MagSpy: Revealing User Privacy Leakage via Magnetometer on Mobile Devices

Yongjian Fu, Lanqing Yang, Hao Pan, *Member, IEEE*, Yi-Chao Chen, *Member, IEEE*, Guangtao Xue, *Member, IEEE*, and Ju Ren, *Senior Member, IEEE*

***Abstract*—Various characteristics of mobile applications (apps) and associated in-app services can reveal potentially-sensitive user information; however, privacy concerns have prompted third-party apps to restrict access to data related to mobile app usage. This paper outlines a novel approach to extracting detailed app usage information by analyzing electromagnetic (EM) signals emitted from mobile devices during app-related tasks. The proposed system, MagSpy, recovers user privacy information from magnetometer readings that do not require access permissions. This EM leakage becomes complex when multiple apps are used simultaneously and is subject to interference from geomagnetic signals generated by device movement. To address these challenges, MagSpy employs multiple techniques to extract and identify signals related to app usage. Specifically, the geomagnetic offset signal is canceled using accelerometer and gyroscope sensor data, and a Cascade-LSTM algorithm is used to classify apps and in-app services. MagSpy also uses CWT-based peak detection and a Random Forest classifier to detect PIN inputs. A prototype system was evaluated on over 50 popular mobile apps with 30 devices. Extensive evaluation results demonstrate the efficacy of MagSpy in identifying in-app services (96% accuracy), apps (93.5% accuracy), and extracting PIN input information (96% top-3 accuracy).**

***Index Terms*—Electromagnetic signal, mobile application usage, privacy.**

I. Introduction

MOBILE devices have become an increasingly integral part of our everyday lives. The way individuals use mobile apps is heavily influenced by their personal interests and preferences, leading to a wide range of usage patterns within the same application. As illustrated in Fig. 1, app usage behavior can provide a unique insight into users' privacy profiles. Many third-party app developers and service providers depend on this data for various purposes, such as offering personalized services and targeted advertising recommendations [1], as well as predicting human mobility patterns [2]. However, growing concerns over the potential for private information leaks have led third-party applications to reduce their access to mobile app usage data. This includes system-kernel information such as the /proc filesystem, battery statistics, and internet traffic data on both Android and iOS devices [3]. Although data from cellular network providers could serve as an alternative source for app usage information, strengthened security protocols within app services are making it progressively more challenging to extract such data. In response to these concerns, governments are implementing regulations to protect user privacy, which further restrict third-party access to data with user labels.

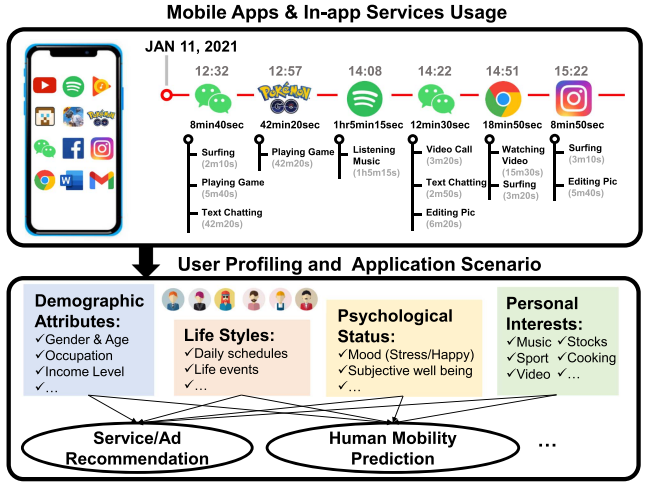


Fig. 1. Extraction of potentially-sensitive user information through analysis of detailed mobile app usage.

Several studies have demonstrated how data from the operating system kernel can be exploited to deduce information about running app and usage behaviors. This includes analyzing sources such as network traffic statistics [4], [5], [6], power consumption traces [7], [8], CPU utilization [9], [10], memory usage statistics [11], [12], and other information available via the procfs pseudo filesystem [13] or system APIs [14]. Developers of mobile operating systems, such as Android and iOS, have attempted to thwart such attacks by limiting access to sensitive system-kernel data resources. In addition, a number of researchers have explored the possibility of using electromagnetic (EM) signals to infer the launch of specific apps [15], [16], [17], [18], [19]. More specifically, these schemes exploit magnetometer readings to monitor CPU activity without

Received 15 July 2024; revised 1 November 2024; accepted 5 November 2024. Date of publication 11 November 2024; date of current version 5 February 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62341201, Grant 62122095, Grant 62432004, Grant 61936015, and Grant 62072306, and in part by grant from the Guoqiang Institute, Tsinghua University. Recommended for acceptance by J.S. Sun. *(Corresponding author: Hao Pan.)*

Yongjian Fu is with the School of Computer Science and Engineering, Central South University, Changsha 410017, China (e-mail: fuyongjian@csu.edu.cn).

Lanqing Yang, Hao Pan, Yi-Chao Chen, and Guangtao Xue are with the College of Computer Science and Technology, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: yanglanqing@sjtu.edu.cn; panh09@sjtu.edu.cn; yichao@sjtu.edu.cn; gt_xue@sjtu.edu.cn).

Ju Ren is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100190, China (e-mail: renju@tsinghua.edu.cn).

Digital Object Identifier 10.1109/TMC.2024.3495506

requiring permissions, enabling the launch of fingerprinting applications from scratch (via a "cold start") on both laptops and smartphones. In real-world scenarios involving mobile devices, the launching of apps varies. Apps launched in a "hot-start" manner (reloading data already residing in memory) result in EM signals with fewer characteristics, making it difficult to identify apps using the same methods described in [15]. Moreover, systems designed to detect the launching of mobile apps are unable to track the detailed time durations of active apps with which the user is closely interacting in the foreground or identify apps running in the background. Furthermore, much of the information required to infer personal behavior and preferences pertains to in-app services usage [20].

This paper proposes a novel system based on the EM side channel for the characterization of the continuous mobile app usage behavior as well as interactive information (e.g., PIN input) in real time. The proposed system, referred to as MagSpy, delineates app functionality across three levels: app identification, in-app activity, and keystroke detection. Specifically, this system can determine: 1) when a user is engaging with a specific app, 2) the duration of app usage and the features being accessed, and 3) the passwords entered for app access or mobile payments. Consequently, MagSpy enables an attacker to discern a user's interests, habits, and even their passwords.

Developing MagSpy imposes a number of daunting challenges: **(1)** The built-in magnetometer captures EM signals associated with running apps from the target device as well as fluctuations in the surrounding geomagnetic field. Thus, our first challenge is to extract EM signals from noisy magnetometer readings. **(2)** Mobile devices handle multiple app tasks simultaneously, both in the foreground and background. The background apps generate significant noise, complicating the task of identifying the specific app in use. Furthermore, many apps offer various in-app services (e.g., surfing, editing pictures, watching videos, chatting, and playing games), and the unique configurations of numerous mobile devices lead to different EM patterns even for identical tasks. Thus, our second challenge is to accurately map the EM signals to the corresponding app and in-app activities while mitigating the impact of background app noise and characterizing device-specific EM patterns. **(3)** The electromagnetic variations caused by human actions (e.g., PIN input) are extremely subtle and intertwined with app behaviors. Therefore, our ultimate challenge is to detect and decouple human actions from complex electromagnetic fluctuations.

To tackle these challenges, we firstly develop a multi-layer perceptron (MLP) regression scheme that leverages 3-axis accelerometer and 3-axis gyroscope data for the fitting of geomagnetic field data in order to effectively negate geomagnetic signals induced by user movements or other external factors. Secondly, to accurately map the EM signals to the corresponding app and in-app activities while mitigating the impact of background app noise and characterizing device-specific EM patterns, we propose a weighted data augmentation method. This method enhances the training dataset by incorporating noise associated with background apps and services. Additionally, we design a Cascade-LSTM classification model to characterize in-app services and app types. This model reduces the effects of diminished characteristic information in EM signals caused by a hot start. It initially assigns labels to in-app services (e.g., watching videos, chatting, or surfing) and then aggregates these labels to accurately infer the associated apps. Preliminary experiments revealed that most mobile devices using a given operating system (OS) generate similar EM signals when performing the same tasks, suggesting that EM signals are primarily influenced by the software environment rather than internal hardware components. Consequently, we train two classification models specifically for each of the mainstream mobile OSs (Android and iOS). Finally, a peak detection algorithm based on the continuous wavelet transform (CWT) and Random Forest (RF) classifier are combined to recognize the peak signal associated with keyboard opening and deduce important key strokes (e.g., PIN). Extensive experiments demonstrate the effectiveness and generalizability of the proposed scheme when applied to a diversity of devices, apps, and users.

The main contributions of this work are as follows:

- The proposed MagSpy system is able to simultaneously inferring fine-grained mobile app usage information and sensitive user actions (such as PIN input) continuously through readings from the built-in magnetometer, all without requiring user permissions.
- We develop a highly elaborate time-series classification algorithm (i.e., Cascade-LSTM) to classify running app and the corresponding in-app services.
- We design a peak detection algorithm based on CWT to identify EM signals triggered by changes in screen color during keystrokes, which are then analyzed to extract detailed PIN input information using a Random Forest classifier.
- Extensive evaluations on Android and iOS devices demonstrate the efficacy of the proposed MagSpy system in accurately identifying the apps being used as well as the in-app services and PIN input information in an energy efficient manner.

II. Preliminary Analysis

Preliminary experiments are conducted to answer four primary questions: i) Does the manner in which an app is launched affect the corresponding EM patterns? If so, can all of the EM patterns be distinguished? ii) What are the characteristics of the EM signals generated when using specific apps involving different in-app services? iii) What other factors impact magnetometer readings? iv) How to decouple subtle keystroke features from complex mixed signals? Our answers to the above questions demonstrate the feasibility of using EM data to infer details pertaining to the detailed app usage. They also help to identify some of the challenges that would be involved in further developing this technology.

A. EM Signals From Mobile Devices

We install a proprietary app on the testing devices to enable the continuous collection of the built-in magnetometer readings in the background at the maximum sampling rate. The magnetometers built into mobile devices are 3-axis Hall-effect sensors, which measure the strength of the surrounding magnetic field along the x-, y-, and z-axes. Note that in this work, we use

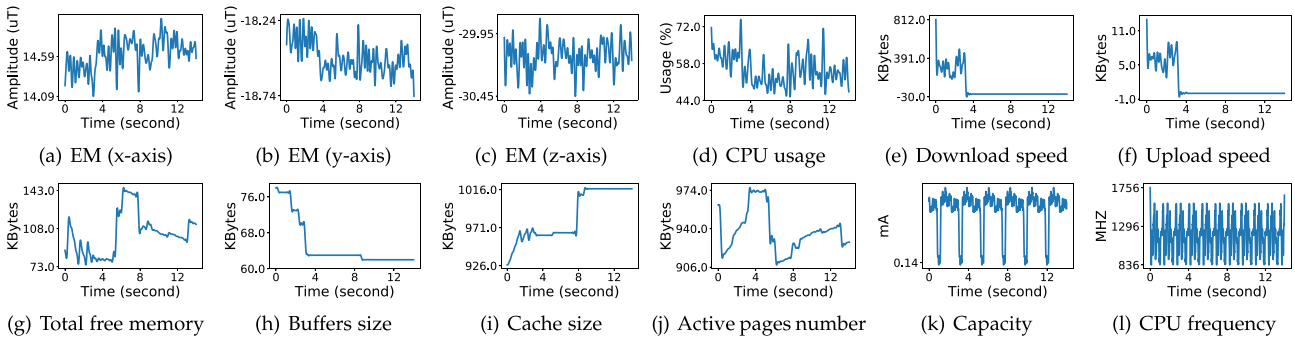


Fig. 2. Signals of magnetometer data and working states of different components when downloading a file with the screen off.

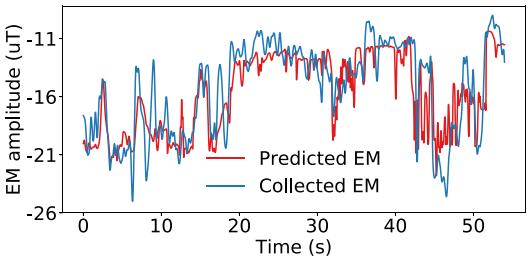


Fig. 3. Comparison between the composite signals of various electronic components and the real signals.

the total magnetic field strength as in [15]. The formula used to calculate the total magnetic field strength $M_T(t)$ is as follows: $M_T(t) = \sqrt{m_x(t)^2 + m_y(t)^2 + m_z(t)^2}$, where $m_x(t)$, $m_x(t)$, and $m_x(t)$ represent magnetometer readings of the three axes.

We next analyze how to characterize the working states of electronic components using EM signals. Using Android Debug Bridge (ADB) [21], we collect EM signals and system kernel data, including CPU usage, frequency, and memory statistics. Since ADB lacks screen data, the smartphone screen remains off. EM signals and system data are recorded over 10 minutes while downloading a file, with Fig. 2 showing 15 seconds of this data. The results show no direct link between EM signal patterns and individual components like the CPU or Cache. Instead, the EM signals reflect a mix from all onboard components, excluding the screen. We then use Ridge Regression [22] to model the EM signals, incorporating data from various scenarios like playing music or downloading files. The relationship for the EM x-axis data can be expressed as follows:

$$\begin{aligned} EM_x = {} & 0.33CPU_{load} + 0.12Buffer - 0.08Rx \\ & + 0.15Tx + 0.31Mem + 0.25Active \\ & + 0.01Cap + 0.03CPU_{freq} + 0.10Cache \end{aligned} \quad (1)$$

The regression results in Fig. 3 show that these components can be used to reconstruct the EM signals.

TABLE I
CLASSIFICATION RESULTS OF EM SIGNALS GENERATED DURING COLD-/HOT-START LAUNCH OF TEN APPS

	kNN	LDA	SVM	MLP	RF
Cold	89.7%	93.5%	93.7%	95.6%	94.9%
Hot	11.67%	12.92%	13.37%	16.14%	15.72%

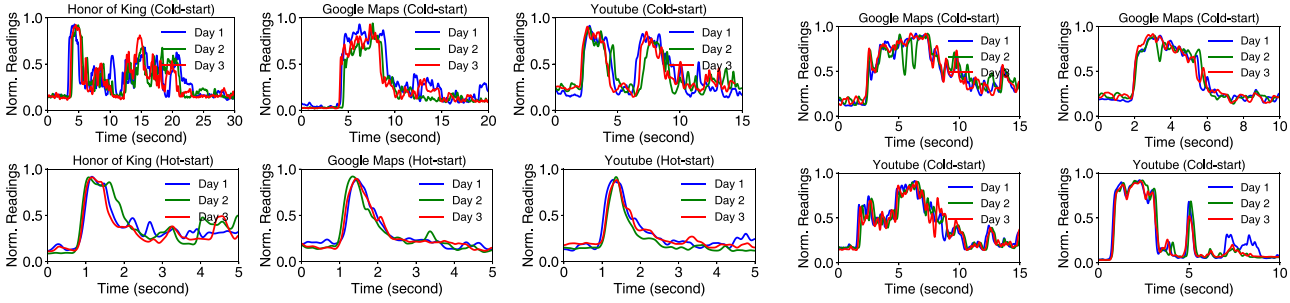
B. EM Signals Associated With Launching an App

We first record EM signals generated during the launch of 10 apps under cold-start (i.e., from the hard disk) and hot-start (i.e., from the "Recent Apps" list) conditions. As shown in Fig. 4(a), the EM patterns exhibit strong spatial and temporal consistency, varying only based on the launch method. This is due to the fact that cold-start requires full app initialization, whereas hot-start involves reloading data already present in memory.

We apply the feature extraction method from [15] to 20 EM signal traces collected from each of the 10 apps under both cold-start and hot-start conditions. Various time-series classification algorithms are then used to assess the distinctiveness of the EM patterns, with results shown in Table I. The initial classification results for EM signals during hot-start are unsatisfactory, barely surpassing random guess accuracy. Thus, the EM side channel provides limited information on hot-start app launches. These findings highlight the need for more efficient feature extraction methods to identify app usage during hot-start.

C. EM Signals Across Devices

The cold-start launch of an app involves the execution of a fixed code set, which can be used to facilitate the exploration of similarities and differences in EM patterns emitted from different mobile devices. Thus, we record the magnetometer readings generated by another two smartphones while the same version of the same app is launched, the results of which are shown in Fig. 4(b)–(c). We find that devices with the different OSs produced totally different EM patterns when performing the same app task, while devices with the same OS produced similar EM patterns. For example, although the corresponding apps in Fig. 4(b) and (a) come from different devices, their waveforms are very similar due to using the same OSs. Furthermore, we use Dynamic Time Warping (DTW) to compute the similarity



(a) Using Oppo Reno4 Pro (b) Using Huawei P20Pro (c) Using iPhone 7Plus

Fig. 4. EM patterns obtained from three smartphones while launching different apps in cold- or hot-start manners.

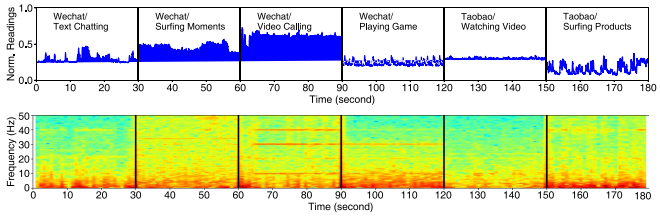


Fig. 5. Magnetometer readings and corresponding spectrograms generated while using various apps and in-app services.

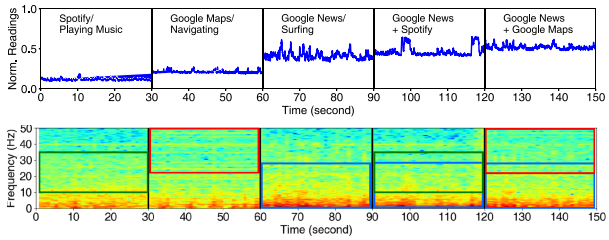


Fig. 6. Magnetometer readings and corresponding spectrograms generated while using multiple apps simultaneously.

of EM signals across different devices running the same apps. The similarity is defined using the Similarity $= 1 - \frac{d_{\text{dtw}}}{\max d_{\text{dtw}}}$. The similarity value is between 0 and 1, and the higher the value, the more similar it is. The results show that devices with the same OS have an average similarity score of 0.73, while devices with different OSs only have an average similarity score of 0.18. This prompts us to train two classification models for each of the mainstream mobile OSs (Android and iOS). Note that the two models share the same architecture, but their parameters are different.

D. EM Signals Involving In-App Services

EM signals also vary when use different in-app services in the same app. Fig. 5 presents EM patterns and the corresponding spectrograms generated while using WeChat (a social/communication app) and Taobao (a shopping app). The resulting EM patterns show that the functional differences between the apps result in different EM patterns. We can also see in Fig. 6 that the internal EM patterns can be distinguished according to the in-app services. Taken together, these results indicate that the EM side channel leaked information pertaining to app usage behavior as well as in-app services.

E. EM Signals Associate With Execution of Multiple Apps

Multi-window schemes (e.g., split screens) allow multiple mobile apps and services to run simultaneously. For example, a user can play a game in one window while browsing the web in another. However, the multi-window mechanism does not change the app activity lifecycle; only the most recently interacted app remains active (RESUMED state) at any time [23], [24]. This means apps not being actively used do not produce foreground EM interference. Background apps (e.g., music/video playback, downloads, or navigation) generate EM signals, which may interfere with those from active foreground apps. As shown in Fig. 6, the EM patterns and spectrograms from multiple running apps resemble the superposition of EM signals from each app, especially in the frequency domain. The green/red/blue boxes in the spectrogram (Fig. 6) highlight features for music playback, navigation, and web browsing, respectively. This suggests the possibility of separating complex EM signals into those associated with individual apps or in-app services.

F. Other Factors Affecting Magnetometer Readings

1) Device Movement: Magnetometers in mobile devices sense geomagnetic signals for electronic compasses. Changes in device position or orientation affect the magnetometer readings. Fig. 9 shows total magnetometer readings during user walking. The blue line represents geomagnetic signal changes due to user movement, while the red line shows EM signals generated by the smartphone during app tasks. We observe that the geomagnetic signal changes caused by movement exceed the EM signal amplitude, indicating a significant impact on data preprocessing (e.g., normalization). The dashed blue line in Fig. 9 represents predicted geomagnetic field signals (see Section IV-A), which can be used for geomagnetic fluctuation cancellation.

2) EM Noise From External Electrical Devices: Other electrical devices (e.g., household appliances) also leak EM emissions at levels that cannot be disregarded out of hand [25]. Theoretically, EM intensity drops off exponentially (rather than

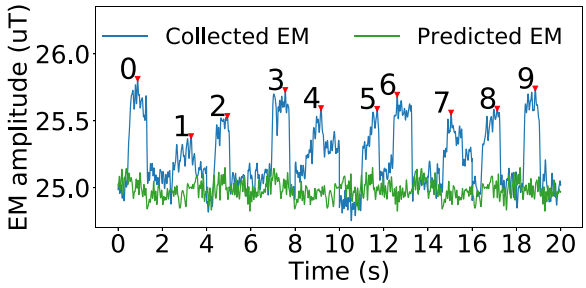


Fig. 7. EM signals generated by input of 10 PINs.

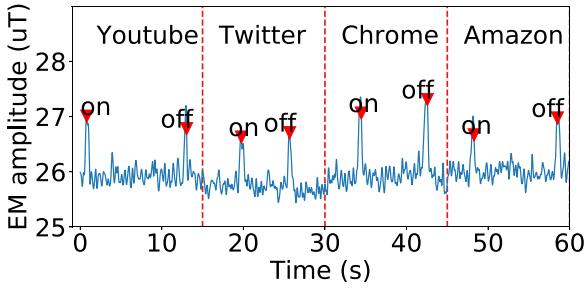


Fig. 8. EM signals generated when keyboard is opened or closed in four apps.

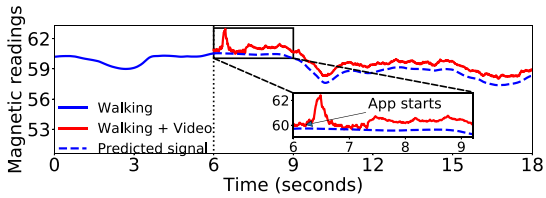


Fig. 9. Magnetometer readings obtained from mobile device while the user was walking with and without a video app playing.

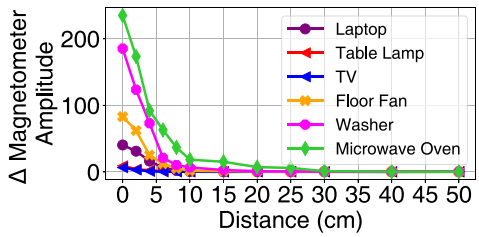


Fig. 10. Changes in amplitude of magnetometer readings at various distances from appliances.

linearly) with an increase in distance. Fig. 10 presents a diagram plotting the attenuation of EM intensity as a function of distance. When the mobile device is ≥ 25 cm from large household appliances (i.e., the washing machine and the microwave oven), the built-in magnetometer is unable to detect the EM signals generated by these machines. In the case of appliances with low EM emissions (i.e., the table lamp and the television), 5 cm is sufficient to eliminate interference. Thus, unless the smartphone is in close proximity to an operating electrical device, external EM noise is negligible.

G. *Decoupling the Features of Keystrokes*

Inputting keystrokes, such as PIN, typically has a negligible effect on the working state of most onboard electronic components; that is, the corresponding EM signals are very weak. To test this hypothesis, we collect EM signals from the screen and other components while PINs are entered with the screen on. As illustrated in Fig. 7, PIN input slightly alters the amplitude of the EM signals. However, when we attempt to predict EM signals using the formula (1) derived from the screen-off state, we find that our predictions fail to capture any of the peaks associated with PIN input. This suggests that the screen itself holds substantial information useful for detecting PIN.

Considering that many input method apps change color upon pressing an on-screen button to enhance user interaction, we aim to detect the EM signals caused by these color changes across different screen areas to identify keystrokes. As depicted in Fig. 8, we initially detect the activation of the keyboard. As shown in Fig. 8, PIN composed of digits from 0 to 9 generated distinctive peaks, distinguishable from the noisy background. The observed variations in EM signals enabled us to decode the PIN.

III. Threat Model

In this section, we present the threat model of the proposed MagSpy system. We assume that the target victim has installed an app designed for seemingly innocuous purposes, like chatting or document editing. However, this app contains malicious code designed to run covertly in the background. For this type of attack, our analysis focuses on 50 mainstream apps available on Google Play (for Android devices) and the Apple App Store (for iOS devices). We envision the overall attack flow as follows:

Step 1: The user unknowingly downloads a chat app that serves as a facade. Embedded within this app is functionality that continuously reads data from the 9-axis IMU sensor–comprising a 3-axis magnetometer, 3-axis accelerometer, and 3-axis gyroscope–while running in the background. This occurs even as the victim uses other apps that may handle sensitive information. Notably, the app does not request permissions to access the IMU sensor data. The duration and frequency of this background data collection are dictated by the operating system's specific constraints and settings. This issue of limitations is detailed in Section VI-A.

Step 2: MagSpy collects historical EM signal data from the 50 most popular apps, updating the historical dataset promptly with each app version update. This data is utilized to train two OS-specific models that function as both app and in-app service classifiers. Additionally, MagSpy gathers EM signal data from PIN input to set parameters for CWT-based peak detection models and to train Random Forest classifiers. The user selects the OS information when installing apps, which means we can assume the OS type is known.

Step 3: The disguise app continuously collects IMU sensor data in real time and stores it locally to be processed when the mobile device is idle. After identifying the apps in use, their specific services, and PIN input details, MagSpy then uploads this data to the attacker's server.

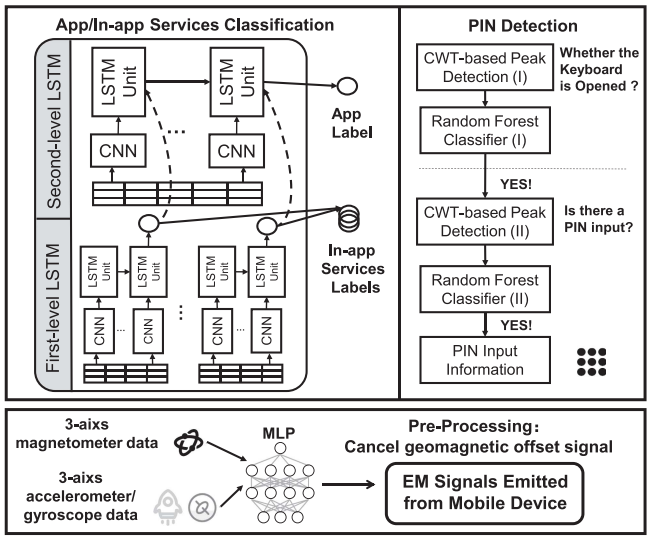


Fig. 11. The Processing pipeline of MagSpy with two sensors data as inputs and three privacy information as output.

TABLE II
THE RESULTS OF CANONICAL REGRESSION ALGORITHMS ON GEOMAGNETIC SIGNAL FITTING

	ARIMA	SARIMA	VAR	LR	RF	SVR	MLP
MSE	1.45	1.33	1.28	1.25	0.82	0.48	0.26

Mean square error (MSE) is the metric to evaluate how well the predicted EM fits the actual EM. The smaller MSE the better.

IV. SYSTEM DESIGN

In this section, we present a comprehensive overview of MagSpy, followed by technical details of each step. Fig. 11 depicts the processing pipeline of the proposed system. Initially, in the preprocessing phase, data from the 3-axis accelerometer and 3-axis gyroscope are utilized to adjust for the effects of geomagnetic variations. Subsequently, the Cascade-LSTM model is employed to identify the current in-app services, which aids in classifying the mobile app. Concurrently, MagSpy detects keyboard opening actions in real time in order to monitor PINs input information.

A. Cancelling the Geomagnetic Offset

We first seek to resolve offset in the geomagnetic noise caused by external factors, such as the user walking or changes in wrist position. To characterize the influence of user movement, we use the additional information of a 3-axis gyroscope and 3-axis accelerometer. The output data is then evaluated using a number of regression algorithms: Autoregressive Integrated Moving Average (ARIMA) [26], Seasonal ARIMA (SARIMA) [27], Vector Auto Regression model (VAR) [28], Logistic Regression (LR) [29], Random Forest Regression (RF) [30], Support Vector Regression (SVR) [31] and Multi-layer Perceptron [32], the results of which are listed in Table II. MLP is a standout among the seven methods based on its ability to capture linear as well as nonlinear relationships, which made it particularly well-suited to the interpretation of user movement and other external influences. Consequently, we employ the MLP model to convert the data from the 3-axis accelerometer and 3-axis gyroscope into meaningful geomagnetic field data. The MLP network is structured with an input layer, four hidden layers (consisting of 64, 64, 32, and 16 neurons, respectively), and an output layer. Fig. 12 displays the real collected signals and the geomagnetic signals synthesized with MLP.

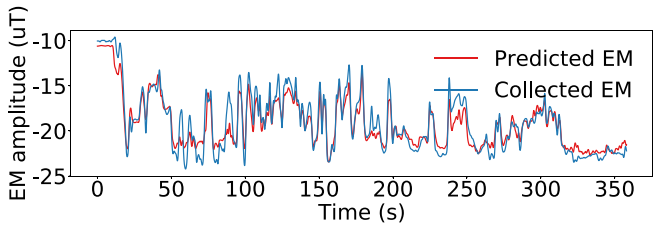


Fig. 12. Fitting results from accelerometer and gyroscope through MLP vs. raw geomagnetic field data (x-axis).

After establishing the MLP model, we partition the EM signal data into discrete time intervals utilizing a sliding window of size w. For each segment, we mitigate the geomagnetic field offset by deducting the predicted geomagnetic noise from the total EM data recorded by the magnetic sensor. This process yields a purified EM signal emanating directly from the computing device. Subsequently, this refined signal is subjected to Gaussian filtering and normalization to ensure its suitability for analysis. These preparations ready the signal for progression to the subsequent phase of processing.

B. App/In-App Services Classification

1) In-App Services Clustering: While users display varied app usage patterns, their activities are constrained by the available in-app services. To categorize these services broadly, we conduct cluster analysis on EM signal datasets from 50 apps. The initial step involves gathering data from diverse app interactions such as playing games on a browser, watching videos, listening to music, etc. Following data compilation, we employ the TSkmeans algorithm [33] for time-series clustering. Unlike traditional clustering techniques like K-Means [34], TSkmeans utilizes analytically derived iterative updating rules to optimize an objective function, which enhances cluster search efficiency. This method also assigns weights to different timestamps, improving the clustering of time-series data by addressing the limitations of relying solely on spatial distances. The extensive historical data available in this study allows us to identify nine distinct categories of in-app services, as depicted in Fig. 13. For further precision, we manually label portions of the dataset based on classification criteria similar to those used by Google Play and the Apple Store. This aids in pinpointing the nearest cluster centers. Each cluster center is named according to the derived data, and these names serve to label the in-app services. Essentially, these clustering labels are then applied to reclassify the original data segments, facilitating the training of the model used to categorize in-app services. The outcomes of this classification process are detailed in the subsequent section.

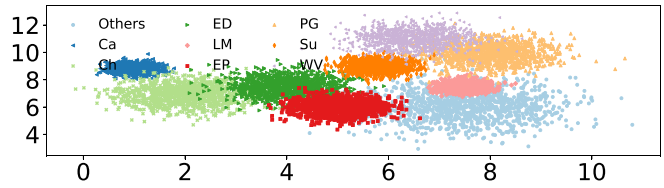


Fig. 13. Clusters related to nine types of in-app service. The notation is as follows: video/voice calls (Ca), text chatting (Ch), editing documents (ED), listening to music (LM), editing photos (EP), playing games (PG), surfing/reading (SU), watching video (WV), and Others.

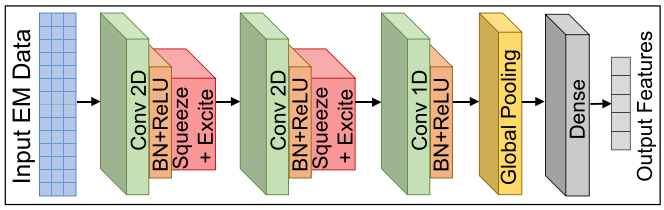


Fig. 14. CNN Structure.

2) Classifier Structure: Assume that we have obtained the OS information of the attacker's mobile device, then it is necessary to select the corresponding app/in-app services classification model. We develop a dual-level Cascade-LSTM method for the classification of app and their in-app services. Including the window length of the input EM signal allows this system to handle the dynamics associated with human usage habits. The proposed Cascade-LSTM framework is illustrated in Figrue 15. This scheme includes two parts: 1) first level, extraction of in-app services representations from each M-second time interval of EM signals, and 2) second level, modeling of in-app services over $M \times N$ seconds and app type.

Feature extraction and classification are key to resolving classification problems, especially considering the reduced signal characterization caused by hot-start. In our models, we apply a layer for the extraction of similar features (based on CNN) and a classification layer (based on LSTM), which are implemented in two levels. We select CNN for feature extraction based on its ability to capture local discriminative features. The five feature vectors are then successively fed into the LSTM network (i.e., in five separate steps). Finally, the LSTM network outputs in-app service labels. We obtain N in-app service labels for each M-second interval from EM signals of $M \times N$ seconds. In the second level, the same EM signal of $M \times N$ seconds is divided into N parts, which are fed into CNN to extract feature vectors one by one. The corresponding in-app service labels associated with each one-second time interval from the first level are concatenated with each feature vector and then fed into the LSTM network (in N steps), whereupon the LSTM network outputs the app label.

3) Feature Extraction Layer: It is typical for a basic CNN to employ a convolution layer in conjunction with batch normalization. The key attribute of a CNN is the ability to alternate

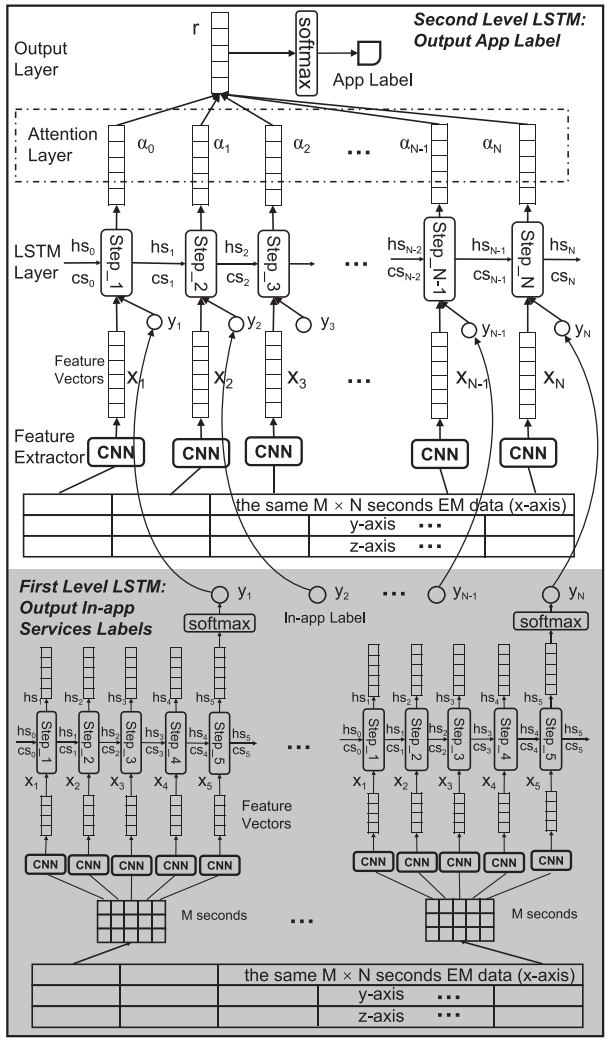


Fig. 15. Structure of Cascade-LSTM.

between different processing units (e.g., for convolution, pooling, sigmoid / hyperbolic tangent squashing, rectification, and normalization). This ability makes CNN easier to obtain a better representation of salient signals. Moreover, the deep architecture makes it possible to stack multiple layers of processing units to facilitate the characterization of salient signals at various scales. The features extracted using CNN are task-dependent and highly discriminative. Therefore, CNN can efficiently extract representative features from EM signals with a high sampling rate and large noise.

The details of the our CNN model are shown in Fig. 14. In each CNN layer, asides with the basic Convolutional-2D structure, batch normalization and activation function of ReLU are used to train deeper network without suffering from the vanishing gradient problem [35]. After a global pooling layer, all features are joined together and fed to a dense layer to generate more representative features. In our CNN design, we apply a set of 1D filters on the EM signals directly and abandon the Pooling and Dropout layer in the network to retain the characteristic

of EM signals [36]. Given M seconds of 3-axis EM signal sequence $x = [x_1, x_2, \ldots, x_M]$, 1-sec 3-axis EM signal x_i is transformed to the CNN network in each time step. The output of the convolutional layer can be written as:

$$C_r(t) = f\left(\sum_{i=1}^{l}\sum_{j=1}^{k} x(i + s(t-1), j)w_r(i,j) + b(r)\right)$$

where $C_r(t)$ represents the t^{th} component of the r^{th} feature map, s denotes the convolution stride, and w_r and $b(r)$ denote the weights and bias of the r^{th} convolutional filter [37].

One of the problems in CNN network is that the features extracted from all filters of the previous layer share the same contribution to the next layer, which can impair classification tasks that are sensitive to feature importance, especially in time-series classification. To address the problem, we apply Squeeze-and-Excitation Network (SE-Net) to our network. SE-Net can adaptively recalibrate channel-wise feature responses by explicitly modeling interdependencies between channels [38]. Therefore, we use it to extract important feature maps and emphasize the representation of the network.

SE-Net is composed of the Squeeze operation and the Excite operation. The Squeeze operation is computed by:

$$z = \frac{1}{H \times W}\sum_{i=1}^{H}\sum_{j=1}^{W} u(i,j)$$

where z represents the target statistics, H and W represent the height and width of the filter, and u represents the features passed through a squeeze operation. And the excite operation is computed by:

$$s = \sigma(W_2\partial(W_1 z))$$

where ∂ refers to the ReLU function, W_1 and W_2 are parameters to limit the model complexity, and σ is a general activation function (e.g., sigmoid, ReLU, tanh, and etc). Finally, SE-Net outputs $s \odot u$, where $\odot$ is the element-wise multiplication. Using this structure, SE-Net enables the CNN network to capture channel-wise dependencies and generate more representative features.

4) Classification Layer: The LSTM networks have been shown to model temporal sequences and their long-range dependencies more accurately than original RNN model. In this part, we apply LSTM network to our classification model.

LSTM layer treats X_i as input. The input of LSTM model at time step i is the output at time step $i-1$ along with new input at time step i. At each time step, LSTM maintains a hidden vector hs and a memory vector cs responsible for controlling state updates and outputs as shown in Fig. 15, which can be represented as:

$$hs_i = LSTM(X_i, hs_{i-1}, cs_{i-1})$$

In each step, hs is sensitive to the input, while cs changes slowly to maintain a long memory. This structure enables LSTM to delivery long-time dependence of EM signals among different time steps, which breaks through the limitation that CNN can only extract local properties.

After a many-to-one LSTM layer and a softmax layer, $y = softmax(hs_N)$ serves as the output of the first level (in-app service classification) and further is added to the feature vectors of the second level (in this case, N=5).

For the second level (app classification), since different time steps may exhibit different significance, while LSTM is known to have difficulty in dealing with long term dependencies in long sequences [39]. To address the problems, we apply attention mechanism to leverage the important features of each time step.

An attention mechanism takes LSTM output sequence hs_i as input and computes a vector $\alpha = [\alpha_1, \alpha_2, \ldots \alpha_T]$ where T is the length of vector hs_i. [40] , which can be represented as:

$$\alpha_t = \sum_{i=1}^{N} f(hs_i)$$

where f is a weighted function trained with all the other components of the model. Then $r = \alpha \odot hs_N$ serves as the output of LSTM with Attention layer. Since α_t integrates information among different time steps, this structure acts as a feature filter unit and can further enhance the representation ability of the network.

5) Mitigate the Diversity of Users: As mentioned previously, users differ in the way they use an app. In fact, when time interval M is sufficiently small, even the EM signals from users performing the same in-app service may differ due to the differences in usage habits, such as typing speed. Thus, the interval M must be extended sufficiently to mask the differences in the using behavior of most users. In this work, we increase time interval M to enable the Cascade-LSTM model to mitigate diversity in the behavior of users.

6) Data Augmentation for Background App Noise: The scale of datasets on noise from background apps is limited by computational overhead. Thus, we develop a data augmentation scheme to avoid over fitting and improve generalizability. [41] shows that when multiple CPU cores work simultaneously, the EM signal is the superposition of that when individual CPU works. Thus, we select the EM signal, B_j, of a number of apps/services that tend to run in the background, such as music players, file downloaders, navigation apps and various app background services. This EM signal is added with the EM signal of other apps X_i with a particular weight $\alpha_{i,j}$, as follows: $\hat{X}_i = X_i + \alpha_{i,j} \times B_j$. The artificial app EM signal with noise from background apps (i.e., $\hat{X}_i$) is then included in the dataset to improve model performance.

C. Keystroke and PIN Input Detection

The PIN detection algorithm is designed to accurately identify and classify PIN inputs from electromagnetic (EM) signal data by leveraging a two-stage peak detection and classification approach. The primary goal is to ensure high precision in detecting the specific moments when a keyboard is operated and accurately identifying the input PIN values.

As described in Algorithm 1, the process of PIN detection comprises two stages. In the first stage, peak-shaped signals are identified using a Continuous Wavelet Transform (CWT)-based peak detection algorithm [42]. For each detected peak,

100 samples before and 100 samples after the peak's maximum point are captured, resulting in a total of 200 samples. These samples are analyzed using a Random Forest classifier, which determines whether the peak EM signal is associated with the opening of the keyboard.

If the classifier confirms that the peak indicates the opening of the keyboard, we proceed to the second stage. Here, another round of peak detection is performed using the CWT-based algorithm with adjusted parameters, such as prominence degree, minimum distance between peaks, maximum peak widths, and peak thresholds. For each newly detected peak, 50 samples before and 50 samples after the peak's maximum point are captured. The captured samples are then processed by a second Random Forest classifier with different hyperparameters, including the number of estimators, tree depth, maximum features per split, and maximum leaf nodes. This classifier identifies individual PIN input values, ranging from 0 to 9.

By employing distinct parameter settings and classifier configurations across the two stages, the system effectively distinguishes between different types of peaks and their associated PIN inputs, thereby enhancing the robustness and accuracy of the PIN detection process.

Algorithm 1: PIN Detection.

Input: EM signal sampling list $X = \{x_1, x_2, \ldots, x_m\}$, each subset X' is a subset of X, window size d
Output: Result list D containing the predicted value of each PIN input

while *samples in* X' *larger than* p **do**
  Fetch subset X' from X and remove p samples from X;
  Run $CWTAlgorithm_1$ on X' and get peaks $PEAK$;
  if $PEAK$ *exists* **then**
    for *each peak* c *in* $PEAK$ **do**
      Generate list Y containing 100 values around c;
      Apply $RandomForest_1$ to Y to determine whether c is an operation of opening the keyboard;
      if c *is an operation of opening the keyboard* **then**
        while *true* **do**
          Fetch another peak d in $PEAK$;
          if d *does not exist* **then**
            Fetch subset X' from X and remove p samples from X;
          Generate a list Z containing 100 values around d;
          Run $RandomForest_1$ on Z to determine whether d is an operation of closing the keyboard;
          if d *is not an operation of closing the keyboard* **then**
            Generate a list U containing 50 values around d;
            Run $RandomForest_2$ on U to obtain a PIN value pin between 0 and 9;
            Insert pin into list D;
            Delete d from $PEAK$;
          else
            Delete d from $PEAK$;
            break;
return Result list D

V. Evaluation

A. Experiment Setup

Apps and Devices: Table III lists the 50 most popular apps on the Google Play Store and Apple Store, spanning a variety of representative categories. The experiments are conducted on 30 mobile devices, detailed in Table IV. These devices include 14 Android phones, 6 Android tablets, 7 iPhones, and 3 iPads.

Participants: We recruit 12 individuals (5 females) ranging in age from 11 to 48 years to represent victims in our experiments.

Data Collection: We deploy an application on mobile devices that continuously gathers data from the magnetometer, accelerometer, and gyroscope in the background. The maximum sampling rate for these sensors varies depending on the device model. Initially, we collect data from these sensors at their maximum available sampling rates specific to each device. Subsequently, we uniformly resample the data to 100 Hz to standardize it for further analysis.

i) In-app services data collection: Our study focuses on nine in-app services identified through clustering results. Participants interact with apps that include at least one of these services for a duration of 10 minutes, conducting sessions in various locations and at different times.

ii) App data collection: Each participant is randomly assigned to use one of the apps on three different devices. These sessions are conducted in various locations and at different times to ensure diverse data collection.

iii) PIN input data collection: 12 participants are instructed to enter a series of 10 digits (ranging from 0 to 9) in random order using five different input methods. These include interactions with the device's screen in portrait orientation using the left and right thumb, as well as the left and right index finger, and in landscape orientation using the right thumb. Each input session lasts one minute and takes place in various locations at different

TABLE III
Top 50 Popular Apps on Mobile Devices

Category	App List (both on Android and iOS)
Video & Music	Youtube, Netflix, Pandora, Hulu, Tiktok, Bandcamp, Spotify, Apple Music, Google Play Music, Netmusic
Social	Instagram, Twitter, Facebook, Snapchat, Facebook Messenger, WeChat, WhatsApp, Reddit, Tumblr, LinkedIn
Game	Shadowgun Legends, PUBG, Pokemon GO, Dragon Ball Legends, Madden NFL Mobile Football, Call of Duty, Clash of Clans, Minecraft, Clash Royale, King of Honor, Toto Chess
Browser	Chrome, Firefox, Microsoft Edge
Productivity	Microsoft Word, Any.do, Notebook, Evernote, Todoist, Rivet
Others	Amazon, Taobao, Wish, Fordeal, Adobe Lightroom, NYTimes, Uber, MyFitnessPal, Google Map, Careem

TABLE IV
30 MOBILE DEVICES USED IN THE EXPERIMENTS

Category	Device List
Android phones	Google Pixel 2, Huawei Nexus 6P, Huawei P10, Huawei P30Pro, Huawei Honor 9, LG G4, LG Nexus 5X, Motorola Nexus 6, Samsung Galaxy A7, Samsung Galaxy S7, Samsung Galaxy S8, Xiaomi Redmi K30, Xiaomi Mi 8, Xiaomi Mi 6
Android tablets	Samsung Galaxy Tab S4, Samsung Tab A T510, Huawei MediaPad 8.4, Huawei Pad M5, Lenovo Tab 4 10 Plus, Xiaomi Pad 4
iPhones	iPhone 11, iPhone XS Max, iPhone X, iPhone 8, iPhone 7 Plus, iPhone 6S, iPhone 6
iPads	iPad Air 2019, iPad Mini 3, iPad Pro 12.9

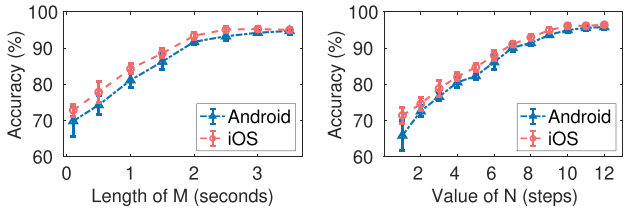
times of the day. After each session, the input digits are manually labeled to establish a baseline for accuracy. This procedure is repeated multiple times, with each repetition considered a separate sample for inclusion in the PIN input dataset.

Data Augmentation: During our experiments, we enhance the training dataset through data augmentation by introducing artificial noise that mimics the EM signals from background apps. Specifically, we select apps categorized under Music, Browser (with download manager), and Navigation to serve as background apps. To assess the EM signals linked to the refresh services of these background apps, we keep the device screen on, activate the app's background services, and collect data from the magnetometer sensor. We then integrate these background app signals, with intensities ranging from 0.2 to 0.6, into the signals from other apps in our study. These artificially generated EM signals are accurately labeled according to their original categories and incorporated into the training dataset to enhance the model's ability to identify app signals in the presence of background noise.

B. Methodologies Evaluation

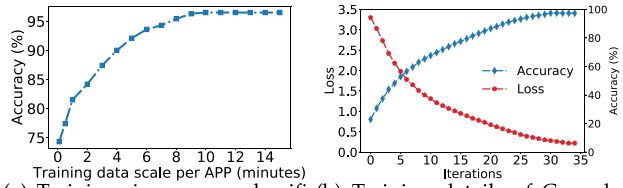
This subsection examines the effectiveness of our proposed Cascade-LSTM model for classifying apps and in-app services.

1) Time Interval vs. Classification Performance: Users typically exhibit significant variability in their behavior, even when interacting with the same app and services. Consequently, we adjust the window size M to accommodate these variations. We perform a 6-fold cross-validation, where datasets from 10 randomly selected users are used for training, and datasets from the remaining 2 users are designated for testing in each fold. Fig. 16(a) displays the outcomes when the app-internal service classification models are trained using various lengths of M. When M is set at 0.1 seconds, the in-app services classification model performs poorly, achieving only 70.8% accuracy. Gradually increasing M enhances performance synchronously; setting M at ≥ 2 seconds allows the classification model to achieve an average accuracy of 98% with a small standard deviation. This improvement is logical as a larger time interval size encompasses more generalized behavior information rather than specific discriminatory information. Therefore, for



(a) M in the first level. (b) N in the second level.

Fig. 16. Impact of the length of time intervals M and N .



(a) Training size vs. app classification accuracy (b) Training details of Cascade-LSTM

Fig. 17. Evaluation results of Cascade-LSTM.

subsequent evaluations, M is maintained at 2 seconds, based on the assumption that there is minimal variation in app-internal services over this duration. The window size N at the second level (app classification) also influences the performance of the proposed Cascade-LSTM model. Fig. 16(b) illustrates the results when M is fixed at 2 seconds and N varies between 2 and 16. Notably, when N is set at ≥ 8, the Cascade-LSTM model attains an average accuracy of 96% in app classification with a small standard deviation. Conversely, when the time step N is set at only 1, the model's performance suffers significantly, resulting in an average accuracy of only 60% and a large standard deviation. Increasing the window size N to 10 steps substantially improves performance, leading to an average accuracy of 96% with a very small standard deviation.

2) Training Size and Converge Length: We vary the length of training data of each app running on 30 devices, ranging from 1 minute to 10 minutes. The remainder of the dataset is used for testing. The time step N is set at 8, and the time interval M is set as 2. The results are presented in Fig. 17(a). When using only 4 minutes for each app, we attain accuracy of 94%. When using ≥ 5 minutes for each app, the accuracy range from 96.2% to 96.5%. This implies that when user use an unknown app out the range of app list of MagSpy and want to obtain the screen time proportion of this app in the future, he/she could collect 5-minute labelled EM data of this app and upload it to the host for re-training classification model.

When we use the whole 10-min training size, set the time step N at 8 and the time interval M at 2, Cascade-LSTM achieve the optimal training accuracy at 34 iterations, beyond which training loss continued to decrease but training accuracy did not increase any further. Thus, we halt training after 40 iterations to avoid over fitting. These results are detailed in Fig. 17(b).

3) Comparison With Baseline Methods: The proposed Cascade-LSTM is compared with six baseline classifiers (SVM, Random Forest, CNN, LSTM, FCN [39], DRCNN [43]). As

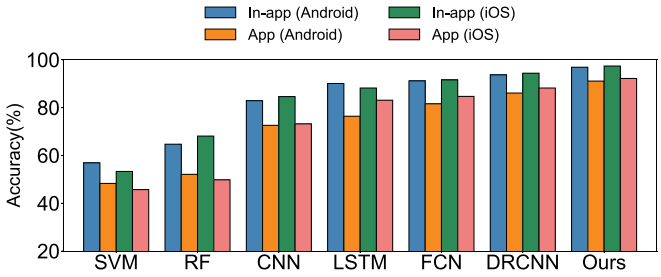


Figure 18: Comparisons with baselines.

Fig. 18. Comparisons with baselines.

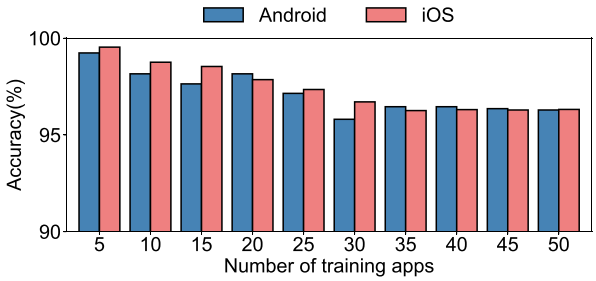


Fig. 19. Impact of number of training apps.

shown in Fig. 18, Cascade-LSTM demonstrates higher accuracy in the app/in-app services classification task compared to baseline algorithms, which can increase app classification accuracy to 96.5%. Compared to the method used in [43], our approach improves accuracy by 5.6%. We attribute this performance improvement to the superior capability of our method in extracting and leveraging temporal features from magnetometer signals.

4) Impact of the Number of Training Apps: Considering the thousands of app types in the market, we also sought to determine whether the increasing number of training apps would decrease the multi-label apps classification performance. As shown in Fig. 19, the identification performance of MagSpy did decrease with the the number of training apps increased, but it also began relatively slowing decreasing or relatively stabilizing when the number exceeded 25, regardless of applying to Android or iOS phones.

C. Robustness Against Geomagnetic Noise

We evaluate the effectiveness of the proposed geomagnetic offset signal cancellation method by eliminating the cancellation step (i.e., Step 1in Fig. 11) and testing our trained model in six real-world scenarios. The comparison results in Fig. 20 show that the inclusion of preprocessing greatly improves the stability and accuracy of app classification.

D. Robustness Against Background Noise

Fig. 21 shows the classification accuracy under the effects of background apps. We can see that the trained model without the data augmentation is unable to perform adequately while background apps are running. On the other hand, with the data

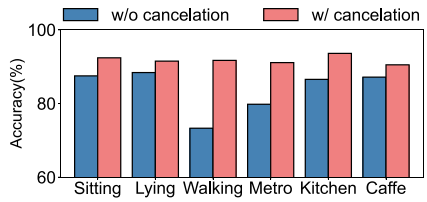


Fig. 20. Performance of w/ and w/o canceling geomagnetic noise.

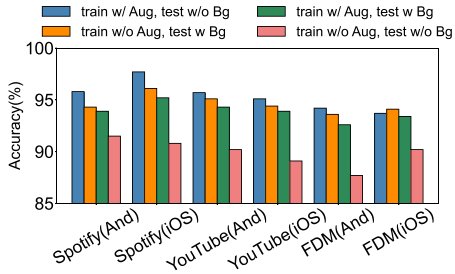


Fig. 21. Performance under background app interference. "Aug" indicates data augmentation; "Bg" indicates background app noise in testing data.

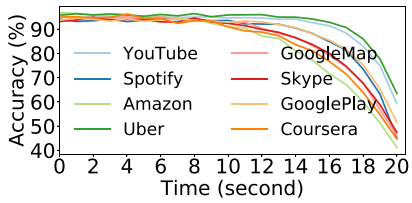


Fig. 22. Influence of users' extreme behaviors (e.g., continuous typing in game apps) on app classification accuracy in 8 different apps.

augmentation, when a background app is running, the performance of the proposed Cascade-LSTM dropped by less than 1%, regardless of the device. Overall, the proposed model is proved to be robust against background noise. These results also demonstrate that our proposed data augmentation technique helps to prevent overfitting and improve the generalizability of our model.

E. Users' Extreme Behavior

The specific habits of users could have a profound effect on the proposed system. For example, continuous typing while using a Gaming app would prompt MagSpy to cluster the data within the document editing category, thereby misclassifying the app as well as the in-app services. It is possible to adjust the windows of M and N (see Section V-B1); however, this would be impractical if the total typing time are close to $M * N$. As shown in Fig. 22, when users spend more than 15 seconds typing ($0.75M * N$) while using apps in the eight categories, app classification accuracy dropped to 80%. This demonstrates that MagSpy is subject to failure when it encounters non-standard user behaviors. However, in another user study involving three volunteers (Fig. 23), we record the time users spent continually

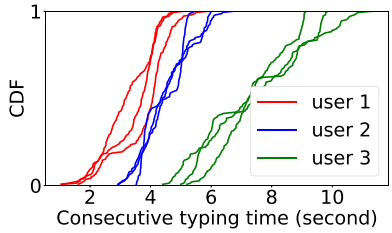


Fig. 23. CDF of continuous typing time in non-text editing apps.

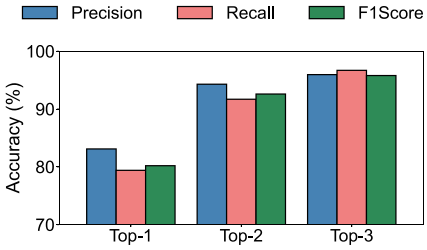


Fig. 24. The classification results of PIN detection. Top-1 to Top-3 results of 10 users are reported.

typing while using apps in a normal manner. Our results indicate that most users do not spend more than 10 seconds typing. This appears reasonable due to the fact that typing operations in gaming, music, or surfing apps is meant for immediate interactions. Thus, the proposed system proves robust to the habits of most users.

F. PIN Detection Performance

We also design experiments to evaluate the proposed PIN detection scheme. We begin by dividing the data from the 12 volunteers into a training set and a test set using the leave-one method; i.e., for every 11 volunteers' PIN inputs dataset used to train the PIN classification model, the remaining volunteer's PIN input dataset is included in the test set. For each test dataset, the top-K (K from 1 to 3) classification results are calculated in terms of average precision, callback, and F1-score. Fig. 24 presents the corresponding classification results. The proposed method is able to classify PINs correctly within 3 predictions. This is a clear indication of its effectiveness in the extraction of private data from EM signals. Furthermore, the fact that MagSpy is able to detect running apps as well as PIN input means that hackers would be able to deduce when a user is performing a particularly sensitive task (e.g., typing in the password of bank card) and extract that information for financial fraud. Notably, even with varying background illumination across different apps, signal differentiation techniques can be applied to isolate changes induced by PIN entry behaviors, thereby ensuring the stability of detection.

The sampling frequency of the built-in magnetometer may also affect the classification accuracy of our MagSpy. Fig. 25 lists the app classification accuracy and PIN detection accuracy using data obtained at different sampling frequencies. We can see that at $50 \sim 100$ Hz , the accuracy of the system is close to the theoretical upper bound, which means that MagSpy should

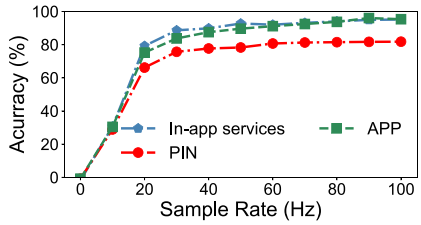


Fig. 25. Robustness against sampling rate.

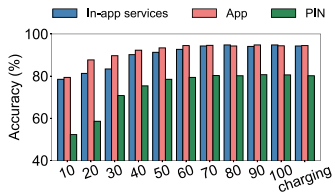


Fig. 26. Influence of battery.

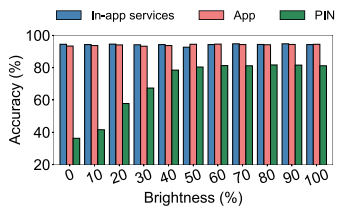


Fig. 27. Influence of brightness.

perform well on most devices. From another perspective, users or companies can protect user privacy by limiting the sampling rate of the magnetometer. Taking into account the performance of its basic functions (e.g., direction estimation) and privacy concerns, a threshold of 20 Hz is generally acceptable.

G. Other Factors

Many other factors could potentially affect the performance of the proposed system, such as the battery level and screen brightness (see Figs. 26 and 27). As shown in Fig. 26, a 90% decrease in battery power led to a 20.6% drop in in-app classification, 16.7% drop in app classification and a 27.3% drop in PIN detection. This is probably due to the fact that the EM signal depends on the availability of battery power (i.e., signal amplitude). When the battery is low, the phone automatically switches to power saving mode, in which the CPU operates at a lower frequency, videos are not preloaded, and background services are turned off. Under low brightness conditions, the accuracy of in-app classification ranged from 97.4% to 93.5%, app classification ranged from 96.2% to 92.8%, and PIN detection ranged from 83.2% to 33.9%, as shown in Fig. 27. This can be attributed to the fact that app/in-app classification does not depend only on screen-driven tasks, but rather focuses on CPU, storage and wireless communications modules. In contrast, PIN detection depends on changes in the colors in specific areas, such that a decrease in intensity leads to a corresponding decrease in signal strength.

H. Resource Consumption

We also think it prudent to evaluate the power consumption and required storage space of MagSpy, considering that it is tasked with the real-time collection of data from the magnetometer, accelerometer and gyroscope as well as complex data processing. We assume that the sampling rate was 50 Hz and that the user would take a break every hour. Under these conditions, the data stored in one hour would be 5.3 M Bytes, and data collection consume around 62.9 mJ; processing the data would consume approximately 8 mJ within about 0.9 seconds, and the wireless transmission of classification results would consume approximately 27 mJ. In other words, MagSpy would consume 97.9 mJ of power each hour, while accounting for 0.6% of the CPU utilization, and 5.3 M Bytes of storage space (note that this storage space is constantly recycled). Overall, the resource consumption of MagSpy should be too low for most users to notice.

VI. Discussion

In this section, we outline the limitations of the MagSpy system as well as countermeasures against our system.

A. Limitations

The MagSpy system is capable of detecting user's privacy information with a high degree of accuracy; however, this requires the continuous collection of data from the built-in sensor in the background. Due to privacy issues, some OSs have placed restrictions on apps running in the background. Our system is able to run stably on Android platforms up to version 7.0 and iOS 11.0, due to the fact that they have no restrictions on background apps reading these three sensors. This accounts for 45% of the Android market [44] and 11.4% of the iOS market [45]. Note however that Android 8.0 (14.3%) forces the background apps to suspend after a period of time (often in few minutes) in order to reduce energy consumption. Then all versions since Android version 9 and iOS 12 directly prohibit the reading of sensor data in the background. Nonetheless, there are many solutions to deal with the problem. In order not to be forced to suspend in Android 8.0, apps can apply for a "wake-lock" permission [46]. We also discover that devices being recharged can get rid of the restriction. For Android 9 or iOS 12 (or above), applying for a GPS permission is enough to keep our data collecting service running in the background [47]. Experiments show that, with these methods, MagSpy can continually collect magnetometer signal data for analysis.

B. Countermeasures

There are some methods that could be used to prevent information leakage via magnetometer disturbance. Physical shielding using ferromagnetic materials is a straightforward approach to reducing the susceptibility of sensors to electromagnetic activity. However, mobile device manufacturers tend to disregard EM leakage from electronic components in the pursuit of lighter and thinner devices. Meanwhile, the standards for EM leakage may still be too loose, so that EM leakage signals can be used to infer the privacy behaviors of users when using mobile apps. Explicit user permissions could be introduced to limit access to magnetometers; however, it is likely that many users would have to be informed of the privacy threats associated with built-in sensors and many mobile devices are running on outdated OSes [44], [45]. One final approach would be to limit access to magnetometers immediately when app running in the background.

VII. Related Work

A. Electromagnetic Side Channel

In previous studies [18], [25], [41], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], the EM side channel is exploited to enable attacks on smartcards, FPGAs, and other mobile devices. In [51], the EM signals emitted by laptops are detected using customized antennas and a software-defined radio receiver for the extraction of RSA and ElGamal keys. In [41], EM side-channel signals are used to create a novel near-field communication system between mobile devices. In [25], electric appliances are characterized based on the EM radiation signals they emitted. In [15], [16], [17], researchers exploit the reaction of magnetometers to EM activity to infer activities corresponding to the launching of apps or opening of browser websites. In the current work, we demonstrate the feasibility of using magnetometer readings to infer the complete mobile app usage behavior, such as the multiple simultaneously running apps (with corresponding in-app service) and sensitive user actions (PIN input).

B. Other Side Channels

Side channels other than EM signals can be used to infer user behavior in the usage of mobile devices: **(i)** Several researchers have shown that power consumption traces (collected in the form of *sysfs* files [58]) are highly correlated with CPU activity patterns, and could therefore be used to infer the opening of apps [7]. However, the sampling rate for battery monitoring is generally low (≤ 5 Hz). **(ii)** Researchers have also employed data-usage statistics (i.e., tcp packages, memory footprint, browser cache) to infer user behavior [4], [12], [59]. Note however that accessing data-usage statistics on mobile devices via the */proc* file system can only be achieved on rooted devices. **(iii)** Researchers have demonstrated the efficacy of using built-in motion sensors to elucidate the motion of users [60], [61], [62]. Researchers have also utilized motion sensor data to infer when the user is tapping and gesturing during data input [63]. In contrast, the EM signals emitted by mobile devices provide valuable information pertaining to app usage and user behavior.

VIII. Conclusion

This paper presents a novel attack scheme capable of capturing the EM signals emitted by the built-in components of mobile devices, and then deciphering current app usage, in-app services, and key stroke information. The system comprises a geomagnetic offset canceller, in-app/app classifier, which is a Cascade-LSTM classifier with CNN-based feature extractor, and PIN detection method via CWT-based peak detection algorithm

and Random Forest classier. The proposed system is capable of classifying app/in-app services types and key stroke information in the noisy EM signals from 30 mobile devices running on the 50 mainstream mobile apps.

REFERENCES

[1] Z. Tu et al., "Your apps give you away: Distinguishing mobile users by their app usage fingerprints," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 2, no. 3, pp. 1–23, 2018.

[2] Y. Qiao, X. Zhao, J. Yang, and J. Liu, "Mobile big-data-driven rating framework: Measuring the relationship between human mobility and app usage behavior," *IEEE Netw.*, vol. 30, no. 3, pp. 14–21, May/Jun. 2016.

[3] S. L. Jones, D. Ferreira, S. Hosio, J. Goncalves, and V. Kostakos, "Revisitation analysis of smartphone app use," in *Proc. 2015 ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2015, pp. 1197–1208.

[4] X. Zhou et al., "Identity, location, disease and more: Inferring your secrets from Android public resources," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 1017–1028.

[5] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on Android," in *Proc. 9th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2016, pp. 49–60.

[6] X. Zhang, X. Wang, X. Bai, Y. Zhang, and X. Wang, "Os-level side channels without procfs: Exploring cross-app information leakage on IoS," in *Proc. Symp. Netw. Distrib. System Secur.*, 2018, pp. 1–15.

[7] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[8] L. Yan, Y. Guo, X. Chen, and H. Mei, "A study on power side channels on mobile devices," in *Proc. 7th Asia-Pacific Symp. Internetware*, 2015, pp. 30–38.

[9] L. Simon, W. Xu, and R. Anderson, "Don't interrupt me while I type: Inferring text entered through gesture typing on Android keyboards," in *Proc. Privacy Enhancing Technol.*, vol. 2016, no. 3, pp. 136–154, 2016.

[10] K. Zhang and X. Wang, "Peeping Tom in the neighborhood: Keystroke eavesdropping on multi-user systems," in *Proc. USENIX Secur. Symp.*, 2009, Art. no. 23.

[11] Q. A. Chen, Z. Qian, and Z. M. Mao, "Peeking into your app without actually seeing it:{UI} state inference and novel android attacks," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 1037–1052.

[12] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 143–157.

[13] R. Spreitzer, F. Kirchengast, D. Gruss, and S. Mangard, "Procharvester: Fully automated analysis of procfs side-channel leaks on Android," in *Proc. 2018 Asia Conf. Comput. Commun. Secur.*, 2018, pp. 749–763.

[14] R. Spreitzer, G. Palfinger, and S. Mangard, "Scandroid: Automated side-channel analysis of Android APIs," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2018, pp. 224–235.

[15] Y. Cheng et al., "Magattack: Guessing application launching and operation via smartphone," in *Proc. 2019 ACM Asia Conf. Comput. Commun. Secur.*, 2019, pp. 283–294.

[16] N. Matyunin et al., "Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting," in *Proc. 18th ACM Workshop Privacy Electron. Soc.*, 2019, pp. 135–149.

[17] R. Ning, C. Wang, C. Xin, J. Li, and H. Wu, "Deepmag: Sniffing mobile apps in magnetic field through deep convolutional neural networks," in *Proc. 2018 IEEE Int. Conf. Pervasive Comput. Commun.*, 2018, pp. 1–10.

[18] T. Ni, G. Lan, J. Wang, Q. Zhao, and W. Xu, "Eavesdropping mobile app activity via {Radio-Frequency} energy harvesting," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 3511–3528.

[19] Z. Zhu et al., "Remote app sensing with your phone," in *Proc. 2016 ACM Int. Joint Conf. Pervasive Ubiquitous Computing: Adjunct*, 2016, pp. 241–244.

[20] Y. Fu, J. Liu, X. Li, and H. Xiong, "A multi-label multi-view learning framework for in-app service usage analysis," *ACM Trans. Intell. Syst. Technol.*, vol. 9, no. 4, pp. 1–24, 2018.

[21] A. Developers, "Android debug bridge (adb)," 2019. [Online]. Available: https://developer.android.com/studio/command-line/adb

[22] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[23] "Multi-window support," (n.d.). [Online]. Available: https://developer.android.com/guide/topics/ui/multi-window

[24] iOS Developer, "Human interface guidelines: Multiple windows on ipad," 2020. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/ios/system-capabilities/multiple-windows/

[25] E. J. Wang, T.-J. Lee, A. Mariakakis, M. Goel, S. Gupta, and S. N. Patel, "MagnifiSense: Inferring device interaction using wrist-worn passive magneto-inductive sensors," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2015, pp. 15–26.

[26] G. E. P. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *J. Amer. Statist. Assoc.*, vol. 65, no. 332, pp. 1509–1526, 1970.

[27] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results," *J. Transp. Eng.*, vol. 129, no. 6, pp. 664–672, 2003.

[28] S. Johansen, "Estimation and hypothesis testing of cointegration vectors in Gaussian vector autoregressive models," *Econometrica: J. Econometric Soc.*, pp. 1551–1580, 1991.

[29] S. Menard, *Applied Logistic Regression Analysis*, Newcastle upon Tyne, U.K.: Sage, 2002.

[30] A. Liaw et al., "Classification and regression by randomforest," *R News*, vol. 2, no. 3, pp. 18–22, 2002.

[31] Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proc. Eur. Conf. Mach. Learn.*, 1998, pp. 137–142.

[32] B. W. Suter, "The multilayer perceptron as an approximation to a Bayes optimal discriminant function," *IEEE Trans. Neural Netw.*, vol. 1, no. 4, pp. 296–298, 1990.

[33] X. Jin and J. Han, "K-means clustering," in *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G. I. Webb, Eds., 2017, pp. 695–697.

[34] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[35] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[36] B. Chen, B.-F. Chen, and H.-T. Lin, "Rotation-blended CNNs on a new open dataset for tropical cyclone image-to-intensity regression," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2018, pp. 90–99.

[37] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *J. Syst. Eng. Electron.*, vol. 28, no. 1, pp. 162–169, 2017.

[38] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, 2018, pp. 7132–7141.

[39] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2017.

[40] F. Wang et al., "Residual attention network for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3156–3164.

[41] H. Pan, Y.-C. Chen, G. Xue, and X. Ji, "Magnecomm: Magnetometer-based near-field communication," in *Proc. 23 rd Annu. Int. Conf. Mobile Comput. Netw.*, New York, NY, USA: ACM, 2017, pp. 167–179.

[42] P. Du, W. A. Kibbe, and S. M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching," *Bioinformatics*, vol. 22, no. 17, pp. 2059–2065, 2006.

[43] H. Pan et al., "Magthief: Stealing private app usage data on mobile devices via built-in magnetometer," in *Proc. 2021 18th Annu. IEEE Int. Conf. Sens. Commun. Netw.*, 2021, pp. 1–9.

[44] Statcounter, "Mobile and tablet Android version market share worldwide," 2019. [Online]. Available: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide

[45] Statcounter, "Mobile and tablet IoS version market share worldwide," 2019. [Online]. Available: https://gs.statcounter.com/ios-version-market-share/mobile-tablet/worldwide

[46] A. Developers, "Keep the device awake," 2019. [Online]. Available: https://developer.android.com/training/scheduling/wakelock

[47] A. Preprint, J.-M. Lin, and J. S. Seibel, "Motion-based side-channel attack on mobile keystrokes," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:197629514

[48] T. Ni, "Sensor security in virtual reality: Exploration and mitigation," in *Proc. 22nd Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2024, pp. 758–759.

[49] D. Ding, Y.-C. Chen, X. Ji, and G. Xue, "LeakThief: Stealing the behavior information of laptop via leakage current," in *Proc. 20th Annu. IEEE Int. Conf. Sensing, Communication, Netw.*, 2023, pp. 186–194.

[50] Z. Wang, F.-H. Meng, Y. Park, J. K. Eshraghian, and W. D. Lu, "Side-channel attack analysis on in-memory computing architectures," *IEEE Trans. Emerg. Topics Comput.*, vol. 12, no. 1, pp. 109–121, First Quarter, 2024.

[51] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—Channel (s)," in *Proc. 4th Int. Workshop Cryptographic Hardware Embedded Syst.*, Redwood Shores, CA, USA, 2003, pp. 29–45.
[52] Y. Long et al., "EM eye: Characterizing electromagnetic side-channel eavesdropping on embedded cameras," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2024, pp. 1–17.
[53] T. Ni, X. Zhang, and Q. Zhao, "Recovering fingerprints from in-display fingerprint sensors via electromagnetic side channel," in *Proc. 2023 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 253–267.
[54] T. Ni et al., "Uncovering user interactions on smartphones via contactless wireless charging side channels," in *Proc. 2023 IEEE Symp. Secur. Privacy*, 2023, pp. 3399–3415.
[55] T. Ni et al., "Exploiting contactless side channels in wireless charging power banks for user privacy inference via few-shot learning," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–15.
[56] R. Xiao, T. Li, S. Ramesh, J. Han, and J. Han, "Magtracer: Detecting GPU cryptojacking attacks via magnetic leakage signals," in *Proc. 29th Annu. Int. Conf. Mobile Comput. Netw.*, 2023, pp. 1–15.
[57] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, "Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel," in *Proc. USENIX Secur. Symp.*, 2022, pp. 4383–4400.
[58] P. Lifshits et al., "Power to peep-all: Inference attacks by malicious batteries on mobile devices," in *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 4, pp. 141–158, 2018.
[59] N. Zhang, K. Yuan, M. Naveed, X. Zhou, and X. Wang, "Leave me alone: App-level protection against runtime information gathering on Android," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 915–930.
[60] C. Liu, L. Zhang, Z. Liu, K. Liu, X. Li, and Y. Liu, "Lasagna: Towards deep hierarchical understanding and searching over mobile sensing data," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Network.*, 2016, pp. 334–347.
[61] K. Sun, C. Xia, S. Xu, and X. Zhang, "StealthyIMU: Stealing permission-protected private information from smartphone voice assistant using zero-permission sensors," in *Proc. Netw. Distrib. System Secur. Symp.*, 2023, pp. 1–16.
[62] S. Zhang, Y. Liu, and M. Gowda, "I spy you: Eavesdropping continuous speech on smartphones via motion sensors," in *Proc. ACM Interactive Mobile Wearable Ubiquitous Technol.*, vol. 6, no. 4, pp. 1–31, 2023.
[63] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *Proc. 6th USENIX Conf. Hot Topics Secur.*, Nov., 2011, pp. 9–9.

Yongjian Fu received the BSc degree in computer science from Central South University, in 2021, China. He is currently working toward the PhD degree in computer science with the School of Computer Science and Engineering, Central South University. His research interests include wireless sensing, mobile computing, and Internet-of-Things.

Lanqing Yang received the received the undergraduate degree from the University of Electronic Science and Technology of China (UESTC), in 2017, and the PhD degree in computer science from Shanghai Jiao Tong University, in 2023, China. He is currently a assistant researcher with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include mobile computing, sensor network, and wireless sensing.

Hao Pan (Member, IEEE) received the received the undergraduate degree from the Yingcai Honors College of University of Electronic Science and Technology of China (UESTC), in 2016, and the PhD degree in computer science from the Shanghai Jiao Tong University (SJTU), in 2022. He is a senior researcher with Microsoft Research Asia (Shanghai). His research interests include networked systems and span the areas of wireless communication and sensing, human-comuputer interaction, and computer vision.

Yi-Chao Chen (Member, IEEE) received the BS and MS degree with the Department of Computer Science and Information Engineering, National Taiwan University, in 2004 and 2006, respectively. and the PhD degree in computer science from the University of Texas at Austin, in 2015. He joined Shanghai Jiao Tong University as a tenure-track associate professor with the Department of Computer Science and Engineering, in 2018. Prior to joining SJTU, He spent a year as a researcher with Huawei Future Network Theory Lab, Hong Kong and then worked as a chief scientist with Hauoli LLC. His research interests include networked systems and span the areas of mobile computing, wireless networking, and cyber-security.

Guangtao Xue (Member, IEEE) received the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China, in 2004. He is currently a professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include vehicular ad hoc networks, wireless networks, mobile computing, and distributed computing. He is a member of the IEEE Computer Society and the IEEE Communication Society.

Ju Ren (Senior Member, IEEE) received the BSc, MSc, and PhD degrees in computer science from Central South University, China. Currently, he is an associate professor with the Department of Computer Science and Technology, Tsinghua University, China. His research interests include Internet-of-Things, edge computing, edge intelligence, as well as security and privacy. He currently serves as an associate editor for many journals, including *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Cloud Computing* and *IEEE Transactions on Vehicular Technology*, etc. He also served as the general co-chair for IEEE BigDataSE'20, the TPC co-chair for IEEE BigDataSE'19, the track co-chair for IEEE ICDCS'24, the poster co-chair for IEEE MASS'18, a symposium co-chair for IEEE/CIC ICCC'23&19, I-SPAN'18 and IEEE VTC'17 Fall, etc. He received several best paper awards from IEEE flagship conferences, including IEEE ICC'19 and IEEE HPCC'19, etc., the IEEE TCSC Early Career Researcher Award (2019), and the IEEE ComSoc Asia-Pacific Best Young Researcher Award (2021). He was recognized as a highly cited Researcher by Clarivate (2020-2022).