SmartSwitch: Efficient Traffic Obfuscation Against Stream Fingerprinting

Haipeng Li[1], Ben Niu[2], and Boyang Wang[1(✉)]

[1] Department of EECS, University of Cincinnati, Cincinnati, USA
li2hp@mail.uc.edu , boyang.wang@uc.edu
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
niuben@iie.ac.cn

Abstract. In stream fingerprinting, an attacker can compromise user privacy by leveraging side-channel information (e.g., packet size) of encrypted traffic in streaming services. By taking advantages of machine learning, especially neural networks, an adversary can reveal which YouTube video a victim watches with extremely high accuracy. While effective defense methods have been proposed, extremely high bandwidth overheads are needed. In other words, building an effective defense with low overheads remains unknown. In this paper, we propose a new defense mechanism, referred to as *SmartSwitch*, to address this open problem. Our defense intelligently switches the noise level on different packets such that the defense remains effective but minimizes overheads. Specifically, our method produces higher noises to obfuscate the sizes of more significant packets. To identify which packets are more significant, we formulate it as a feature selection problem and investigate several feature selection methods over high-dimensional data. Our experimental results derived from a large-scale dataset demonstrate that our proposed defense is highly effective against stream fingerprinting built upon Convolutional Neural Networks. Specifically, an adversary can infer which YouTube video a user watches with only 1% accuracy (same as random guess) even if the adversary retrains neural networks with obfuscated traffic. Compared to the state-of-the-art defense, our mechanism can save nearly 40% of bandwidth overheads.

Keywords: Encrypted traffic analysis · Machine learning · Feature selection

1 Introduction

Millions of Internet users steam videos from service providers, such as YouTube, Amazon Prime, Netflix, Hulu, etc., on a daily basis. As streaming videos has become a routine to Internet users, privacy of streaming services has become one of the primary concerns to both service providers and customers. For example, all the network traffic between a service provider and a user are encrypted to prevent eavesdroppers from learning the content of video streams.

N. Park et al. (Eds.): SecureComm 2020, LNICST 335, pp. 1–21, 2020.
https://doi.org/10.1007/978-3-030-63086-7_15

However, recent research studies [5,11,18,20,21,28] have shown that steaming services are vulnerable under encrypted traffic analysis. An attacker can compromise user privacy due to the use of adaptive bitrate streaming technique (e.g., Dynamic Adaptive Streaming over HTTP), which is the key for service providers to offer high quality streaming. Specifically, the side-channel information of encrypted traffic (e.g., packet size) has a *strong correlation* with the content of a video. As a result, an attacker, who can eavesdrop a user's network traffic, can infer which video a user watches based on the side-channel information of encrypted traffic and achieve extremely high accuracy with machine learning. For example, an attacker can achieve 99% accuracy by leveraging Convolutional Neural Networks [21,28]. Revealing users' sensitive information and interests through streaming fingerprinting can lead to unintended disclosure and can be leveraged by other cyberattacks, such as email phishing [16] and targeted advertising [25], which will cause more severe damages to individuals.

Traffic obfuscation, which adds noise (i.e., dummy data) to preserve real packet size, is one of the primary approaches to mitigate privacy leakage against encrypted traffic analysis [7,17,28]. However, it is often challenging for traffic obfuscation to be both efficient and effective in defense. Namely, producing small noise would introduce low bandwidth overheads but is often not effective. On the other hand, significant amounts of noises can effectively preserve user privacy but could easily introduce high bandwidth. For instance, Zhang et al. [28] proposed an effective defense to obfuscate packet sizes. Unfortunately, their method produces over 600% bandwidth overheads. The extremely high overhead is an enormous burden to both service providers and users, and impedes the implementation of traffic obfuscation in streaming services.

In this paper, we develop a new defense mechanism, referred to as SmartSwitch. The main idea of our proposed mechanism is to *smartly switch* the noise level in traffic obfuscation. More specifically, our proposed method obfuscates side-channel information of each packet (i.e., the size of each packet) differently, where more significant packets are obfuscated with higher noises while others are obfuscated with lower noises. The reason that switching the noise level can optimize efficiency while remain effective in defense is because *not every packet leaks privacy equally*. In other words, adding higher noises to more significant packets can maintain efficacy in defense and applying lower noises on less significant packets can save bandwidth overheads.

To identify which packets are more critical and need to be protected with higher noises, we formulate this problem as a feature selection problem ([3,4,15,27]), where packet sizes are considered as features. However, addressing feature selection in the context of encrypted traffic is not trivial as the number of dimensions (i.e., the number of packets) is over hundreds or even thousands, which faces the *curse of dimensionality* [3,4,15,27]. In this study, we investigate and customize several feature selection methods to examine their different trade-offs in the context of encrypted traffic. The main contributions of this paper are summarized below:

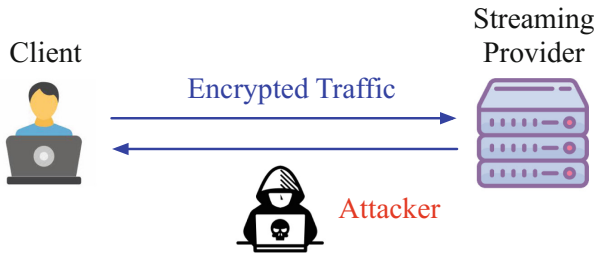


Fig. 1. The system model.

- To advance the understanding of privacy leakage in stream fingerprinting, we collect a large-scale encrypted traffic dataset of YouTube videos. The dataset consists of 100 classes (i.e., 100 YouTube videos) with 200 traffic traces per classes. Our dataset has a greater number of classes and a greater number of traces per class compared to datasets examined in previous studies [21,28].
- We build a Convolutional Neural Network (CNN) for stream fingerprinting. Our CNN is more comprehensive than the CNN used in previous studies, and outperforms it in attack accuracy. Specifically, our CNN achieves 91.4% accuracy while the previous one achieves 80.1% accuracy over our dataset.
- We investigate a fundamental research problem—*which packets are more critical against encrypted traffic analysis?*—and leverage feature selection to address it. We leverage and customize several feature selection methods, including feature permutation importance and mutual-information-based algorithms, to evaluate which packets are more critical than others and should be protected with higher noises.
- While our mechanism is generic, we leverage d^*-privacy used in [26,28] as the underlying noise generation algorithm to demonstrate its efficacy and efficiency. Specifically, even an attacker retrains neural networks with obfuscated traffic, our mechanism reduces attack accuracy to around 1%, which is the same as random guess over 100 classes. Compared to the previous research [28], our mechanism can reduce nearly 40% of bandwidth overhead.
- Our study promotes the *interpretability* of encrypted traffic analysis as many of the recent attacks [12,19,21–23,28] utilize neural networks as *black boxes* and do not reason which parts of an encrypted traffic trace leak more privacy. Our study also sheds lights on optimizing overheads of defenses against other encrypted traffic analysis, such as website fingerprinting.

2 Background

System Model. In the system model, which is described in Fig. 1, we assume there is a client and a streaming service provider. The network traffic between the client and service provider is encrypted with AES (Advanced Encryption Standard) with TLS (Transport Layer Security) protocol. The streaming service provider utilizes Dynamic Adaptive Streaming over HTTP (MPEG-DASH)

technique in order to delivery high quality streaming services. MPEG-DASH creates different sizes of segments from a video and the service provider sends those segments to the client through TLS [21]. We assume the client watches a single YouTube video each time.

Threat Model. In this study, we assume an attacker is an eavesdropper who has *passive on-path access* to the network traffic between a client and the service provider [9]. For instance, an adversary could be someone who can sniff a client's WiFi traffic or on the same local-area network as the client. We also assume an attacker knows the IP addresses of the client and the service provider. The network packets between the client and the service provider are encrypted. The attacker dose not have the secret key to decrypt the content of traffic packets.

We denote the packets sent by the client as outgoing packets and packets received by the client as incoming packets. We assume that the attacker knows the start time of each traffic trace. Note that this is feasible to learn in the context of streaming traffic as outgoing packets usually locate at the beginning of each trace to initiate the connection and a significant amount of incoming packets would be received by the client when a video starts to play.

Closed-World Setting. We measure stream fingerprinting attacks and defenses in the *closed-world setting* by following the literature in this line of research. Specifically, the closed-world setting assumes that an attacker knows a list of stream videos that a client could watch (e.g., popular videos on YouTube). In addition, the closed-world setting assumes a traffic trace that is captured from a victim is associated with one of the stream videos in the attacker's list.

The attacker collects labeled traffic traces by itself to train its machine learning models, and then infers the label (i.e., which video) of a captured traffic trace from a victim. We leverage the *accuracy* of the classification to measure the privacy leakage of stream fingerprinting. A higher accuracy indicates more privacy leakage in encrypted stream traffic.

3 Stream Fingerprinting Attack

To advance our understanding of privacy leakage in stream fingerprinting, we collected a dataset in a greater scale and built a more comprehensive neural network compared to previous studies.

Data Collection. We collected a large-scale dataset of encrypted traffic traces from YouTube videos. The dataset consists of 20,000 traffic traces in total. Specifically, we selected 100 videos (classes), played each selected video 200 times and captured a traffic trace each time. For the 100 videos we investigate in this study, we selected videos recommended by YouTube in five categories, including gaming, music, talk, news and sports. In each category, we selected 20 different videos. For each video, we collected the encrypted traffic for the first 3 min and

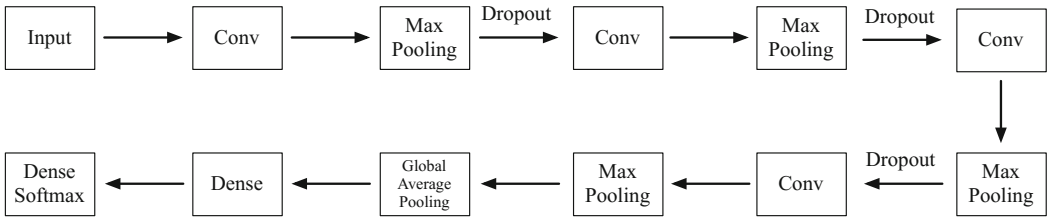


Fig. 2. Structure of our CNN model.

discarded the rest if a video lasts longer than 3 min. If there are advertisements at the beginning of each video, we keep the corresponding traffic of advertisements in our data collection.

We implemented a traffic crawler in `Python` and ran it on a Linux machine (Ubuntu 18.04, 3.40 GHz CPU, 16 GB Memory) to collect traffic. In our traffic crawler, we leveraged `Selenium` to automatically open a YouTube video page and utilized `pyshark` to capture the corresponding traffic. We used `Chrome` (Version 80.0) as the web browser during our data collection. It took one month of efforts (from September 2019 to October 2019) to complete the data collection.

Compared to the largest YouTube encrypted traffic dataset in the existing literature [28], where this dataset has only 40 classes and 100 traces per class, our dataset outperforms it in both the number of classes and the number of traces per class. The increases in those two aspects are critical to advance the understanding of the privacy leakage under stream fingerprinting in the real world.

Data Format. Raw traffic traces in our dataset are further processed to extract side-channel information of traffic, such that they can be used as inputs for neural networks. Existing studies [21,28] in stream fingerprinting aggregate packets into *bins* and use the size of aggregated traffic in a bin as a feature, where each bin contains all the packets in a fixed interval.

We follow the same method to extract bins from raw traffic traces. For example, given a 180-second traffic trace and an interval size of $w = 1$ s, a traffic trace is transformed into a vector of 180 elements, where each element is the size of aggregated traffic in a bin. When interval size $w = 0$, it indicates that there is no bins anymore and we use the size of each packet as a feature in that case. As the majority of packets (over 99% packets) are incoming packets in each trace, *we only keep incoming packets by following previous studies.*

Convolutional Neural Networks. We built a new Convolutional Neural Network and leveraged it as the classifier in stream fingerprinting. As shown in Fig. 2, our CNN consists of 11 layers, including 1 input layer, 4 convolutional layers, 5 pooling layers, and 1 output layer. It is more comprehensive than the CNN used in previous studies. For instance, the CNN used in [21,28] has only 6 layers at most.

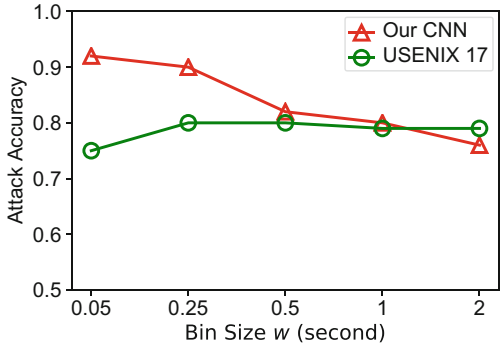


Fig. 3. Comparison of Our CNN and USENIX17 CNN in attack accuracy.

Attack Results in Stream Fingerprinting. Similar as [28], we studied the attack results of stream fingerprinting over a dataset by considering different bin sizes. Specifically, we applied five different bin sizes, $w = \{0.05, 0.25, 0.5, 1, 2\}$ seconds on our YouTube dataset, and obtained 5 versions of the dataset, where each version maps to a bin size. The number of elements n in a vector in each version depends on the corresponding bin size w and can be computed as $n = 180/w$.

To implement our CNN, we used `Keras` (front end) and `Tensorflow` (back end). We trained our CNN on a Linux machine with Ubuntu 18.04, 2.80 GHz CPU, 16 GB Memory and a GPU (NVIDIA Titan RTX). We used 64% of data for training, 16% of data for validation, and 20% of data for test. We also performed 5-fold cross-validation in our evaluation. In addition, we used `BatchNormalization()` function from `Keras` to normalize the data. We tuned hyperparamters of our CNN using NNI (Neural Network Intelligence) [1], a free toolkit offered by Microsoft. Specifically, we ran at most 50 epochs with NNI or stopped the search if the accuracy did not further improve after 10 consecutive epochs. It took us about 6 h to tune hyperparameters for bin size $w = 0.05$. These tuned hyperparameters are described in Appendix. If a different bin size was used, we retuned hyperparameters in our experiments. The hyperparameters for other bin sizes are skipped in this paper due to space limitations.

The attack results of our CNN on test data are summarized in Fig. 3. Overall, the attack achieves high accuracy and successfully infers user privacy across different bin sizes. For instance, given $w = 0.05$ s, our CNN achieves 91.4% accuracy in stream fingerprinting compared to 1% of random guess over 100 classes. We also noticed that the accuracy decreases if we increase the bin size. This is reasonable as a greater bin size leads to more coarse side-channel information, which reveals less privacy over encrypted traffic analysis.

To compare with previous attacks, we also implemented the CNN used in [21] and [28], which we referred to as USENIX17 CNN in this paper. As the tuned hyperparameters rendered in [21] only achieved 1% of accuracy over our YouTube dataset, we retuned the parameters of USENIX17 CNN again based on our YouTube dataset, and the attack results are reported in Fig. 3. As we can

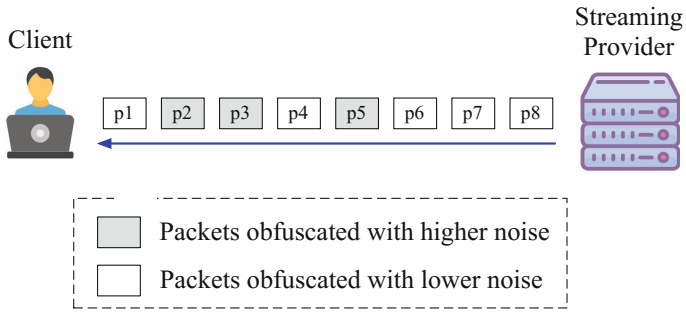


Fig. 4. A high-level overview of SmartSwitch.

MarkFeature(F, D): Given a set of features $F = \{f_1, ..., f_n\}$ over a traffic dataset D, where D has m traffic traces and each trace has n elements, run and output a binary vector $\boldsymbol{b}$

$$\boldsymbol{b} = (b_1, ..., b_n) \leftarrow \mathsf{FeatureSelectAlgo}(F)$$

where $b_i = 1$ if f_i is a selected feature and $b_i = 0$ otherwise.

ObfuTraffic$(\boldsymbol{a}, \boldsymbol{b})$: Given a traffic trace $\boldsymbol{a} = \{a_1, ..., a_n\}$ and a binary vector $\boldsymbol{b} = \{b_1, ..., b_n\}$, generate two noise parameters e_0 and e_1, where e_0 represents for a lower noise level and e_1 represents for a high noise level, for $1 \leq i \leq n$, run

$$a'_i = \begin{cases} a_i + \mathsf{NoiseGeneAlgo}(e_0) \text{ if } b_i = 0 \\ a_i + \mathsf{NoiseGeneAlgo}(e_1) \text{ if } b_i = 1 \end{cases}$$

and output an obfuscated traffic trace $\boldsymbol{a'} = (a'_1, ..., a'_n)$.

Fig. 5. The Details of SmartSwitch.

observe, our CNN outperforms USENIX17 CNN in almost each bin size except $w = 2\,\text{s}$. For instance, given bin size $w = 0.05\,\text{s}$, the attack accuracy of our CNN is 11.4% higher than the one derived by USENIX17 CNN.

We would like to point out that the USENIX17 CNN achieved 94.4% accuracy on the dataset of 40 classes in [28] and 99% accuracy on the dataset of 20 classes in [21]. The accuracy of USENIX17 CNN dropped over our dataset as our evaluation involves more classes. For a machine learning problem, it is common to see that the accuracy of a same model decreases when the number of classes increases.

4 SmartSwitch: Our Proposed Defense Mechanism

We present our defense mechanism in this section. It consists of two building blocks, including a feature selection method and a noise generation algorithm. A feature selection method decides which packets are more significant, and can

be one of the methods we will discuss in the next section. A noise generation algorithm produces noises to obfuscate the size of each packet using privacy-preserving techniques, such as differential privacy [26,28] or padding [6,7].

SmartSwitch first runs a feature selection method to distinguish packets, where more significant packets are marked as 1s and others are marked as 0s. Then, to generate an obfuscated traffic trace, the noise generation algorithm will apply a higher noise level to packets marked with 1s and add a lower noise level to packets marked with 0s. The high-level idea of SmartSwitch is illustrated in Fig. 4. The more rigorous technical details are summarized in Fig. 5. We use MarkFeature to describe the feature selection step and use ObfuTraffic to illustrate the generation of an obfuscated trace. We denote the two underlying building blocks as FeatureSelectAlgo and NoiseGeneAlgo.

Discussion. SmartSwitch is a *generic* defense mechanism, which can be integrated with concrete feature selection methods and noise generation algorithms. It is also feasible to apply it to defend against other fingerprinting attacks over encrypted traffic, such as website fingerprinting [6,10,13,14,19,22].

5 Which Packets Are More Significant?

In this section, we investigate a fundamental research problem—*which packets are more significant against encrypted traffic analysis?* Specifically, we consider the side-channel information of each bin as a feature, and formulate the question above as a feature selection problem. However, evaluating important features over encrypted stream traffic is not trivial as the number of features could be more than hundreds or even thousands. For instance, if the bin size is $w = 0.05\,\mathrm{s}$, then our dataset has $3,600$ dimensions, which is challenging to identify critical features from others.

As how to effectively and accurately select features over high dimensional data remains open in the current literature, we explored and customized two approaches, including permutation feature importance and mutual-information-based algorithms, in the context of encrypted traffic analysis.

5.1 Permutation Feature Importance

Feature selection methods can be grouped into two categories, wrapper methods and filter methods. A wrapper method modifies data associated with one feature each time and evaluates the corresponding change of accuracy of a trained classifier. A greater change in accuracy suggests a feature is more important.

Permutation Feature Importance (PFI) is one of the most common wrapper methods. It modifies the input to a classifier by permuting data in one feature each time. However, *directly permuting data in one feature each time in our problem is not effective as the number of dimensions in our dataset is high.*

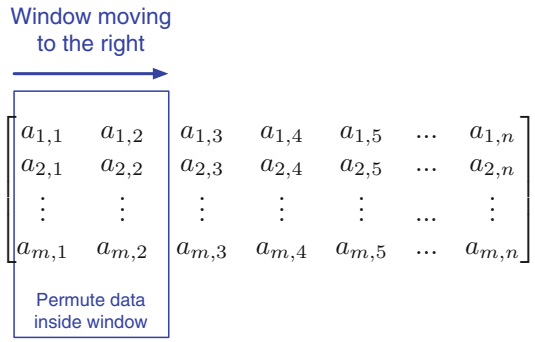


Fig. 6. An example of Sliding Window PFI, where a window include 2 features. In this figure, Sliding Window PFI permutes data inside the window to evaluate the importance of the 2 features.

Specifically, in our preliminary experiments, we did not notice changes of accuracy on a classifier and failed to evaluate the importance of a feature when we only permuted data related to one feature each time.

To overcome this limitation, we customize the algorithm of PFI by permuting data within a *sliding window* each time, where a sliding window consists of multiple consecutive features. In addition, the permutation within each sliding window is performed both vertically (i.e., across traces) and horizontally (i.e., across features) to create differences in data in order to examine the potential change of accuracy of a classifier. We refer our method as Sliding Window Permutation Feature Importance (Sliding Window PFI). An example of Sliding Window PFI is illustrated in Fig. 6.

Details of Sliding Window PFI. Given a dataset with m traces and n features, which can be represented as a $m \times n$ matrix, Sliding Window PFI initiates from the very left and takes a $m \times p$ submatrix, where p is the width of the sliding window and $p < n$. Within the sliding window, Sliding Window PFI first performs row-permutation to shuffle data across traces and then operates column-permutation to shuffle data across features. The rest of the matrix outside the current window keeps unchanged. The entire dataset after this permutation is fed into the trained CNN to measure the change in terms of accuracy in classification. This accuracy change is recorded to indicate the importance of the p consecutive features within the sliding window. A greater change on accuracy indicates the consecutive features within the sliding window is more significant. Our method keeps the size of the window the same but strides the sliding window to the right with one feature to measure the importance of the p features inside the sliding window for the next iteration. Our method iterates until it records the accuracy change of the last p features on the right in the matrix.

Tradeoff of Sliding Window PFI. Sliding Window PFI can effectively evaluate the changes of a classifier compared to the original PFI. As a necessary tradeoff, the significant features selected by Sliding Window PFI are groups of *consecutive* features, where each group of consecutive features is derived from one sliding window.

5.2 Mutual-Information-Based Algorithms

Unlike wrapper methods, filter methods analyze features based on statistic information. Therefore, filter methods are independent of machine learning classifiers and they are more efficient and generic. Mutual Information (MI) is a primary metric to measure the importance of features in filter methods. Given two random variables F, C, the mutual information of these two random variables can be calculated as below:

$$I(F;C) = H(C) - H(F|C) \tag{1}$$

where $H(F)$ is the entropy of random variable F and $H(F|C)$ is the conditional entropy for F given C. Mutual information of two random variables is greater than zero, and a higher mutual information indicates two random variables are more dependent.

In the context of feature selection, F is a random variable over data in a feature (or a subset of features) and C is a random variable over all the classes. A greater mutual information between F and C suggests that the feature (or the subset of features) is more important to classify the data. Given the number of features k that a method would like to select, the goal of this method is to maximize the mutual information of random variable F based on a subset of k selected features and target variable C [2].

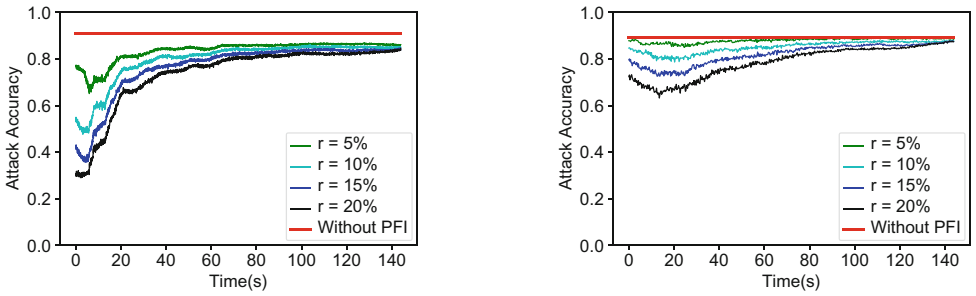
As feature selection over high dimensional data using mutual information is a NP-hard problem [12], we leveraged three greedy algorithms, including Max-Relevance [2,15], Minimal-Redundancy-Maximal-Relevance [15] and Joint Mutual Information Maximisation [3,27], to address feature selection over encrypted stream traffic. The main idea of each greedy algorithm is briefly discussed below. More details can be found in the references.

Max-Relevance (MR). MR [2,15] evaluates the *relevance* between features and classes. It first calculates the MI score between each single feature and classes. Then, the features with top-k highest MI scores will be selected.

Given a set of features $F = \{f_1, ..., f_n\}$, the MI score of feature f_i and random variable of classes C can be computed as $I(f_i, C)$. MR selects a set of features $S = \{f_{s_1}, ..., f_{s_k}\}$, where $S \subset F$ and $f_{s_j} \in F$ for $1 \leq j \leq k$, such that

$$\arg\max_{S} D(S, C) \tag{2}$$

where $D(S, C) = \frac{1}{|S|} \sum_{f_{s_j} \in S} I(f_{s_j}; C)$.



(a) Bin size $w = 0.05s$ (b) Bin size $w = 0.25s$

Fig. 7. The results of Sliding Window PFI. The x-axis is the time difference compared to the starting point of traffic traces.

Minimal Redundancy Maximal Relevance (mRMR). In addition to measuring the relevance between features and classes as in MR, mRMR [15] also measures the *redundancy* among features. When two features highly depend on each other, classification accuracy would not change significantly if only one of them is selected. Therefore, the other feature is considered as redundant and can be removed from the result of feature selection.

Specifically, given a set of features $F = \{f_1, ..., f_n\}$, the MI score of feature f_i and random variable of classes C is denoted as $I(f_i, C)$. mRMR selects a set of features $S = \{f_{s_1}, ..., f_{s_k}\}$, where $S \subset F$ and $f_{s_j} \in F$ for $1 \leq j \leq k$, such that

$$\arg\max_{S} D(S, C) - R(S), \tag{3}$$

where redundancy $R(S) = \frac{1}{|S|^2} \sum_{f_{s_i}, f_{s_j} \in S} I(f_{s_i}; f_{s_j})$. mRMR uses a greedy selection, which iterates each feature to find these top-k features.

Joint Mutual Information Maximisation (JMIM). JMIM [3,27] utilizes joint mutual information score to examine redundancy. Specifically, given two features f_i, f_j and random variable of classes C, the joint mutual information can be evaluated as

$$I(f_i, f_j; C) = I(f_i; C|f_j) + I(f_j; C) \tag{4}$$

Given a set of features $F = \{f_1, ..., f_n\}$ and an empty set $S = \varnothing$, JMIM first selects the feature with the maximum mutual information and adds it to set S. Then, JMIM adds one feature to set S in each iteration. Specifically, an unselected feature $f_u \in \{F - S\}$ is selected in an iteration such that it maximizes the minimum joint mutual information of itself, any selected feature $f_s \in S$ and random variable C, which can be formulated as below:

$$\arg\max_{f_u \in \{F-S\}} (\min_{f_s \forall S}(I(f_u, f_s; C))) \tag{5}$$

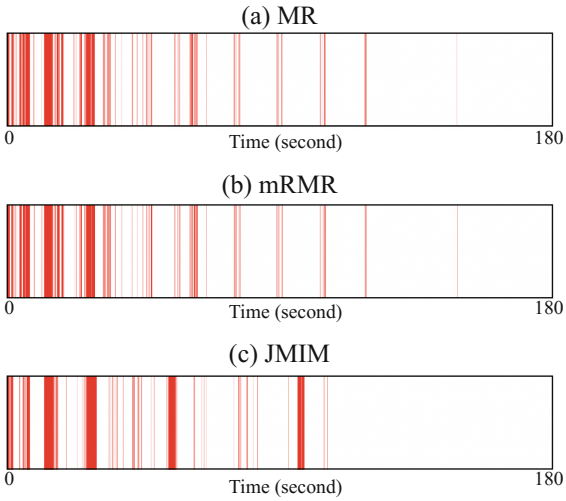


Fig. 8. Spectrum of selected features given bin size $w = 0.05\,\text{s}$ and $t = 10\%$. If a feature is selected, it is marked with red; otherwise, it is marked with white. (Color figure online)

Table 1. Attack accuracy with data from selected features only

	$n = 3600$ (bin size $w = 0.05\,\text{s}$)		$n = 720$ (bin size $w = 0.25\,\text{s}$)	
	$k = 180$	$k = 360$	$k = 72$	$k = 144$
Sliding window PFI	**35.1%**	48.5%	46.4%	**61.7%**
MR	30.5%	**57.7%**	39.8%	47.6%
mRMR	30.1%	57.5%	39.7%	46.7%
JMIM	32.2%	52.3%	**47.8%**	50.2%

Feature f_u is added to set $S = S \cup f_u$ and removed from the unselected set $\{F - S\}$ at the end of this iteration. JMIM continues the iterations until a total number of k features is selected.

6 Evaluation of Feature Selection

Next, we evaluate the results of feature selection over our YouTube dataset and show that each packet is not equally significant in stream fingerprinting.

Results of Sliding Window PFI. We examine Sliding Window PFI over our dataset by examining different sizes of sliding windows. Specifically, given the overall number of features n of a dataset, we examine sliding window size p, where $p = r \cdot n$ and parameter r is the ratio of the sliding window size to the overall number of features.

Given each n, which is decided by bin size, we examine parameter $r = \{5\%, 10\%, 15\%, 20\%\}$. We implemented Sliding Window PFI in `Python` and utilized our CNN described in Sect. 3 as the trained classifier. Specifically, after

each permutation within a sliding window is completed, the accuracy change of classification on test data is recorded.

Due to the limitation of space, we only report the results of Sliding Window PFI when bin size is $w = 0.05$ and $w = 0.25$ s as our CNN achieves higher attack accuracy given these two bin sizes. As we can observe from Fig. 7, significant accuracy changes introduced by the permutation on data are identified at the first 40 s of traffic traces. This indicates that the beginning of encrypted traffic traces are more significant and leaks more privacy.

In addition, we can observe that, given the same bin size (or the number of features), a wider sliding window (i.e., a greater value of r) will cause more changes in attack accuracy. This is expected as a wider sliding window will cause more data to be permuted in each iteration. Moreover, we notice that permutation on data over a smaller bin size will cause more differences in the classification results. This is consistent with our attack results, where data over a smaller bin size leaks more privacy.

Results of Mutual-Information-Based Algorithms. Next, we examine the results of the three mutual-information-based greedy algorithms. Compared to the evaluation of Sliding Window PFI, where the feature importance is reported based on each sliding window, the three greedy algorithms report feature important based on each feature.

We implement the three methods using `feast` in Matlab [4]. `feast` is an open source framework for mutual-information-based feature selections. For each bin size, we select the top $t\%$ of features by following each greedy algorithm. The results of feature selection for bin size $w = 0.05$ and $t = 10\%$ are described in Fig. 8. Given $w = 0.05$ s and $t = 10\%$, there are $3,600$ features in total and 360 features are selected. As we can see that, all the three greedy algorithms indicate the features at the beginning of traffic traces are more significant. We have the same observation for other bin sizes. We skip further details due to space limitation. In addition, we observe that although the selected top $t\%$ of features from each greedy algorithm might be different, but overall the selected feature set is relatively similar.

Validation of Results with CNN. In addition to visualizing the selected features, we also run experiments to valid selected features are more significant to the classification in stream fingerprinting. Specifically, we perform stream fingerprinting, retune hyperparameters and re-train CNN based on data that is associated with selected features. As shown in Table 1, by selecting a much smaller number of features, the CNN model can achieve an accuracy that is much higher than random guess of 1%. For instance, give $n = 3600$ (bin size $w = 0.05$ s), selecting top $k = 360$ ($t = 10\%$) features can achieve more than 50% of accuracy in stream fingerprinting. In addition, we can also confirm that Sliding Window PFI and the three greedy algorithms obtain similar attack accuracy if they select a same number of features on the same dataset. We also notice that mRMR is outperformed by other methods in most of the cases shown in Table 1.

7 Evaluation of SmartSwitch

In this section, we use a concrete noise generation algorithm, named d^*-privacy [26], as a case study to demonstrate that our proposed defense mechanism SmartSwitch can effectively mitigate privacy leakage against fingerprinting attacks but can also significantly save bandwidth overhead.

To prove the advantage of our defense mechanism, we compare SmartSwitch with an existing defense [28], which also leverages d^*-privacy as the underlying noise generation algorithm but applies the same level of noise on all packets. We denoted this defense mechanism as NDSS19 in the rest of this paper. To conduct a fair comparison, we report the efficacy and efficiency of the two defense mechanisms over the same dataset—our YouTube dataset and utilize the same classifier—our CNN.

Details of d^*-privacy can be found in Appendix. The security analysis of it can be found in [26]. We would like to point out that applying the same level of noise in the context of d^*-privacy means that using the same privacy parameter ϵ to generate noise for all the packets in a traffic trace. In our evaluation, we assume a *strong attacker* who can adapt in aware of a defense and can re-train CNN with obfuscated traffic generated by a defense.

7.1 Defense Performance of NDSS19 on YouTube Dataset

We first reproduce the results of the defense mechanism, NDSS19, on our dataset. By reproducing it, we aim to validate that NDSS19 is effective against our CNN model, but not efficient in bandwidth. This evaluation will be used as the baseline in our comparison. We examine YouTube dataset with bin size $w = \{0.05, 0.25\}$ seconds, and we choose privacy parameter $\epsilon = \{5 \times 10^{-7}, 5 \times 10^{-6}, 5 \times 10^{-5}, 5 \times 10^{-4}, 5 \times 10^{-3}, 5 \times 10^{-2}\}$ to obfuscate data with each bin size by using NDSS19. We would like point out that a smaller privacy parameter indicates a higher level of noise in differential privacy. We skip other bin sizes ($w = \{0.5, 1, 2\}$ seconds) investigated in Sect. 3 as those bin sizes derived lower attack accuracy.

Attack Accuracy on Obfuscated Traffic Generated by NDSS19. We performed the attack on obfuscated traffic by using our CNN model. We re-tuned hyperparameters and re-trained CNN over each obfuscated dataset for each combination of bin size w and privacy parameter ϵ. The attack accuracy is shown in Fig. 9. As we can observe, NDSS19 is effective on our dataset when we choose privacy parameter $\epsilon \leq 5 \times 10^{-5}$, which we denote it as *privacy parameter threshold*. When choosing a greater privacy parameter than the threshold, the noise level is no longer effective against the attack using CNN. In addition, given the same privacy parameter, a smaller bin size leads to a higher attack accuracy, which is expected.

The overall observations in Fig. 9 are consistent with the results in the paper of NDSS19, which suggests that we successfully reproduced the results

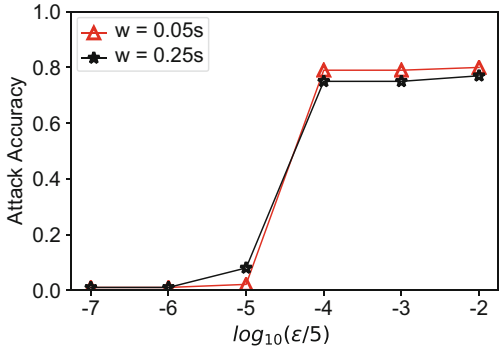


Fig. 9. Attack accuracy on obfuscated traffic generated by NDSS19.

Table 2. Bandwidth overhead of NDSS19

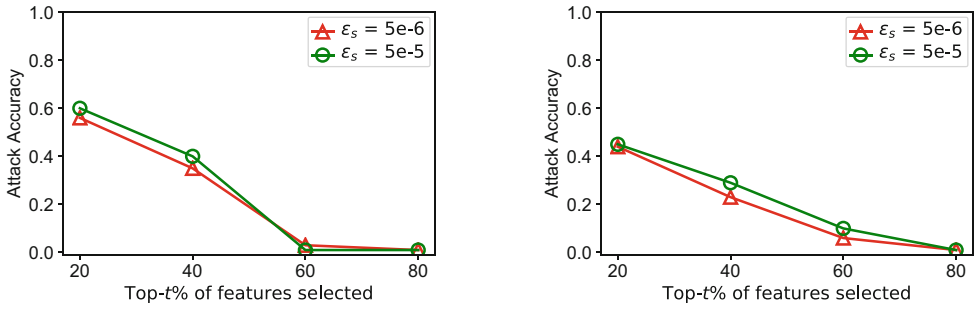
Privacy parameter ϵ	$w = 0.05\,\mathrm{s}$	$w = 0.25\,\mathrm{s}$
5×10^{-4}	50.8%	8.7%
5×10^{-5}	487.5%	87.6%
5×10^{-6}	4895.1%	881.4%

of NDSS19 on our dataset. It is worth to mention that, in the paper of NDSS19, the defense remains effective when privacy parameter $\epsilon \leq 5 \times 10^{-6}$, which is different from our privacy parameter threshold $\epsilon \leq 5 \times 10^{-5}$. This is likely because our dataset is different from the dataset used in their paper.

Bandwidth Overhead of NDSS19. We report the bandwidth overhead of NDSS19 on our YouTube dataset. As shown in Table 2, given bin size $w = 0.25\,\mathrm{s}$ privacy parameter $\epsilon = 5 \times 10^{-5}$, NDSS19 introduces, on average, 87.6% (14.9 MB) bandwidth overhead per trace to generate obfuscated traffic traces. If we keep the bin size the same but change the privacy parameter to privacy parameter $\epsilon = 5 \times 10^{-6}$, the bandwidth overhead increases to 881.4% (149.8 MB) per trace on average. We also observed that a smaller bin size will need more bandwidth overhead given the same privacy parameter. The observation we have in bandwidth overhead is also consistent with the results in the paper of NDSS19. Given a same privacy parameter, *the degree of bandwidth overhead* is similar. The actual numbers are different due to the difference in datasets.

7.2 Defense Performance of SmartSwitch on YouTube Dataset

Next, we report the defense performance of SmartSwitch using d^*-privacy as the underlying noise generation algorithm. In terms of the underlying feature selection algorithms, we investigate Sliding Window Permutation Feature Importance, Max-Relevance, and Joint Mutual Information Maximisation, respectively. We skip Minimal Redundancy Maximal Relevance in the rest of this



(a) MR, $w = 0.05s$ (b) MR, $w = 0.25s$

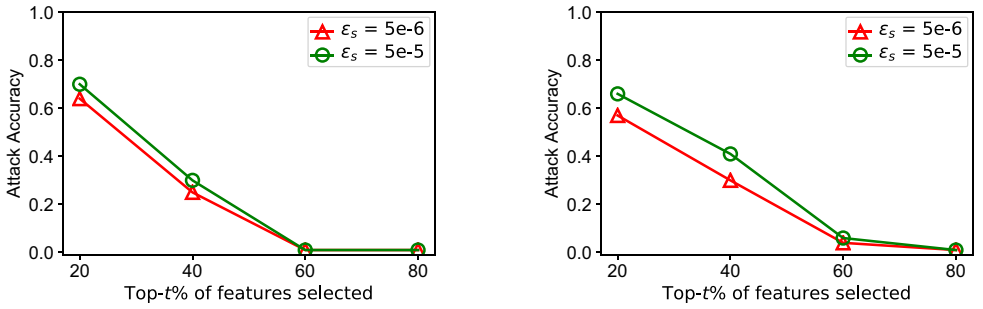
Fig. 10. Attack accuracy on obfuscated dataset generated by SmartSwitch with Max-Relevance.

evaluation as it has a similar but weaker results in feature selection than the other two mutual-information-based algorithms as we shown in Table 1.

The Number of Selected Features. While we have addressed how SmartSwitch can select features and how SmartSwitch can apply noise to obfuscate packets, another critical question we have not answered yet is—*how many features should SmartSwitch select?* To answer this question, given bin size $w = 0.25\,\text{s}$, we selected top-t% of features using different feature selection algorithms on YouTube dataset, where $t = \{20, 40, 60, 80\}$. We applied privacy parameter $\epsilon_u = 5 \times 10^{-3}$ to generate noise for unselected features and privacy parameter $\epsilon_s = \{5 \times 10^{-5}, 5 \times 10^{-6}\}$ respectively to produce noise for selected features. We re-tuned hyperparameters and re-trained CNN over each obfuscated dataset for each combination of bin size w, privacy parameter ϵ and the proportion of selected features t.

As we can see from Fig. 10, if Max-Relevance is the underlying feature selection algorithm, SmartSwitch should select more than top-60% features to effectively defend against stream fingerprinting. For instance, given $\epsilon_s = 5 \times 10^{-5}$, SmartSwitch can reduce the attack accuracy close to 1%. We can also observe that, if we compare the defense performance between $\epsilon_s = 5 \times 10^{-5}$ and $\epsilon_s = 5 \times 10^{-6}$, a higher noise level on selected feature is more effective in defense. If Sliding Window PFI serves as the underlying feature selection algorithm, as shown in Fig. 11, our observations are consistent.

Note that we did not report the cases with Joint Mutual Information Maximization for bin size $w = 0.05\,\text{s}$ (i.e., $3,600$ features), as it is computationally challenging to select more than top-20% of features from a total number of $n = 3,600$ features. For example, after running 2 days with Joint Mutual Information Maximization, our desktop ran out of memory. Thus, we only used JMIM to analyse the datasets with $w = 0.25\,\text{s}$ which is shown in Fig. 12. As we shown above, SmartSwitch in that case needs to obfuscate top-60% of features to be effective in defense.



(a) Sliding Window PFI, $w = 0.05s$ (b) Sliding Window PFI, $w = 0.25s$

Fig. 11. Attack accuracy on obfuscated dataset generated by SmartSwitch with Sliding Window PFI.

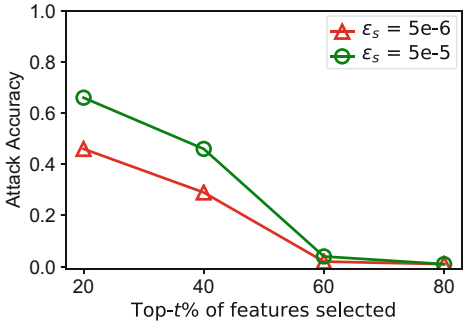


Fig. 12. Attack accuracy on obfuscated dataset generated by SmartSwitch with Joint Mutual Information Maximisation (bin size $w = 0.25$ s).

Bandwidth Overhead of SmartSwitch. As SmartSwitch only needs to primarily protect top-60% of all the features, it can save bandwidth overhead compared to NDSS19. As shown in Table 3, SmartSwitch can reduce the overhead from 87.6% to 52.9% when $w = 0.25$ s. And also, when $\epsilon_s = 5 \times 10^{-6}$, which is the threshold to make the defense effective in [28], SmartSwitch can reduce the overhead from 881.4% to 528.9%. In other words, SmartSwitch can save nearly 40% ($\approx \frac{881.4\% - 528.9\%}{881.4\%}$) bandwidth compared to NDSS19.

8 Related Work

Website Fingerprinting. During the early stage, researchers focused on using traditional machine learning methods to identify encrypted traces [6,10,13,14], These studies rely on hand-crafted features as inputs. Panchenko et al. [13] proposed the state-of-art fingerprinting method which leverages the cumulative size of traffic data as features and their proposed method is able to achieve an accuracy of 93%. Recently, researchers adapted deep learning model as attack which can automate the feature extraction process and achieve a very high accuracy. [19,22] used a carefully designed and tuned Convolutional Neural Network

Table 3. Comparison between NDSS19 and SmartSwitch (bin size $w = 0.25$ s and $\epsilon_u = 5 \times 10^{-3}$ in SmartSwitch)

	NDSS19	SmartSwitch	NDSS19	SmartSwitch
	$\epsilon = 5 \times 10^{-5}$	$\epsilon_s = 5 \times 10^{-5}$	$\epsilon = 5 \times 10^{-6}$	$\epsilon_s = 5 \times 10^{-}6$
Attack accuracy	8.1%	**4.5%**	1.1%	**1.0%**
Bandwidth	87.6%	**52.9%**	881.4%	**528.9%**

(CNN) and achieved 98% accuracy. Besides website fingerprinting, Schuster et al. [21] demonstrated that fingerprinting video streaming traffic is also feasible. They applied CNN on video streams collected from different service providers and their classification accuracy can reach as high as 99%. Recent studies [8,23] also demonstrated that it is feasible to fingerprint voice commands from encrypted traffic of smart speakers.

Defense Against Fingerprinting. Dryer et al. [6] revealed that burst-related information is one of the most important feature for website fingerprinting and they proposed BuFLO (Buffered Fixed-Length Obfuscation) which sends packets at a fixed size within a fixed interval. Compared with BuFLO, WTF-PAD [7] introduces no latency. Wang et al. [24] designed Walkie-Talkie, which changes the communication pattern into half-duplex and also apply burst-modeling to change the burst patterns of traffic. However, both WTF-PAD and Walkie-Talkie can be compromised by deep learning model based attacks. The attack in [22] can achieve 90% accuracy against WTF-PAD and 49.7% against Walkie-Talkie with their CNN model. In [28], Zhang et al. explored differential privacy in order to against the deep learning based attacks. They applied d^*-privacy on video streaming traffic and is able to reduce the accuracy to nearly 1%.

Feature Selection. In general, feature selection methods can be divided into two different categories, classifier dependent (wrapper methods) or classifier independent (filter methods). Wrapper methods analyse feature space by evaluate the classifier's results on each subset [12]. Therefore, the performance of Wrapper methods heavily relies on a well-designed and fine-tuned classification model.

Filter methods use information theory to estimate the relevance between features and class labels. Mutual Information is one of the most popular approaches. Battiti [2] proposed mutual information based feature selection. This method leverages the mutual information between candidate features and labels to select the informative subset. Yang et al. [27] proposed a feature selection method based on JMI (Joint Mutual Information), which estimates the relevance between pairs of features and class labels. Compared to MI, JMI considers conditional mutual information between each two individual features such that JMI treats features dependently. Peng et al. [15] devised a MI-based feature selection criterion, called

mRMR (Minimal-Redundancy-Maximal-Relevance). The redundancy and relevance of candidate features are both considered for the purpose of reducing the dimension of feature set. Bennasar et al. [3] proposed JMIM (Joint Mutual Information Maximisation) which selects features based on JMI. The major difference between mRMR and JMIM is that, JMIM considers labels when it estimates the redundancy of features. In general, relevance and redundancy are two major factors for MI and JMI based filter methods when they evaluate candidate features.

9 Conclusion

We propose a novel defense mechanism to reduce the bandwidth overhead against stream fingerprinting. Our analysis results show that not every encrypted packet leaks privacy evenly, and protecting more significant packets with higher noise level is sufficient to maintain the efficacy in defense. Our defense mechanism is generic and can also be extended to defenses again other encrypted traffic analysis, such as website fingerprinting.

Acknowledgement. Our source code and datasets can be found on GitHub (https://github.com/SmartHomePrivacyProject/SmartSwitch). Authors from the University of Cincinnati were partially supported by National Science Foundation (CNS-1947913), UC Office of the Vice President for Research Pilot Program, and Ohio Cyber Range at UC.

Appendix

Hyperparameters of CNN. The tuned hyperparameters of our CNN are described in Table 4. For the search space of each hyperparameter, we represent it as a set. For the activation functions, dropout, filter size and pool size, we searched hyperparameters at each layer. The tuned parameters we report in the table are presented as a sequence of values by following the order of layers we presented in Fig. 2. For instance, the tuned activation functions are `selu` (1st Conv), `elu` (2nd Conv), `relu` (3rd Conv), `elu` (4th Conv), `tanh` (the second-to-last Dense layer).

d^*-privacy. Xiao et al. [26] proposed d^*-privacy, which is a variant of differential privacy on time-series data, to preserve side-channel information leakage. They proved that d^*-privacy can achieve $(d^*, 2\epsilon)$-privacy, where d^* is a distance between two time series data and ϵ is privacy parameter in differential privacy.

Let $\boldsymbol{x} = (x_1, ..., x_n)$ and $\boldsymbol{y} = (y_1, ..., y_n)$ denote two time series with the same length. The d^*-distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as:

$$d^*(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i \geq 2} |(x_i - x_{i-1}) - (y_i - y_{i-1})| \tag{6}$$

d^*-privacy produces noise to data at a later timestamp by considering data from an earlier timestamp in the same time series. Specifically, let $D(i)$ denote the greatest power of 2 that divides timestamp i, d^*-privacy computes noised

Table 4. Tuned hyperparameters of CNN When $w = 0.05\,\mathrm{s}$

Hyperparameters	Search Space	CNN ($w = 0.05$) s
Optimizer	{Adam, SGD, Adamax, Adadelta}	Adam
Learning rate	{0.001, 0.002, ..., 0.01}	0.006
Decay	{0.00, 0.01, 0.02, ..., 0.90}	0.71
Batch size	{32, 64, 128, 256, 512}	64
Activation function	{softsigh, tanh, elu, selu, relu}	[selu; elu; relu; elu; tanh]
Dropout	{0.1, 0.2, ..., 0.7}	[0.3; 0.4; 0.4; 0.7]
Dense layer size	{100, 110, ...,200}	170
Convolution number	{32, 64, 128, 256, 512}	[256; 128; 128; 512]
Filter size	{4, 6, ..., 26}	[14; 20; 16; 24]
Pool size	{1, 3, 5, 7}	[1; 3; 1; 3]

data $\tilde{x}_i$ at timestamp i as $\tilde{x}_i = \tilde{x}_{G_{(i)}} + (x_i - x_{G_{(i)}}) + r_i$, where $x_1 = \tilde{x}_1 = 0$, function $G(\cdot)$ and r_i are defined as below

$$G(i) = \begin{cases} 0 & \text{if } i = 1 \\ i/2 & \text{if } i = D(i) \\ i - D(i) & \text{if } i > D(i) \end{cases} \tag{7}$$

$$r_i = \begin{cases} \text{Laplace}(\frac{1}{\epsilon}) & \text{if } i = D(i) \\ \text{Laplace}(\frac{\lfloor \log_2 i \rfloor}{\epsilon}) & \text{otherwise} \end{cases} \tag{8}$$

References

1. NNI: An open source AutoML toolkit for neural architecture search and hyper-parameter tuning. https://github.com/Microsoft/nni
2. Battiti, R.: Using mutual information for selecting features in supervised neural net learning. IEEE Trans. Neural Netw. **5**, 537–550 (1994)
3. Bennasar, M., Hicks, Y., Setchi, R.: Feature selection using joint mutual information maximisation. Exp. Syst. Appl. **42**, 8520–8532 (2015)
4. Brown, G., Pocock, A., Zhao, M.J., Lujan, M.: Conditional likelihood maximisation: a unifying framework for information theoretic feature selection. J. Mach. Learn. Res. **13**, 27–66 (2012)
5. Dubin, R., Dvir, A., Hadar, O., Pele, O.: I know what you saw last minute – the Chrome browser case. In: Black Hat Europe (2016)
6. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Peek-a-Boo, I still see you: why efficient traffic analysis countermeasures fail. In: Proceedings of IEEE S&P'12 (2012)
7. Juarez, M., Imani, M., Perry, M., Diaz, C., Wright, M.: Toward an efficient website fingerprinting defense. In: Proceedings of ESORICS 2016 (2016)
8. Kennedy, S., Li, H., Wang, C., Liu, H., Wang, B., Sun, W.: I can hear your alexa: voice command fingerprinting on smart home speakers. In: Proceedings of IEEE CNS 2019 (2019)

9. Kohls, K., Rupprecht, D., Holz, T., Popper, C.: Lost traffic encryption: fingerprinting LET/4G Traffic on Layer Two. In: Proceedings of ACM WiSec 2019 (2019)
10. Liberatore, M., Levine, B.N.: Inferring the source of encrypted HTTP connections. In: Proceedings of ACM CCS'06 (2006)
11. Liu, Y., Ou, C., Li, Z., Corbett, C., Mukherjee, B., Ghosal, D.: Wavelet-based traffic analysis for identifying video streams over broadband networks. In: Proceedings of IEEE GLOBECOM 2008 (2008)
12. Molnar, C.: Interpretable machine learning a guide for making black box models explainable. (2019). https://christophm.github.io/interpretable-ml-book/
13. Panchenko, A., et al.: Website fingerprinting at internet scale. In: Proceedings of NDSS 2016 (2016)
14. Panchenko, A., Niessen, L., Zinnen, A., Engel, T.: Website fingerprinting in onion routing based anonymization networks. In: Proceedings of Workshop on Privacy in the Electronic Society (2011)
15. Peng, H., Long, F., Ding, C.: Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. IEEE Trans. Pattern Anal. Mach. Intell. **27**(8), 1226–1238 (2005)
16. Peng, P., Yang, L., Song, L., Wang, G.: Opening the blackbox of virustotal: analyzing online phishing scan engines. In: Proceedings of ACM SIGCOMM Internet Measurement Conference (IMC 2019) (2019)
17. Rashid, T., Agrafiotis, I., Nurse, J.R.C.: A new take on detecting inside threats: exploring the use of hidden markov models. In: Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats (2016)
18. Reed, A., Klimkowski, B.: Leaky streams: identifying variable bitrate DASH videos streamed over encrypted 802.11n connections. In: 13th IEEE Annual Consumer Communications & Networking Conference (CCNC) (2016)
19. Rimmer, V., Preuveneers, D., Juarez, M., Goethem, T.V., Joosen, W.: Automated website fingerprinting through deep learning. In: Proceedings of NDSS 2018 (2018)
20. Saponas, T.S., Lester, J., Hartung, C., Agarwal, S.: Devices that tell on you: privacy trends in consumer ubiquitous computing. In: Proceedings of USENIX Security 2007 (2007)
21. Schuster, R., Shmatikov, V., Tromer, E.: Beauty and the burst: remote identification of encrypted video streams. In: Proceedings of USENIX Security 2017 (2017)
22. Sirinam, P., Imani, M., Juarez, M., Wright, M.: Deep fingerprinting: understanding website fingerprinting defenses with deep learning. In: Proceedings of ACM CCS 2018 (2018)
23. Wang, C., et al.: Fingerprinting encrypted voice traffic on smart speakers with deep learning. In: Proceedings of ACM WiSec 2020 (2020)
24. Wang, T., Goldberg, I.: Walkie-Talkie: an efficient defense against passive website fingerprinting attacks. In: Proceedings of USENIX Security 2017 (2017)
25. Weinshel, B., et al.: Oh, the places you've been! user reactions to longitudinal transparency about third-party web tracking and inferencing. In: Proceedings of ACM CCS 2019 (2019)
26. Xiao, Q., Reiter, M.K., Zhang, Y.: Mitigating storage side channels using statistical privacy mechanisms. In: Procedings of ACM CCS 2015 (2015)
27. Yang, H.H., Moody, J.: Feature selection based on joint mutual information. In: Proceedings of International ICSC Symposium on Advances in Intelligent Data Analysis (1999)
28. Zhang, X., Hamm, J., Reiter, M.K., Zhang, Y.: Statistical privacy for streaming traffic. In: Proceedings of NDSS 219 (2019)