MagDefender: Detecting Eavesdropping on Mobile Devices using the Built-in Magnetometer

Abstract—**The microphones and cameras on smartphones are highly susceptible to eavesdropping by malicious applications (apps) and even the operating systems (OSes) installed by unscrupulous phone vendors. Nowadays, several third-party apps and newer OS versions are able to monitor the working status of on-board devices via OS media related APIs; however, so-called transplantation attacks have proven highly effective in evading these measures. This study explored the possibility of using the magnetometer built into mobile devices to detect instances of eavesdropping via the on-board microphones and cameras. Our results revealed that on-board hardware (e.g., cameras and microphones) and related modules (e.g., codec chip and CPU) leak electromagnetic (EM) emissions whenever audio or video data is accessed. Nonetheless, this type of EM leakage is weak and tends to be buried beneath extraneous EM signals (noise) from other sources. We therefore adopted a deep learning approach (pseudo-Siamese convolutional neural network) to the analysis of EM signals. The proposed system, referred to as MagDefender, enables the continuous monitoring of EM signals in the background without the need for hardware modification, root/jailbreak operations, or additional system permissions. Empirical results demonstrate the efficacy of MagDefender in recognizing instances of eavesdropping on camera/microphones data, with average accuracy of 95.1% when applied to devices on which it was trained, and 87.5% on unseen devices.**

I. Introduction

The availability of high-fidelity sensors and ubiquitous internet connectivity has prompted the development of numerous mobile applications (apps) and services that rely on multimedia sensors. The cameras and microphones built into mobile devices make it possible to capture and share image, audio, and video files. They have also made it possible to develop voice assistants, speech input, music identification, and face/object recognition. Unfortunately, many malicious apps also use these sensors in ways that violate user expectations and privacy. The New York Times reported on apps using code from a company called Alphonso, which makes it possible to listen for audio signals indicative of user behavior and preferences in order to more precisely target them with advertisements [1]. Notable vulnerabilities have also been identified in official apps. For example, it was revealed that Apple FaceTime allowed unauthorized access to iPhone cameras and microphones, even by attackers who lack advanced hacking skills [2].

The example in Fig. 1 illustrates two online advertising schemes based on information obtained by eavesdropping on smartphones via the built-in camera or microphone. A conversation with your friend on topic *X* is first recorded by the microphone. This process can be implemented by the operating system itself, a third-party app in-stalled on the device, or the third-party

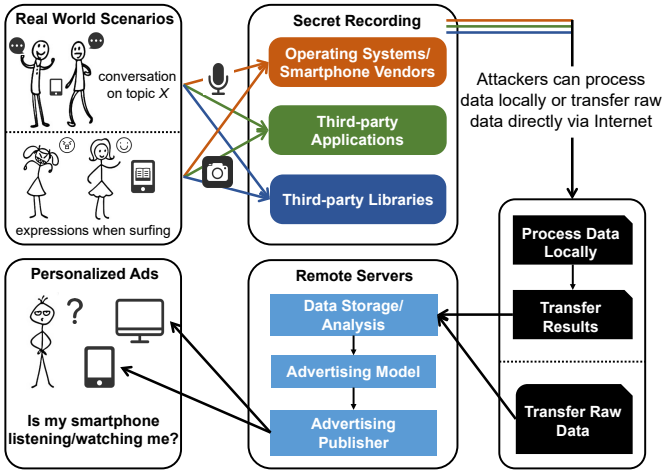


Fig. 1: Working principle of the online advertisement scheme based on information obtained by eavesdropping on smartphones via the built-in camera or microphone.

libraries included in an app. After being processed and filtered (locally on the device or on a remote server), the data is fed into a profiling model, which advises ad publishers about the types of ads to which you are likely to respond favorably, such as those associated with topic *X*. Another strategy uses the front camera to monitor your expressions while looking at various items or reading various news articles on a third-party app. As above, the ad publisher could then use this information to create a profile of you in order to select ads specific to your tastes.

Motivation: Current mainstream operating systems (Android and iOS) are equipped with mechanisms that control access to hardware sensors. They also impose restrictions on the degree to which background apps can access multimedia sensor data. All Android installations since Android 9 (accounting for 10.4% of the market [3]) block background apps from accessing camera and microphone data [4]. iOS 9+ prohibits background camera usage, while permitting background microphone usage with a red recording indicator appearing in the upper part of the screen [5]. Note however that these restrictions have no effect at all on the actions of the operating system itself. It is a trivial matter for OS providers and mobile phone manufacturers to access private sensor data in the background without the need for permissions. Furthermore, many malicious third-party apps and libraries are able to evade restrictions imposed by the operating system to gain access to sensor data using a scheme referred to as a transplantation attack [6]. Essentially, the attacker transplants the system libraries required for a Media Server process into the

library for a malicious app, thereby giving it direct access to camera/microphone hardware drivers (or hardware interface libraries). Transplantation allows the attacker to access the cameras or microphones without the need to call up APIs, even on unrooted phones (see Sec. II-B).

A number of developers have implemented version upgrades with indicators warning the user about apps accessing the microphones and/or cameras. iOS 14 flashes an orange/green light whenever an app is using these devices in the background [7]. The MIUI 12 modification of Android 10 includes an "application behavior record" function in the security system detailing app behaviors, such as access to mic/camera data) [8]. Third-party apps, such as Access Dots [9] utilize Android APIs (`CameraManager.AvailabilityCallback` and `AudioManager.AudioRecordingCallback`) to indicate the working status of the cameras and microphones on most Android devices since Android 7+. Note however that all of the above solutions are based on OS APIs, which makes it impossible for them to identify instances of eavesdropping by the OS itself. Furthermore, they are unable to identify malicious apps that does not use system APIs to access camera/microphone data.

Our solution: In this study, we used the magnetometer built into the mobile devices to detect instances of eavesdropping via the on-board microphones and cameras. The proposed scheme, referred to as MagDefender, uses the magnetometer to capture EM signals emitted by electronic components and then analyzes them for evidence of an app accessing sensor data in the background. Our development of MagDefender was predicated on the assumption that all of the hardware modules involved in collecting sensor data (e.g., microphone, camera, codec chip, CPU, and RAM) work in a unique and consistent manner. MagDefender can be implemented on any commercial off-the-shelf (COTS) mobile device using any version of any OS without the need for hardware modification, root/jailbreak, or additional system permissions.

A number of daunting challenges had to be overcome in developing the MagDefender system. First, the built-in magnetometer is sensitive to all forms of magnetic interference. These high sensitivity devices are able to detect EM signals from electronic components in mobile devices; however, they also detect geomagnetic signals, which must be extracted from the overall readings. In this study, we developed a geomagnetic signal cancellation scheme uses support vector regression (SVR) and IMU motion data (from the built-in 3-axis gyroscope and 3-axis accelerometer) to deal with changes in the geomagnetic signals caused by device motion. In an effort to minimize power consumption, built-in magnetometers operate at a low sampling rate ($50 \sim 200$Hz), the resolution of which is insufficient for the extraction of features required to differentiate among EM signals. Furthermore, the EM signals generated while accessing camera/microphone data are easily overwhelmed by the EM signals generated during the execution of other app tasks/services. Conventional classification algorithms are unable to cope with these situations. Thus, our second challenge was to design a deep learning model as an alternative to conventional classification schemes with the aim of deriving the information required to identify EM emissions from only a few training samples when encountering unfamiliar mobile devices for which the system was not trained. Finally, we had to deal with the fact that manufacturers vary widely in terms of hardware configuration and sensor selection, which can result in widely divergent EM emission patterns. We developed a pseudo-Siamese convolutional network (one-shot learning) to determine whether the observed EM signals (instance input) generated by the device contain the EM signals generated only when accessing camera/microphone data (exemplar input). Thus, only the exemplar EM signals are required to identify instances of eavesdropping, even when applied to an unknown mobile device (i.e., not included in the training dataset).

The main contributions of this work are as follows:

- We conducted a pilot study to verify that the media sensors and corresponding hardware modules in mobile devices generate unique and consistent EM signals detectable using the built-in magnetometer.
- We developed a third-party app solution capable of monitoring the working status of the microphones and cameras in mobile devices by analyzing magnetometer readings without utilizing any OS media related APIs.
- We developed a Pseudo-Siamese CNN as a one-shot learning method to identify instances of EM signals generated when accessing camera/microphone data.
- We evaluated the efficacy of the proposed scheme on 30 commercial Android and iOS devices (various OS versions) in detecting eavesdropping events. The proposed scheme achieved average accuracy of 95.1% when applied to devices on which it was trained, and 87.5% on unseen devices.

II. Background and Vulnerability Analysis

In this section, we examine the operations of a mobile device when an app requests camera/microphone data. We also examine the vulnerability of solutions that rely on OS media related APIs to monitor camera/microphone usage. Note that this analysis focuses on Android devices, and the same is true for iOS devices.

A. Android Media Service

In the Android environment, camera/microphone programming is based on the client/server architecture. The client refers to any app using the Android Camera/Audio Service, whereas the server refers to the Media Server. The workflow of a normal camera app accessing the camera via the Android Camera Service is presented on the left side of Fig. 2. The client process is on a standard 5-layer structure. The Application layer and the Framework layer are written using Java code. The Runtime layer contains a Java Virtual Machine to

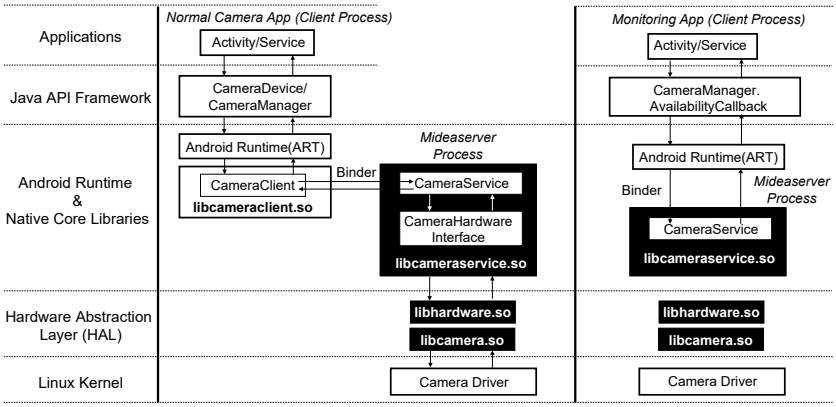


Fig. 2: left) Workflow of normal camera app when using the Android Camera APIs; right) Process involved in monitoring app to detect instances of camera device calls by camera clients using the *CameraManager*.

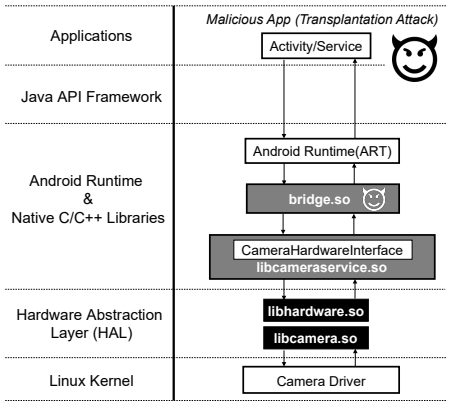


Fig. 3: Model of transplantation attack involving stealthy use of camera device by avoiding Android APIs [6].

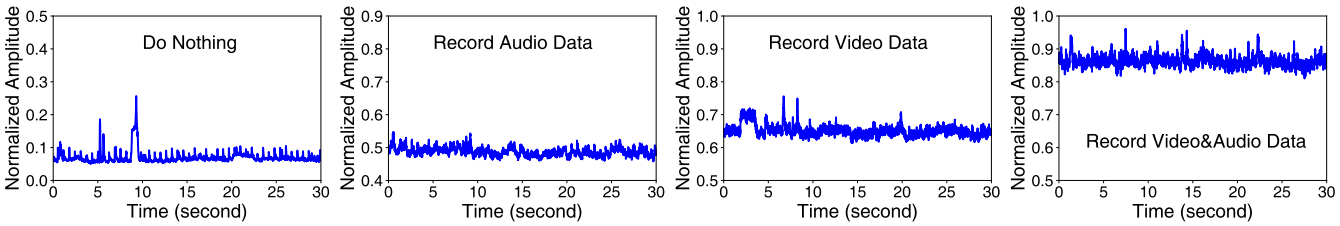
execute Java code. This layer also includes native libraries required for Interprosses Communication (IPC) (e.g. *libcameraclient.so*). The client process does not interact directly with the Camera device; therefore, the HAL layer uses *.so* libraries instead of code to talk with the camera driver. The *Mediaserver* process uses the lower 3 layers. The top Native Core Library layer contains system libraries written in native code (*.so* libraries), which are used to handle requests from clients, forward requests to the HAL layer, obtain responses from the HAL layer, and forward responses to the client. The HAL layer deals with most of the tasks associated with *Mediaserver* processes. Note that the camera driver is contained in the Linux kernel layer used for *Mediaserver* processes.

Consider the following example. Any client process that involves taking a photo must first establish a connection with the Camera Service in order to access the associated functions. The client first calls an Android API using Java code. The Android API uses a system library (*libcameraclient.so*) to send a CONNECT Binder request to the *Mediaserver* process, which is then parsed by the Camera Service according to type based on the structure of the Binder data. If the request type is CONNECT, then the Camera Service instructs the *Systemserver* the check the client permissions. As long as the client passes the permission check, the Camera Service calls the *CameraHardwareInterface* to interact with the camera driver in the Linux kernel. A notification thread in the HAL layer monitors the camera driver while awaiting camera-related events (e.g., focusing or exposure operations). When the camera finishes taking a photo, the camera driver sends an event signal to the notification thread, which then transfers the image data back to the *Mediaserver* from the bottom up by calling the callback functions. Note that by this point, the image data has already been compressed by the camera driver into a pre-specified photo format (e.g., jpeg). The Camera Service then forwards the image data directly to the client process via the Binder IPC. Image data received by the Camera Client in the Runtime layer is then forwarded to the Framework layer, which posts it to the screen for the user to see. Image data can be saved as a photo file in the Application layer. The above workflow involved in acquiring camera data using the Android Camera Service is similar to the workflow involved in acquiring audio data using the Android Audio Service.

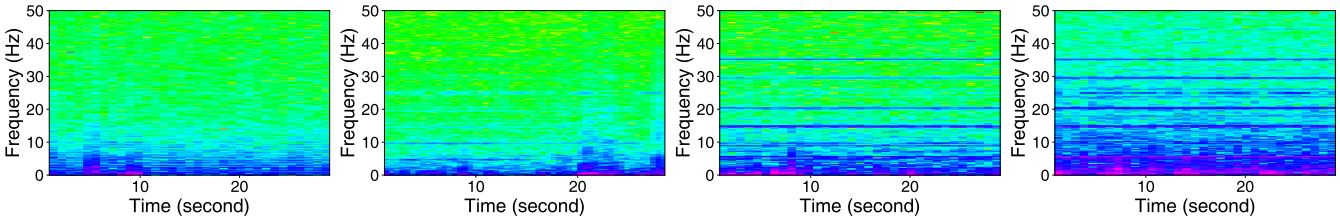
The right part of Fig. 2 illustrates the working principle used in the monitoring of apps. Camera usage can be monitored using `getSystemService("camera")` to retrieve the *CameraManager* in order to interact with camera devices, in conjunction with `CameraManager.AvailabilityCallback` (added in API level 21) to determine whether a camera device has been called by any camera API client via the Binder IPC [10]. Microphone usage can be monitored using `getSystemService("audio")` to retrieve the *AudioManager* in order to interact with microphone-related devices in conjunction with `AudioManager.AudioRecordingCallback` to determine whether any microphone-related devices are recording audio data [11]. Note that these methods do not require `Manifest.permission.CAMERA` or `.RECORD_AUDIO`. Essentially, this is the means by which third-party apps (e.g., Access Dots) monitor the working status of the built-in cameras and microphones in real time. Note that many Android phone vendors use a similar approach to implement indicators of camera and microphone operations.

B. Transplantation Attacks

Common attacks that involve capturing a photo on an Android phone call camera-related APIs provided by an Android SDK. However, this type of spy-on-user attack is not stealthy, as it is unable to evade monitoring apps (see the right side of Fig. 2). Sophisticated attackers are able to take photos and record audio/video files without calling Android APIs. These stealthy methods are referred to as transplantation attacks [6]. The first step in the transplantation attack model is to choose an app that uses CAMERA permissions (e.g., QRCode Scanner) for repackaging. Note that such apps are widely available on Google Play, and repackaging and redistributing them is generally straightforward. Fig. 3 illustrates the process after the malicious repackaged app has



(a) Normalized EM signals when apps executing different tasks.



(b) Corresponding spectrum of the EM signals when apps executing different tasks.

Fig. 4: Magnetometer readings and corresponding spectra of EM signals generated by Huawei P20Pro during the execution of various tasks.

been installed on the victim's mobile device. Basically, transplantation attacks involve loading the *Mediaserver* process into the address space of the malicious app, thereby allowing the Java code in the application layer to pass through the Framework layer and access the *libcameraservice_transplanted.so* in the Android Runtime layer. Functions in the *CameraHardwareInterface* then call functions in the HAL layer to talk with the camera driver in the Linux kernel layer. Java Native Interface (JNI) programming is utilized as a bridge to connect Java code in the application layer with transplanted *.so* libraries in the Runtime and HAL layers. The native code is compiled as an *.so* file (*bridge.so* in Fig. 3). Unlike normal camera apps, this type of malicious app jumps over the Android API Framework and eliminates the Binder IPC between *Mediaserver* processes by deleting the *libcameraclient.so* library and Camaera Service in the *libcameraservice.so* library. Under these conditions, the monitoring app tasks/services mentioned above are unable to detect eavesdropping behavior that involves taking photos or recording video.

III. Attack and Defense Models

In this section, we first envision two attack scenarios in which attackers use built-in cameras and microphones to eavesdrop on victims. We then present a novel defense model to alert the victim when the cameras and/or microphones are being used.

Attack Scenarios. This study addresses two attack scenarios involving (1) Unscrupulous mobile phone vendors modifying the operating system kernel and related libraries; and (2) Sophisticated attackers repackaging an app (e.g., QRCode Scanner) that already has CAMERA permissions to include malicious code for eavesdropping without using OS media related APIs.

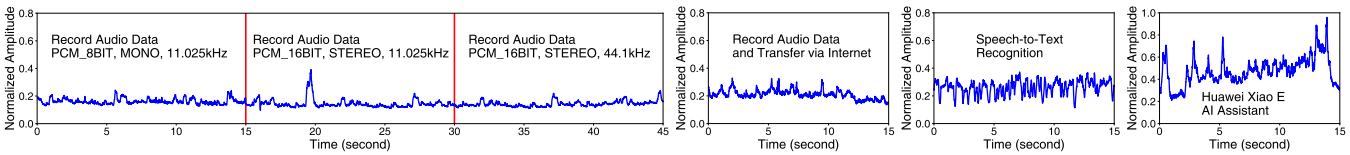
Defense Model. The fact that the above mentioned attack strategies can easily evade the status monitoring of Camera/Audio Service APIs means that software-based defense models infeasible. In this study, we developed a novel defense model, which uses the on-board magnetometer in mobile devices to detect electromagnetic (EM) emissions generated whenever the microphone and/or cameras are being used, even surreptitiously.

IV. Preliminary Analysis

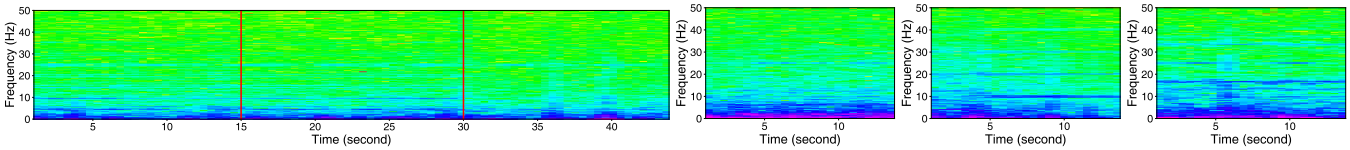
Preliminary experiments were conducted to answer three fundamental questions: i) When apps access camera/microphone data, does the mobile device generate EM emission signals that could be captured using the built-in magnetometer? ii) If so, what are the characteristics of these EM signals? iii) What other factors affect magnetometer readings? Our answers to these questions demonstrated the potential of using EM side-channel sensing to detect instances of eavesdropping and helped to elucidate the challenges involved in developing this technology. In the following section, we describe preliminary testing on the detection of EM signals emitted by two representative smartphones (Huawei P20Pro and iPhone 7 Plus). A proprietary app was installed on both devices to enable the continuous recording of background readings from the built-in magnetometer using sampling rates of 100Hz (Huawei P20Pro) and 100Hz (iPhone 7 Plus). For the sake of brevity, we visualized only the total EM intensity in the time and frequency domains, as calculated using readings from the three-axis magnetometer, as follows: $mag_t(t) = \sqrt{mag_x(t)^2 + mag_y(t)^2 + mag_z(t)^2}$.

A. EM signals captured by magnetometer

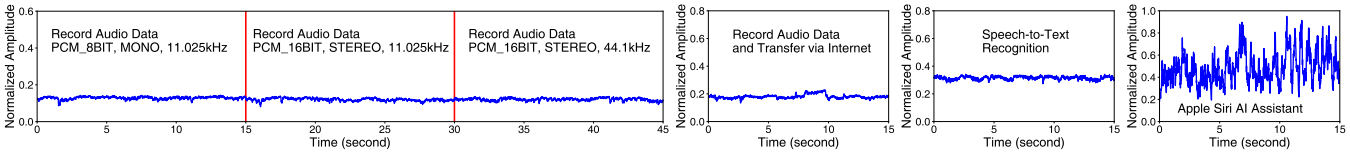
The first experiment was meant to verify whether mobile devices generate EM emission signals that can be detected by the built-in magnetometer when an app accesses camera/microphone data. Throughout the experiment, the phone remained static on a table to minimize fluctuations in the magnetometer readings caused by variations in geomagnetic signals. Readings were



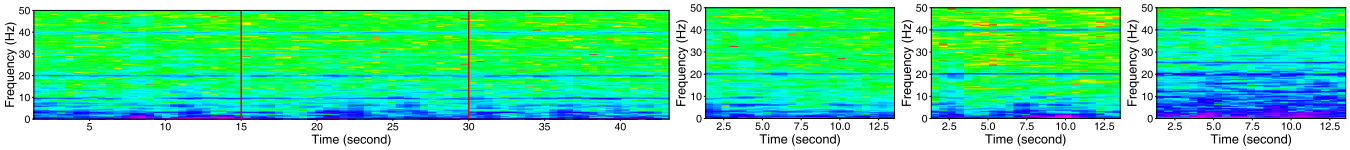
(a) Magnetometer readings when Huawei P20Pro executing different audio tasks.



(b) Corresponding EM spectrum when Huawei P20Pro executing different audio tasks.



(c) Magnetometer readings when iPhone 7Plus executing different audio tasks.



(d) Corresponding EM spectrum when iPhone 7Plus executing different audio tasks.

Fig. 5: Magnetometer readings and corresponding EM spectrum when using different mobile phones with different OSes to execute different audio-related tasks.

recorded continuously while performing a variety of tasks: doing nothing, recording audio data, recording video data, recording audio and video data simultaneously. Fig. 4 shows EM signal fluctuations in the time and frequency domains corresponding to functions involved in accessing audio and video data. Note that accessing audio and image data simultaneously resulted in stronger EM emission signals, due to the superimposing of one signal over the other. Overall, the results of this first preliminary test indicated that the built-in magnetometer is well-suited to capturing EM emission signals generated by mobile devices while accessing camera/microphone data.

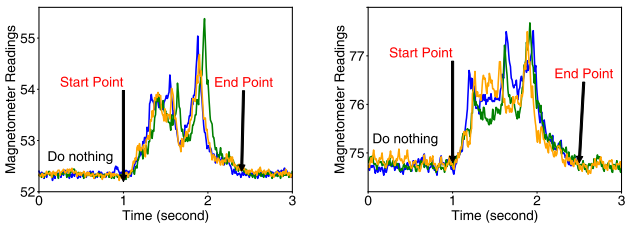
B. EM signals when accessing audio data

The second experiment was meant to characteristize of EM signals generated by the smartphone while executing a variety of tasks that involve the camera and/or microphone. We also examined the differences between the EM signals generated by different smartphone models. We first programed the devices to access audio data using various parameter settings (*i.e.*, `sampleRateInHz`, `channelConfig`, `audioFormat`), while simultaneously recording the corresponding magnetometer readings. We then examined the EM signals generated while executing apps that use audio data in conjunction with other functions, like apps that record audio and transmit data over the internet, speech recognition apps, and system AI assistant services 'Siri' on iOS and 'Xiao E' on EMUI.

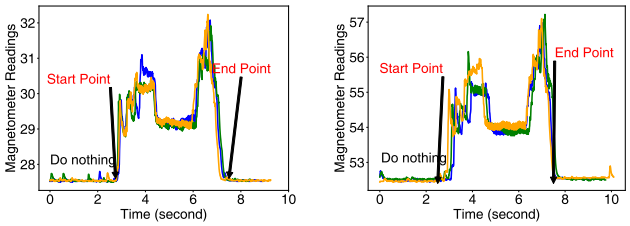
The magnetometer readings in Fig. 5 revealed the following: (i) The EM signals generated by the device while recording audio data did not vary, despite changes in parameter settings; (ii) The EM signals generated by the device while performing complex tasks presented the basic EM waveform (generated by accessing audio data) in both the time and frequency domains. From this, we surmise that regardless of the audio processing methods employed by the attacker, it should be possible to detect eavesdropping based on the corresponding EM signals. A comparison of Figs. 5(c) with 5(a) and Figs. 5(d) with 5(b) revealed that different devices generated different EM signals while performing the same task. This can no doubt be attributed to the fact that manufacturers differ in their selection of camera modules and other hardware components (*e.g.*, CPU, RAM), and corresponding software (*e.g.*, hardware drivers, interface libraries).

Huawei P20Pro and iPhone 7Plus both come equipped with three built-in microphones. We observed no differences in the EM signals generated while accessing data streams from any of the three microphones separately or all of the microphones simultaneously. This can be explained by the fact that the acquisition of an audio stream involves the use of microphones, the audio codec chip, the CPU, and RAM; however, the power consumption of the microphones is proportionally very low ($50mW$). The large power consumption of the CPU and RAM is sufficient to produce EM emission signals that dwarf the contribution of the microphone(s).

Magnetometer readings generated while simultane-

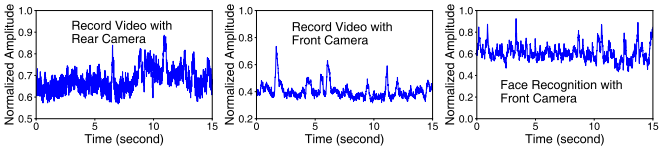


(a) Huawei P20Pro Rear Camera (b) Huawei P20Pro Front Camera

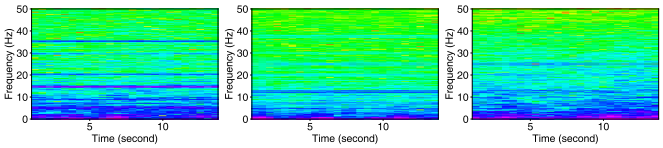


(c) iPhone 7Plus Rear Camera (d) iPhone 7Plus Front Camera

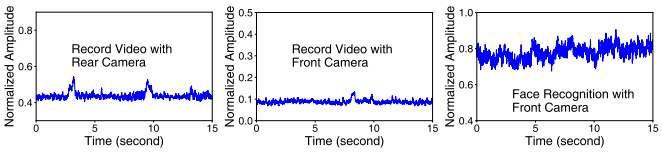
Fig. 6: Magnetometer readings generated while capturing image data (without preview) using various camera modules on different smartphones. Each camera module performed the task of taking a photo three times.



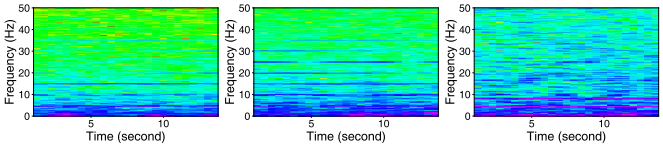
(a) Magnetometer readings when Huawei P20Pro executing different camera-related tasks.



(b) Corresponding EM spectrum when Huawei P20Pro executing different camera-related tasks.



(c) Magnetometer readings when iPhone 7Plus executing different camera-related tasks.



(d) Corresponding EM spectrum when iPhone 7Plus executing different camera-related tasks.

Fig. 7: Magnetometer readings and corresponding EM spectrum when using different mobile phones (different OSes) to execute different camera-related tasks.

ously accessing multimedia sensor data in the background and other apps in the foreground.

C. EM emission signals when accessing camera data

Camera data is accessed when taking a photo (instantaneous behavior) or recording video (steady-state

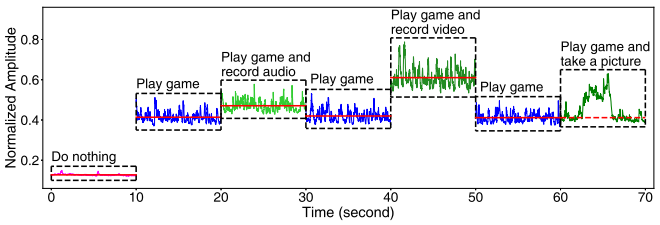
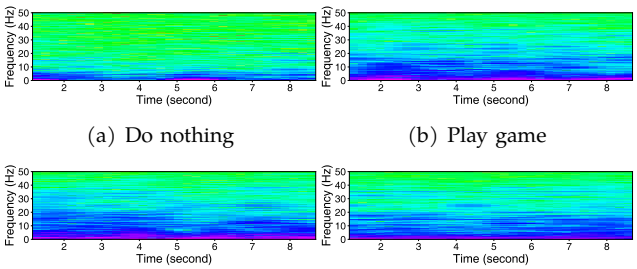


Fig. 8: Magnetometer readings generated while simultaneously accessing camera/microphone data in the background and running other apps in the foreground.



(a) Do nothing (b) Play game

(c) Play game and recordg audio (d) Play game and record video

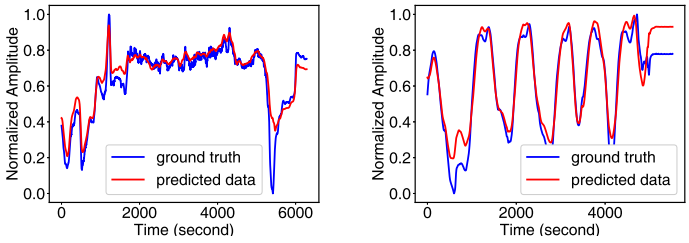
Fig. 9: EM signals generated by foreground running apps drowning out the spectral characteristics of EM signals generated by apps/services accessing audio and video data in the background.

TABLE I: Results obtained using canonical regression algorithms for geomagnetic signal fitting. Mean Square Error (MSE) was the metric used to evaluate how well the predicted EM fits the actual EM, where a smaller MSE value is better.

	ARIMA	SARIMA	VAR	LR	RF	SVR
MSE	1.45	1.33	1.28	1.25	0.82	0.28

behavior). Thus, we first conducted an experiment to characterize the EM emission signals generated while taking a photo, respectively using the rear and front camera modules. Note that the photos were taken without preview in automatic mode, which is the scheme preferred by many attackers. Fig. 6 presents the EM emission patterns generated while taking photos using an iPhone 7Plus and Huawei P20Pro. Overall, the EM emission patterns were unique and consistent, regardless of whether the front or rear camera modules were used. Here, the term unique refers to distinct EM signal patterns resulting from the different hardware used in the front and rear cameras. The term consistent refers to the fact that the EM emission signals remained stable over time.

We then conducted an experiment to characterize the EM emission signals generated while recording video data respectively using the front and rear camera modules. Note that video was recorded in automatic mode without audio data or preview. As shown in Fig. 7, the EM signal generated while accessing the data stream from the front camera differed from the signal generated while accessing the rear camera. Fig. 7 presents the results of an experiment in which a face recognition app (without preview) continuously accessed the photo flow. The EM signals generated while running the face recognition tasks presented the same basic EM waveform (generated by accessing video data) in both the time and frequency domains.



(a) Setting (b) Walking

Fig. 10: Fitting results from 6-axis IMU data based on SVR vs. raw geomagnetic field data (x-axis).

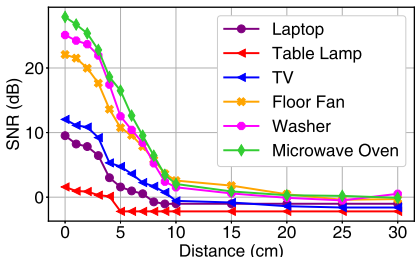


Fig. 11: SNR in magnetometer readings obtained with the iPhone 7Plus at various distances from appliances.

D. Other factors affecting magnetometer readings

Other running apps/services on mobile devices: Mobile devices are equipped with numerous electronic components in addition to sensors. Theoretically, any electronic component (e.g., CPU, GPU, LED display, Wi-Fi module, or battery) could generate EM signals while running. We previously demonstrated that the EM signals generated by a device vary with the app/tasks being performed. Thus, we sought to characterize the EM signals generated while simultaneously running eavesdropping operations in the background and the legitimate apps running in the foreground: 1) Minecraft (a mobile gaming), 2) an app accessing audio data, 3) an app accessing the image data stream from the front camera module, and 4) taking a photo using the front camera module. Fig. 8 shows the readings obtained by magnetometer throughout the experiments. Overall, we determined that any app accessing camera/microphone data generates a non-negligible EM emission signal, which is superimposed on the EM emission signal generated by the foreground app. In Sec. V, we will introduce the method developed for the extraction of EM emission signals associated with the acquisition of camera/microphone data from complex superimposed EM emission signals. Note that characterizing and cancelling out extraneous signals proved exceedingly difficult.

Device Movement: Magnetometers were first incorporated in mobile devices to measure changes in geomagnetic field lines. This means that while a mobile device is being used, any movement by the user could alter the position and/or orientation of the device, resulting in changes in the geomagnetic reading it produces. The blue line in Fig. 10 indicates the magnetometer readings (along the x-axis) generated by an iPhone 7Plus, while the user was sitting and walking around. Note that during this analysis, no tasks other than the acquisition of magnetometer readings were performed. Overall, the movement-induced variations in the amplitude of the geomagnetic signal far exceeded the changes in amplitude of the EM signal generated by the mobile device. This is a clear indication that variations in the geomagnetic flux can have a huge impact on data preprocessing (*i.e.*, normalization). We also characterized the geomagnetic fluctuations caused by user motion based on data obtained using a 3-axis gyroscope and 3-axis accelerometer. The output data was evaluated using a number of regression algorithms: ARIMA [12], SARIMA [13], VAR [14], LR [15], RF [16] and SVR [17], the results of which are listed in Table I. Overall, SVR provided the best performance. The red line in Fig. 10 presents the geomagnetic signal predicted using SVR. By minimizing the influence of geomagnetic signals, we were able to obtain clean EM emission signals for subsequent EM signal processing.

Nearby Electrical Devices: Other electrical devices (*e.g.*, household appliances) also leak EM emissions at levels that cannot be disregarded out of hand [18]. Theoretically, EM intensity drops off exponentially (rather than linearly) with an increase in distance. Nonetheless, we conducted a series of experiments to determine the actual attenuation of EM emission signals as a function of distance. We collected EM emission signals emitted from various electrical devices using the built-in magnetometer in the iPhone 7Plus at various distances from the appliances. We then plotted a corresponding SNR attenuation diagram with distance, as shown in Fig. 11. We found that when the mobile device was more than $20cm$ away from large household appliances (washing machine and microwave oven), the built-in magnetometer was unable to detect the EM signals the emitted. In the case of small appliances (table lamp), $5cm$ was sufficient to eliminate any interference.

V. MagDefender Design

A. System Overview

Our objective in this work was to determine from captured EM signals whether an app/service was accessing multimedia sensor data. We also sought to identify which multimedia sensor was being accessed (e.g., microphone or camera) and the means by which access was achieved (e.g., taking a photo or recording video/audio).

The MagDefender system comprises two main components for EM signal preprocessing and Pseudo-Siamese Network analysis. Fig. 12 presents an overview of the proposed system. EM signal preprocessing was meant to eliminate interference due to variations in the geomagnetic signal in order to obtain clean EM signals from the magnetometer readings. It was also meant to disentangle 3-axis data into information pertaining to direction and amplitude. The Pseudo-Siamese Network was meant to determine whether the EM signals from the mobile device contained components indicative of a request to access camera/microphone data. It was also meant to categorize sensor data according to its source and the

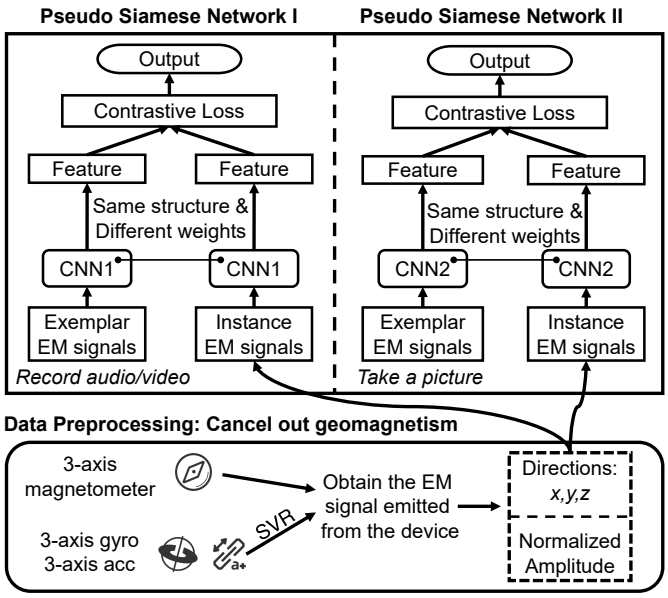


Fig. 12: Overview architecture and processing pipeline of our MagDefender.

manner by which it was accessed. Before outlining the details of the MagDefender system, we will first outline the process of data collection used to train the proposed Pseudo-Siamese CNNs.

B. Data Collection

Training of the proposed networks began with the assembly of a dataset comprising EM signals captured from a variety of mobile devices. Specifically, the dataset comprised the exemplar EM signals and instance (target for identification) EM signals, which are subsequently paired for use in training the pseudo-Siamese CNNs.

Exemplar Data Collection. We first collected Exemplar EM signals generated when accessing camera/microphone data. Results obtained in preliminary experiments prompted us to focus on the following seven types of exemplar EM signals:

- **Accessing audio data.** With the mobile device stationary on a table and all tasks disabled, we ran a proprietary app requesting access respectively to each RAW audio data stream (i.e., for each microphone), while using a 3-axis magnetometer to record the corresponding EM signals generated by the device. The resulting data were labeled using the term "audio".
- **Accessing video(or & audio) data.** With the device stationary on a table and all tasks disabled, we ran a proprietary app requesting access to video (w/ or w/o audio) data streams without preview, respectively from the rear and front camera modules, while simultaneously recording the corresponding EM signals. We respectively labeled these EM signals using the terms "front-video", "rear-video", "front-video-audio", and "rear-video-audio".
- **Taking a picture.** Using the same setup, we captured single photos (without preview), respectively using the rear and front cameras and saved them locally. We respectively labeled these EM signals using the terms "front-pic" and "rear-pic".

Instance Data Collection. The collection and labeling of instance EM signals involved turning off permissions to access the multimedia sensors (for all apps on the mobile devices) and allowing users to operate the mobile devices at will. The magnetometer readings were labeled using the term "none". Initially, we ran our proprietary apps with various media-related types of tasks in the background, while respectively running the above seven programs, which were labeled using the terms "audio", "rear-video", "front-video", "rear-video-audio", "front-video-audio", "rear-pic", and "front-pic", respectively.

C. System Description

1) EM Signal Preprocessing: EM signal preprocessing involved the use of SVR and IMU motion sensor data (3-axis accelerometer and 3-axis gyroscope data) to eliminate interference associated with changes in geomagnetic field lines. We then extracted information related to the direction and amplitude of the magnetic field from the raw 3-axis EM data. The details of data preprocessing are presented in Algorithm 1, as follows:

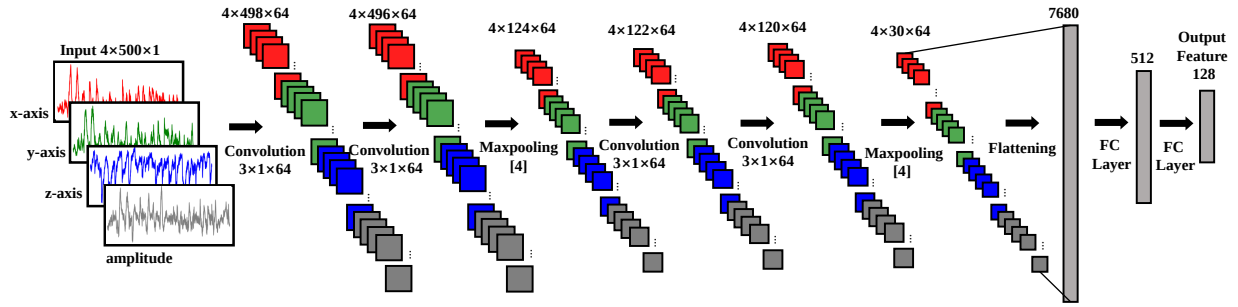
Algorithm 1: Extraction of Direction and Amplitude Information

Input:
- $mag(t) = \{mag_x(t), mag_y(t), mag_z(t)\}, t = 1 \dots n.$
- $acc(t) = \{acc_x(t), acc_y(t), acc_z(t)\}, t = 1 \dots n.$
- $gyro(t) = \{gyro_x(t), gyro_y(t), gyro_z(t)\}, t = 1 \dots n.$

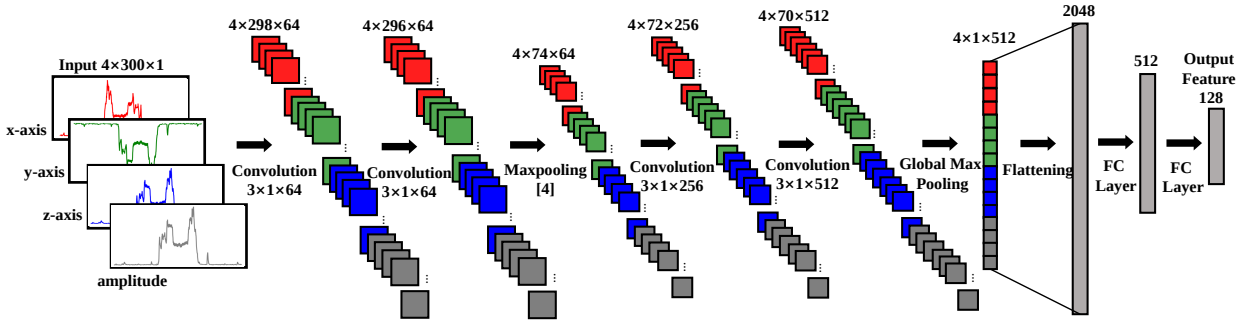
Output:
- $Dir(t) = \{dir_x(t), dir_y(t), dir_z(t)\}, t = 1 \dots n$ (direction information).
- $Amp_{norm} = Amp_{norm}(t), t = 1 \dots n$ (normalized amplitude information).

$EM = mag - SVR(acc, gyro)$
for $t \in [1, 2, \dots, n]$ **do**
  $Amp(t) = \sqrt{EM_x(t)^2 + EM_y(t)^2 + EM_z(t)^2}$;
end
for $i \in \{x, y, z\}$ **do**
  $dir_i = \frac{EM_i}{Amp}$;
end
$Amp_{norm} = \frac{Amp - min(Amp)}{max(Amp) - min(Amp)}$;

2) Pseudo Siamese CNN: Our primary objective was to determine whether instance EM signals contained EM signal patterns characteristic of tasks involved in accessing camera/microphone data. This could be achieved using labeled instance data to train classifiers; however, there are inherent differences in the EM signals generated by different mobile devices. Furthermore, it would be impractical to collect EM signals for every mobile device on the market just to create a training dataset. Even if this were attempted, it is unlikely that the classifier would perform well when applied to new (unseen) devices. We therefore implemented a novel deep neural network, referred to as a Pseudo-Siamese network, to assist in EM signal matching and discrimination.



(a) Extracting features of steady-state eavesdropping type EM signals.



(b) Extracting features of instantaneous eavesdropping type EM signals.

Fig. 13: CNN models for Feature Extraction Subnetwork.

Siamese Network. A Siamese network is a neural network architecture comprising two or more subnetworks with identical structures, parameters, and weights [19]. As shown in the upper part of Fig. 12, parameter updating is mirrored across both subnetworks, which are joined via a loss function at the top through the computation of a similarity metric, such as Euclidean distance, between representative features in each subnetwork branch. Siamese networks are meant to bring the output feature vectors closer together (when dealing with input pairs that are labeled as similar), and push the feature vectors apart (when dealing with input pairs that are labeled as dissimilar). Each branch in a Siamese network can be seen as a function that embeds an input image within a space. For example, the Siamese network must bring together the output feature vectors for a pair of input signals, where one comprises instance EM data labeled "audio" and the other comprises exemplar EM data labeled "audio". Conversely, the Siamese network must push apart the feature vectors for a pair of input signals, where one comprises instance data labeled "none" and the other comprises exemplar EM data labeled "front-video". Exemplar data with labels would automatically be collected for a new (not previously encountered) mobile device.

Siamese networks are well-suited to situations requiring verification involving a very large number of classes and/or scenarios where examples of all classes are unavailable at the time of training. However, a classical Siamese network sharing identical network structures and weights would be ill-suited to dealing with EM signals containing a variety of components. In our scenario, the right subnetwork (see Pseudo Siamese Network I in Fig. 12) was tasked with filtering out EM signals generated by other apps/services prior to the feature extraction, while the left subnetwork was tasked with extracting EM signal features directly. Thus, we settled on a Pseudo-Siamese network with two branches sharing the same structure but different weights.

Feature Extraction Subnetwork. Deep Convolutional Neural Networks (CNNs) have been used with considerable success in image processing and time-series analysis. The effectiveness of this approach lies in its ability to learn discriminative features automatically by exploring the deep architecture at multiple levels of feature abstraction without the need for a priori domain knowledge.

In this work, 1D CNN [20] was used as the subnetwork for feature extraction. Note that we had to deal with two types of EM signal. One signal type remained relatively stable over time (e.g., exemplar signals in the training dataset labeled "audio", "front/rear-video", or "front/rear-video-audio") and one signal type was an instantaneous impulse (e.g., exemplar signals labeled "front-pic" or "rear-pic"). Due to the inherent differences between the two signal types, we developed two 1D CNNs with different structures for the respective extraction of features from steady-state EM signals and instantaneous impulse type EM signals.

The two model structures are respectively illustrated in Fig. 13(a) and 13(b), with many of the details suppressed for clarity. The first 6-layer CNN model is denoted as $Conv(64)-Conv(64)-Maxpool[4]-Conv(64)-Conv(64)-Maxpool[4]-Flattening-FC(512)-FC(128)$. Each convolution layer has 64 filters, the first Fully Connected (FC) layer has 512 neurons with ReLU activation, and the second FC layer has 128 neurons with linear activation. Two Maxpooling layers are used to reduce

the input dimensions by $1/4$. The second a 6-layer CNN model is denoted as $Conv(64)-Conv(64)-Maxpool[4]-Conv(256)-Conv(512)-Globalpool-Flattening-FC(512)-FC(128)$. The first two convolution layers have 64 filters, the third convolution layer has 256 filters, and the forth has 512. We replaced the second Maxpooling layer with Global Max Pooling in order to deal with instantaneous impulse type signals and thereby capture the activation along the time dimension.

Loss Function. The proposed network was optimized using a cost function capable of distinguishing between pairs. Specifically, the cost function was meant to keep matched pairs close, and ensure that unmatched pairs were separated in Euclidean distance by at least margin m computed in the embedded feature space. This was implemented using the margin-based contrastive loss function proposed in [21], which is defined as follows:

$$L(s_1, s_2, y) = \alpha(1-y)D_w^2 + \beta y \max(0, m - D_w)^2 \quad (1)$$

where s_1 is the exemplar EM signals, s_2 is the instance EM signals, and y is a binary indicator function denoting whether the two inputs belong to the same class; α and β are two constants and m is the margin equal to 1 in our case. $D_2 = ||f(s_1; w_1) - f(s_2; w_2)||_2$ is the Euclidean distance computed in the embedded feature space, f is an embedding function that maps 4-dimension EM signals (3 for directions and 1 for amplitude) to a real vector space through CNN, and w_1 and w_2 are the learned weights of each subnetwork. Unmatched pairs contribute to the loss function only if their distance is within margin m. This loss function brings matching pairs closer together in the feature space and moves non-matching pairs apart. Obviously, negative pairs with a distance exceeding the designated margin would not contribute to the loss (second part of Eq. 1).

3) Training and Testing:

Training. Two pseudo-Siamese networks were trained. The first network used the subnetwork in Fig. 13(a) with a training set derived from the collected dataset described in Sec. V-B, which included 20000 matched pairs for each label ("audio", "front/rear-video", and "front/rear-video-audio") with 60000 unmatched pairs selected at random. The second network used the subnetwork in Fig. 13(a), with a training dataset that included 20000 matched pairs for each label ("front/rear-pic") and 40000 unmatched pairs selected at random.

Both of the Pseudo-Siamese Networks were trained from scratch in an end-to-end manner using a batch size of 64 pairs per iteration. The weight parameters (i.e., filters) used in the networks were initialized uniformly in accordance with the protocol outlined in [22]. The gradients related to the feature vectors in the previous layer were computed using the contrastive loss function and back-propagated to the lower layers of the network. Once all of the gradients in all of the layers were computed, mini-batch stochastic gradient descent (SGD) was used to update the parameters of the network. Specifically, we employed the adaptive per-parameter update strategy referred to as RMSProp for the updating of weights [23]. The decay parameter for RMSProp was fixed at 0.95 (in accordance with previous works) and the margin for the contrastive loss function was maintained at 1. Training was performed through 20 epochs using an early stopping strategy based on saturation of the validation set performance. The initial learning rate was set at 0.002 and reduced by a factor of 0.9 following each epoch. The entire framework was implemented using Keras library with TensorFlow as the backend. Training using a GeForce GTX 1070 required 6.8 hours to finish the 20 epochs for each Pseudo-Siamese Network.

Testing. During the testing stage, we first paired each instance EM signal with the five labeled exemplar EM signals and passed on the results to the corresponding Pseudo-Siamese Network. Specifically, we divided the instance EM signal stream into segments of 5 seconds to match the duration of the "audio", "front/rear-video" and "front/rear-video-audio" exemplar EM signals. We then respectively paired the segmented instance signal with the three exemplar signals using the first trained Pseudo-Siamese Network. In a parallel operation, we divided the testing EM signal into segments of 3 seconds to match the duration of the "front/rear-pic" and then paired them with the two exemplar signals using the second trained network. When all of the exemplar signals had been paired, the final results were obtained by determining whether the testing EM signals contained EM signals associated with requests to access camera/microphone data.

TABLE II: Summary of mobile devices used in experiments.

Category	Device List
Training (20)	Huawei Nexus 6P, Huawei P10, Huawei P20 Pro, LG G7, LG V40, Samsung Galaxy A7, Samsung Galaxy S8, OnePlus 7 Pro, Xiaomi Redmi K20 Pro, Xiaomi Mi 8, Moto Z2 Force, Huawei Pad M5, Xiaomi Pad 4, iPhone 11, iPhone XS Max, iPhone 7 Plus, iPhone 6S, iPad 2018
Testing (10)	Samsung Galaxy S7, Huawei Mate 9, Xiaomi Mi 6, LG V30, iPhone X, Samsung Galaxy Tab A T510, Samsung Galaxy Tab S4, iPhone 8, iPhone 6, iPad Air2

VI. Evaluation

A. Experiment Setup

Mobile Devices: Table II lists the thirty mainstream mobile devices with different OS versions as representative examples (Android and iOS) in the experiments. Twenty of the devices were used to train the proposed MagDefender system, and ten were used to evaluate the performance when applied to new (i.e., unseen) devices.

Participants: We recruited ten users (4 females), each of which was allocated three mobile devices.

Training Dataset: We collected and labeled the EM dataset using the methods outlined in Sec. V-B.

Testing Dataset: The testing dataset included 1) exemplar EM signals from each mobile device and 2) actual

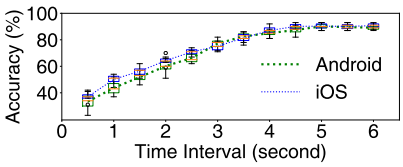
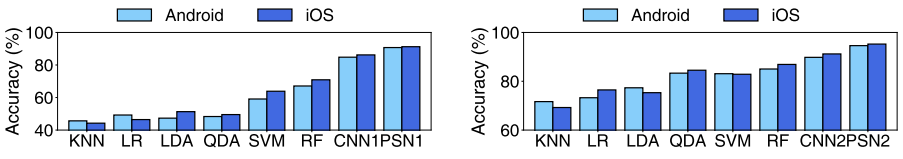
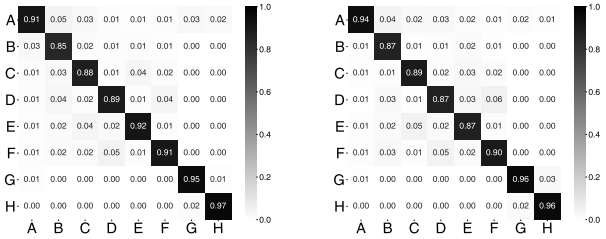


Fig. 14: Length time interval lengths vs. classification performance.



(a) Steady-state eavderopping (b) Instantaneous eavderopping

Fig. 15: Comparisons with conventional classifiers in terms of eavesdropping EM signal classification on trained devices.



(a) Android Devices (b) iOS Devices

Fig. 16: Confusion matrix while classifying different eavesdropping behaviors on different devices.

EM signals with labels. All thirty of the devices remained stationary on a table during the initial collection of exemplar EM signals generated while accessing camera/microphone data. Following this initial assessment, the participants were free to use their three devices in any manner that suited them, which resulted in randomly using multimedia related apps during which the corresponding built-in magnetometer readings were labeled as follows: "none", "audio", "front/rear-video", "front/rear-video-audio" and "front/front-pic".

B. Methodologies Evaluation

1) Time Interval Selection: As mentioned previously, the proposed CNN is tasked with extracting effective features in EM data over a fixed time interval. Assigning the time interval a small value would enable fine-grained differentiation between eavesdropping behaviors; however, this would greatly reduce the subsequent classification performance, due to a lack of time series information for feature extraction. For the steady-state eavesdropping classification task, we divided the training dataset according to different time interval values, and used the CNN (as Fig. 13(a)) with an additional softmax layer to extract features and classify them. Fig. 14 shows that when the time interval was set at over 5 seconds, the average accuracy of the classification model reached around 90% with negligible standard deviation. For the instantaneous eavesdropping classification task, we chose 3 seconds as the time interval value based on the maximum value of taking a picture among the while mobile devices.

2) Eavesdropping Behavior Classification: We compared the effectiveness of the proposed model in detecting instances of continuous eavesdropping (audio and video) with seven basic time-series classification algorithms (kNN [24], LR [15], LDA [25], QDA [26], SVM [17], RF [16] and CNN [27]). For the first six traditional algorithms, we used the following features in the time and frequency domain: min/max, min/max 10th/90th, mean and standard deviation over each sliding window on the 2-second EM signals; we used energy, domain frequency ratio, FFT peaks as frequency domain features. For CNNs, we selected the same structure as the subnetwork (Fig. 13(a) and 13(a)) used in pseudo-Siamese Network I/II (PSN1/2) with a softmax layer added to the output feature layer. The results are shown in Fig. 15(a) and 15(b), the convolution-based methods outperformed the conventional classifiers. This can be attributed to the fact that conventional classifiers rely on the quality of the extracted features, and simple feature extraction methods in the time-domain and frequency-domain often fail to extract key features of the EM signals. The excellent classification accuracy of CNN and our proposed method can be attributed to the powerful feature extraction capability of CNN. The confusion matrix of classifying seven eavesdropping behaviors and "no eavesdropping" is shown in Fig. 16, where A means "no eavesdropping", B means "audio", C and D means "front/rear-video", E and F means "front/rear-video-audio", G and H means "front/rear-pic". Overall, our proposed system obtained 94.71% of accuracy, and 95.61% of recall when detecting eavesdropping on Android devices, while 95.47% of accuracy, and 96.23% of recall on iOS devices.

C. Robustness on Unseen Devices

We also assessed the classification performance of the proposed scheme when applied to unseen devices. This was achieved using the testing dataset in Table II, which was collected using ten mobile devices. The experiment results are presented in Fig. 17. Overall, the conventional CNN with softmax layer performed very poorly (average 36.4% on steady-state types and 41.7% on instantaneous types) in this analysis, due mainly to the fact that the robustness of those models depends on the availability of a priori information related to the device. If we included a labeled EM dataset from the unseen device, it is very likely that CNN would provide better results; however, the process of collecting and labeling EM signals for new devices can be troublesome. The excellent performance of the proposed Pseudo-Siamese Networks when applied to unseen devices can be attributed to the structure of the network. When the classification results of the two networks were combined, the model achieved an average classification accuracy of 84.2% on the steady-state eavesdropping classification and 92.9% on the instantaneous type when applied to unseen devices. We added the exemplar EM signals of the unseen mobile

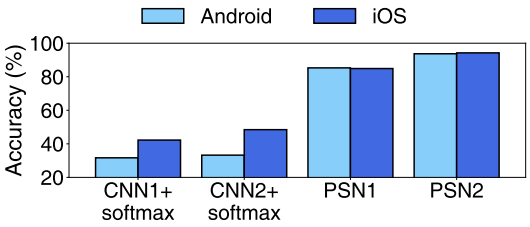


Fig. 17: Comparisons with the conventional CNN in terms of eavesdropping EM signal classification accuracy on the unseen devices.

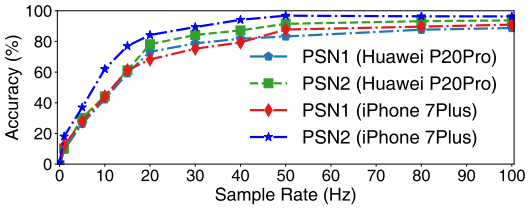


Fig. 18: Influence of the magnetometer sampling rate on eavesdropping behavior classification accuracy.

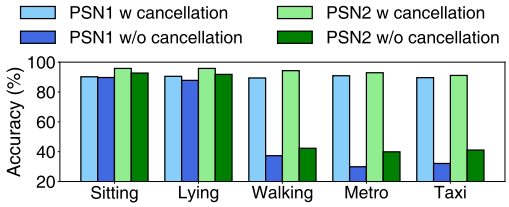
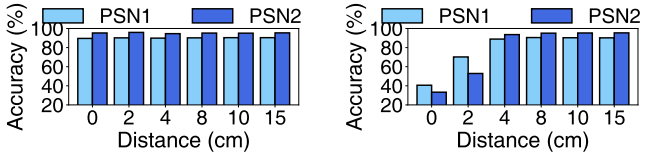


Fig. 19: EM signals related to eavesdropping classification accuracy with mobile devices in static position and in motion.



(a) Laptop (b) Washer

Fig. 20: Classification of EM signals: accuracy vs. distance to electronic devices.

device at the input end to enable the entire network to learn the characteristics of the EM signals generated by eavesdropping-related functions. We then compared the features extracted from the instance EM signals to obtain the final classification result. Fortunately, collecting exemplar EM signals for a new mobile device is a simple matter, requiring only a few minutes.

D. Influence of Sampling Rate

We sought to determine whether the sampling frequency of the built-in magnetometer would affect the classification accuracy of the MagDefender system. This was achieved using down sampling to alter sampling rates in the dataset of Huawei P20Pro and iPhone 7Plus. Then we retrained two PSNs models and evaluated the results. Fig. 18 lists the eavesdropping classification accuracy using data obtained over a range of sampling rates. At a sampling rate of $\geq 50Hz$, the accuracy of the system was close to the theoretical upper bound, indicating that MagDefender should perform well on most devices.

E. Robustness Against Geomagnetic Noise

We evaluated the effectiveness of the proposed geomagnetic noise cancellation method (detailed in the first part of Sec. IV-D), and testing our trained model in five real-world scenarios with different motion status. The volunteer was asked to use the mobile device freely (here we chose the iPhone 7Plus) in five different environments, while collecting readings from the built-in magnetometer and labeling them. Fig. 19 shows that the geomagnetic noise cancellation method improved the stability and accuracy of detection.

F. Influence of Nearby Electronic Devices

We also evaluated the impact of EM signals generated by two household electronic devices on the performance of the classifier in a real-world environment. We placed the mobile device (iPhone 7Plus) close to the electronic appliances while collecting EM signals for testing. As shown in Fig. 20(a), placing the mobile device close to a laptop computer had almost no effect on classification accuracy. As shown in Fig. 20(b), classification accuracy remained high as long as the device was kept $\geq 3cm$ from the washing machine. Placing the mobile device $1cm$ from large household appliances (microwave oven, washing machine) greatly hindered system performance, due to the fact that those appliances produce EM signals similar to a sine wave with wide variations in signal amplitude, which can lead to large deviations in the dataset during data preprocessing (i.e., normalization).

VII. Discussion

In this section, we discuss four other eavesdropping scenarios applicable to the MagDefender system. We also discuss the adversary model of MagDefender and how attackers could potentially cause the system to fail.

Other eavesdropping scenarios. Eavesdropping attacks are not limited to cameras and microphones. It is possible to access GPS data to obtain location information or utilize the Bluetooth channel to transfer data surreptitiously. Using third-party libraries, attackers have even been known to obtain screenshots or record screen in order to steal private information, such as photos and input PINs [28]. We conducted experiments involving the detection of four types of eavesdropping attack. Five mobile devices (in Table II) were used for training, and another two devices to evaluate the performance on unseen devices. We employed the same methods for the collection of exemplar and instance EM emissions with which to compile four datasets to train Pseudo-Siamese CNNs corresponding to the four eavesdropping strategies. As shown in TableIII, the proposed Pseudo-Siamese CNNs approach proved highly effective in detecting and differentiating among the additional four types of eavesdropping attack.

TABLE III: Classification results of MagDefender in identifying four types of eavesdropping attack. Accuracy1 means the classification accuracy on the trained devices. Accuracy2 means on the unseen devices.

Type	GPS	Buletooth Transmitting	Screen Recording	Screenshots
Accuracy1(%)	91.6	89.2	96.4	93.8
Accuracy2(%)	89.3	87.6	95.7	90.4

Potential countermeasures to inhibit MagDefender. In the following, we examine three methods by which the MagDefender could potentially be thwarted:

(1). Generating confounding EM signals: Attackers could alter CPU usage by running irregular code in the background in order to generate EM signals to mask evidence of eavesdropping behavior. This strategy could perhaps be used to mask actions that occur instantaneously (i.e., taking a photo or screenshot); however, applying it to continuous eavesdropping actions (e.g., recording audio or video) would disrupt the normal operation of the device, leaving it open to detection.

(2). Hacking the magnetometer and/or motion sensor: Theoretically, MagDefender could be defeated by modifying the hardware drivers of the magnetometer and motion sensors (in the Linux Kernel), or transplanting relevant core libraries (in the HAL layer) to provide erroneous data in the upper layers. However, magnetometer and motion sensors are commonly used for tasks such as gesture recognition. Modifying the readouts from these sensors would again disrupt the normal operation of the device, leaving it open to detection.

(3). Modifying the hardware: The internal electronic components could be physically shielded using ferromagnetic materials to hinder the detection of EM signals by the magnetometer; however, shielding all the components would not be feasible due to the tightness of device packaging. Another approach would be to limit the sampling rate of the magnetometer (*e.g.*, $5Hz$ and corresponding result is shown in Fig. 18) in order to compromise classification accuracy; however, this would also have a negative effect on the performance of legitimate apps.

VIII. Related Work

A. Eavesdropping Detection Techniques

Numerous researchers in the field of data security have addressed the issue of eavesdropping on mobile devices. Pan et al. scanned 17260 popular Android apps from a variety of app markets to identify instances of code associated with the leakage of data from multimedia sensors [28]. They examined media permissions, privacy policies, and outgoing network flows in an attempt to identify apps that upload audio recordings to the Internet without explicitly informing the user. Note however that they were unable to examine media exfiltration from background activity and completely disregarded iOS apps. They also failed to address the realistic attack scenario in which an app transforms audio recordings into less detectable text transcripts before sending out the information. By contrast, the MagDefender system is able to detect access to multimedia data directly from magnetometer readings. MagDefender can also detect the eavesdropping behaviors from the OS itself and the malicious apps in means of transplantation attacks.

B. Side-channel Sensing on Mobile Devices

Researchers have developed numerous side-channel sensing technologies, based on system information or data from sensors in mobile devices.

System information: Shmatikov [29] illustrated how the memory footprint left by browsers can be used for website fingerprinting. Several researchers have shown that power consumption traces [30], [31], [32] can be used to infer the opening of apps and websites. In [33], [34], the authors designed learning systems to automatically fingerprint apps using encrypted network traffic. Note however that it is very difficult for third-party apps to acquire the system kernel data (*i.e.*, CPU/memory usage, power consumption) with high sampling rates (usually $\leq 20Hz$). By contrast, the MagDefender system utilizes EM side-channel information instead of system kernel data to detect eavesdropping. Our approach does not require hardware modification, root/jailbreak operations, or additional system permissions.

Magnetometer data: The EM side channel, specifically, has been exploited for attacking electronic devices. In [39], the EM signals emitted by laptops were detected for the extraction of keys. In [40], EM side-channel signals were used to create a novel near-field communication system between mobile devices. In [41], mobile devices were characterized based on their near-field electromagnetic radiation signals. Several researches exploited the reaction of the built-in magnetometer to EM activity to infer apps and webpages opened on victim's laptop/phones [42], [43]. In this work, we prove that the built-in magnetometer readings accurately capture the EM emission signal generated by the app accessing the multimedia sensor data, and also show the feasibility of identifying the eavesdropping behaviours with the elaborate deep learning network.

IX. Conclusion

This study used the magnetometer built into mobile devices to detect instances of eavesdropping via the on-board microphones and cameras. Systematic analysis of the EM signals emitted by mobile devices made it possible for us to identify the specific EM signals generated when apps access camera/microphone data. We also developed two Pseudo-Siamese Networks to identify two types of eavesdropping: steady-state (via audio/video) and instantaneous (*e.g.*, taking a photo). Both networks proved highly effective in detecting instances of eavesdropping, even when encountering previously unseen mobile devices. Furthermore, the proposed MagDefender system can be implemented on any mobile device using any version of any OSes.

References

[1] S. Maheshwari. (2017) That game on your phone may be tracking what you're watching on tv. [Online]. Available: https://www.nytimes.com/2017/12/28/business/media/alphonso-app-tracking.html?_r=1

[2] I. Bogost. (2019) Facetime is eroding trust in tech - the atlantic. [Online]. Available: https://www.theatlantic.com/technology/archive/2019/01/apple-facetime-bug-you-cant-escape/581554/

[3] Statista. (2019) Android operating system share worldwide by os version from 2013 to 2019*. [Online]. Available: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[4] AndroidDevelopers. (2019) Behavior changes: all apps. [Online]. Available: https://developer.android.com/about/versions/pie/android-9.0-changes-all

[5] AppleDeveloper. (2018) Microphone background service. [Online]. Available: https://forums.developer.apple.com/thread/106415

[6] Z. Zhang, P. Liu, J. Xiang, J. Jing, and L. Lei, "How your phone camera can be used to stealthily spy on you: Transplantation attacks against android camera service," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 99–110.

[7] M. Huilgol. (2020) ios 14 has new orange and green indicators to warn you about apps using microphone and camera. [Online]. Available: http://www.iphonehacks.com/2020/06/ios-14-orange-light-recording-indicator.html

[8] Roman.T. (2020) Miui 12 review – is it the best android ui on the planet? [Online]. Available: https://www.thephonetalks.com/miui-12-review-features/

[9] T. Mehta. (2020) Access dots - ios 14 cam/mic access indicators! [Online]. Available: https://play.google.com/store/apps/details?id=you.in.spark.access.dots&hl=en_US

[10] AndroidDevelopers. (2020) Cameramanager. [Online]. Available: https://developer.android.com/reference/android/hardware/camera2/CameraManager

[11] ——. (2020) Audiomanager. [Online]. Available: https://developer.android.com/reference/android/media/AudioManager

[12] G. E. P. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American Statistical Association*, vol. 65, no. 332, pp. 1509–1526, 1970.

[13] . H. L. A. Williams, B. M., "Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results," *Journal of transportation engineering*, vol. 6, no. 129, pp. 664–672, 2003.

[14] S. Johansen, "Estimation and hypothesis testing of cointegration vectors in gaussian vector autoregressive models," *Econometrica: Journal of the Econometric Society*, pp. 1551–1580, 1991.

[15] S. Menard, *Applied logistic regression analysis*. Sage, 2002, vol. 106.

[16] . W. M. Liaw, A., "Classification and regression by randomforest," *R news*, pp. 18–22, 2002.

[17] Joachims, "Text categorization with support vector machines: Learning with many relevant features," *In European conference on machine learning*, pp. 137–142, 1998.

[18] 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 *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 15–26.

[19] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015.

[20] L. Dan, J. Zhang, Z. Qiang, and X. Wei, "Classification of ecg signals based on 1d convolution neural network," in *IEEE International Conference on E-health Networking*, 2017.

[21] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016.

[22] P. Krhenbühl, C. Doersch, J. Donahue, and T. Darrell, "Data-dependent initializations of convolutional neural networks," *Computer Science*, 2015.

[23] Y. N. Dauphin, H. D. Vries, J. Chung, and Y. Bengio, "Rmsprop and equilibrated adaptive learning rates for non-convex optimization."

[24] M. Tsypin and H. Röder, "On the reliability of knn classification," in *Lect Notes Eng Comput Sci. Proceedings of the World Congress on Engineering and Computer Science 2007 WCECS 2007*. Citeseer, 2007, pp. 24–26.

[25] S. Balakrishnama and A. Ganapathiraju, "Linear discriminant analysis-a brief tutorial," *Institute for Signal and information Processing*, vol. 18, pp. 1–8, 1998.

[26] P. A. Lachenbruch and M. Goldstein, "Discriminant analysis," *Biometrics*, pp. 69–85, 1979.

[27] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv: Learning*, 2018.

[28] E. Pan, J. Ren, M. Lindorfer, C. Wilson, and D. R. Choffnes, "Panoptispy: Characterizing audio and video exfiltration from android applications," 2018.

[29] S. Jana and V. Shmatikov, "Memento: Learning secrets from process footprints," pp. 143–157, 2012.

[30] Y. Chen, X. Jin, J. Sun, R. Zhang, and Y. Zhang, "Powerful: Mobile app fingerprinting via power analysis," *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pp. 1–9, 2017.

[31] L. Yan, Y. Guo, X. Chen, and H. Mei, "A study on power side channels on mobile devices," *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, pp. 30–38, 2015.

[32] P. Lifshits, R. Forte, Y. Hoshen, M. Halpern, M. Philipose, M. Tiwari, and M. Silberstein, "Power to peep-all: Inference attacks by malicious batteries on mobile devices," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 4, pp. 141–158, 2018.

[33] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," *IEEE Conference on Communications and Network Security (CNS)*, pp. 433–441, 2015.

[34] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2018.

[35] J. Quisquater and D. Samyde, "Electromagnetic analysis (ema): Measures and counter-measures for smart cards," *Lecture Notes in Computer Science*, vol. 2140, pp. 200—210, 2001.

[36] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," *cryptographic hardware and embedded systems*, vol. 2162, pp. 251–261, 2001.

[37] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side-channel(s)," *CHES*, pp. 29—45, 2002.

[38] A. Zajic and M. Prvulovic, "Experimental demonstration of electromagnetic information leakage from modern processor-memory systems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 3, no. 56, pp. 885—893, 2014.

[39] J. Longo, E. De Mulder, D. Page, and M. Tunstall, "Soc it to em: electromagnetic side-channel attacks on a complex system-on-chip." *In International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 620–640, 2015.

[40] H. Pan, Y.-C. Chen, G. Xue, and X. Ji, "Magnecomm: Magnetometer-based near-field communication," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '17. ACM, 2017, pp. 167–179.

[41] 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 *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 15–26.

[42] Y. Cheng, X. Ji, W. Xu, H. Pan, Z. Zhu, C. You, Y. Chen, and L. Qiu, "Magattack: Guessing application launching and operation via smartphone," *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp. 283–294, 2019.

[43] N. Matyunin, Y. Wang, T. Arul, K. Kullmann, J. Szefer, and S. Katzenbeisser, "Magneticspy: Exploiting magnetometer in mobile devices for website and application fingerprinting," in *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 135–149.