s$ since w is not the major factor affecting security and utility based on our evaluation in Sect. 7. We used $\epsilon = \{5 \times 10^{-8}, 5 \times 10^{-7}, \ldots, 50\}$. The settings in the FPA_k method remain the same, *i.e.*, $k = 10$. We assume that the attacker uses the noised data ($\tilde{x}$) for training and testing. We used fivefold cross-validation for security evaluation.

8.1 Impact on security

The classification accuracy and one standard deviation of a fivefold cross-validation with different video lengths are shown in Fig. 13.

FPA_k. The result for the FPA_k method is shown in Fig. 13a. When ϵ is small (*e.g.*, $\epsilon = 0.05$ and $\epsilon = 0.5$), the classification accuracy remains low since the noise level is high. However, when ϵ is larger (*e.g.*, $\epsilon = 5$ and $\epsilon = 50$), longer videos yield higher classification accuracy: when $\epsilon = 50$, the accuracy goes down by over 30% when $l = 30s$, compared to $l = 150s$. The reason is that longer videos have more features in the traces, which gives the classifier more information to better separate them.

d^*. The result with d^* as the defense mechanism is shown in Fig. 13b. When $\epsilon \leq 5 \times 10^{-6}$, the video length does not affect the accuracy. When $\epsilon = 5 \times 10^{-5}$, however, the accuracy keeps increasing when l is increasing. The reason is the same as that in FPA_k: longer videos indicate more features, thus easier classification.

d_{L1}. The result of the d_{L1} mechanism is shown in Fig. 13c. As shown in the figure, d_{L1} requires tighter parameters ($\epsilon \leq 5 \times 10^{-7}$) to maintain the baseline accuracy, compared to d^*. When the noise level is low ($\epsilon \geq 5 \times 10^{-6}$), d_{L1} mechanism cannot keep the videos indistinguishable. The accuracy also increases when the video length increases.

8.2 Implications of longer videos

Our evaluation shown in Fig. 13 indicates that longer videos need tighter parameters (ϵ) to maintain the same security level. However, if the video length is longer than the defender expected, what are the implications on the three mechanisms? Note that FPA_k is *not applicable* in this scenario, since the video length must be known before it performs the Fourier Perturbation. Therefore, we focus on d^* and d_{L1} in this subsection.

Methodology To answer this question, we need to find a way to set d^* and d_{L1} in the same security level when the video length is short and evaluate the impact when the length is longer. To achieve this, for each method, we first define $Acc(m_0, w_0, \epsilon_0, l_0)$ as the classification accuracy when $method = m_0, w = w_0, \epsilon = \epsilon_0, l = l_0$, and $thres(m_0, w_0, l_0)$ as the maximum ϵ that can maintain the baseline accuracy when $method = m_0, w = w_0, l = l_0$.

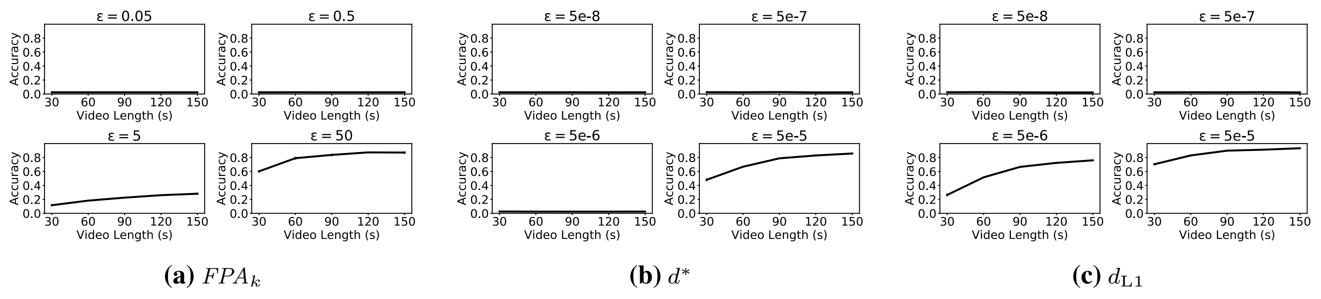


Fig. 13 Classification accuracy with different video lengths

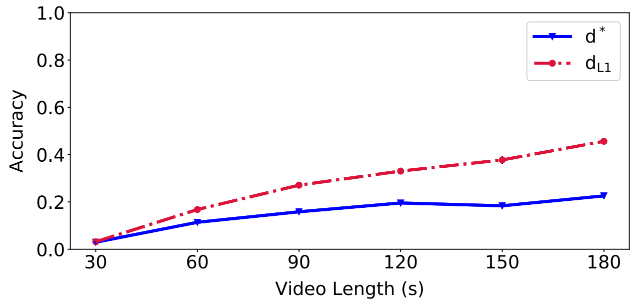


Fig. 14 Classification accuracy when ϵ is set to *thres* (d^*, 0.25, 30) and *thres* (d_{L1}, 0.25, 30) for d^* and d_{L1}, respectively

In our case, since the baseline accuracy is 0.025, for each method, the $thres(m_0, w_0, l_0)$ needs to satisfy the following two conditions:

$$Acc(m_0, w_0, thres, l_0) = 0.025,$$
$$\forall \epsilon \geq thres, Acc(m_0, w_0, \epsilon, l_0) > 0.025.$$

For d^* and d_{L1}, we use the two conditions in *binary search* to find the approximate *thres* when $w = 0.25s, l = 30s$. Suppose the search range for the ith round in the binary search is $[L_i, R_i]$. Since we use Python for evaluation and the floats in Python follow the IEEE 754 *double precision* standard, which contain 53 bits of precision (16 digits)[2], the stopping condition is set as $R_i - L_i \leq$ 1e-16. After the binary search ends (n rounds), we set $thres = R_n$.

Evaluation For d^* and d_{L1}, we use the evaluation result presented in Fig. 13 to set the $[L_0, R_0]$ for the binary search. For d^*, L_0 = 5e-6, R_0 = 5e-5; for d_{L1}, L_0 = 5e-7, R_0 = 5e-6. After the binary search ends, we have the *thres* for the two mechanisms[3]. Then, we set the ϵ to the *thres* for d^* and d_{L1}, respectively, and evaluate the classification accuracy with fivefold cross-validation when $l = [60s, 90s, 120s, 150s, 180s]$. The results are shown in Fig. 14. We can learn that when the video length is longer than 30 seconds, the classification accuracy increases no matter which defense mechanism is chosen. However, the impacts on the two mechanisms are different: for d^*, the accuracy increases to about 18% when $l = 180s$; for d_{L1}, the accuracy goes up to over 40%, which is more than twice as that of d^*. The results indicate that d_{L1} is less effective when the video length is longer than the expected setting, compared to d^*. Therefore, if the video to be protected may have a longer length than the defender expected, it is better to choose d^* over d_{L1}.

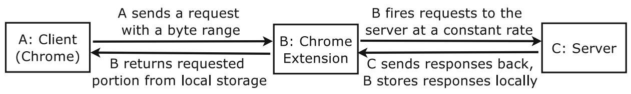

A: Client (Chrome)
A sends a request with a byte range
B returns requested portion from local storage
B: Chrome Extension
B fires requests to the server at a constant rate
C sends responses back, B stores responses locally
C: Server

Fig. 15 Workflow of the `Chrome` extension

9 Real-world implementation

To demonstrate the practicality of our approach, we implemented the FPA_k privacy mechanism in a Chrome extension that proxies Youtube streaming. The workflow of the extension is illustrated in Fig. 15. 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.

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

[2] https://docs.python.org/3/tutorial/floatingpoint.html

[3] *thres*(d^*,0.25,30) = 0.0000111524321020, *thres*(d_{L1},0.25,30) = 0.0000024160161657.

Table 3 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*)

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

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`[4] 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`[5] library, which is similar to Python's `numpy`, to implement numeric computation, and used the `Random` library in `SIM.JS`[6] 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 Sect. 3 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 of bytes per bin (*BPB*), the number of packets per bin (*PPB*), the average packet length per bin (*LPB*), the size of bursts[7] (*BURST*).

In *BPB* series, each element represents the volume in one bin; in *PPB* series, each element is the number of packets collected in one bin; and in *LPB* series, each element represents the average packet length in one bin, *i.e.*, $LPB[i] = BPB[i]/PPB[i]$ if $PPB[i] > 0$. In *BURST* series, each element is the value of burst in one bin. 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*.[8]

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 Sect. 3 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 3. 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 1. This result suggests that differential privacy is effective in defeating machine learning adversaries. According to the Post-processing Lemma [15], 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 3, 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 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 cannot 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.

[4] https://github.com/jpillora/xhook

[5] https://github.com/nicolaspanel/numjs

[6] https://github.com/mvarshney/simjs-source

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

[8] In previous sections, the evaluations were performed on the *BPB* feature; here, we show that extracting more features does not really help the classification.

10 Discussion

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

Reducing waste 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. It would be more reasonable to find an empirical clip bound based on real-world statistics.

Leakage through video length None of the 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-minute to 1-hour could be considered in one group and padded so that all of them appear to be a 1-hour video.

Applying the mechanisms to protect longer videos In our paper, the videos are only 3-minute long, which may not be realistic in reality. While the methods present in the paper can be applied to longer videos, due to the increase in the video length, more noise needs to be added in order to make them indistinguishable. To make it more practical and reduce the utility cost, the user can utilize the approaches to reduce the *waste* as mentioned above, as well as grouping the videos before adding noise. Moreover, in practice, it may not be necessary to maintain a baseline accuracy; lowering the security guarantee to obtain a reasonable balance between security and utility may be a better approach.

Comparing the three mechanisms In Sect. 7.3, we compared the utility of the three 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.

Open-world settings In this paper, we performed our evaluation on a 40-video dataset. While 40 may seem to be a small number, in real attacks, the attacker may utilize some auxiliary information to narrow down the set of videos the victim may be watching. Moreover, the closed-world setting is the *most advantageous* to the attacker and the *least favorable* to the defender. As a result, it would be much harder to build a defense against attackers in the closed-world settings than open-world settings. Our evaluation demonstrates the effectiveness of the defense method in a more challenging scenario. Therefore, the proposed defense should also work in the open-world settings.

Practical deployment To deploy our defense approaches in real-world systems, it is important to make the end users aware of the trade-off between privacy and cost. Better privacy leads to higher *waste* and/or higher *deficit*. If the *deficit* is too high, the video cannot be played smoothly; if the *waste* is too high, the cost of network usage will increase. To lower the *deficit*, the service provider can enforce larger packet sizes of the first few seconds with differential privacy guarantee. Fortunately, all mechanisms offer tunable security parameters to adjust privacy levels. Service providers can configure a set of privacy levels for end users to choose from; the privacy gain and utility loss should be properly explained so that the users can adjust their levels accordingly.

11 Related work

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 [32,33,59]. Researchers have also shown that adding noise to shared resources can be an effective defense [4,25,66]. Particularly relevant to our work is due to Xiao et al. [62], which introduced the d^* algorithm to mitigate storage side channels resulting from `procfs` in Linux, 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.

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. [47] proposed the Fourier Perturbation Algorithm (FPA_k) algorithm to ensure differential privacy for time-series data. Shi et al. [50] 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. [16] presented a framework, FAST, to release real-time aggregate statistics under differential privacy based on filtering and adaptive sampling. Cao et al. [6] proposed two methods to answer a subset of representative sliding window queries with differential privacy. Kellaris et al. [24] introduced ω-event privacy over infinite streams, which protects any event sequence occurring in ω successive timestamps. None of these works considered applying differential privacy to defeat traffic analysis, however.

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 [43,44,51,52,60]. Accordingly, many research projects have explored mechanisms [5,23,44,61] to address this security threat. 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.

Private messaging systems Prior works have applied differential privacy techniques in private messaging systems. One of the first systems is Vuvuzela [58], which is a large-scale private messaging system that protects against both passive and active adversaries with differential privacy guarantee. There are other works that extend Vuvuzela for private messaging systems [27,28,57]. 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.

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. Oh et al. [42] specifically considered methods to interfere with automated person recognition in an image. Marohn et al. [36] similarly explored the effectiveness of an image-obfuscation technique dubbed "thumbnail preserving encryption" against ML classifiers. A recent paper [40] has also explored adversarial ML to defeat traffic analysis attackers. However, they assume that the defender can arbitrarily inject/remove/change packets, which is unrealistic in our case.

Differential privacy and adversarial samples There are works that use differential privacy to increase the robustness of classifiers against adversarial samples [31,46]. These papers have different objectives from ours. Their goals are to improve the robustness of classification; our paper has the opposite goal: using differential privacy to make classification difficult.

12 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 differentially private mechanisms used in the paper offer good security protection with moderate utility loss.

Funding 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

Declarations

Conflicts of interest The authors have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Appendix A: Appendix

Theorem 1 *c $\mathbb{D}$, 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:

$$A : \mathbb{P}(A(x) \in Z) \leq exp(\epsilon_A) \times \mathbb{P}(A(x') \in Z) \tag{A.1}$$

$$B : \mathbb{P}(A(x) \in Z) \leq exp(\epsilon_B \times d^*(x, x')) \times \mathbb{P}(A(x') \in Z) \tag{A.2}$$

For B, we have:

$$\epsilon_B \times d_{min} \leq \epsilon_B \times d^*(x, x') \leq \epsilon_B \times d_{max} \tag{A.3}$$

If B is (d^*, ϵ)-private,

$$\frac{\mathbb{P}(A(x) \in Z)}{\mathbb{P}(A(x') \in Z)} = exp(\epsilon \times d^*(x, x')) \leq exp(\epsilon \times d_{max}) \tag{A.4}$$

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

$$\epsilon' = \frac{\epsilon}{d^*(x, x')} \leq \frac{\epsilon}{d_{min}} \tag{A.5}$$

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

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems (2016). arXiv preprint arXiv:1603.04467
2. Bengio, Y., et al.: Learning deep architectures for ai. Foundations and trends® in Machine Learning (2009)
3. Benhamouda, F., Joye, M., Libert, B.: A new framework for privacy-preserving aggregation of time-series data. ACM Trans. Inf. Syst. Secur. (TISSEC) **18**, 1–21 (2016)
4. Brickell, E., Graunke, G., Neve, M., Seifert, J.P.: Software mitigations to hedge AES against cache-based software side channel vulnerabilities. In: IACR Cryptology ePrint Archive (2006)
5. Cai, X., Nithyanand, R., Wang, T., Johnson, R., Goldberg, I.: A systematic approach to developing and evaluating website fingerprinting defenses. In: 2014 ACM Conference on Computer and Communications Security. ACM (2014)
6. Cao, J., Xiao, Q., Ghinita, G., Li, N., Bertino, E., Tan, K.L.: Efficient and accurate strategies for differentially-private sliding window queries. In: 16th International Conference on Extending Database Technology. ACM (2013)
7. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy. IEEE (2017)
8. Chan, T.H.H., Shi, E., Song, D.: Private and continual release of statistics. ACM Trans. Inf. Syst. Secur. (TISSEC) **14**, 1–24 (2011)
9. Chatzikokolakis, K., Andrés, M.E., Bordenabe, N.E., Palamidessi, C.: Broadening the scope of differential privacy using metrics. In: International Symposium on Privacy Enhancing Technologies Symposium. Springer (2013)
10. Chen, Q.A., Qian, Z., Mao, Z.M.: Peeking into your app without actually seeing it: Ui state inference and novel android attacks. In: USENIX Security Symposium (2014)
11. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE (2013)
12. Diao, W., Liu, X., Li, Z., Zhang, K.: 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)
13. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. VLDB Endowment (2008)
14. Dwork, C.: Differential privacy. In: 33rd International Conference on Automata, Languages and Programming (ICALP) (2006)
15. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Found. Trends Theor. Comput. Sci. **9**, 211–407 (2014)
16. Fan, L., Xiong, L.: An adaptive approach to real-time aggregate monitoring with differential privacy. IEEE Trans. Knowl. Eng. **26**, 2094–2106 (2014)
17. Farabet, C., Couprie, C., Najman, L., LeCun, Y.: Learning hierarchical features for scene labeling. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 1915–1929 (2013)
18. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). arXiv:1412.6572
19. Hamm, J.: Machine vs machine: minimax-optimal defense against adversarial examples (2017). arXiv:1711.04368
20. Hayes, J., Danezis, G.: k-fingerprinting: a robust scalable website fingerprinting technique. In: USENIX Security Symposium (2016)
21. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**, 82–97 (2012)
22. Jean, S., Cho, K., Memisevic, R., Bengio, Y.: On using very large target vocabulary for neural machine translation (2014). arXiv:1412.2007
23. Juarez, M., Imani, M., Perry, M., Diaz, C., Wright, M.: Toward an efficient website fingerprinting defense. In: European Symposium on Research in Computer Security. Springer (2016)
24. Kellaris, G., Papadopoulos, S., Xiao, X., Papadias, D.: Differentially private event sequences over infinite streams. VLDB Endow **7**, 1155–1666 (2014)
25. Keramidas, G., Antonopoulos, A., Serpanos, D.N., Kaxiras, S.: Non deterministic caches: a simple and effective defense against side channel attacks. Design Autom. Embed. Syst. **12**, 221–230 (2008)
26. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)

27. Kwon, A., Corrigan-Gibbs, H., Devadas, S., Ford, B.: Atom: Horizontally scaling strong anonymity. In: 26th Symposium on Operating Systems Principles. ACM (2017)
28. Lazar, D., Zeldovich, N.: Alpenhorn: Bootstrapping secure communication without leaking metadata. In: OSDI (2016)
29. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature (2015)
30. LeCun, Y., Bengio, Y., et al.: Convolutional networks for images, speech, and time series. In: The Handbook of Brain Theory and Neural Networks (1995)
31. Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., Jana, S.: Certified robustness to adversarial examples with differential privacy. In: 2019 IEEE Symposium on Security and Privacy (SP). IEEE (2019)
32. Li, P., Gao, D., Reiter, M.K.: Mitigating access-driven timing channels in clouds using stopwatch. In: 43rd International Conference on Dependable systems and networks. IEEE (2013)
33. Liu, W., Gao, D., Reiter, M.K.: On-demand time blurring to support side-channel defense. In: European Symposium on Research in Computer Security. Springer (2017)
34. Liu, X., Zhou, Z., Diao, W., Li, Z., Zhang, K.: When good becomes evil: Keystroke inference with smartwatch. In: 22nd ACM Conference on Computer and Communications Security. ACM (2015)
35. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks (2017). arXiv:1706.06083
36. Marohn, B., Wright, C.V., Feng, W.C., Rosulek, M., Bobba, R.B.: Approximate thumbnail preserving encryption. In: 1st International Workshop on Multimedia Privacy and Security (2017)
37. Mikolov, T., Deoras, A., Povey, D., Burget, L., Černockỳ, J.: Strategies for training large scale neural network language models. In: 2011 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU). IEEE (2011)
38. Mnih, A., Teh, Y.W.: A fast and simple algorithm for training neural probabilistic language models (2012). arXiv:1206.6426
39. Mondal, A., Sengupta, S., Reddy, B.R., Koundinya, M., Govindarajan, C., De, P., Ganguly, N., Chakraborty, S.: Candid with youtube: Adaptive streaming behavior and implications on data consumption. In: 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). ACM (2017)
40. Nasr, M., Bahramali, A., Houmansadr, A.: Blind adversarial network perturbations (2020). arXiv:2002.06495
41. Nicolae, M.I., Sinn, M., Tran, M.N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., Edwards, B.: Adversarial robustness toolbox v1.2.0. CoRR (2018). arXiv:1807.01069
42. Oh, S.J., Fritz, M., Schiele, B.: Adversarial image perturbation for privacy protection—a game theory perspective. In: IEEE International Conference on Computer Vision (2017)
43. Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., Wehrle, K.: Website fingerprinting at internet scale. In: NDSS (2016)
44. Panchenko, A., Niessen, L., Zinnen, A., Engel, T.: Website fingerprinting in onion routing based anonymization networks. In: 10th Annual ACM Workshop on Privacy in the Electronic Society. ACM (2011)
45. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. (2011)
46. Pinot, R., Yger, F., Gouy-Pailler, C., Atif, J.: A unified view on differential privacy and robustness to adversarial examples (2019). arXiv:1906.07982
47. Rastogi, V., Nath, S.: Differentially private aggregation of distributed time-series with transformation and encryption. In: 2010 ACM SIGMOD International Conference on Management of data. ACM (2010)
48. Sainath, T.N., Mohamed, A.R., Kingsbury, B., Ramabhadran, B.: Deep convolutional neural networks for LVCSR. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE (2013)
49. Schuster, R., Shmatikov, V., Tromer, E.: Beauty and the burst: Remote identification of encrypted video streams. In: USENIX Security Symposium (2017)
50. Shi, E., Chan, H., Rieffel, E., Chow, R., Song, D.: Privacy-preserving aggregation of time-series data. In: NDSS (2011)
51. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep fingerprinting: Undermining website fingerprinting defenses with deep learning (2018). arXiv:1801.02265 (2018)
52. Sun, Q., Simon, D.R., Wang, Y.M., Russell, W., Padmanabhan, V.N., Qiu, L.: Statistical identification of encrypted web browsing traffic. In: IEEE Symposium on Security and Privacy. IEEE (2002)
53. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems (2014)
54. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., et al.: Going deeper with convolutions. In: CVPR (2015)
55. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). arXiv:1312.6199
56. Tompson, J.J., Jain, A., LeCun, Y., Bregler, C.: Joint training of a convolutional network and a graphical model for human pose estimation. In: Advances in Neural Information Processing Systems (2014)
57. Tyagi, N., Gilad, Y., Leung, D., Zaharia, M., Zeldovich, N.: Stadium: A distributed metadata-private messaging system. In: 26th Symposium on Operating Systems Principles. ACM (2017)
58. Van Den Hooff, J., Lazar, D., Zaharia, M., Zeldovich, N.: Vuvuzela: Scalable private messaging resistant to traffic analysis. In: 25th Symposium on Operating Systems Principles. ACM (2015)
59. Vattikonda, B.C., Das, S., Shacham, H.: Eliminating fine grained timers in xen. In: 3rd ACM Workshop on Cloud Computing Security Workshop. ACM (2011)
60. Wang, T., Cai, X., Nithyanand, R., Johnson, R., Goldberg, I.: Effective attacks and provable defenses for website fingerprinting. In: USENIX Security Symposium (2014)
61. Wang, T., Goldberg, I.: Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In: USENIX Security Symposium (2017)
62. Xiao, Q., Reiter, M.K., Zhang, Y.: Mitigating storage side channels using statistical privacy mechanisms. In: 22nd ACM Conference on Computer and Communications Security. ACM (2015)
63. Yi, B.K., Faloutsos, C.: Fast time sequence indexing for arbitrary LP norms (2000)
64. Zhang, X., Wang, X., Bai, X., Zhang, Y., Wang, X.: Os-level side channels without PROCFS: Exploring cross-app information leakage on IOS. In: NDSS (2018)
65. Zhang, Y., Juels, A., Reiter, M.K., Ristenpart, T.: Cross-VM side channels and their use to extract private keys. In: 19th ACM conference on Computer and communications security. ACM (2012)
66. Zhang, Y., Reiter, M.K.: Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud. In: 2013 ACM conference on Computer and communications security. ACM (2013)