Video Aficionado: We Know What You Are Watching

Jialing He, Zijian Zhang*, *Member, IEEE,* Jian Mao, *Member, IEEE,* Liran Ma, *Member, IEEE,* Bakh Khoussainov, Rui Jin, and Liehuang Zhu*, *Member, IEEE*

Abstract—Users enjoy the convenience of watching videos on smart devices. However, video watching records can be exposed without users' knowledge and be exploited to infer private information. In this paper, we design and implement a new side-channel attack system, named *video aficionado*, which can identify video watching information without violating any access control policies on Android. Our system only needs to collect power consumption data of a video playing app, which does not require explicit user permission. The collected data is sent to a remote server, where noise is cleaned and identified by a Multi-Layer Perceptron (MLP) trained classifier. We evaluate our proposed system through a set of carefully designed experiments. Experimental results demonstrate that our system can make an identification with 74.5% accuracy on average for each 20-second power measurement segment out of 3918 segments collected from 20 videos. To the best of our knowledge, video aficionado is the first real-time power consumption-based video identification system on smart devices.

Index Terms—Power consumption, Privacy analysis, Video identification, Deep learning.

1 Introduction

With recent advances in wireless communication technology, there has been an upward trend towards streaming videos on mobile devices (e.g., smartphones and tablets). For instance, eMarketer [1] reports that more than 75% of video viewing worldwide was on mobile in 2018. Users shift to mobile video viewing for reasons such as convenience, i.e., without being limited by geographic location or show schedule. Video viewing record (e.g., time, duration, and genres) has been exploited to infer user personalities, preferences, and habits [2]. Furthermore, sensitive private information like emotional status, depression, and violence tendency can be deduced from a number of video watching records. For example, Honorato et al. [3] develop an antisocial personality disorder criteria based on movie watch history, and Romer et al. [4] demonstrate various types of movie effects on children. These works and other similar studies show that mobile video viewing identification can be utilized to extract user privacy, or more precisely, can serve as a fundamental step for in-depth privacy mining.

Typically, mobile devices such as smartphones have a set of rigorous access control mechanisms (e.g., formal permission requests to access personal information) to protect private or otherwise sensitive user data. As a result, these control mechanisms prohibit a third party (e.g., an app) from directly identifying video viewing information. In this paper, our goal is to design a system that indirectly infers videos displayed on a mobile device without violating the control mechanisms. We name our system *video aficionado*. In the design and implementation of the proposed system, we need to go through a number of steps and address challenges presented in each step. Next, we will informally describe those steps and challenges.

Firstly, we need to find appropriate and accessible information that can be used for video identification. Our approach is inspired by the following observation. The power consumption of a video stream is closely correlated to its contents. For example, a dramatic scene (e.g., high-speed car-chasing) likely contain more data bits than a slow-motion scene, which leads to more power consumption in transmission and play. In the literature, power consumption has been used for location tracking (e.g., PowerSpy [5]), browsing activity inference (e.g., USB power analysis [6]), and application identification (e.g., Powerful [7]). Our proposed scheme exploits distinguishable differences in power consumption to identify videos. To the best of our knowledge, we are the first to employ power consumption analysis in video identification.

Secondly, we need to be able to acquire, in real-time, the power consumption of a running application. Accordingly, we design and implement an application named *PowerShot* that utilizes Android internal APIs and Android Debug Bridge (ADB) to fetch power consumption data during video viewing. Note that *PowerShot* does not violate any access control policies on Android.

Thirdly, we need to filter out noise from power consumption data by collected *PowerShot*. The noise arises due to factors such as temperature changes, background services

- *Jialing He, Rui Jin, Liehuang Zhu are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081 P. R. China.*
- *Zijian Zhang is with the School of Computer Science and Technology, Beijing Institute of Technology, China and with the Department of Computer Science, University of Auckland, New Zealand.*
- *Jian Mao is with the School of Cyber Science and Technology at Beihang University, China.*
- *Liran Ma is with the Department of Computer Science at Texas Christian University.*
- *Bakh Khoussainov is with the Department of Computer Science, University of Auckland, New Zealand.*
- *Zijian Zhang (zhangzijian@bit.edu.cn) and Liehuang Zhu (liehuangz@bit.edu.cn) are corresponding authors.*

in the operating system, and battery non-linearity. To address this challenge, we develop a two-step data cleansing procedure. The procedure includes two processes: (1) a Cumulative Distribution Function (CDF)-based process and (2) an Exponential Moving Average (EMA)-based process.

Fourthly, the challenge is to extract unique features with distinguishing power from cleaned power consumption data. After a thorough investigation, we choose 22 features from the time domain (e.g., energy and time difference between two power consumption spikes) and 7 features from the frequency domain (e.g., spectral centroid and entropy). The features are extracted from power consumption measurements divided by sliding windows.

Lastly, we need a suitable classification algorithm that can recognize complex patterns because of subtle differences among power consumption segments. The algorithm also needs to be able to work with complex and imprecise data without underlying assumptions on its probabilistic features. Furthermore, the algorithm shall be capable of performing inference on unseen data, which can help to address unseen videos in open-world scenarios. We choose a Multi-Layer Perceptron (MLP) as the base algorithm.

We conduct a set of comprehensive experiments (using different apps, encoding formats, resolutions, genres, viewers, and classification algorithms) to evaluate the performance of our proposed scheme. The evaluation results demonstrate that our scheme can make an identification with an accuracy of 74.5% on average based on a video segment as short as 20 seconds.

The rest of the paper is organized as follows. Section 2 lays out the background information. Section 3 details the design of our proposed scheme. Evaluation of our proposed scheme is shown in Section 4. Some remarks including defense measures and some discussions about the experiments are presented in Section 5. The related work is shown in Section 6, and then the conclusion of the paper in Section 7.

2 Background

This section provides technical background information on the feasibility of video identification through power consumption profiling. This involves answering two questions: 1) How to accurately capture power consumption of a running app in real-time? 2) What is the relationship between a video and its power consumption?

2.1 Real-time power data acquisition

Power measuring tools (and building them) are of interest since they can provide power consumption information to video service providers (such as iQIYI and Youku) in real time. Many smart devices are also shipped with power measuring mechanisms. For example, Android has included built-in power consumption statistic tools for apps since version 2.0.1. An example of statistics showing app power consumption on a smartphone (OPPO R17) is shown in Fig. 1.

The power consumption of an app on a smartphone is calculated by eq.(1) below

$$\begin{aligned} P_A &= \textstyle\sum_i^n (p_m^i), \\ p_m(t) &= I * t, \end{aligned} \tag{1}$$

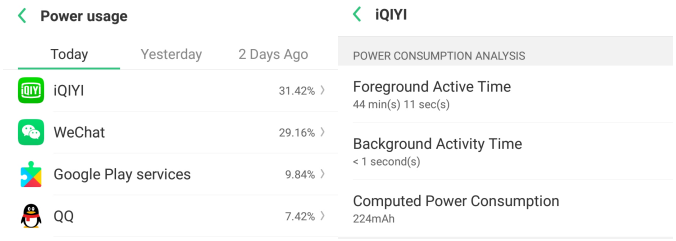


Fig. 1: App power consumption on Android systems.

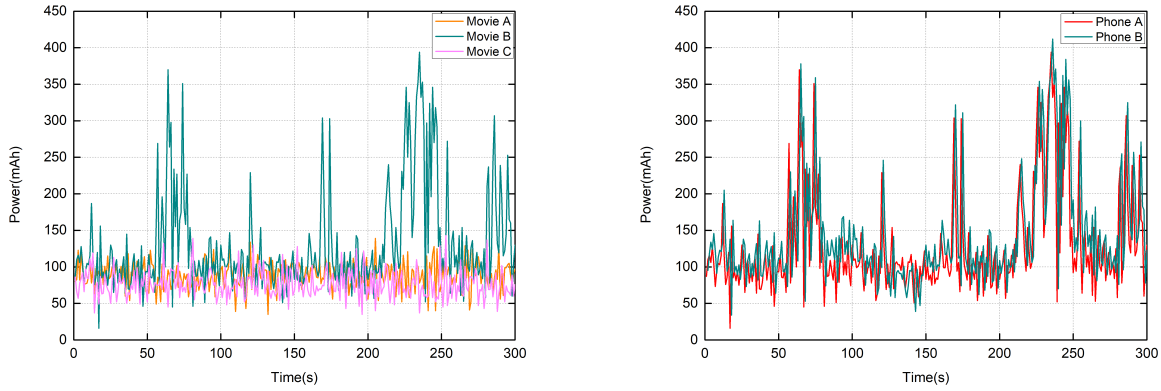
where P_A denotes the power consumption of an installed app, p_m^i stands for the i-th module's (such as CPU, Bluetooth, sensor and so on) power consumption, I means the real-time current, t is the time variable, and the index m indicates the module under consideration.

The duration time t is typically recorded in the system file ($/data/system/batterystats.bin$) and the current I is in the battery profile ($power\_profile.xml$). Using eq.(1), we devise an app named *PowerShot* to record real-time power consumption of apps and output a log file (which is not stored by Android built-in measuring tools) containing the specific power consumption data. *PowerShot* captures the real-time power consumption through Android internal APIs using an ADB and a compliation tool.

2.2 Why can power consumption work?

A video stream consists of frames of images, which can be quite large if not compressed [8]. For example, under a resolution of 1280×720 and a frame rate of 30 (i.e., 30 frames are transmitted and displayed per second), data volume per second is $1280 \times 720 \times 30 \times \frac{3}{2}(Byte)/1024/1024 = 39.5MB$ ($\frac{3}{2}$ is used because the YUV format is adopted; 3 will be used if the RGB format is adopted). Therefore, an 90-minute movie would entail $208.6GB$ data to be transmitted and processed, which is prohibitively highly for mobile devices. Hence, videos need to be compressed before streaming, and decompressed at the receiving devices.

Video compression is based on the strong correlation between image pixels, eliminating some redundant data will not lead to information loss. Specifically, for a short period video that consists of a series of images with little difference in pixel, brightness, and chrominance, we need only preserve the complete information of the first image along with the difference information existing in the other images. Therefore, the more dynamic is a video (e.g., more moves and changes), the larger volume data is generated when compressing the video. In contrast, less computing power would be needed for decompressing. In a commonly used video decoding format H.264, a video is compressed as a series of Group of Pictures (GoP), each GoP is built from a fixed number of I, P and B frames, where I is normally the first frame of GoP, containing the most information, and P and B have much smaller size frames than I frame but they need more computing power to decode (often this means more power consumption). Subsequently, playing different videos can cause different power consumption since the different computing power is needed to decompress different I, P, and B frames. In addition, one compressed video may pro-



(a) Power consumption of three movies on one smartphone.

(b) Power consumption of one movie on two smartphones.

Fig. 2: Power consumption of movies (only part of each is shown for clear demonstration).

duce similar power consumption across different devices. Based on this observation, we postulate that every video has its own signature determined by its power consumption.

To verify our postulation, we employ our power measuring tool *PowerShot* to obtain several videos' power consumption data. The original power consumption information of each video is kept in a log file stored in the smartphone. We capture the power data in the log file and draw a graph for each video's power consumption. This is presented in Fig. 2. Fig. 2(a) contains the power consumption profiles of 3 different movies (Movie A, B, and C) which are played by the same device. Two curves in Fig. 2(b) represent the power consumption of the same movie watched by two people whose devices are both Huawei Honor 7X. Fig. 2(b) indicates that the power consumption data acquired from two devices about one the same movie are similar. These experiments confirm our postulation that the power consumption data can uniquely determine the video. Moreover, the power consumption of any video is, in some respect, device independent; the power consumption data of the same video played on different devices should be identical.

3 The Design of Video Aficionado

3.1 Threat model

We assume a covert app is installed on a victim's Android smartphone and runs in the background. The covert app cannot violate any access control mechanisms on the smartphone, particularly that of video playing apps.

We assume the covert app can only access power measurements on the smartphone, which requires no explicit user permission. In addition, a network connection is required to communicate with a remote server where the video inference process is performed.

We assume the attacker has a prior knowledge of the video collection containing all possible videos the victim may choose. Accordingly, the attacker can measure the power consumption of each video in this collection in advance. The attacker can associate the collected data to the video playing app's power consumption measured by the covert app. On the one hand, the attacker needs to cleanse the noise contained in power measurements, which makes videos identifiable. On the other hand, it quires the attacker to identify the video as soon as possible within tens of seconds rather than after watching the entire video.

3.2 System design

The covert app in our system can be disguised as any apps running on the smartphone. We utilize *PowerShot* to collect the power measurements, which is available for the Android versions ranging from 5.1 to 8.1. Fig. 3 illustrates the work flow of *video aficionado*. The flow contains five steps: power acquisition, data processing, feature extraction, model training, and video inference.

- *Power acquisition.* We need to acquire power consumption data in real-time when a video is being played on a mobile device. In this paper, we develop an app named *PowerShot* to perform this task.
- *Data processing.* The collected raw power consumption data contains noises (e.g., abnormal peaks of power consumption caused by the battery's temperature) that can negatively impact the subsequent steps. We filter out the nosies from the collected raw data in this step.
- *Feature extraction.* We extract a number features from time and frequency domains based on the cleansed power consumption data. Examples of features include spectral entropy and spectral centroid.
- *Model training.* We train our classification model based on the extracted features. A number of recommended movies and TV shows from doban.com [1] are used in this step.
- *Video inference.* Given a piece of power consumption data of a video being watched, the trained classification model can identify what the video is.

We now explain each of these in more details.

1. https://movie.douban.com/top250

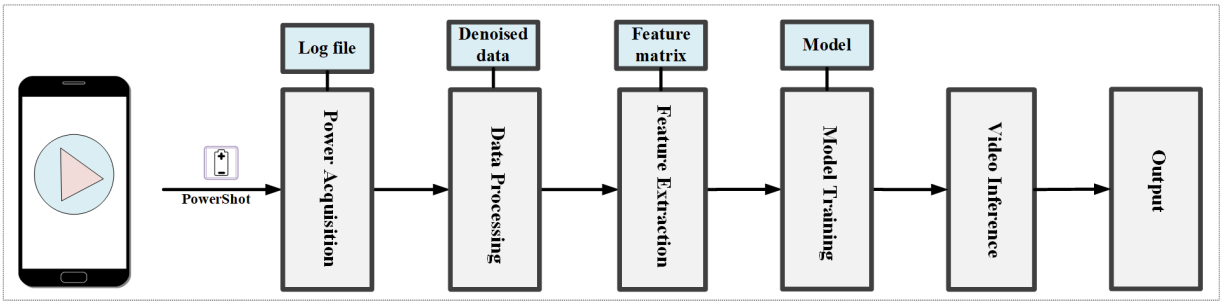


Fig. 3: The framework of Video Aficionado.

3.2.1 Power acquisition

The real-time power consumption data of a video is the basis of the our analysis. So, a power measuring tool is needed to meet the following requirements:

1) The tool should be able to record the video's power consumption in real time.
2) The tool should be able to preserve digital data that can further be used in later operations and analysis of data such as data processing, feature extraction, model training, etc.
3) The tool should be device independent in the sense that it should run on different models of smartphones with various Android versions.

We note that there are several power consumption tools, however, they cannot meet all the above requirements. For example, PowerBooter is a power profiling tool that can record the power consumption of an running app in real-time, but only aiming at ADP1 phones. PowerTutor is an open source tool based on PowerBooter which targets at HTC G1, G2, and Nexus 1 phones with low Android versions. None of these tools can preserve the power consumption data in a file, like a log file. In general, as far as we know, no power consumption tool preserves the digital power data and satisfies all the three requirements.

To satisfy all the requirements above, we devise a power measuring app *PowerShot*. The *PowerShot* tool fetches power consumption data of each running app in real-time and output the measurement data to a corresponding log file. In addition, the *PowerShot* app is compatible with almost all mobile devices running on Android versions from 5.1 to 8.1. Specifically, *PowerShot* can measure power consumption data up to 8 modules (e.g., CPU and sensor) inside a mobile device. In the *PowerShot* app there are also several sampling rates available. The tool measures the power consumption data by invoking $/power\_profile.xml$ and $/data/system/batterystats.bin$ at each sampling moment and outputs the measurements to a log file. The log file mainly contains 5 fields: 'Uid', 'Interval', 'CPU', 'Sensor', and 'All'. 'Uid' refers to the id of the running app, 'Interval' denotes the sampling rate, 'CPU' and 'Sensor' represent the power consumption data of each module, respectively, and 'All' means the total power consumption up to the current time.

3.2.2 Data processing

The captured raw power consumption data contains noises which can be caused by measuring errors generated from the battery of a mobile device (e.g., the temperature of the battery may increase because of its continuous use and this can impact the power consumptions). Therefore, we first perform cleansing operations on the raw power consumption data. Our cleansing operation consists of two common methods used in the signal processing: the first method is based on the CDF and the second method is based on EMA function. The CDF-based method copes with some obvious abnormal peaks and a EMA-based method handles small abnormal fluctuations.

The CDF is the integral of Probability Density Function (PDF) describing the probability distribution of a random variable X. The definition of CDF is given by the equation:

$$F_X(x) = P(X \leq x), \tag{2}$$

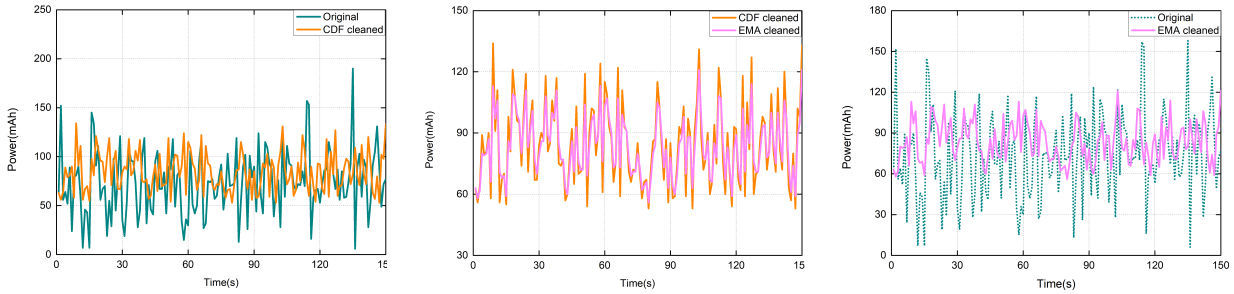
where the right-hand part of (2) denotes the probability the value of the random variable X is equal to or less than x.

In this paper, we remove the power consumption data points whose CDF do not lie between 4.5% to 95.5% (4.2.4 demonstrates the analysis of the parameter). Fig. 4(a) depicts the original power measurements and the denoised power data after the removing process. It shows that some obvious abnormal peaks are eliminated. Note that the denoised data does not need to follow the original since some data points of the original data might be removed.

Although the CDF-based denoising process can remove obvious abnormal peaks. However, small fluctuations also need to be eliminated. We exploit an EMA-based filtering process to cut off these small fluctuations. Moving Average (MA) is an approach to evaluate data points through generating a series of averages for various subsets inside the full data set. There are several types of MA including Simple Moving Average (SMA), Weighted Moving Average (WMA), Exponential Moving Average (EMA) and etc. Here, we utilize EMA that is primarily based on (3):

$$S_t = \begin{cases} Y_1, & t = 1 \\ a \cdot Y_t + (1-a) \cdot S_{t-1}, & t > 1 \end{cases} \tag{3}$$

where Y_t denotes the original data point, S_t represents the data point after the EMA-based filtering process, and a is a parameter whose value lies between 0 to 1. In this paper, we set it as 0.8 (4.2.4 demonstrates the analysis of the parameter). Fig. 5(b) shows the original power measurements and the filtered power data processed by the EMA-based filtering step. We can see that small fluctuations have been removed after this filtering step.



(a) CDF-based noise removal. (b) EMA-based noise removal. (c) Raw data vs. processed data.

Fig. 4: Data processing results.

In summary, with a set of raw power measurements $P(p_1, p_2, ..., p_n)$. Note that these values are from formula (1); so, $P(p_1, p_2, ..., p_n)$ stands for the power consumption of a video, where n means the number of the data points. We first employ a CDF-based denoising step to eliminate the obvious abnormal peaks, and then utilize an EMA-based filtering step to remove small abnormal fluctuations. After these two steps, we obtain the processed data $S(s_1, s_2, ..., s_m)$. The whole denoising process is explained in Algorithm 1.

3.2.3 Feature extraction

Video aficionado is required to infer video information regardless which part of the video is being watched. Therefore, we process the cleansed data in units of sliding windows. In addition, the size of the sliding window needs to be as short as possible, which enables video identification in tens of seconds but rather after watching the entire video. The length of a sliding window is Z and an offset factor is denoted by f. So, the cleansed data $S(s_1, s_2, ..., s_m)$ is proceeded as a sequence of sliding window-based samples:

$$R_i = (r_{(i-1)fZ+1}, r_{(i-1)fZ+2}, ..., r_{(i-1)fZ+Z}),$$

where $i \in \{1, 2, ..., \lfloor \frac{m-Z}{fZ} \rfloor\}$. For each sample R_i, we first perform a max-min peak search in order to capture the skeleton of power consumption fluctuations. Specifically, assuming the sample $R(r_i, r_2, ..., r_Z)$, we will process each measurement r_i, where $i \in \{1, 2, ..., Z\}$) in the sample as follows. The measurement will be selected as a local max (min) peak if it meets the following requirements:

- Its value is larger (smaller) than the previous and the next measurements, which implies that $r_i \geq r_{i+1}$ and $r_i \geq r_{i-1}$ ($r_i \leq r_{i+1}$ and $r_i \leq r_{i-1}$)
- Its value is at least α larger (smaller) than that of the prior local max (min) peak, i.e., $pmax_i \geq \alpha \cdot pmax_{i-1}$ ($pmin_i \leq \alpha \cdot pmin_{i-1}$).

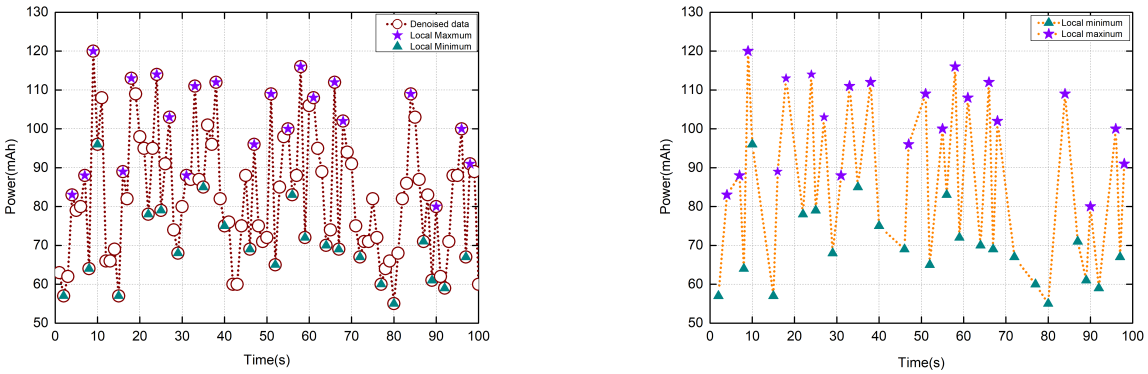
Thus, the two steps above select the measurements representing the local max (min) peaks in each sample R_i.

Parameter α is critical in the max-min peak searching process, which is the basis for feature extraction. If the value of α is too large, many valuable local max and min peaks that contain significant features may be overlooked. On the other hand, if α is too small, too many local peaks would be obtained, which in turn may obscure the real features of a video's power consumption. Based on [7], we set $\alpha = \sigma$ in this paper, where σ is the standard deviation of the power consumption data of a given video. Fig. 5(a) depicts an example of the max-min search process, from which we can see that a series of local max and min peaks are chosen

Algorithm 1 Data denoising

Input: Raw power measurements P
Output: Denoised data S

Remove all zero elements:
$n = size(P), m = 0, Q = []$;
for $i = 1 : N$ **do**
  if $p_i \neq 0$ **then**
    $m = m + 1$;
    $q_m = p_i$;
  end if
end for
for $j = 1 : size(y)$ **do**
  if $f(j) == q_i$ **then**
    break;
  end if
end for
CDF-based denoising:
$[y, f] = ecdf(Q)$;
$g = 0, O = []$;
for $i = 1 : m$ **do**
  for $j = 1 : size(y)$ **do**
    if $f(j) == q_i$ **then**
      break;
    end if
  end for
  if $y(j) < 1 - h \&\& y(j) > h$ **then**
    $g = g + 1$;
    $o_g = q_i$;
  end if
end for
EMA-based denoising:
$s_1 = o_1$
for $i = 2 : g$ **do**
  $s_i = floor(a * o_i + (1 - a) * o_{i-1})$;
end for
return S



(a) Marked local maximums and minimums.

(b) The skeleton constructed by the marked maximums and minimums.

Fig. 5: Selection of local max and min peaks.

through our searching process.

The skeleton of a movie's power consumption is deemed attained when all the local max and min peaks are captured. The information about the location, value, and transformation between the max and min peaks are vital features that are needed to be recorded. We use several vectors to enclose this information. Specifically, the vector $W = ((r_1, t_1), ..., (r_c, t_c))$ contains the information about all local max and min peaks, where r_i is the power measurement of the $i-th$ local max/min peak, t_i is the relevant time stamp, and c is the total number of local max/min peaks. The vector $U = (u_1, ..., u_c)$ is derived from W such that $u_j = 1$ when r_j is a local max peak and$u_j = -1$ when r_j is a local minimum.

We further calculate $\triangle W$, $\triangle T$, and Re vectors to describe the variation of local max and min peaks. For $i \in \{1, ..., c\}$, if $u_i u_{i+1} = -1$ then the value of $\mid r_i - r_{i+1} \mid$ is recorded as an element of $\triangle W$, while the value of $t_{i+1} - t_i$ is recorded as an element of $\triangle T$. Moreover, the vector Re can be computed as $Re = \frac{\triangle W}{\triangle T}$, where L denotes the total number of Re.

Once we have the skeleton of the power consumption of a video, the video features can be calculated in the given time. We use a simple library LibXtract [9] that contains a series of audio feature extraction functions and captures several features for the video power consumption data.

For each sliding window-based power sample $R = (r_1, r_2, ..., r_Z)$, a feature vector $feature = (f_1, f_2, ..., f_K)$ representing the features of a video piece is calculated, where K denotes the total number of features. In our case $K = 29$. Below we describe those features in more detail.

Time-domain features:

Below we list 22 features related to time domain.

- Average power consumption in a sliding window.

$$f_1 = \frac{\sum_{i=1}^{Z} r_i}{Z} \tag{4}$$

- The standard deviation of power consumption in a sliding window.

$$f_2 = \sqrt{\frac{1}{Z-1}\sum_{i=1}^{Z}(r_i - f_1)^2} \tag{5}$$

- The maximum and minimum values of power consumption in a sliding window.

$$f_3 = max(R) \qquad f_4 = min(R) \tag{6}$$

- The 30%-th, 60%-th, 90%-th power measurements in a sliding window.

$$f_5 = r_{\lfloor Z*0.3 \rfloor} \qquad f_6 = r_{\lfloor Z*0.6 \rfloor} \qquad f_7 = r_{\lfloor Z*0.9 \rfloor} \tag{7}$$

- Features related to $\triangle W$

$$f_8 = \frac{\sum_{i=1}^{L} \triangle W(i)}{L} \tag{8}$$

$$f_9 = \sqrt{\frac{1}{L-1}\sum_{i=1}^{L}(\triangle W(i) - f_8)^2} \tag{9}$$

$$\begin{aligned} f_{10} &= \triangle W(\lfloor L*0.3 \rfloor) \\ f_{11} &= \triangle W(\lfloor L*0.6 \rfloor) \\ f_{12} &= \triangle W(\lfloor L*0.9 \rfloor) \end{aligned} \tag{10}$$

- Features related to $\triangle T$

$$f_{13} = \frac{\sum_{i=1}^{L} \triangle T(i)}{L} \tag{11}$$

$$f_{14} = \sqrt{\frac{1}{L-1}\sum_{i=1}^{L}(\triangle T(i) - f_{13})^2} \tag{12}$$

$$\begin{aligned} f_{15} &= \triangle T(\lfloor L*0.3 \rfloor) \\ f_{16} &= \triangle T(\lfloor L*0.6 \rfloor) \\ f_{17} &= \triangle T(\lfloor L*0.9 \rfloor) \end{aligned} \tag{13}$$

- Features related to Re

$$f_{18} = \frac{\sum_{i=1}^{L} Re(i)}{L} \tag{14}$$

$$f_{19} = \sqrt{\frac{1}{L-1}\sum_{i=1}^{L}(Re(i) - f_{18})^2} \tag{15}$$

$$\begin{aligned} f_{20} &= Re(\lfloor L * 0.3 \rfloor) \\ f_{21} &= Re(\lfloor L * 0.6 \rfloor) \\ f_{22} &= Re(\lfloor L * 0.9 \rfloor) \end{aligned} \tag{16}$$

Frequency-domain features:

Features from the frequency sphere are also vital for video identification. We adopt seven features in the frequency domain for each sliding window-based power sample. Considering a sliding window $R = (r_1, r_2, ..., r_Z)$, we first apply Fast Fourier Transform (FFT) to the sliding window and obtain the vector $D = (d_1, d_2, ..., d_Z)$. We then calculate the seven features, which are root mean square (RMS) (f_{23}), spectral centroid (f_{24}), spectral entropy (f_{25}), spectral spread (f_{26}), spectral skewness (f_{27}), spectral kurtosis (f_{28}) and spectral flatness (f_{29}). The detailed calculation formulas are given as follows.

- Root Mean Square (RMS)

$$f_{23} = \sqrt{\frac{1}{Z}\sum_{i=1}^{Z} d_i^2} \tag{17}$$

- Spectral centroid

$$f_{24} = \frac{\sum_{i=1}^{Z} d_i \frac{i}{2}}{\sum_{i=1}^{Z} d_i s} \tag{18}$$

- Spectral entropy

$$f_{25} = \frac{\sum_{i=1}^{Z} d_i \frac{i}{2}}{\sum_{i=1}^{Z} d_i s} \tag{19}$$

- Spectral spread

$$f_{26} = \sum_{i=1}^{Z} \beta_i log_2^{\beta_i} \quad where \quad \beta_i = \frac{d_i}{\sum_{j=1}^{Z} d_j} \tag{20}$$

- Spectral skewness

$$f_{27} = \sqrt{\sum_{i=1}^{Z}[\beta_i(\frac{i}{2} - f_{25})^2]} \tag{21}$$

- Spectral kurtosis

$$f_{28} = \frac{\sum_{i=1}^{Z} \beta_i(\frac{i}{2} - f_{25})^3}{f_{27}^3} \tag{22}$$

- Spectral flatness

$$f_{29} = \frac{(\prod_{i=1}^{Z} d_i)^{\frac{1}{Z}}}{\frac{1}{Z}\sum_{i=1}^{Z} d_i} \tag{23}$$

3.2.4 Model training

We use MLP to train a classifier based on the features generated from the previous step. We choose MLP for the following reasons. Firstly, MLP is capable of extracting patterns that are too complex to be picked up by traditional classification algorithms [10]–[12], which helps capture subtle differences among short (i.e., 20 seconds) power consumption measurement segments of videos. Secondly, MLP can work with complex and imprecise data without underlying assumptions on its probabilistic features such as PDF. Last, MLP can perform inference on unseen data after learning from the initial inputs, which can deliver consistent performance when encountering an extended video set or unseen videos in open-world scenarios.

To balance the efficiency and accuracy, we construct a simple MLP network with one hidden layer. The network contains 20 neurons based on the number of features and classification labels. The sigmoid function is utilized as the activation function of the network.

3.2.5 Video inference

Once a test video segment is obtained, we perform data preprocessing and feature extraction steps for this power consumption segment. After that, several feature instances are obtained. We then test each instance with the trained classifier and finally output the video label of the instance.

4 EVALUATION

4.1 Settings

The smartphone model used in our evaluation is Huawei Honor 7X. Specifically, this model hosts a Kirin 659 chipset (16 nm) with an Octa-core ARM CPU (4×2.36 GHz Cortex-A53 and 4×1.7 GHz Cortex-A53), and runs on Android 7.0 with EMUI 5.1. Note that more than 95 percent of the world's smartphones are built on the ARM architecture. More importantly, the power consumption patterns used by our proposed system are identical under different generations of ARM CPUs. That is, we do not measure the absolute power consumption values, rather the changes in time and frequency domains (e.g., number of peaks in a given time). Thus, it is reasonable to expect our proposed system would be able to deliver replicable performance on different smartphones with different chipsets. We recruit 10 volunteers to watch the 20 videos including 10 movies and 10 TV shows. Each volunteer is required to watch all the movies and TV shows once through video players such as iQIYI[2] or Baofeng[3]. In this work, all the videos are watched offline. The power consumption of the video player measured by our app *PowerShot* is regarded as the power consumption of the video being watched. More specifically, only the power consumption of CPU is recorded. We choose the CPU module's power consumption as the video playing app's power measurements for two-fold reasons: when a user watches videos off-line, CPU and screen consume the most energy compared to other modules such as sensors and Bluetooth; screen power consumption varies significantly from user to user (e.g., preference setting for screen brightness), which

2. https://www.iqiyi.com/
3. http://www.baofeng.com/

can degrade identification accuracy across different users. In contrast, CPU power consumption is consistent regardless of users. Following experimental results demonstrate the effectiveness of using the CPU module as well. A sound system should be agnostic to video resolutions. That is, regardless of the resolution of a video, the system should be able to correctly identify the video. Therefore, we utilize different resolution videos ranging from 360P to 1080P for evaluating our system.

The collected power consumption data of these 20 videos are divided into a training set and a test set. Specifically, we firstly record the videos' power consumption on 9 different devices as the training data set. Then, the videos' power consumption on another different device is collected into the test data set. Segments (for instance 20-second segment) of each video from the test data set are tested by the trained classifier.

4.2 Results

4.2.1 Overview

We run each experiment for 20 times and record the average identification *accuracy*, *precision*, and *recall*, which is shown in Table 1. Specifically, our proposed system achieves an average identification accuracy of 74.5% over 3918 power measurement segments. These segments include diverse video configurations across different resolutions, encoding, and genres. The experiment results prove that the selected features are able to capture the skeleton of power measurements of each video. Furthermore, the results demonstrate that our MLP based classifier can deliver consistent performance under a wide range of video configurations.

TABLE 1: Average identification accuracy

Number of instances	Accuracy	Precision	Recall
3918	74.5%	0.748	0.746

In our experiments, we also discover some real-world factors that negatively affect the identification accuracy. For example, our volunteers use different devices of the maker and model. Nonetheless, they have different apps running on each device during video watching. Subsequently, the app difference adds additional random variations to the power measurements, which in turn degrades the identification accuracy.

Next, we conduct comprehensive experiments to show the impacts of different parameters on the performance of our proposed system.

4.2.2 Impacts of the number of sample per video

As we mentioned before, the training data set is composed of power measurements recorded by another 9 devices. The test dataset is the power data from the victim's smartphone. In this experiment, we compare 9 different sets of the training dataset, from 1 sample to 9 samples per video. Fig. 6 demonstrates that the identification accuracy increases with the increase of sample number per video in the training data set. It is worth noting that Fig. 6 shows that the identification accuracy can reach above 50% when 5 samples per video are utilized to collect the training data.

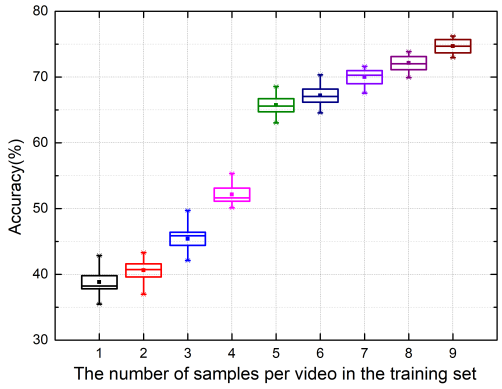


Fig. 6: Accuracy vs. number of samples per video.

4.2.3 Impacts of sliding window size

Fig. 7 demonstrates the impacts of different sliding window sizes on identification accuracy. The sample rate here is set as 1 Hz, i.e., *PowerShot* fetches the power consumption of the video at each second. The identification accuracy increases first as the size of the sliding window grows. Then, it reaches the highest when the size is set as 20. After that, the accuracy decreases as the sliding window size becomes larger.

Identifying a 20-second video piece is differentiating hundreds of I, P, B frames in fact. The likelihood of changes between hundreds of frames is high, which makes the video unique and identifiable. However, it is still possible that there are very little changes for two "stationary" video pieces, which is a factor that degrades the identification accuracy.

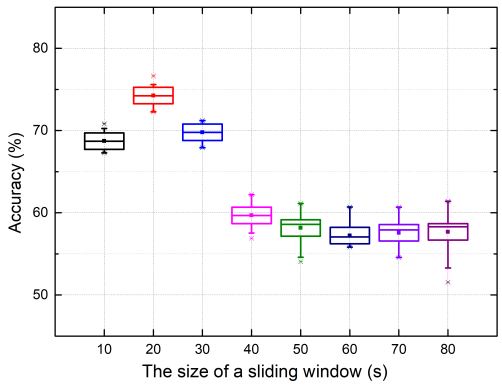
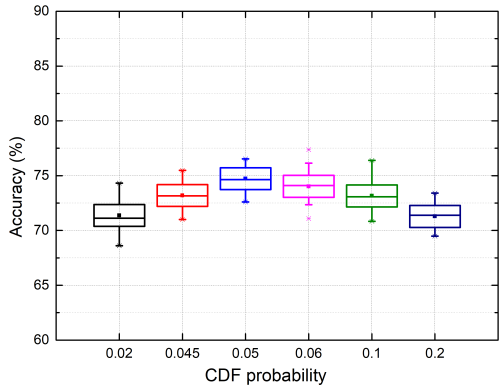


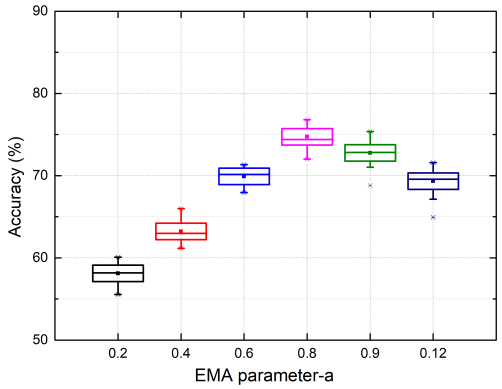
Fig. 7: Accuracy vs. sliding window size.

4.2.4 Impacts of data processing

As mentioned in Section 3.B, the data processing operations of our scheme contain a CDF-based step and an EMA-based step. Fig. 8 demonstrates the impacts of these two processing steps on the identification accuracy. Fig. 8(a) indicates when the CDF probability is set as 0.05, the average identification accuracy achieves the highest. Fig. 8(b) shows



(a) CDF-based denoising results.



(b) EMA-based denoising results.

Fig. 8: Impacts of data processing.

the identification accuracy can reach to the highest when the parameter a of EMA is set as 0.8.

4.2.5 *Impacts of different features and their settings*

We take 29 features in total involving both time and frequency domains. Fig. 9 demonstrates the impacts of these 29 features, where the higher the information gain is, the more effective of the corresponding feature for the identification accuracy. From Fig. 9, we can see that several features that have the greatest impacts on identification accuracy are f_3, f_{10}, f_{15}, and f_{20}. These features are calculated from the local maximums and minimums, indicating the drastic fluctuations of the video's power data are the most effective factor for video identification. These unique fluctuations of each different video often caused by shot switching and editing tempo. In this paper, some features like (f_5, f_6, f_7), (f_{10}, f_{11}, f_{12}), (f_{15}, f_{16}, f_{17}), and (f_{20}, f_{21}, f_{22}) are all set as the $30\% - th$, $60\% - th$, and $90\% - th$. These percentages are selected in accordance with library LibXtract [9]. Furthermore, experimental results depicted in Fig. 10 indicates that 30%, 60%, and 90% are the optimal ones compared to other percentage settings.

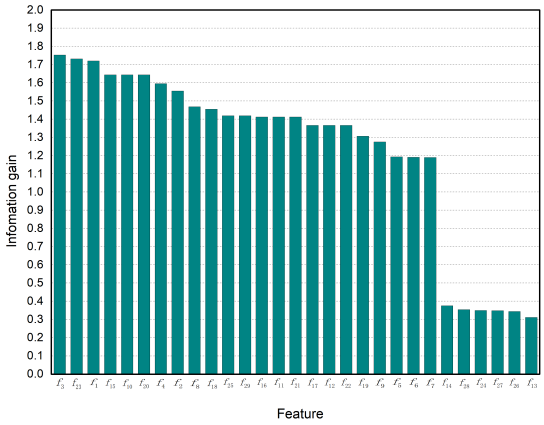


Fig. 9: The impacts of different features.

5 Remarks

5.1 Defence

One way to defend the attack described in this paper is to restrict access to the power consumption statistics on Android. However, power profiles can be useful in various domains such as assisting developers to optimize hardware design or software implementation, which is beneficial and legitimate. Therefore, completely banning the access to power measurements is not ideal. One possible defense measure is to add random redundant P, or B frames in the compressed data of videos as mentioned in 2.2. Thus, the power consumption patterns can be changed while it does not impact the video viewing experience.

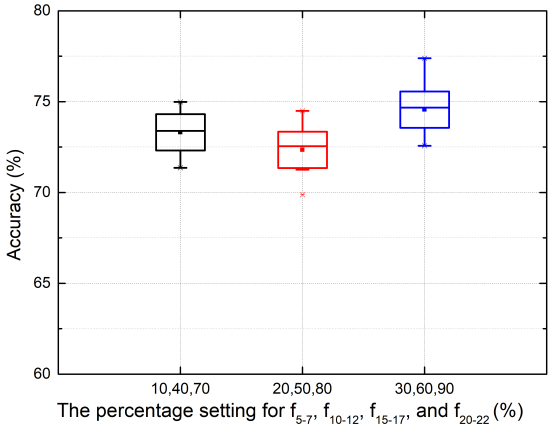


Fig. 10: The impacts of differen percentages for (f_5, f_6, f_7), (f_{10}, f_{11}, f_{12}), (f_{15}, f_{16}, f_{17}), and (f_{20}, f_{21}, f_{22}).

5.2 Discussion

First of all, one could argue that only 20 videos have been used in our evaluation, which is a small number. It is worth

noting that there are 3918 20-second video power consumption instances out of these 20 videos, which is a reasonable large data set. We think it would be beneficial to expand the video selection in future work, and the identification accuracy may be degraded with more videos. We argue that due to the advance features of an MLP based classifier on large data sets, the performance degradation could be insignificant. Otherwise, it would be an interesting research direction on how to maintain desirable identification accuracy under a large amount of videos.

Secondly, another concern is the "open-world" issue, namely, how to deal with the videos that do not belong to our data set. We can address this issue by adding a confidence threshold. Specifically, if the confidence of a prediction is below a certain threshold, it will be labeled as "other", which refers to a video that is not among the training set of videos. As to how to tune the threshold on our classifier to better fit the "open-world" situation, it can be another future work.

Thirdly, we only consider off-line video watching in this paper. It is true that power consumption can be fluctuated by the dynamics of network connections. However, our scheme only uses the CPU power consumed by the video playing app. As a result, the generated power profile for each video shall be independent of network dynamics. We argue that our results are also applicable to online video watching.

Lastly, when we did our experiment, our power consumption measuring app *PowerShot* was able to run successfully on the latest version of Android OS to capture the power consumption of videos being played. However, Android has been constantly updated to prevent newly discovered security threats by restricting access to certain APIs of the system. Therefore, it is possible that our app would need to be modified to be compatible with newer Android versions in the future. However, power consumption data is generally considered as insensitive, which makes restricting access to power data a lesser priority in Android. Last but not least, our proposed system could play a positive role in helping Android to improve its security due to its potential in spurring other security-related works.

6 RELATED WORK

In this section, we briefly introduce the related work in the domain of video streams identification by network traffic analysis, measuring the power consumption of phones, and data analyzing by mobile devices' sensor or power data.

6.1 Video fingerprinting by traffic analysis

Inspired by the technique that attacking or fingerprinting websites by traffic analysis, some works that utilize traffic analysis to infer videos have emerged. Saponas et al. [13] found that video contents have correlations with their traffic traces. They captured the signature of traffic trace, which further being utilized to identify the video. Some similar mechanisms improved the identification accuracy of video inference by adding some other features like frame sizes [14], aggregated traffic throughput [15], etc. Later some advanced schemes exploited some other parameters like Pearson correlation to implement video fingerprinting frameworks for Netflix streams [16], [17]. Schuster et al. [18] scaled the scheme for "open-world" videos, namely the videos do not belong to the collected data set.

Most of the work collected traffic data of video streams by directly or remotely attack victims devices by Wi-Fi sniffing or JavaScript etc. All the schemes could only apply for PC devices and oft worked based on strong assumptions that victims devices can be attacked with relatively low user permissions.

6.2 Power data acquisition

All the works can be roughly divided into three categories: hardware-based [20], [21], model-based [19], [22] and software-based power profiling [23], [24].

Hardware-based power profiling methods mainly rely on several peripheral devices that directly measure a phone's battery and compute the power consumption accordingly. Trestian et al. [20] utilize the Arduino Duemilanove to measure the voltage of a smartphone's battery and then compute the current by measuring the voltage drop generated by a resistance. The power consumption of this tested smartphone can eventually be calculated. Similarly, Hindle et al. [21] capture a smartphone's power consumption through the Arduino Duemilanove and an INA219 current meter. All of these methods can only measure the aggregated power consumption of the phone.

Model-based power profiling approaches estimate the power consumption of specific applications on smartphones by constructing a mathematical model consisting of a series of functions that represent relations between system parameters (states) and power consumptions. Zhang et al. [22] propose a power profiling model referred as PowerBooter, which involves six modules containing CPU, Wi-Fi, GPS, cellular, LCD screen and audio. This just aims at ADP1 phones. PowerTutor [19] is an open source tool based on PowerBooter which targets at HTC G1, HTC G2, and Nexus one phones on low Android versions.

Software-based power profiling methods capture the power consumption through Android system API without peripheral devices or mathematical models. Invoking Android API makes these methods amenable to almost all kinds of smartphones. PETrA [23] is a power measuring tool utilizing Android system power consumption statistics, which can obtain the power consumptions of all smartphones with Android version 5.0 or over. However, it can only run on PC and is relies on USB to collect tested devices, which is not very practical. Battery Historian [24] is another power measuring tool constructed by Google, which exploits Docker and ADB toolkits to visualize power consumptions of tested smartphones, however, can only run on PC like PETrA.

6.3 Data analysis on mobile devices

Sensor or power data of mobile devices can be utilized to execute the location inference, human activity recognition, continuous authentication, and mobile app fingerprinting. One way of collecting personal sensitive data is through sensors on the mobile devices like accelerometers, gyroscopes, microphones, cameras, etc. Several schemes [25]–[33] utilize

the motion sensors' (accelerometers, gyroscopes, and magnetometers) data to track the users, infer the running app, and authenticate the users; Another common approach of amassing personal sensitive data is analyzing power consumption profiles of mobile devices: Chen et al. [7] propose a system which can perform mobile app fingerprinting via apps' power profiles; Michalevsky et al. [5] exploit machine learning algorithms and the phone's power consumption profiles to infer the phone's location.

7 Conclusion

We propose a video identification system named *video aficionado* to infer video viewing information on a mobile device withour violating access control policies on Android. To the best of our knowledge, it is the first system that can accurately identify the videos being watched on mobile devices in near real time with only power consumption data. Our proposed system combines effective signal processing techniques and powerful classification algorithms, and strikes a balance between accuracy and complexity. Our experiment results demonstrate the effectiveness of our proposed system under various settings.

ACKNOWLEDGMENTS

This paper is partially supported by National Natural Science Foundation of China No. 61872041, and the National Natural Science Foundation of China under Grant 61772070.

References

[1] https://www.emarketer.com/newsroom/index.php/threequarters-video-viewing-mobile/
[2] I. Cantador, I. Fernńdez-Tobás, A. Bellogín. Relating personality types with user preferences in multiple entertainment domains[C]//CEUR workshop proceedings. Shlomo Berkovsky, 2013.
[3] T.G. Honorato, V.H.S. Oliva, et al. The antisocial personality disorder in the Brazilian movies[J]. Jornal Brasileiro de Psiquiatria, 2018, 67(3): 143–150.
[4] D. Romer, P.E. Jamieson, K.H. Jamieson, et al. Parental desensitization to gun violence in PG-13 movies[J]. Pediatrics, 2018, 141(6): e20173491.
[5] Y. Michalevsky, A. Schulman, G.A. Veerapandian, et al. Powerspy: Location tracking using mobile device power analysis[C]//24th USENIX Security Symposium (USENIX Security 17). 2015: 785–800.
[6] Q. Yang, P. Gasti, G. Zhou, et al. On inferring browsing activity on smartphones via USB power analysis side-channel[J]. IEEE Transactions on Information Forensics and Security, 2016, 12(5): 1056–1066.
[7] Y. Chen, X. Jin, J. Sun, et al. POWERFUL: Mobile app fingerprinting via power analysis[C]//IEEE INFOCOM 2017-IEEE Conference on Computer Communications. IEEE, 2017: 1–9.
[8] W. Hu, G. Cao. Energy-aware video streaming on smartphones[C]//2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, 2015: 1185–1193.
[9] https://www.jamiebullock.com/LibXtract/documentation/index.html
[10] L. Breiman: Random forests[J]. Machine learning, 2001, 45(1): 5–32.
[11] Vapnik, V. and Chervonenkis, A., 1964. A note on class of perceptron. Automation and Remote Control, 24.
[12] Murphy K P. Naive bayes classifiers[J]. University of British Columbia, 2006, 18: 60.
[13] T Scott Saponas, Jonathan Lester, Carl Hartung, Sameer Agarwal, and Tadayoshi Kohno. Devices that tell on you: Privacy trends in consumer ubiquitous computing. In USENIX Security 2007.
[14] Yali Liu, Canhui Ou, Zhi Li, Cherita Corbett, Biswanath Mukherjee, and Dipak Ghosal. Waveletbased traffic analysis for identifying video streams over broadband networks. In GLOBECOM 2008.
[15] Yali Liu, Ahmad-Reza Sadeghi, Dipak Ghosal, and Biswanath Mukherjee. Video streaming forensicCcontent identification with traffic snooping. In ISC 2010.
[16] Andrew Reed and Benjamin Klimkowski. Leaky streams: Identifying variable bitrate DASH videos streamed over encrypted 802.11n connections. In CCNC 2016.
[17] Andrew Reed, and Michael Kranch. Identifying HTTPS-protected Netflix videos in real-time. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 361–368. 2017.
[18] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In 26th USENIX Security Symposium pp. 1357–1374. 2017.
[19] http://ziyang.eecs.umich.edu/projects/powertutor/index.html
[20] R. Trestian, A.N. Moldovan, O. Ormond, et al. Energy Consumption Analysis of Video Streaming to Android Mobile Devices[C]//2012 IEEE Network Operations and Management Symposium. IEEE, 2012: 444–452.
[21] A. Hindle, A. Wilson, K. Rasmussen, et al. Greenminer: A Hardware Based Mining Software Repositories Roftware Energy Consumption Framework[C]//Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014: 12–21.
[22] L. Zhang, B. Tiwana, Z. Qian, et al. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones[C]//Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 2010: 105–114.
[23] D. Di Nucci, F. Palomba, A. Prota, et al. Petra: A Software-based Tool For Estimating the Energy Profile of Android Applications[C]//2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). IEEE, 2017: 3–6.
[24] "Battery Historian 2.0," https://github.com/google/battery-historian, 2016.
[25] Y. Liang, Z. Cai, Q. Han, et al. Location privacy leakage through sensory data[J]. Security and Communication Networks, 2017.
[26] J. Han, E. Owusu, L.T. Nguyen, et al. Accomplice: Location inference using accelerometers on smartphones[C]//2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012). IEEE, 2012: 1–9.
[27] W. Xu, G. Lan, Q. Lin, et al. KEH-Gait: Towards a Mobile Healthcare User Authentication System by Kinetic Energy Harvesting[C]//NDSS. 2017.
[28] S. Eberz, N. Paoletti, M. Roeschlin, et al. Broken hearted: How to Attack ECG Biometrics[C]//NDSS. 2017.
[29] A.M. Khan, Y.K. Lee, S.Y. Lee, et al. Human Activity Recognition Via An Accelerometer-enabled-smartphone Using Kernel Discriminant Analysis[C]//2010 5th international conference on future information technology. IEEE, 2010: 1–6.
[30] H. Saevanee, N. Clarke, S. Furnell, et al. Continuous User Authentication Using Multi-modal Biometrics[J]. Computers & Security, 2015, 53: 234–246.
[31] H. Xu, Y. Zhou, M.R. Lyu. Towards Continuous and Passive Authentication Via Touch Biometrics: An experimental Study on Smartphones[C]//10th Symposium On Usable Privacy and Security (SOUPS 2014). 2014: 187–198.
[32] Z. Sitová, J. Šeděnka, Q. Yang, et al. HMOG: New Behavioral Biometric Features For Continuous Authentication of Smartphone Users[J]. IEEE Transactions on Information Forensics and Security, 2016, 11(5): 877–892.
[33] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, Networkprofiler: Towards Automatic Fingerprinting of Android Apps, in IEEE INFOCOM' 13, Turin, Italy, Apr. 2013.

Jialing He (hejialing@bit.edu.cn) received the B.Eng. and M.S. degrees from the Beijing Institute of Technology, Beijing, China, in 2016 and 2018, respectively, where she is currently pursuing the Ph.D. degree with the School of Computer Science and Technology. Her current research interests include NILM, Internet of Things security, and analysis of entity behavior and preference.

Zijian Zhang (zhangzijian@bit.edu.cn) received his Ph.D. degree in Beijing Institute of Technology. He is an Associate Professor with the School of Computer Science and Technology, Beijing Institute of Technology. His research interests include authentication and key agreement, behavior recognition, and privacy preserving.

Jian Mao (maojian@buaa.edu.cn) is an Associate Professor in the School of Cyber Science and Technology at Beihang University, China. Her interests include applied cryptography and cloud security, web security, and mobile security. She received her Ph.D. degree from Xidian University, China.

Liran Ma (l.ma@tcu.edu) is an Associate Professor in the Department of Computer Science at Texas Christian University. His current research interests include wireless, mobile, and embedded systems, including security and privacy, smartphones, smart health, mobile computing, Internet of Things, and cloud computing. It involves building prototype systems, and conducting real experiments and measurements. He received his D.Sc. degree in Computer Science from The George Washington University.

Bakh Khoussainov (bmk@cs.auckland.ac.nz) received the Ph.D. degree in Mathematics from the Algebra and Logic Department, Novosibirsk University, USSR. He is currently a professor in the Computer Science Department, the University of Auckland, New Zealand. His research interests include computable algebraic systems and model theory, automata and automatic structures, games on finite graphs and complexity, abstract data types and algebraic specifications, computably enumerable reals and randomness. He is also an editor of the JSL, and consider quality papers in effective algebra and model theory, automatic structures, automata and logic.

Rui Jin is a master student in the School of Computer Science and Technology, Beijing Institute of Technology, China. His research interests include NILM and Network Security.

Liehuang Zhu (liehuangz@bit.edu.cn) received the B.Eng. and M.S. degrees from Wuhan University, Wuhan, China, in 1998 and 2001, respectively, and the Ph.D. degree from the School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China, in 2004. He is a Professor with the School of Computer Science and Technology, Beijing Institute of Technology. He is selected into the Program for New Century Excellent Talents in University from the Ministry of Education, Beijing. His current research interests include Internet of Things, cloud computing security, and blockchain.